UNIVERSITY OF BELGRADE
FACULTY OF PHYSICS

Dušan Žigić

DEVELOPMENT OF THE DREENA MODEL FOR QUARK-GLUON PLASMA TOMOGRAPHY

Doctoral Dissertation

Belgrade, 2024

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФИЗИЧКИ ФАКУЛТЕТ

Душан Жигић

РАЗВОЈ ДРЕЕНА МОДЕЛА ЗА ТОМОГРАФИЈУ КВАРК-ГЛУОНСКЕ ПЛАЗМЕ

докторска дисертација

Београд, 2024. година

Thesis Defense Committee

Thesis advisor:

Dr. Magdalena Đorđević
Research Professor
Institute of Physics Belgrade
University of Belgrade

Thesis advisor:

Dr. Igor Salom
Research Professor
Institute of Physics Belgrade
University of Belgrade

Committee member:

Dr. Bojana Ilić
Assistant Research Professor
Institute of Physics Belgrade
University of Belgrade

Committee member:

Prof. Dr. Maja Burić
Professor
Faculty of Physics
University of Belgrade

Committee member:

Prof. Dr. Voja Radovanović
Professor
Faculty of Physics
University of Belgrade

Acknowledgements

This thesis was completed under the guidance of Dr. Magdalena Djordjević, Research Professor at the Laboratory for High Energy Physics and Dr. Igor Salom, Research Professor at the Group for Gravitation, Particles and Fields at the Institute of Physics Belgrade, University of Belgrade. The presented research was funded by the Ministry of Science, Technological Development and Innovations of the Republic of Serbia and by the European Research Council under the grant ERC-2016-COG: 725741.

I extend my sincerest thanks to my advisors, Dr. Magdalena Djordjević and Dr. Igor Salom, for their exceptional mentorship and guidance. Their collaborative spirit and willingness to share their knowledge and experience have been invaluable to my development as a researcher.

I am sincerely grateful to my colleagues Stefan, Veljko, Ana, Bithika, Bojana, Pasi and Jussi for their friendship, insightful discussions, and contributions to a stimulating work environment. Their support and collaboration have been invaluable throughout my academic journey.

I am deeply grateful to my parents, Zoran and Zorica, and to my sister, Milana, for their encouragement, and unwavering support throughout my academic journey. Their belief in me has been a constant source of strength and inspiration.

Abstract

The study of the quark-gluon plasma (QGP) in heavy-ion collisions provides a window into the fundamental properties of Quantum Chromodynamics (QCD), the theory governing strong interactions. This thesis focuses on understanding partonic energy loss mechanisms, medium properties, and the evolution of QGP. Combining theoretical advancements, computational frameworks, and experimental comparisons, the research highlights the dynamical energy loss formalism as a critical tool for probing the QGP and advancing our understanding of strongly interacting matter.

Chapters 1 and 2 introduce the theoretical foundations of QCD, exploring the QCD phase diagram, heavy-ion collisions, and the dynamical energy loss formalism. These chapters provide a comprehensive overview of confinement, asymptotic freedom, and deconfinement transitions, offering the context necessary for understanding energy loss in the QGP. The importance of experimental observables such as nuclear modification factors (R_{AA}) and flow coefficients (v_2) is discussed in relation to the formalism's role in modeling parton-medium interactions.

Chapter 3 focuses on testing the path-length dependence of energy loss mechanisms using the dynamical energy loss formalism. The chapter introduces appropriate observables and systems for such studies, emphasizing suppression ratios in smaller collision systems. Using the DREENA-C framework, the analysis investigates the robustness and reliability of these observables in capturing path-length effects, thereby validating their suitability for QGP tomography.

Chapter 4 details the development and application of the DREENA-B framework, which models the QGP as a longitudinally expanding medium. Results from DREENA-B are compared with experimental data, demonstrating its predictive power for high-$p_\perp$ observables. This chapter highlights the advantages of including medium evolution and temperature gradients in modeling energy loss phenomena.

Chapter 5 explores the initial stages of heavy-ion collisions, focusing on the theoretical and computational modeling of early-stage dynamics. The results provide insights into the early-time behavior of the QGP and its impact on partonic energy loss. This chapter bridges the gap between initial state modeling and the QGP's subsequent hydrodynamic evolution, emphasizing the role of initial conditions in determining final-state observables.

Chapter 6 introduces the DREENA-A framework, which incorporates full (2+1)D hydrodynamical temperature profiles to model QGP evolution. As a powerful tomography tool, DREENA-A enables precise extraction of QGP transport coefficients and energy loss analysis in varying collision geometries. The framework's accuracy in reproducing experimental results across different systems underscores its utility in QGP studies.

Chapter 7 examines the significance of higher-order flow harmonics in QGP tomography. Using the DREENA-A framework, event-by-event fluctuations, bulk evolution, and initial-state effects on

flow coefficients are investigated. The analysis demonstrates the importance of higher harmonics in revealing medium properties and constraining theoretical models.

This thesis combines advanced theoretical models, computational innovations, and experimental validation to enhance our understanding of QGP properties and energy loss mechanisms. The findings contribute to the broader field of heavy-ion physics, offering new tools and perspectives for exploring strongly interacting matter under extreme conditions.

Keywords: quark-gluon plasma, high-$p_\perp$ data, numerical simmulation
Research field: Physics
Research subfield: High-energy and nuclear physics
UDC number:

Сажетак

Проучавање кварк-глуонске плазме (КГП) у сударима тешких јона пружа увид у основна својства квантне хромодинамике, теорије која описује јаке интеракције. Ова дисертација се фокусира на разумевање механизама губитка енергије партона, својстава медијума и еволуције КГП-а. Комбиновањем теоријских унапређења, рачунских модела и упоређивањем са експерименталним подацима, истраживање истиче формализам динамичког губитка енергије као кључни алат за проучавање КГП-а и напредовање у разумевању јако интерагујуће материје.

Поглавља 1 и 2 представљаju теоријске основе квантне хромодинамике, истражују фазни дијаграм, сударе тешких јона и формализам динамичког губитка енергије. Ова поглавља пружају свеобухватан преглед конфинирања, асимптотске слободе и прелаза у слободно стање кваркова, пружајући контекст неопходан за разумевање губитка енергије у КГП-у. Дискутује се о важности експерименталних опсервабли, као што су фактор нуклеарне модификације (R_{AA}) и елиптички ток (v_2), у контексту улоге формализма у моделирању интеракција партона и медијума.

Поглавље 3 се фокусира на тестирање зависности механизама губитка енергије од дужине пређеног пута партона, користећи формализам динамичког губитка енергије. Поглавље уводи одговарајуће опсервабле и системе за таква истраживања, са нагласком на односе супресије у мањим сударним системима. Анализа користи DREENA-C модел за испитивање поузданости ових опсервабли у опису ефеката дужине пута, чиме се потврђује њихова погодност за томографију КГП-а.

Поглавље 4 описује развој и примену DREENA-B модела, који моделира КГП као средину која се лонгитудинално шири. Резултати DREENA-B модела упоређени су са експерименталним подацима, показујући предиктивну моћ модела за високо-енергијске опсервабле. Ово поглавље истиче предности укључивања еволуције медијума и температурних градијената у моделовање губитка енергије.

Поглавље 5 истражује почетне фазе судара тешких јона, фокусирајући се на теоријско и рачунарско моделирање динамике у раним фазама. Резултати пружају увид у понашање КГП-а у раним тренуцима еволуције и њихов утицај на губитак енергије партона. Ово поглавље премошћава јаз између моделирања почетних стања и хидродинамичке еволуције КГП-а, наглашавајући улогу почетних услова у одређивању финалних опсервабли.

Поглавље 6 уводи DREENA-A модел, који укључује (2+1)-димензионе хидродинамичке температурне профиле за моделирање еволуције КГП-а. Као моћан алат за томографију, DREENA-A омогућава прецизну екстракцију транспортних коефицијената КГП-а и анализу губитка енергије у различитим сударним геометријама. Тачност модела у репродук-

цији експерименталних резултата у различитим системима наглашава његову корисност у проучавању КГП-а.

Поглавље 7 анализира значај виших хармоника тока у томографији КГП-а. Коришћењем DREENA-A модела истражују се флуктуације, еволуција и ефекти почетног стања на коефицијенте тока. Анализа показује важност виших хармоника за откривање својстава средине и ограничавање параметара теоријских модела.

Ова дисертација комбинује напредне теоријске моделе, рачунарске иновације и експерименталну валидацију како би унапредила разумевање својстава КГП-а и механизама губитка енергије. Налази доприносе широј области физике судара тешких јона, нудећи нове алате и перспективе за проучавање јако интерагујуће материје у екстремним условима.

Кључне речи: кварк-глуонска плазма, високоенергијски подаци, нумеричке симулације
Научна област: Физика
Ужа научна област: Физика високих енергија и нуклеарна физика
УДК број:

Contents

List of figures

List of Tables

Chapter 1

Introduction

1.1 Structure of this thesis

Focus of this PhD thesis is the investigation of quark-gluon plasma. Modern cosmology suggests that quark-gluon plasma existed shortly after the Big Bang and is now produced in "Little Bangs," which are collisions of heavy ions at relativistic energies. The behavior of this unique form of matter is governed by quantum chromodynamics - theory of strong interaction. To provide the reader with a foundation for the topics explored in the subsequent chapters this section offers an overview of quantum chromodynamics, quark-gluon plasma, the theoretical models used to describe it and the collider experiments where it is generated.

The findings outlined in this thesis stem from the publications listed below [1, 2, 3, 4, 5]

- M. Djordjevic, D. Zigic, M. Djordjevic and J. Auvinen, *How to test path-length dependence in energy loss mechanisms: analysis leading to a new observable*, Phys. Rev. C **99**, no.6, 061902 (2019)
- D. Zigic, I. Salom, J. Auvinen, M. Djordjevic and M. Djordjevic, *DREENA-B framework: first predictions of R_{AA} and v_2 within dynamical energy loss formalism in evolving QCD medium*, Phys. Lett. B **791** (2019), 236-241
- D. Zigic, B. Ilic, M. Djordjevic and M. Djordjevic, *Exploring the initial stages in heavy-ion collisions with high-$p_\perp$ R_{AA} and v_2 theory and data*, Phys. Rev. C **101**, no.6, 064909 (2020)
- D. Zigic, I. Salom, J. Auvinen, P. Huovinen and M. Djordjevic, *DREENA-A framework as a QGP tomography tool*, Front. in Phys. **10**, 957019 (2022)
- D. Zigic, J. Auvinen, I. Salom, M. Djordjevic and P. Huovinen, *Importance of higher harmonics and v4 puzzle in quark-gluon plasma tomography*, Phys. Rev. C **106**, no.4, 044909 (2022)

1.2 Theory of strong interaction and Quantum Chromodynamics

The strong interaction, governed by Quantum Chromodynamics (QCD), is the force responsible for binding quarks and gluons into protons, neutrons, and other hadrons. QCD is a gauge theory based on the symmetry group $SU(3)_C$, where the mediators of the interaction—gluons—carry color charge. This unique property leads to phenomena such as confinement and asymptotic freedom [6, 7].

1.2.1 The QCD Lagrangian

The mathematical foundation of QCD is encoded in its Lagrangian, which describes the dynamics of quarks and gluons. The QCD Lagrangian is given by:

$$\mathcal{L} = \sum_{k=1}^{n_q} \bar{\psi}_k \left(i\gamma^\mu D_\mu - m_k \right) \psi_k - \frac{1}{4} F^a_{\mu\nu} F^{\mu\nu a}, \tag{1.1}$$

where ψ_k represents the quark fields, m_k is the quark mass, and $F_{\mu\nu}$ is the gluon field strength tensor. The covariant derivative, D_μ, ensures gauge invariance and is defined as:

$$D_\mu = \partial_\mu - igT^a A^a_\mu, \tag{1.2}$$

where A^a_μ are the gluon fields and T^a are generators of $SU(3)_C$ and g is the coupling constant. The gluon field strength tensor, which encapsulates gluon self-interactions, is expressed as:

$$F^a_{\mu\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g f^{abc} A^b_\mu A^c_\nu, \tag{1.3}$$

with f^{abc} being the structure constants of the $SU(3)_C$ group. This term introduces non-linearities unique to QCD, giving rise to phenomena such as the running coupling and gluon self-interactions [8].

1.2.2 Confinement and asymptotic freedom

The following equation can describe the potential between quarks in QCD [9]:

$$V(r) = -\frac{4}{3}\frac{\alpha_s}{r} + \sigma r \tag{1.4}$$

where α_s represents the strong coupling constant, r is the distance between quarks, and $\sigma \sim 0.18\text{GeV}$ [10]. This potential accurately explains the energy levels of heavy quarkonium systems, such as charmonium and bottomonium. Figure 1.1 illustrates the dependence of the QCD potential on the distance between quarks. At short distances, the first term of $V(r)$ dominates, resembling a Coulomb-like interaction. As the distance increases, the second term grows linearly with r and becomes significant. This linear dependence is directly related to the confinement of quarks within hadrons. The confinement mechanism can be visualized as color force lines forming a tube or string due to gluon-gluon interactions. As the string stretches, the energy increases proportionally to kr. When the energy reaches a critical threshold, producing a new quark-antiquark ($q\bar{q}$) pair becomes energetically favorable. This results in the fragmentation of the original string into two shorter strings, a process known as string fragmentation [12].

Conversely, at high energies or short distances, the coupling constant, α_s, decreases logarithmically, enabling quarks to behave as nearly free particles. This phenomenon, known as asymptotic

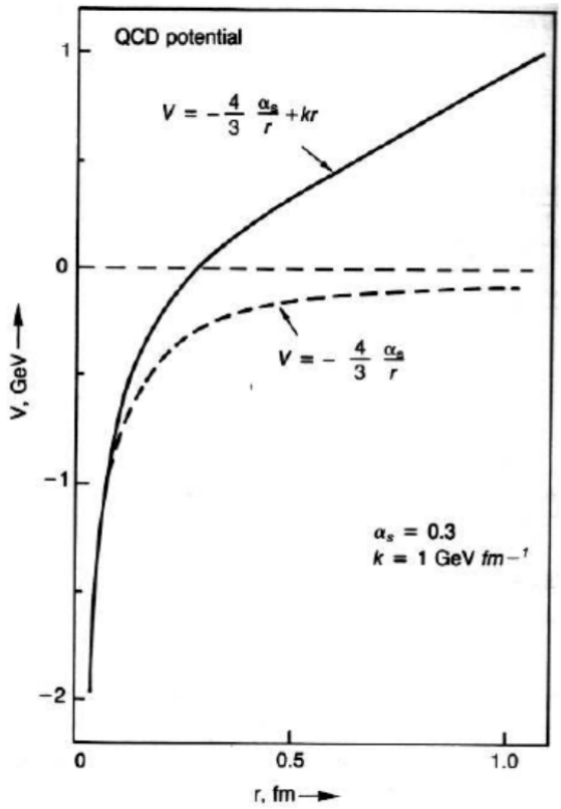

Figure 1.1: The linear potential between quarks as a function of separation distance, illustrating confinement. Figure adapted from [11].

freedom [13]. From renormalization group equation, running coupling can be expressed in terms of β function as [13]:

$$\alpha_s\left(Q^2\right) \propto \frac{1}{\beta_0 ln\left(Q^2/\Lambda^2_{QCD}\right)}, \tag{1.5}$$

where β_0 is is one-loop approximation, and Λ_{QCD} is the QCD scale parameter [6]. Figure 1.2 shows the running coupling constant α_s , which decreases with increasing Q^2, the energy of the process involved.

1.2.3 Non-perturbative techniques

At low energies, where the coupling constant becomes large, perturbative methods are insufficient. Lattice QCD provides a powerful non-perturbative approach by discretizing spacetime, enabling numerical simulations of QCD phenomena, such as the equation of state (EoS) and the nature of phase transitions [15]. Additionally, effective models like the Polyakov-loop extended Nambu-Jona-Lasinio (PNJL) model approximate QCD dynamics by incorporating thermal and density effects [16].

These approaches have been critical for understanding QCD under extreme conditions, such as the formation of the quark-gluon plasma (QGP) shortly after the Big Bang and the behavior of matter in neutron stars [17, 18].

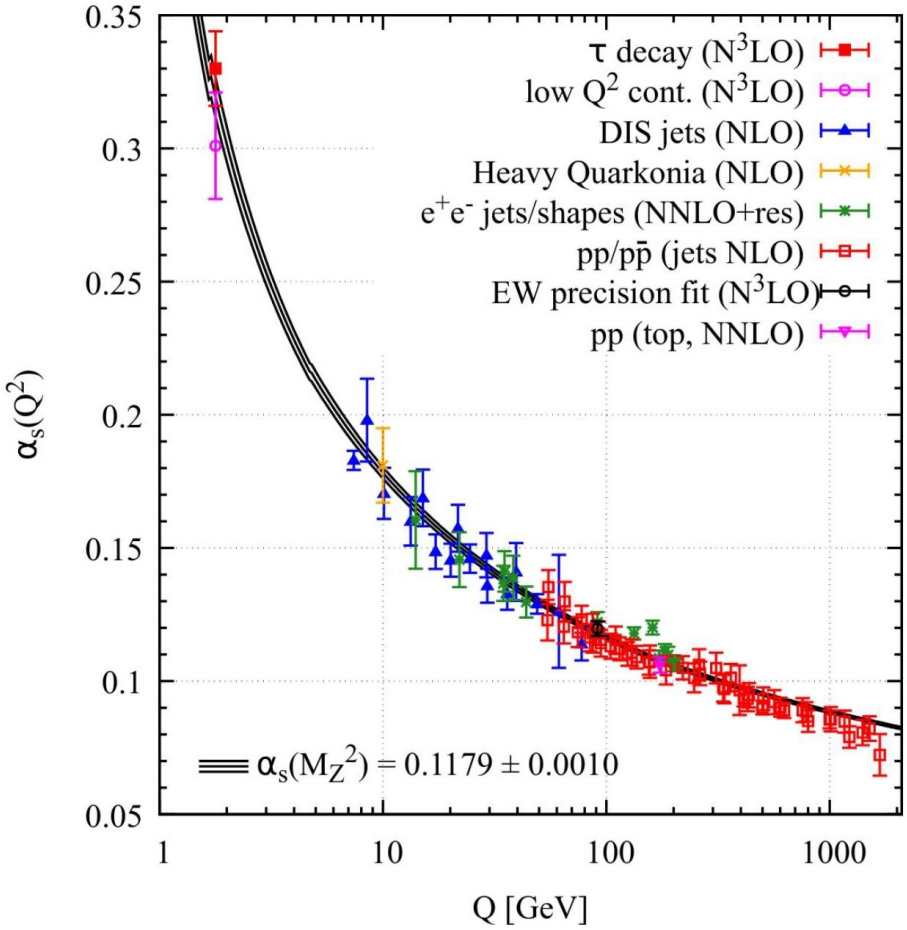

Figure 1.2: QCD running coupling, α_s as a function of momentum scale, Q. Figure adapted from [14].

1.3 QCD phase diagram

The QCD phase diagram encapsulates the possible states of strongly interacting matter under varying temperature (T) and baryon chemical potential (μ_B). This framework is essential for exploring the transitions between confined hadronic matter and deconfined quark-gluon plasma (QGP), as well as other exotic phases predicted by Quantum Chromodynamics (QCD).

1.3.1 Phases of QCD matter

At low T and μ_B, quarks and gluons are confined within hadrons due to the strong interaction, forming what is referred to as the hadronic phase. This phase is stable under ordinary conditions and dominates in the current universe. However, as the temperature increases, the system undergoes a transition to the QGP, where quarks and gluons exist in a deconfined state. Lattice QCD simulations have shown that this transition at low μ_B is a smooth crossover characterized by a gradual change in thermodynamic quantities, such as the energy and entropy density. The critical temperature for this crossover is $T_c \approx$ 155-160MeV [15, 19].

The chiral condensate $\langle\bar{\psi}|\psi\rangle$ serves as an order parameter for this transition. In the hadronic phase, $\langle\bar{\psi}|\psi\rangle$ is non-zero due to spontaneous chiral symmetry breaking. As the temperature rises, $\langle\bar{\psi}|\psi\rangle$ diminishes, signaling the restoration of chiral symmetry in the QGP phase. This crossover behavior is illustrated in Figure 1.3, which shows the QCD phase diagram with the crossover region

delineated.

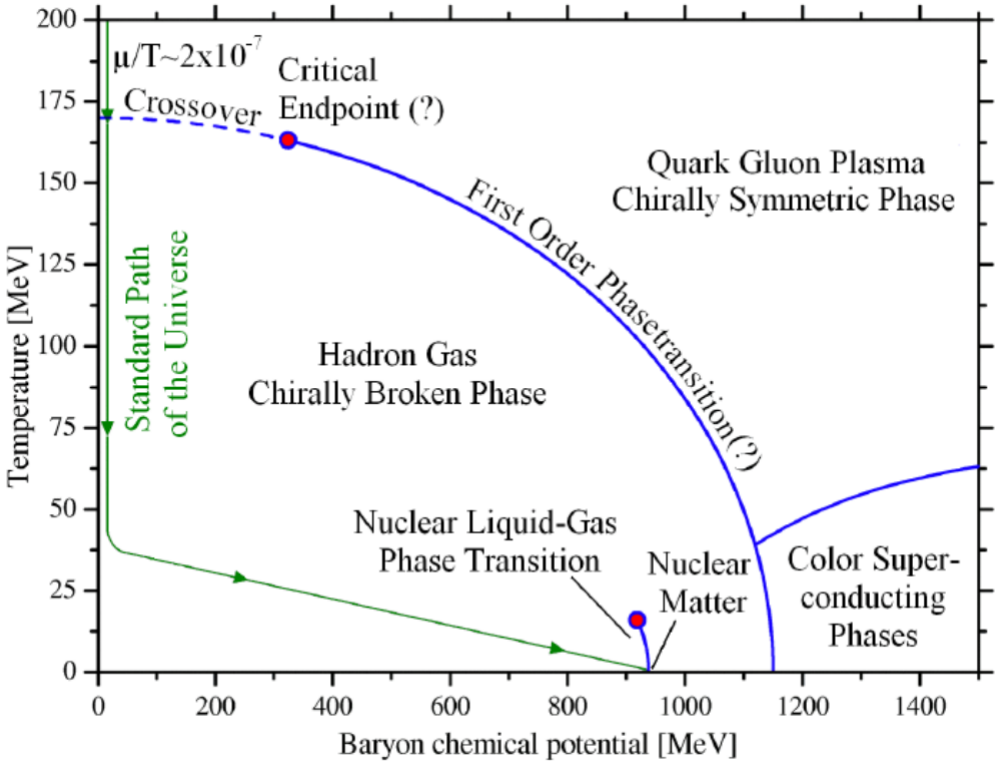

Figure 1.3: The QCD phase diagram with the crossover transition at low μ_B, as predicted by lattice QCD. Figure adapted from [20].

It is widely considered that the QGP created in heavy-ion collisions at RHIC (Relativistic Heavy Ion Collider) and LHC (Large Hadron Collider) exhibits properties of a nearly perfect fluid, with an exceptionally low shear viscosity-to-entropy density ratio (η/s) close to the conjectured lower bound of $1/4\pi$ [21]. Observables such as elliptic flow coefficients and jet quenching provide experimental evidence for the existence of QGP, linking it to the high-temperature region of the phase diagram.

1.3.2 First-Order phase transition and critical point

The QCD phase diagram predicts a first-order phase transition at higher μ_B, relevant to systems with extreme baryon densities (e.g., neutron star interiors) [22]. This transition is characterized by a discontinuity in the energy density and other thermodynamic variables as the system crosses the phase boundary. The first-order transition line terminates at the critical point, a singularity where the nature of the transition changes, and thermodynamic fluctuations in conserved quantities, such as baryon number and charge, diverge [22].

Experimental programs, mainly the RHIC Beam Energy Scan (BES), have focused on identifying signatures of the critical point. Observables such as net-proton fluctuations, kurtosis, and skewness [23] provide critical insights into the location of the endpoint. Data measurement fluctuations from BES suggests possible critical behavior in the intermediate collision energy range [24]. Enhanced fluctuations in these measurements indicate proximity to the critical point, though further experimental precision is required to confirm its exact location.

1.3.3 Exotic phases at high density

At extremely high μ_B and low T, QCD predicts the emergence of exotic phases, such as color superconducting phases. In these phases, quarks form Cooper pairs through attractive interactions mediated by gluons, resulting in phenomena analogous to electron pairing in conventional superconductors. Some of these states are the 2-flavor color superconductor (2SC) and color-flavor locked (CFL) phases [18].

These phases are particularly relevant to the interiors of neutron stars, where densities are several times higher than nuclear saturation density [25]. The presence of these phases can influence the mass, radius, and cooling properties of neutron stars, connecting the QCD phase diagram to astrophysical observations.

1.3.4 Challenges and future directions

Despite significant advancements, many aspects of the QCD phase diagram remain uncertain, particularly at high μ_B. Lattice QCD calculations are hindered in this region due to the sign problem [26], which complicates the evaluation of the fermion determinant. To address these limitations, effective models like the Polyakov-loop extended Nambu-Jona-Lasinio (PNJL) model and Dyson-Schwinger equations have been employed to approximate the behavior of QCD matter at high density [16].

Future experiments, including upgrades to RHIC and the development of new facilities like the Facility for Antiproton and Ion Research (FAIR) and the Nuclotron-based Ion Collider Facility (NICA), aim to explore the high-μ_B region in greater detail. These efforts will provide new data on phase transitions, critical phenomena, and the properties of QCD matter under extreme conditions.

1.4 Heavy-ion collisions

Heavy-ion collisions serve as a powerful tool for investigating strongly interacting matter under extreme temperature and density conditions, enabling the recreation of the quark-gluon plasma (QGP). Cutting-edge experiments conducted at facilities like the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory and CERNS's Large Hadron Collider (LHC) have significantly advanced our understanding of QCD matter. These collisions offer a window into the QCD phase diagram and facilitate the exploration of key phenomena, including deconfinement, chiral symmetry restoration, and the emergence of collective behavior.

1.4.1 Space-time evolution of heavy-ion collisions

The dynamics of heavy-ion collisions are typically described in terms of their space-time evolution. The process can be divided into several key stages, as illustrated in Figure 1.4:

1. **Initial Stages:** The colliding nuclei generate a dense system dominated by gluon fields. This stage is characterized by the formation of a far-from-equilibrium system, described by the different models such as IP-Glasma [27, 28] and EKRT [29, 30, 31].

2. **Thermalization and QGP Phase:** The system quickly equilibrates and transitions into the QGP phase, a strongly coupled state exhibiting fluid-like behavior with a low viscosity-to-entropy density ratio (η/s) near critical temperature (T_c) [32].

3. **Hadronization:** As the QGP expands and cools, quarks and gluons recombine into hadrons, marking the transition to the hadronic phase.

4. **Freeze-Out:** The system undergoes chemical freeze-out (where inelastic collisions cease) and kinetic freeze-out (where elastic collisions stop). The final hadrons propagate to detectors.

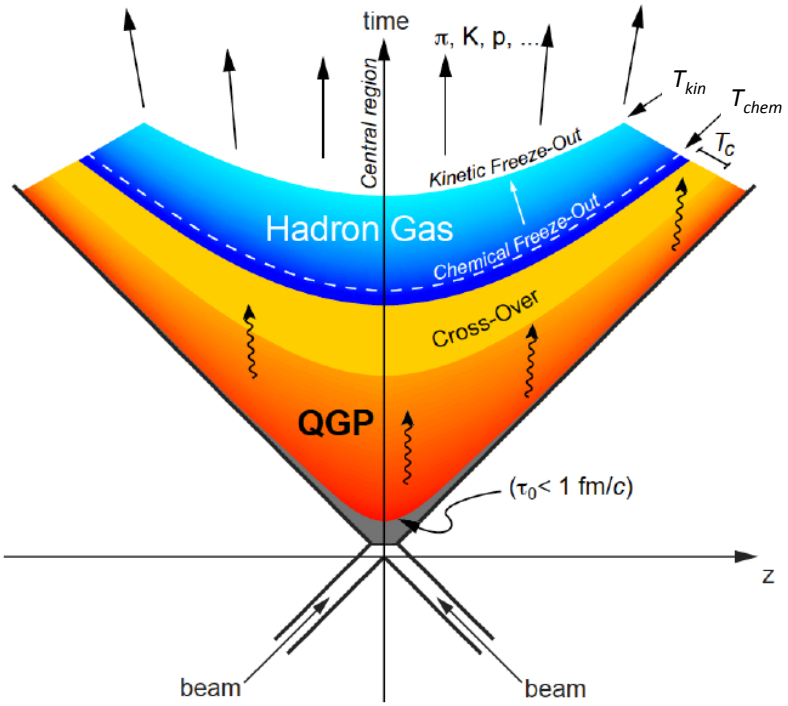

Figure 1.4: A space-time diagram depicting the various stages of QCD matter evolution in heavy-ion collisions. The beam axis is labeled as z, with time represented by t. The hyperbolic curves separate the different stages and correspond to constant Lorentz-invariant proper time. Figure adapted from [33].

This sequence, represented schematically in Figure 1.4, encapsulates the complex evolution of QCD matter from its initial gluon-dominated state to the final observable particles.

1.4.2 Participants, spectators, and reaction plane

In a heavy-ion collision, the colliding nuclei partially overlap, dividing nucleons into two categories: participants and spectators. Participants are the nucleons that undergo interactions, while spectators do not interact and continue moving along the beam path. This distinction is important as it influences the geometry of the collision and plays a critical role in the initial energy distribution of the system [34]. The degree of overlap between the nuclei is measured by the impact parameter (b), which is the transverse distance between the centers of the two colliding nuclei (Figure 1.5).

In heavy-ion collisions, the reaction plane (Figure 1.6) is defined by the impact parameter vector (b) and the beam axis, representing the geometric plane of the initial collision. It is determined by the global spatial anisotropy in the area where the two colliding nuclei overlap. It serves as a reference for analyzing anisotropic flow and collective behavior in the produced matter. In contrast, the participant plane (Figure 1.6) is determined by the spatial distribution of the participant nucleons, i.e. those involved in the interaction in the transverse plane. Unlike the reaction plane, the participant plane accounts for fluctuations in the positions of these nucleons, leading to deviations from the ideal

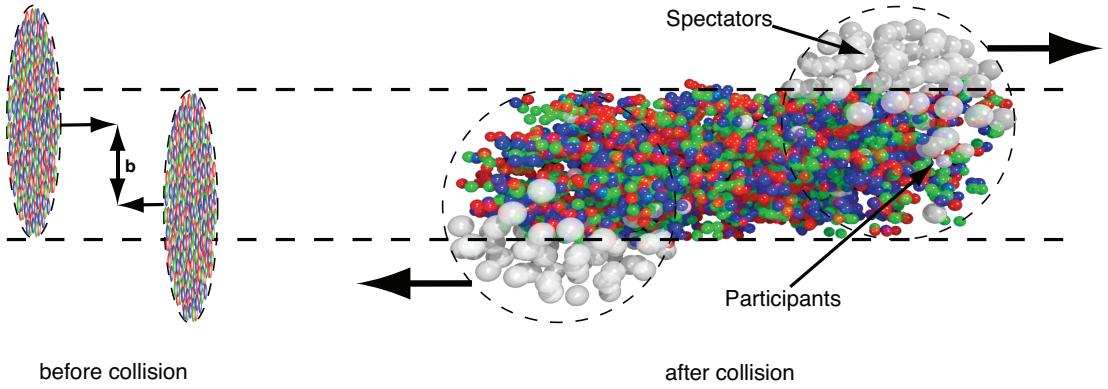

Figure 1.5: Left:Colliding ions just before the interaction, illustrating the impact parameter (b) Right: The participant zone, where new matter is created, and the spectator region, consisting of unaffected nucleons. Figure adapted from [35].

geometric plane. These fluctuations play a significant role in understanding event-by-event variations in the initial energy density distribution. They are crucial for studying higher-order flow harmonics, such as triangular and quadrangular flow.

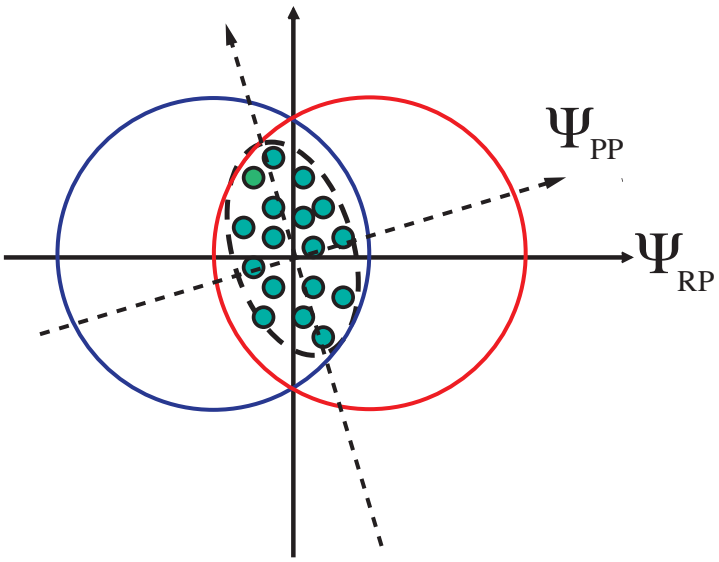

Figure 1.6: Reaction, Ψ_{RP}, and participant, Ψ_{PP}, planes coordinate systems. Figure adapted from [35].

1.4.3 Rapidity and pseudorapidity

Rapidity [36] is a measure of a particle's velocity along the beam axis and is widely used in heavy-ion collision analyses due to its Lorentz invariance. It is defined as:

$$y = \frac{1}{2} ln \left(\frac{E + p_z}{E - p_z} \right), \tag{1.6}$$

where E is the particle's total energy, and p_z is the longitudinal momentum. Rapidity simplifies the comparison of particle production in different reference frames, making it an essential variable in describing the longitudinal dynamics of a collision.

In experimental analyses, pseudorapidity (η) is often used as an approximation to rapidity, particularly for highly relativistic particles where $E \gg m$. Pseudorapidity [36] is defined as:

$$\eta = -\ln\left(tan\frac{\theta}{2}\right) \tag{1.7}$$

where θ is the angle between the particle's momentum and the beam axis. Unlike rapidity, pseudorapidity depends only on the particle's angular distribution, making it straightforward to calculate in detectors.

The distribution of produced particles in rapidity or pseudorapidity provides critical insights into the collision's energy deposition and thermalization. At mid-rapidity ($y \approx 0$), the system reaches the highest energy density, reflecting the thermalized QGP. Forward rapidity regions ($|y| \gg 0$) probe the fragmentation of the initial nuclei and the contributions from spectator nucleons.

1.4.4 Particle multiplicity and centrality

In heavy-ion collisions, particle multiplicity represents the total number of particles produced during the collision. It strongly correlates with the collision's energy density, centrality, and thermalization. Typically measured as the number of charged particles (N_{ch}) within a specific pseudorapidity or rapidity range, multiplicity offers insights into the collision dynamics and properties of the created medium.

At mid-rapidity ($y \approx 0$), the charged-particle multiplicity reflects the thermodynamic properties of the created system, including the initial energy density. Experimental measurements at RHIC and LHC confirm that higher multiplicities correspond to larger initial energy densities, especially in central collisions [37, 38].

Multiplicity strongly depends on the collision centrality, which quantifies the overlap of the colliding nuclei. Central collisions exhibit the highest particle multiplicities due to the maximum number of participant nucleons (N_{part}) involved in the interaction. Peripheral collisions, where the overlap is minimal, produce significantly fewer particles. This correlation is well-documented in experiments at RHIC and LHC, where multiplicity measurements have revealed a nearly linear relationship between $N_{\text{ch}}/d\eta$ and N_{part} for central collisions [39, 40].

Multiplicity fluctuations provide additional information about the collision dynamics. In central collisions, narrower distributions are observed, reflecting more stable collective behavior, while peripheral collisions exhibit broader distributions due to larger relative fluctuations in the number of participants and energy deposition.

At RHIC ($\sqrt{s_{NN}} = 200$GeV) and LHC ($\sqrt{s_{NN}} = 2.76$TeV), charged-particle multiplicity has been extensively studied to characterize the QGP. For example, in central Pb-Pb collisions at the LHC, dN_{ch} approaches values significantly higher than those at RHIC, reflecting the increased energy density at higher collision energies [38]. These studies provide vital benchmarks for hydrodynamic models and the study of QGP properties.

Centrality quantifies how head-on or central a collision is between two colliding nuclei in heavy-ion collisions [36]. It is a crucial concept for classifying collisions based on the extent of overlap and interaction between the nuclei. During a heavy-ion collision, the degree of overlap depends on the impact parameter (b), the transverse distance between the bases of the colliding nuclei. A small impact parameter corresponds to a more central collision with significant overlap, while a large impact parameter indicates a peripheral collision with minimal interaction.

Centrality is typically expressed as a percentage of the total nuclear cross-section, with collisions grouped into centrality classes or bins [39]. These classes represent specific ranges of impact pa-

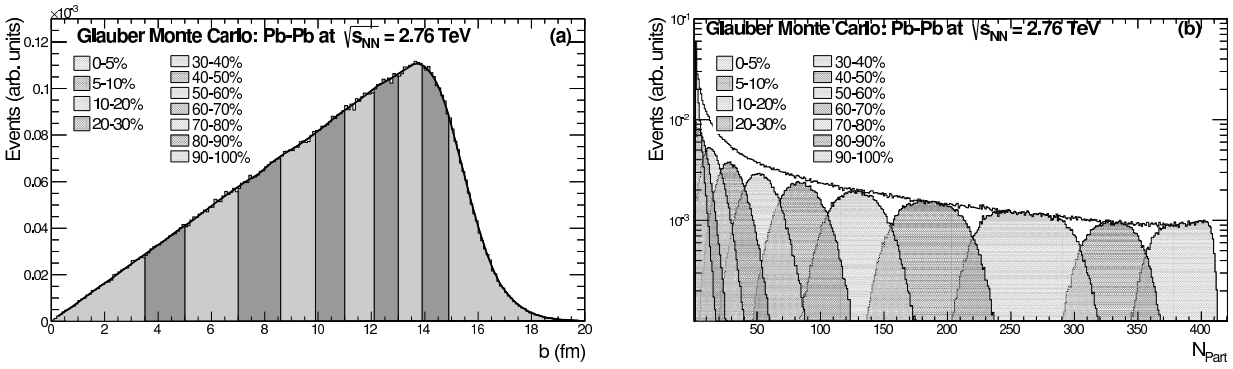

Figure 1.7: An example of categorizing events into various centrality groups is depicted. The figures present outcomes generated using the Monte Carlo Glauber model [41] for Pb+Pb collisions at $\sqrt{s_{NN}} = 2.76$TeV. The graph on the left illustrates the distribution of impact parameters, where larger values of b correspond to higher centrality percentages. The graph on the right displays the distribution of participant nucleons, showing that a greater number of participants corresponds to lower centrality percentages. The different centrality groups are labeled on the graphs. Figure adapted from [41].

rameters. The most central collisions, characterized by maximal overlap, are assigned to the lowest centrality bins (e.g., 0-5% centrality), whereas the most peripheral collisions, with minimal overlap, fall into the highest centrality bins (e.g., 90-100% centrality).

The centrality percentile, c, for a heavy ion collision can be mathematically defined as [41]:

$$c = \frac{\int_0^b \frac{d\sigma}{db'} db'}{\int_0^\infty \frac{d\sigma}{db'} db'} = \frac{1}{\sigma_{AA}} \int_0^b \frac{d\sigma}{db'} db', \tag{1.8}$$

where b is the impact parameter, $d\sigma/db'$ is the distribution of the impact parameter and σ_{AA} is the total inelastic nucleus-nucleus cross section.

Figure 1.7 presents an example of how theoretically generated collision events are sorted into centrality bins.

1.4.5 Nuclear modification factor

The nuclear modification factor (R_{AA}) is a key observable used to study the behavior of high-$p_\perp$ particles in heavy-ion collisions. It provides a quantitative measure of how particle production in heavy-ion collisions differs from that in proton-proton collisions, scaled by the number of binary nucleon-nucleon collisions. It is defined as [42]:

$$R_{AA} = \frac{\text{Yield in A+A collisions}}{\langle N_{coll} \rangle \times \text{Yield in p+p collisions}}, \tag{1.9}$$

where $\langle N_{coll} \rangle$ is the average number of binary collisions in the nucleus-nucleus interaction, and the denominator represents the expectation for particle production if no nuclear medium effects were present.

R_{AA} value less than 1 indicates suppression, often attributed to jet quenching, a phenomenon where high-energy partons lose energy as they traverse the dense QGP medium. This energy loss occurs via interactions with the QGP, such as gluon bremsstrahlung or collisional energy dissipation. Suppression of high-$p_\perp$ particles has been observed at RHIC and LHC, confirming the presence of a dense, strongly interacting medium [43].

Interestingly, at intermediate $p_\perp$, enhancement in R_{AA} can occur due to phenomena like Cronin effects [44], where initial-state scattering broadens parton transverse momentum, and coalescence [45], where quarks recombine to form hadrons. These effects are sensitive to the underlying dynamics of particle production and medium modification. However, these effects diminish at large enough transverse momentum ($p_\perp \geq 8$GeV).

R_{AA} also varies with collision centrality, providing insights into the geometry and density of the medium. Central collisions, which produce the densest QGP, show the strongest suppression at high $p_\perp$, while peripheral collisions exhibit weaker suppression. This centrality dependence supports the interpretation of jet quenching as a medium-induced phenomenon. Experimental studies, such as those conducted at RHIC and LHC, have demonstrated this relationship across various collision systems, including Au+Au and Pb+Pb collisions [40].

1.4.6 Collective flow

Collective flow is one of the most important observables in heavy-ion collisions, providing direct evidence of the hydrodynamic behavior of the QGP. It refers to the anisotropic expansion of the medium created in the collision, driven by pressure gradients established in the initial stages. Flow is quantified using Fourier decomposition of the azimuthal particle distribution [34, 46]:

$$\frac{dN}{dyd^2p_\perp} = \frac{dN}{2\pi dyp_\perp dp_\perp}\left[1 + \sum_n 2v_n cos\left(n\left(\phi - \psi_n\right)\right)\right], \tag{1.10}$$

where ϕ is the azimuthal angle, ψ_n is the event-plane angle, and v_n are the Fourier coefficients that characterize different flow components.

Even though there are infinite Fourier coefficients, most importnat ones are: *i)* elliptic flow, v_2, arises from the initial spatial anisotropy in non-central collisions. Due to the almond-shaped overlap region of the colliding nuclei, the pressure gradients are stronger along the short axis, leading to preferential expansion in this direction. Elliptic flow is a key observable for studying the QGP's viscosity, as it is sensitive to the medium's shear viscosity-to-entropy density ratio, η/s [32]. *ii)* triangular, v_3 and higher order flow coefficients arise from initial-state fluctuations in the positions of participant nucleons. These fluctuations create localized hot spots in the energy density distribution, resulting in azimuthal anisotropies that are independent of the reaction plane. Triangular flow has been instrumental in understanding the initial state and the response of the QGP to fluctuations [47]. *iii)* radial flow refers to the isotropic expansion of the system due to thermal pressure gradients. It results in a characteristic "blue shift" of the particle momentum spectra, with heavier particles exhibiting higher transverse momenta due to their stronger coupling to the collective motion of the medium [48].

Flow measurements strongly depend on particle species. Light particles, such as pions, exhibit stronger flow signals than heavier particles like protons and kaons. This mass ordering arises because heavier particles acquire more momentum from the collective expansion, but their larger masses result in smaller transverse velocities. This behavior reflects the integrated dynamics of the QGP and the hadronic phase.

Collective flow also shows a strong dependence on collision centrality. In central collisions, where the overlap region is more symmetric, v_2 is reduced, while higher-order flow components (v_3, v_4) are enhanced due to fluctuations. In peripheral collisions, elliptic flow dominates due to the pronounced almond-shaped geometry of the initial overlap region.

Elliptic flow and higher-order harmonics are sensitive to the QGP's shear viscosity-to-entropy density ratio (η/s), a critical parameter for characterizing the QGP as a near-perfect fluid. Hydrody-

namic simulations indicate that near critical temperature, T_c, lower η/s values lead to stronger flow signals, providing key insights into the properties of the QGP.

Experiments at RHIC and LHC have provided extensive data on flow coefficients, spanning a wide range of collision energies, system sizes, and particle species. These measurements confirm the collective behavior of the QGP and support the hydrodynamic description of its evolution.

At high $p_\perp$, v_2 reflects the anisotropy in the suppression of high-$p_\perp$ particles traversing the almond-shaped medium created in non-central heavy-ion collisions. This suppression arises from the path-length dependence of parton energy loss in the anisotropic medium, where particles moving along the shorter axis lose less energy compared to those traversing the longer axis. Consequently, high-$p_\perp$ v_2 originates from jet-medium interactions and energy loss mechanisms, distinct from the hydrodynamic flow responsible for low-$p_\perp$ v_2. This distinction highlights the importance of studying high-$p_\perp$ v_2, which provides unique insights into the QGP's temperature profile and anisotropy at higher temperatures.

In addition to R_{AA} (the nuclear modification factor) discussed earlier, high-$p_\perp$ v_2 and higher-order flow harmonics, such as v_3 and v_4, serve as complementary probes for the QGP's bulk properties. These observables combine the sensitivity of R_{AA} to the medium's density and temperature with the directional information provided by flow harmonics, enabling a more comprehensive characterization of the QGP.

A major challenge lies in accurately modeling the interplay between parton energy loss and medium properties to simultaneously describe R_{AA} and v_2 across various collision systems and energies. This challenge motivated the development of the DREENA (Dynamical Radiative and Elastic ENergy loss Approach) framework. DREENA integrates radiative and collisional energy loss mechanisms within a dynamically evolving medium, offering a robust tool for capturing the complexity of jet-medium interactions. It provides a systematic approach to exploring the QGP's bulk properties using high-$p_\perp$ observables and addressing discrepancies in experimental v_2 data, thereby refining theoretical descriptions of QGP dynamics.

Experiments at RHIC and LHC have provided extensive data on low-$p_\perp$ and high-$p_\perp$ R_{AA}, v_2, and higher harmonics across various collision energies, system sizes, and particle species. Before the development of the DREENA framework, these data were often treated as separate fields of study. DREENA enabled the integration of these domains, allowing for a unified approach to constraining the properties of this fascinating state of matter.

Chapter 2

Methodology

2.1 The dynamical energy loss formalism

The dynamical energy loss formalism [49] provides a comprehensive framework for understanding how high-energy partons lose energy as they traverse the quark-gluon plasma (QGP). This formalism addresses the limitations of static models by considering the QGP as a dynamic medium, characterized by moving partonic constituents, temperature gradients, and finite-size effects. It combines collisional (elastic) and radiative (inelastic) energy loss mechanisms, enabling precise predictions of high-$p_{\perp}$ observables such as the nuclear modification factor (R_{AA}) and azimuthal anisotropy (v_2).

Early models like the GLV (Gyulassy-Levai-Vitev) [50, 51, 52, 53, 54] formalism focused on radiative energy loss in a static medium. The GLV model described the energy dissipation of massless partons through multiple scatterings in a QGP. Building on this, Djordjevic and Gyulassy introduced the DGLV [55] model, which incorporated the effects of quark mass into radiative energy loss calculations. What they discovered was that the general result is accured by shifting the frequencies in the GLV series by $\left(m_g^2 + x^2 M^2\right) / (2xE)$, where m_g is the effective gluon mass while x is the fractional energy of the radiated gluon, M is the quark mass and E is the initial jet energy. Furthermore, DGLV recovers GLV results in the massless limit.

Despite its successes, the DGLV formalism assumed a static medium and neglected collisional energy loss. Experimental results from RHIC [56, 57] revealed that radiative mechanisms alone could not account for the observed suppression patterns, particularly for heavy-flavor mesons. These limitations motivated the development of the dynamical energy loss formalism, which integrates dynamic medium properties and combines radiative and collisional contributions. Dynamical energy loss formalism has the following features:

- Radiative energy loss [58] suitable to both light and heavy flavour.
- Collisional energy loss [59], formulated within the same theoretical framework, which is also applicable to both light and heavy flavor particles
- The formalism models finite size and temperature QCD medium, comprised of dynamic (i.e. moving) partons, distinguishing it from approaches that rely on static approximations and vacuum-based propagators [43, 60, 50, 61].

- The calculations utilize a generalized Hard-Thermal-Loop approach [62, 63], with naturally regulated infrared divergences [58, 64, 65].
- The formalism takes into account finite magnetic mass [66] and running coupling [49].
- The soft-gluon approximation was recently relaxed, broadening the formalism's range of applicability [67].

Previous studies [68] demonstrated that all the aforementioned model components influence the high-$p_\perp$ data and are therefore essential for accurate explanations.

Collisional energy loss arises from elastic scatterings between high-energy partons (quarks and gluons) and the constituents of the quark-gluon plasma (QGP). Unlike radiative energy loss, which involves the emission of gluons, collisional energy loss is characterized by momentum transfer through direct interactions with the QGP's dynamic medium particles.

In the dynamical energy loss formalism, collisional energy loss is calculated using a temperature-dependent framework that incorporates the dynamic properties of the QGP. The effective gluon propagator [59] plays a central role in these calculations:

$$D^{\mu\nu}\left(\omega,\vec{\boldsymbol{q}}\right)=-P^{\mu\nu}\Delta_T\left(\omega,\vec{\boldsymbol{q}}\right)-Q^{\mu\nu}\Delta_L\left(\omega,\vec{\boldsymbol{q}}\right), \tag{2.1}$$

where $q=(\omega,\vec{\boldsymbol{q}})$ is the 4-momentum of the exchanged gluon, $\Delta_T\left(\omega,\vec{\boldsymbol{q}}\right)$ and $\Delta_L\left(\omega,\vec{\boldsymbol{q}}\right)$ are the effective transverse and longitudinal propagators [59, 2]:

$$\Delta_T^{-1}=\omega^2-\vec{\boldsymbol{q}}^2-\frac{\mu_E\left(T\right)^2}{2}-\frac{\left(\omega^2-\vec{\boldsymbol{q}}^2\right)\mu_E\left(T\right)^2}{2\vec{\boldsymbol{q}}^2}\left(1+\frac{\omega}{2|\vec{\boldsymbol{q}}|}\ln\left|\frac{\omega-|\vec{\boldsymbol{q}}|}{\omega+|\vec{\boldsymbol{q}}|}\right|\right), \tag{2.2}$$

$$\Delta_L^{-1}=\vec{\boldsymbol{q}}^2+\mu_E\left(T\right)^2\left(1+\frac{\omega}{2\left|\vec{\boldsymbol{q}}\right|}\ln\left|\frac{\omega-|\vec{\boldsymbol{q}}|}{\omega+|\vec{\boldsymbol{q}}|}\right|\right), \tag{2.3}$$

where $\mu_E\left(T\right)$ is the Debye chromo-electric screening mass [2].

$P^{\mu\nu}$ and $Q^{\mu\nu}$ from 2.1 are transverse and longitudinal projection tensors whose only non-zero terms are:

$$P^{ij}=\delta^{ij}-\frac{q^iq^j}{\left|\vec{\boldsymbol{q}}\right|^2} \tag{2.4}$$

$$Q^{00}=1 \tag{2.5}$$

Collisional energy loss per unit length is given by [59]:

$$\begin{aligned}\frac{dE_{coll}}{d\tau}=&\frac{2C_R}{\pi v^2}\alpha_s(ET)\alpha_s(\mu_E^2(T))\int_0^\infty n_{eq}(|\vec{\mathbf{k}}|,T)d|\vec{\mathbf{k}}|\\&\times\Big[\int_0^{|\vec{\mathbf{k}}|/(1+v)}d|\vec{\mathbf{q}}|\int_{-v|\vec{\mathbf{q}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega+\int_{|\vec{\mathbf{k}}|/(1+v)}^{|\vec{\mathbf{q}}|_{max}}d|\vec{\mathbf{q}}|\int_{|\vec{\mathbf{q}}|-2|\vec{\mathbf{k}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega\Big]\\&\times\Big[|\Delta_L(q,T)|^2\frac{(2|\vec{\mathbf{k}}|+\omega)^2-|\vec{\mathbf{q}}|^2}{2}+\Delta_T(q,T)|^2\frac{(|\vec{\mathbf{q}}|^2-\omega^2)((2|\vec{\mathbf{k}}|+\omega)^2+|\vec{\mathbf{q}}|^2)}{4|\vec{\mathbf{q}}|^4}(v^2|\vec{\mathbf{q}}|^2-\omega^2)\Big].\end{aligned} \tag{2.6}$$

In Eq 2.6, $n_{eq}(|\vec{\boldsymbol{k}}|,T)=\frac{N_c}{e^{|\vec{\boldsymbol{k}}|/T}-1}+\frac{N_f}{e^{|\vec{\boldsymbol{k}}|/T}+1}$ is the equilibrium momentum distribution [69] at temperature T with N_c and N_f being the number of colors and flavours respectively. Running coupling is

given by α_s^2, while $C_R = \frac{4}{3}$ for quark jet and 3 for gluon jet. v is velocity of the initial jet, k is the 4-momentum of the incoming medium parton and $|\vec{\boldsymbol{q}}_{max}|$ is defined in [59].

Radiative energy loss is critical in understanding jet quenching in a quark-gluon plasma (QGP). It occurs when a propagating parton radiates gluons due to interactions with the QGP constituents, leading to a significant energy loss. This phenomenon is significant for high-energy partons, whose suppression in transverse momentum spectra is a diagnostic for the QGP properties.

In a dynamical QCD medium, where the constituents are in motion rather than static, the inclusion of medium dynamics introduces significant refinements to energy-loss models. The formalism involves summing over Feynman diagrams representing gluon radiation induced by interactions with the QGP. Each diagram can exhibit infrared divergences, but these are naturally regulated when all contributions are summed [58, 64, 65].

The medium-induced radiative energy loss is sensitive to the finite size of the QGP. In contrast to infinite-medium approximations, finite-size effects lead to nonlinear path-length dependencies, reconciling both incoherent (Gunion-Bertsch) [70] and destructive (Landau-Pomeranchuk-Migdal) [71, 72] limits.

The radiation spectrum [2] is:

$$\frac{dN_{rad}}{dxd\tau} = \frac{C_2(G)C_R}{\pi}\frac{1}{x}\int\frac{d^2\mathbf{q}}{\pi}\frac{d^2\mathbf{k}}{\pi}\frac{\mu_E^2(T)-\mu_M^2(T)}{[\mathbf{q}^2+\mu_E^2(T)][\mathbf{q}^2+\mu_M^2(T)]}T\alpha_s(ET)\alpha_s\Big(\frac{\mathbf{k}^2+\chi(T)}{x}\Big)$$
$$\times\Big[1-\cos\Big(\frac{(\mathbf{k}+\mathbf{q})^2+\chi(T)}{xE^+}\tau\Big)\Big]\frac{2(\mathbf{k}+\mathbf{q})}{(\mathbf{k}+\mathbf{q})^2+\chi(T)}\Big[\frac{\mathbf{k}+\mathbf{q}}{(\mathbf{k}+\mathbf{q})^2+\chi(T)}-\frac{\mathbf{k}}{\mathbf{k}^2+\chi(T)}\Big]. \tag{2.7}$$

Here $C_2(G) = 3$; $\chi(T) \equiv M^2x^2 + m_g(T)^2$, where x is the longitudinal momentum fraction of the jet carried away by the emitted gluon, and $m_g(T) = \mu_E(T)/\sqrt{2}$ is the effective gluon mass in finite temperature QCD medium [65]; $M = 1.2$ GeV for charm, 4.75 GeV for bottom and $\mu_E(T)/\sqrt{6}$ for light quarks; $\mu_M(T)$ is magnetic screening, where different non-perturbative approaches suggest $0.4 < \mu_M(T)/\mu_E(T) < 0.6$ [73, 74]; $\mathbf{q}$ and $\mathbf{k}$ are transverse momenta of exchanged (virtual) and radiated gluon, respectively. $Q_k^2 = \frac{\mathbf{k}^2+\chi(T)}{x}$ in $\alpha_S(\frac{\mathbf{k}^2+\chi(T)}{x})$ corresponds to the off-shellness of the jet prior to the gluon radiation [58]. Note that, all α_S terms in Eqs. (2.6) and (2.7) are infrared safe (and moreover of a moderate value) [49]. Thus, contrary to majority of other approaches, we do not need to introduce a cut-off in $\alpha_S(Q^2)$.

2.2 Software implementation

2.2.1 DREENA-C

The implementation of the DREENA framework began with the simplest model, DREENA-C ("C" for "constant") [75], which assumes a constant medium temperature throughout. This model served as the foundational step in the framework's development.

Initially, the theoretical computational procedure described in [49] was directly implemented. However, this brute-force approach was found to be impractical due to exceedingly long execution times on the available hardware, a shared memory machine with 112 cores and 224 threads. Even with substantial computational resources, this approach could not deliver results in a reasonable timeframe.

To make the implementation viable, a series of optimizations were applied, leading to a speedup of approximately two orders of magnitude compared to the unoptimized method described in [49]. These optimizations included:

- **Tabulation and Interpolation:** The intermediate functions arising in energy loss calculations were precomputed (tabulated) and subsequently interpolated. This drastically reduced the number of numerical integrations required while preserving precision. A thorough analysis of these functions' behavior guided the design of non-uniform sampling grids, ensuring that interpolation errors remained negligible.
- **Improved Numerical Integration:** Quasi-Monte Carlo integration methods replaced the traditional approaches used in [49], yielding improved precision, numerical stability, and faster execution times.
- **Parallelization:** The computational workload was parallelized to exploit the full potential of contemporary multi-core processors, further reducing execution time.

These enhancements not only improved computational performance but also enabled refinements to the physical model:

1. The multi-gluon fluctuation procedure, which was previously limited to three radiated gluons due to numerical constraints, was redeveloped to allow for an arbitrary number of gluons. Analysis showed that including 4-5 gluons provided an optimal balance between computational feasibility and numerical accuracy in the constant-temperature case.
2. The combination of radiative and collisional energy losses along the parton's path was implemented. In contrast, the earlier approach in [49] treated these loss mechanisms separately, simplifying the calculations at the cost of physical accuracy.

The results of the DREENA-C implementation and its numerical findings were presented in the [75].

2.2.2 DREENA-B

While DREENA-C represented a significant step forward, the assumption of a constant temperature medium was a crude approximation of the QGP's actual evolution. Recognizing this limitation, the development progressed to DREENA-B ("B" for "Bjorken") [2], which incorporates the Bjorken approximation for a longitudinally expanding medium. This model introduced a medium whose temperature depends on proper time but remains spatially uniform, marking a gradual advancement toward modeling a fully evolving QGP.

The transition from DREENA-C to DREENA-B necessitated significant changes to the computational algorithm:

- **Algorithmic Modifications:** In the constant-temperature scenario of DREENA-C, certain integrations (e.g., over time) could be performed analytically. However, the introduction of proper-time dependence in DREENA-B required a numerical integration over time, significantly increasing computational complexity. For example, calculating radiative energy loss for a single probe in the Bjorken scenario took approximately 10 hours on the available hardware. Given that producing meaningful results required ~100 such runs, this approach was computationally prohibitive.
- **Optimization Techniques:** To address these challenges, the following strategies were implemented:

- **Adaptive Sampling and Tabulation:** As in DREENA-C, intermediate function values were tabulated and interpolated, but with non-uniform grids adapted to the regions where functions varied most rapidly.
- **Dynamic Integration Parameters:** The number of quasi-Monte Carlo sampling points and required integration accuracy were adjusted dynamically across the parameter space to balance precision and efficiency.
- **Reimplementation in C:** The code, initially developed in symbolic computation software, was rewritten in C, leveraging its efficiency for numerical computations.

These improvements collectively achieved a computational speedup of nearly three orders of magnitude. This acceleration not only made DREENA-B practical for analyzing Bjorken expansion scenarios but also established a foundation for further developments of the DREENA framework. Additional details on the DREENA-B implementation and results are provided in Section 4.

2.2.3 DREENA-A

The final stage of development in the DREENA framework, DREENA-A [4], represented a significant leap in complexity and capability. Unlike its predecessors, DREENA-A was designed to accommodate arbitrary spatiotemporal temperature profiles, offering a fully general treatment of the medium's evolution.

In DREENA-C and DREENA-B, the simplifying assumptions about the medium's evolution allowed parton energy loss to depend only on the path length, independent of the direction or production point. This simplification enabled analytical integration of certain factors in the energy-loss formulas and precomputation of path-length distributions, yielding efficient computational algorithms.

In contrast, DREENA-A required a complete reevaluation of the computational approach. Its inputs include:

1. **Temperature Profile (Tprofile):** A 3D matrix of temperature values at spatial and temporal coordinates (x, y, τ).
2. **Initial Parton Momentum Distributions:** $d^2\sigma/dp_\perp^2$
3. **Jet Production Probability Distribution.**

For each parton trajectory, determined by its transverse origin (x_0, y_0) and direction angle ϕ, the combined radiative and collisional energy losses were calculated by integrating along the path until the medium temperature dropped below $T_c = 155$MeV, signaling the parton's exit from the QGP phase. This required averaging energy losses over all possible trajectories, a process that substantially increased computational demands.

To address these challenges, additional optimization techniques were employed:

- **Reorganization of Integration Order:** The sequence of numerical integrations was adjusted to suit the specific behavior of the functions involved. For example, integrating over initial momentum distributions first was computationally expensive for heavy-flavor particles, so this step was deferred until the final stage of computation.
- **Efficient Trajectory Averaging:** Monte Carlo sampling of production points and directions was replaced with equidistant sampling, where the transverse plane was divided into a grid, and

trajectory angles were sampled uniformly. This approach weighted energy loss by jet production probabilities, reducing execution time by over two orders of magnitude compared to the Monte Carlo method.

The realization of DREENA-A, along with its results, is discussed in detail in Section 6. The complete DREENA-A source code is provided in the Appendix.

Chapter 3

Testing path-length dependence in energy loss mechanisms

Understanding properties of Quark-Gluon Plasma (QGP) [76] created at LHC and RHIC experiments is a major goal of ultra-relativistic heavy ion physics [77], which would allow understanding properties of QCD matter at its most basic level. Energy loss of high $p_\perp$ partons traversing this medium, is an excellent probe of its properties [78], which provided a crucial contribution [77] to establishing that QGP is created in these experiments. Comparing predictions of different energy loss models [79, 80], and consequently different underlying energy loss mechanisms, with experimental data, is therefore crucial for understanding properties of created QGP. However, an open question is how to provide the most direct comparison of energy loss predictions with experimental data.

The most basic signature for distinguishing different energy loss models, is how the predicted energy loss depends on the length of the traversed QCD medium (so-called path-length dependence). This path-length dependence directly relates to different underlying energy loss mechanisms, such as pQCD collisional (with typically linear [59, 81, 82]), radiative (with typically quadratic [43, 83, 60, 84, 85, 50, 61, 86]) or alternatively conformal AdS holography models (with third power [87, 88] energy loss path length dependence). Moreover, even in such cases, the division is not so clear, as there are numerous other effects that can significantly alter these path-length dependencies [89, 90, 58, 64]: inclusion of the mass of the leading particle, finite size and finite temperature effects in QGP, interference effects, etc. Therefore, accurately assessing the path-length dependence is also crucial for understanding mechanisms that underly the observed energy loss, which is in turn necessary for investigating the properties of QCD matter created at RHIC and LHC, i.e. for precision QGP tomography.

However, despite its essential importance and longstanding interest in this subject, it is still not possible to directly infer the energy loss path-length dependence from experimental measurements, and consequently provide a possibility to discriminate between different energy loss models. To our knowledge, the most comprehensive study in this subject [91, 92], attempted to extract the energy loss path-length dependence from a thorough simultaneous study of R_{AA} and v_2 predictions and data (at $Au + Au$ collisions at RHIC and $Pb + Pb$ collisions at the LHC), but was not able to constrain this dependence based on the existing observables and data. With this in mind, the goal of this paper is to propose a novel approach for extracting the energy loss path-length dependence.

It is intuitively clear that the most direct probe of the path-length dependence would involve comparing experimental data (and the related theoretical predictions) for two collision systems of different

size. Moreover, it would be optimal if the size would be the only property distinguishing these two systems, i.e. that other properties/parameters needed for generating relevant predictions would be the same between the two systems. Equally important, it is necessary to propose an appropriate observable from which the path-length dependence can be reliably extracted. Consequently, the aim of the analysis presented in this paper, is to infer an optimal system and an optimal observable, for assessing the energy loss path-length dependence. We will also test how reliable and robust is the inferred observable to different types of energy loss, probes, centralities and collision systems.

3.1 Appropriate observable

In this section, we first start by asking what is an appropriate observable to assess the energy loss path-length dependence? To start addressing this question, we note that such observable should be sensitive to jet-medium interactions (so that energy loss path-length dependence can be reliably extracted). On the other hand, it should not be sensitive to the medium evolution, as the details of the medium evolution would, for such a purpose, present an unwanted background. Having this in mind, it is evident that such observable should be a function of R_{AA}, since R_{AA} has exactly these desired properties - i.e. it is highly sensitive to the energy loss mechanisms in QGP [68, 93, 94, 95], while being insensitive to the medium evolution (i.e. it can be characterized by mean QGP temperature) [93, 94, 95]. The medium evolution insensitivity is also consistent with results from Section 4 of almost identical R_{AA} for constant medium temperature and 1+1 D Bjorken expansion; however, this still remains to be further verifed by using more realistic medium evolution calculations, including event-by-event fluctuations [96, 92].

3.2 Appropriate systems

Measurements for 5.02 TeV $Pb+Pb$ collisions are available, while precision measurements for 5.44 TeV smaller systems (Xe+Xe, Kr+Kr, Ar+Ar and O+O) will become available in the future, with the planned Beam Size Scan (BSS) at the LHC. As these systems have similar collision energies but different sizes (atomic mass numbers are $A = 208, 129, 78, 40, 16$ for Pb, Xe, Kr, Ar, O), comparison of Pb+Pb with smaller systems appears to be a good candidate for the path-length dependence study. Note that BSS at the LHC is complementary to the current Beam Energy Scan (BES) at RHIC, as in BES the systems of the same size but different collision energies are tested, while in BSS the systems of the same energy but different sizes will be explored, thus providing a crucial insight in how properties of the created matter depend on the size of the colliding ions.

3.3 Computational framework

In this study, the R_{AA} predictions will be generated by our full-fledged numerical procedure, recently developed in [75]. The procedure is based on our state-of-the-art dynamical energy loss formalism [58, 64, 59], which contains different important effects (some of which are unique to this model): *i) Finite size, finite temperature* QGP, consisting of *dynamical* (that is moving) constituents. This abolishes the widely used approximations of static scattering centers, vacuum-like propagators and/or infinite size QGP (e.g. [43, 83, 60, 50, 61, 86]). *ii)* Our calculations are based on the finite temperature generalized Hard-Thermal-Loop approach [62], in which the infrared divergencies are naturally regulated [58, 64]. *iii).* Both collisional [59] and radiative [58, 64] energy losses are computed under *the same* theoretical framework, which is applicable to *both light and heavy flavor. iv)* The model is

generalized to the case of finite magnetic mass [66] and running coupling [49]; we also applied first steps towards removing widely used soft-gluon approximation [67]. Moreover, in [68], we showed that all these ingredients are necessary for accurately explaining the high-$p_\perp$ parton-medium interactions in QGP.

To generate the final medium modified distribution of high-$p_\perp$ hadrons, the formalism was integrated into fully optimised numerical framework DREENA [75], which integrates initial $p_\perp$ distribution of leading partons [97, 98], energy loss with multi-gluon [54] and path-length [99] fluctuations and fragmentation functions [100, 101]. To generate R_{AA} predictions for $Pb+Pb$ collisions, we use the set of parameters specified in [75], which correspond to standard literature values (details can be found in Section 2).

The dynamical energy loss formalism was previously used to obtain a comprehensive set of R_{AA} predictions at RHIC and LHC [75]; it shows wide agreement with the existing data [49], explaining puzzling data and generating nonintuitive predictions for future experiments [102, 103] (some of which were already confirmed by subsequent data [104, 105]). This then strongly indicates that our formalism can realistically describe high $p_\perp$ parton-medium interactions, and that it provides a suitable framework for the goal that we want to achieve in this study.

3.4 Smaller systems

For R_{AA} predictions in smaller systems, and their comparison with Pb+Pb collisions, one should note that R_{AA} depends on *i)* initial distribution of high-$p_\perp$ partons, *ii)* average temperature of the created QGP, and *iii)* path-length distributions. Regarding initial distributions, we previously showed [102] that, when the collision energy is changed almost two times (from 2.76 to 5.02 TeV), the influence of the change of $p_\perp$ distributions leads to only a small change (less than 10%) in the resulting suppression. Consequently, for the increase of less than 10% in the collision energy (from 5.02 to 5.44 TeV), the same high-$p_\perp$ distributions can be assumed. The average temperature ($\overline{T}$) for each centrality region in 5.02 TeV $Pb+Pb$ collisions is estimated according to [75]. Note that $\overline{T}$ is directly proportional to the charged multiplicity, while inversely proportional to the overlap area and average size of the medium, i.e. $\overline{T} = (\frac{dN_{ch}/d\eta}{A_\perp \overline{L}})^{\frac{1}{3}}$ [75, 106]. To estimate $\overline{T}$ in smaller systems, we note that, for each centrality region, all the above quantities change in the two collision systems: $A_\perp \sim A^{2/3}$; $\overline{L} \sim A^{1/3}$ [107, 108]; $dN_{ch}/d\eta \sim N_{part}$, where $N_{part} \sim A$, since, for the same collision energy, $\frac{dN_{ch}/d\eta}{N_{part}}$ should remain constant with decreasing the systems size [109, 110]. This therefore leads to $\overline{T} \sim (\frac{A}{A^{2/3}A^{1/3}})^{1/3} \sim const$, i.e. we expect that, for a fixed centrality region, $\overline{T}$ will remain unchanged when moving from large $Pb+Pb$ to smaller systems.

Finally, the path-length distributions for smaller systems, at different centralities, can be calculated in the same manner as previously for Pb+Pb [75]. It is straightforward to see that the two distributions are similar up to a rescaling factor corresponding to $A^{1/3}$. Consequently, we see that comparison of $Pb+Pb$ with smaller systems is in fact close to ideal, when it comes to probing the path-length dependencies.

3.5 Suppression ratio

The next question is, what is the exact variable (i.e. its functional dependence on R_{AA}) that should be compared for the two systems, in order to extract the path-length dependence. Since R_{AA} increases when the system size decreases, it may seem that the ratio of R_{AA} for the two systems is a natural

choice [111]. To test this proposal, in Fig. 3.1, we show momentum dependence of R_{AA} ratio for the $Xe+Xe$ and $Pb+Pb$ systems (note that, for easier reading, we will first concentrate on $Xe+Xe$ and $Pb+Pb$, and we will discuss smaller systems subsequently). We see that it would be very hard to extract the path-length dependence from such ratio, e.g. for high $p_\perp$ this ratio approaches 1, naively suggesting that the underlying model has no (or only weak) path-length dependence. However, the dynamical energy loss model has, in fact, a strong (between linear and quadratic) path-length dependence. The same problem would emerge if experimental data would be plotted in that way, i.e. one may naively conclude that high $p_\perp$ suppression does not depend on the system size. Moreover, we see that this quantity is not robust with respect to the changes in collision centrality, which would further complicate extracting the path-length dependence from simple R_{AA} ratio.

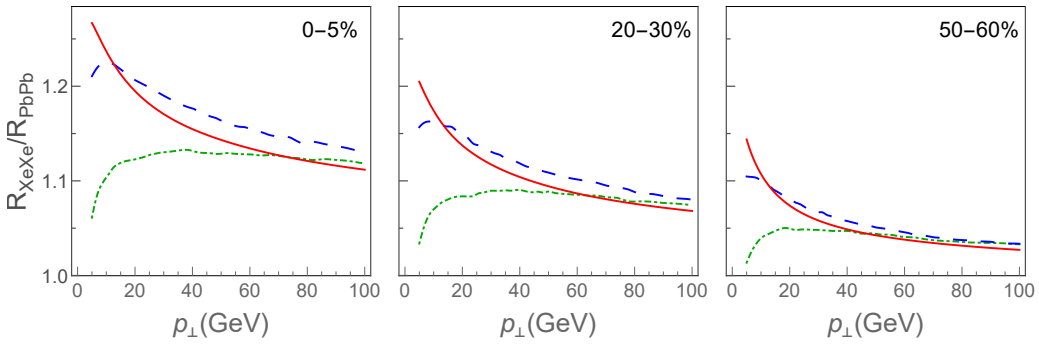

Figure 3.1: Ratio of R_{XeXe} and R_{PbPb} is shown as a function of $p_\perp$ for charged hadrons, D and B mesons (full, dashed and dot-dashed curves, respectively). Centrality regions are denoted in the upper right corners of each panel. Figure adapted from [1].

The problem above can be intuitively understood by using scaling arguments. Fractional energy loss $\Delta E/E$, can be estimated as [75]:

$$\Delta E/E \sim \chi \overline{T}^a \overline{L}^b, \tag{3.1}$$

where a, b are proportionality factors, $\overline{T}$ is the average temperature of the medium, $\overline{L}$ is the average path-length traversed by the jet and χ is a proportionality factor (which depends on initial jet $p_\perp$). $b \to 1$ corresponds to the linear, while $b \to 2$ corresponds to the quadratic (LPM like) dependence of the energy loss.

If $\Delta E/E$ is small (i.e. for higher $p_\perp$ of the initial jet, and for higher centralities), we can make the following estimate [75]

$$R_{AA} \approx 1 - \xi \overline{T}^a \overline{L}^b, \tag{3.2}$$

where $\xi = (n-2)\chi/2$, and n is the steepness of the initial momentum distribution function.

The ratio of R_{XeXe} and R_{PbPb} then becomes

$$\frac{R_{XeXe}}{R_{PbPb}} \approx 1 + \xi \overline{T}^a \overline{L}_{Pb}^b \left(1 - \left(\frac{A_{Xe}}{A_{Pb}}\right)^{b/3}\right). \tag{3.3}$$

This quantity is rather complicated, depending explicitly on the initial jet energy (through ξ), average medium temperature, and average size of the medium. Also, it explicitly depends on centrality (through $\overline{T}$ and $\overline{L}_{Pb}$, which decrease with increasing centrality), consistently with what is seen in Fig. 3.1. Furthermore, as centrality and initial energy of the jet increase, ξ, $\overline{T}$ and $\overline{L}_{Pb}$ become smaller, explaining why the ratio in Fig. 3.1 goes to 1 for high $p_\perp$ and high centrality, which results in the problem of concealing the path-length dependence. Consequently, the ratio of R_{AA}s for different collision systems is not a suitable observable for extracting path-length dependence.

3.6 Suitable observable

It is clear that such observable should expose coefficient b in a simplest possible manner. To initially gauge the appropriate functional dependence, we again resort to the scaling arguments given above, for which we have shown to provide a reasonable description of the full fledged numerical model results in Fig. 3.1. We proceed by subtracting R_{AA}s (obtained from Eq. 3.2) from 1, which, in the case of Xe and Pb, reduces to:

$$R_L^{XePb} \equiv \frac{1-R_{XeXe}}{1-R_{PbPb}} \approx \frac{\xi\overline{T}^a\overline{L}^b_{Xe}}{\xi\overline{T}^a\overline{L}^b_{Pb}} \approx \left(\frac{A_{Xe}}{A_{Pb}}\right)^{b/3}. \tag{3.4}$$

This new quantity R_L^{XePb} has a very simple form, which depends only on the medium size (through A_{Xe}/A_{Pb}), and on the path length dependence, i.e. coefficient b, which is now directly exposed. Note again that this simple dependence is expected to hold for *higher centralities and higher initial* $p_\perp$, where Eqs. 3.2 and 3.4 are applicable. Consequently, as one plots R_L^{XePb} at higher centrality regions, one may expect that this value will approach a limit that directly reflects the path-length dependence, i.e. relation given by Eq. 3.4.

To numerically test our proposal and assess the applicability of the analytically derived scaling in Eq. (3.4), we further concentrate only on higher centrality regions, and calculate $(1-R_{XeXe})/(1-R_{PbPb})$ using our full-fledged numerical procedure [75]. This ratio is shown in Fig. 3.2; full, dashed and dot-dashed curves show our full results for charged hadrons, D and B mesons, respectively; the dashed lines correspond to the $b=1$ and 2 limit from Eq. (3.4). From Fig. 3.2, one can see that R_L^{XePb} is almost independent of centrality, which is exactly what one needs for such observable. At high $p_\perp \to 100$ GeV, we clearly see that R_L^{XePb} for all types of particles reaches a limiting value, as expected. Moreover, this limiting value ($R_L^{XePb} \approx 0.8$) directly reflects the underlying path-length dependence, which is in our case (the dynamical energy loss formalism, with radiative and collisional energy loss in a finite size QCD medium) between linear and quadratic (i.e. $b \approx 1.4$), regardless of the particle flavor; note that this extracted path-length dependence is different from a common assumption of heavy flavor having linear, while light flavor having quadratic (LPM-like) dependance. It is, therefore, clear that making such plots from experimental data, and extracting the corresponding path-length dependence (exponent b), can be used to differentiate between different energy loss models in a simple and direct manner. Also, note that, in distinction to Fig. 3.2, where the gray dashed lines are simple and intuitive (allowing straightforward inference of path-length dependence), defining such lines in Fig. 3.1 would not be possible.

3.7 Testing robustness and reliability

To address the robustness of R_L^{AB} observable, i.e. if the observable is applicable to systems of diverse sizes, we further test R_L^{AB} on other smaller systems ($Kr+Kr$, $Ar+Ar$ and $O+O$). With this goal in mind, in Fig. 3.3, we concentrated on charged hadrons, and generated full-fledged predictions for R_L^{AB} for Xe–Pb, Kr–Pb, Ar–Pb and O–Pb, as a function of $p_\perp$. From this figure, we first observe that, for all four systems, this observable is almost independent on centrality, as expected from the arguments presented above. Secondly, we also observe that, independent on the collision system, this observable shows the same behavior, so it is very robust with respect to extracting path-length dependence. We moreover observe that going to smaller systems makes extracting the path-length dependence even more straightforward, since the separation between L and L^2 lines becomes larger when going to smaller systems, i.e. it increases for a factor of 2, when going from Xe–Pb to Ar–Pb

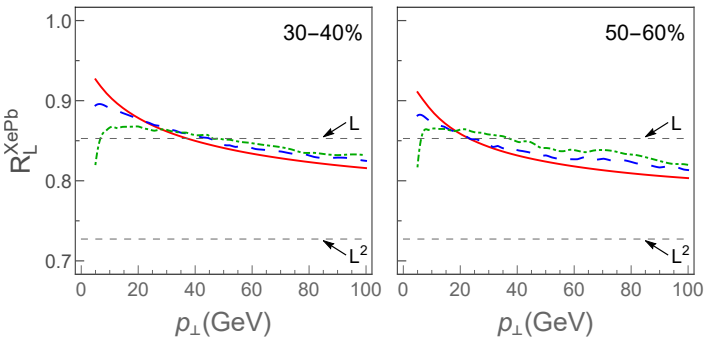

Figure 3.2: Predictions for R_L^{XePb} as a function of $p_\perp$ are shown for charged hadrons (full curves), D mesons (dashed curves) and B mesons (dot-dashed curves). Upper (lower) dashed gray line corresponds to the case in which energy loss path-length dependence is linear (quadratic). Centrality regions are denoted in the upper right corners of each panel. Figure adapted from [1].

and $O-Pb$. This then motivates using this observable across systems of different sizes, and provides another argument for utility of high $p_\perp$ measurements at BSS at the LHC.

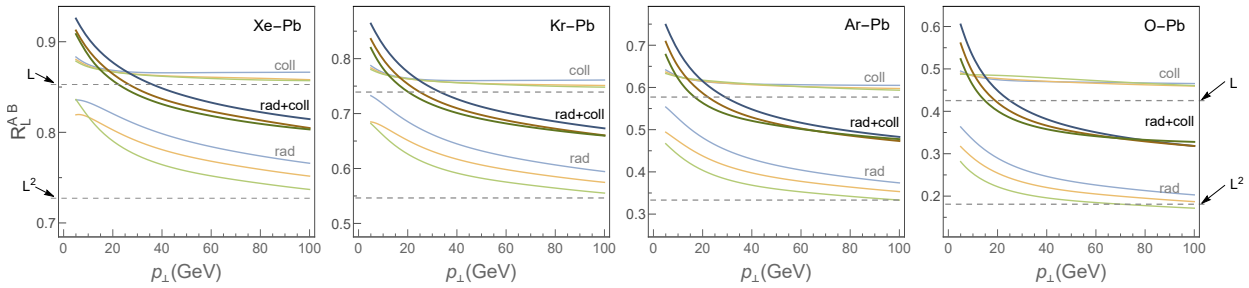

Figure 3.3: Predictions for R_L^{AB} as a function of $p_\perp$ are shown for charged hadrons, where darker set of curves are obtained by using full dynamical energy loss, while upper and lower lighter set or curves, correspond, respectively to the cases where only collisional, or only radiative, energy loss is considered. 1^{st} to 4^{th} panel correspond to, respectively, R_L^{XePb}, R_L^{KrPb}, R_L^{ArPb} and R_L^{OPb}. In each panel, three centrality regions $30-40\%$, $40-50\%$ and $50-60\%$ are, respectively, marked by blue, orange and green. Figure adapted from [1].

Finally, to address the reliability of this R_L^{AB} observable, in Fig. 3.3, we also show R_L^{AB}, calculated by using full numerical procedure stated above, but if only collisional [59] (upper curves) or radiative [58, 64] (lower curves) energy loss are taken into account - we here again concentrate on higher centrality regions where Eqs. 3.2 and 3.4 are applicable. Within the dynamical energy loss model, collisional energy loss is close to - though somewhat less then - linear ($b \approx 0.9$), due to finite size effects [59]. From Fig. 3.3, we see that this path-length dependence scenario is directly recovered; where approach to the appropriate dashed line (indicating $\lesssim L$ dependence) is almost ideal. For the radiative energy loss, due to LPM effect, path-length dependence approaches L^2 for higher $p_\perp$ [58, 64], and we see that, for such scenario, R_L^{AB} also unambiguously recovers this tendency, though the spread of curves for different centralities is somewhat larger compared to the collisional energy loss case. This therefore leads to the conclusion that, in addition to being simple and robust, R_L^{AB} is also an accurate observable for extracting path-length dependence.

Experimental measurements for smaller collision systems at future BSS at the LHC, will provide previously unprecedented opportunity to distinguish between different energy loss mechanisms, and consequently to better understand properties of created QGP. We here proposed a new - simple, robust and reliable - observable for assessing the path-length dependence of the energy loss, which is a

main signature of high $p_\perp$ parton-medium interactions. Based on our results, this observable can be used to straightforwardly extract the path-length dependence from experimental data, which can, consequently, be directly compared with such dependencies from various theoretical models, as a major test of our understanding of energy loss mechanisms.

Furthermore, our study also suggests that $(1 - R_{AA})$ might be a more suitable observable for the exploration of QGP than commonly used R_{AA}, as we have here shown that it more directly reflects the underlying energy loss of the jet traversing the QGP. Furthermore, $(1 - R_{AA})$ observable appears to be highly correlated to v_2 (as noted in study [112]). Since high $p_\perp$ observables are shown [112, 113] to be sensitive to global QGP properties, we expect that including the full medium evolution models (together with event-by-event fluctuations) into the high $p_\perp$ predictions, and providing a detailed joint study of high $p_\perp$ $(1 - R_{AA})$ and v_2 (and possibly higher harmonics) for different collision systems will prove as an excellent tool for high precision QGP tomography, which is a future major goal of relativistic heavy ion physics.

Chapter 4

DREENA-B framework

It is by now established that quark-gluon plasma (QGP), being a new state of matter [76, 114] consisting of interacting quarks, antiquarks and gluons, is created in ultra-relativistic heavy ion collisions at the Relativistic Heavy Ion Collider (RHIC) and the Large Hadron Collider (LHC). Energy loss of rare high $p_\perp$ particles, which are created in such collisions and which transverse QGP, is considered to be an excellent probe of this form of matter [77, 115, 116, 117, 118]. Such energy loss is reflected through different observables, most importantly angular averaged (R_{AA}) [104, 119, 105, 120, 38, 121, 122, 123, 124] and angular differential (v_2) [125, 126, 127, 128, 129, 130, 131, 132] nuclear modification factor, which can be measured and predicted for both light and heavy flavor probes. Therefore, comparing a comprehensive set of predictions, created under the same model and parameter set, with the corresponding experimental data, allows for systematical investigation of QCD medium properties, i.e. QGP tomography.

We previously showed that the dynamical energy loss formalism [58, 64, 59] provides an excellent tool for such tomography. In particular, we demonstrated that the formalism shows a very good agreement [49, 133, 102, 134, 103] with a wide range of R_{AA} data, coming from different experiments, collision energies, probes and centralities. Recently, we also used this formalism to generate first v_2 predictions, within DREENA-C framework [75]. These predictions were compared jointly with R_{AA} and v_2 data, showing a very good agreement with R_{AA} data, while visibly overestimating v_2 data. This overestimation also clearly differentiates the dynamical energy loss from other models, which systematically underestimated the v_2 data, leading to so called v_2 puzzle [135, 91, 136]. On the other hand, it is also clear that v_2 predictions have to be further improved - in particular v_2 was shown to be sensitive to medium evolution, while in DREENA-C medium evolution was introduced in the simplest form, through constant medium temperature. This problem then motivated us to introduce medium evolution in DREENA framework.

While several energy loss models already contain a sophisticated medium evolution, they employ simplified energy loss models. On the other hand, the dynamical energy loss formalism corresponds to the other "limit", where constant (mean) medium temperature was assumed, combined with a sophisticated model of parton-medium interactions, which includes: *i)* QCD medium composed of dynamical (i.e. moving) scattering centers, which is contrary to the widely used static scattering centers approximation, *ii)* finite size QCD medium, *iii)* finite temperature QCD medium, modeled by generalized HTL approach [62, 137], naturally regularizing all infrared and ultraviolet divergencies [65, 58, 64, 59]. *iv)* collisional [59] and radiative [58] energy losses, calculated within the same theoretical framework, *v)* finite parton mass, making the formalism applicable to both light and heavy

flavor, *vi)* finite magnetic [66] mass and running coupling [49].

Note that we previously showed that all the ingredients stated above are important for accurately describing experimental data [68]. Consequently, introducing medium evolution in the dynamical energy loss, is a major step in the model development, as all components in the model have to be preserved, and no additional simplifications should be used in the numerical procedure. In addition to developing the energy loss expressions with changing temperature, we also wanted to develop a framework that can efficiently generate a set of predictions for all types of probes and all centrality regions. That is, we think that for a model to be realistically compared with experimental data, the comparison should be done for a comprehensive set of light and heavy flavor experimental data, through the same numerical framework and the same parameter set. To implement this principle, we also had to develop a numerical framework that can efficiently (i.e. in a short time frame) generate such predictions, which will be presented in this chapter.

We will start the task of introducing the medium evolution in the dynamical energy loss formalism with DREENA-B framework presented here, where "B" stands for Bjorken. In this framework, QCD medium is modeled by the ideal hydrodynamical $1+1D$ Bjorken expansion [138], which has a simple analytical form of temperature (T) dependence. This simple T dependence will be used as an intermediate between constant (mean) temperature DREENA-C framework and the full evolution QGP tomography tool. While, on one hand, inclusion of Bjorken expansion in DREENA framework is a major task (having in mind complexity of our model, see above), it on the other hand significantly simplifies the numerical procedure compared to full medium evolutions. This will then allow step-by-step development of full QGP tomography framework, and assessing improvements in the predictions when, within the same theoretical framework, one is transitioning towards more complex QGP evolution models within the dynamical energy loss framework.

4.1 Computational frameworks

To calculate the quenched spectra of hadrons, we use the generic pQCD convolution, while the assumptions are provided in [49]:

$$\frac{E_f d^3\sigma}{dp_f^3} = \frac{E_i d^3\sigma(Q)}{dp_i^3} \otimes P(E_i \to E_f) \otimes D(Q \to H_Q) \otimes f(H_Q \to e, J/\psi),; \tag{4.1}$$

where "i" and "f", respectively, correspond to "initial" and "final", Q denotes quarks and gluons. $E_i d^3\sigma(Q)/dp_i^3$ denotes the initial quark spectrum, computed at next to leading order [97, 98] for light and heavy partons. $D(Q \to H_Q)$ is the fragmentation function of parton (quark or gluon) Q to hadron H_Q; for charged hadrons, D and B mesons we use DSS [100], BCFY [101, 139] and KLP [140] fragmentation functions, respectively. $P(E_i \to E_f)$ is the energy loss probability, generalized to include both radiative and collisional energy loss in a realistic finite size dynamical QCD medium in which the temperature is changing, as well as running coupling, path-length and multi-gluon fluctuations. In below expressions, running coupling is introduced according to [49], where we note that temperature T now changes with proper time τ; the temperature dependence along the jet path is taken according to the ideal hydrodynamical 1D Bjorken expansion [138]. Partons travel different paths in the QCD medium, which is taken into account through path length fluctuations [141]. Multi-gluon fluctuations take into account that the energy loss is a distribution, and are included according to [54, 49] (for radiative energy loss) and [142, 141] (for collisional energy loss).

The dynamical energy loss formalism was originally developed for constant temperature QCD medium, as described in detail in [58, 64, 59]. We have now derived collisional and radiative energy loss expressions for the medium in which the temperature is changing along the path of the jet; detailed calculations will be presented elsewhere, while the main results are summarized below.

For collisional energy loss, we obtain the following analytical expression:

$$\frac{dE_{coll}}{d\tau} = \frac{2C_R}{\pi v^2}\alpha_s(ET)\alpha_s(\mu_E^2(T))\int_0^\infty n_{eq}(|\vec{\mathbf{k}}|,T)d|\vec{\mathbf{k}}| \times\Big[\int_0^{|\vec{\mathbf{k}}|/(1+v)} d|\vec{\mathbf{q}}|\int_{-v|\vec{\mathbf{q}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega + \int_{|\vec{\mathbf{k}}|/(1+v)}^{|\vec{\mathbf{q}}|_{max}} d|\vec{\mathbf{q}}|\int_{|\vec{\mathbf{q}}|-2|\vec{\mathbf{k}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega\Big] \times\Big[|\Delta_L(q,T)|^2\frac{(2|\vec{\mathbf{k}}|+\omega)^2-|\vec{\mathbf{q}}|^2}{2} + \Delta_T(q,T)|^2\frac{(|\vec{\mathbf{q}}|^2-\omega^2)((2|\vec{\mathbf{k}}|+\omega)^2+|\vec{\mathbf{q}}|^2)}{4|\vec{\mathbf{q}}|^4}(v^2|\vec{\mathbf{q}}|^2-\omega^2)\Big], \tag{4.2}$$

Here E is initial jet energy, τ is the proper time, T is the temperature of the medium, α_S is running coupling [49] and $C_R = \frac{4}{3}$. k is the 4-momentum of the incoming medium parton, v is velocity of the incoming jet and $q = (\omega, \vec{\mathbf{q}})$ is the 4-momentum of the gluon. $n_{eq}(|\vec{\mathbf{k}}|,T) = \frac{N}{e^{|\vec{\mathbf{k}}|/T}-1} + \frac{N_f}{e^{|\vec{\mathbf{k}}|/T}+1}$ is the equilibrium momentum distribution [69] at temperature T including quarks and gluons (N and N_f are the number of colors and flavors, respectively). $\Delta_L(T)$ and $\Delta_T(T)$ are effective longitudinal and transverse gluon propagators [143, 144]:

$$\Delta_L^{-1}(T) = \vec{\mathbf{q}}^2 + \mu_E(T)^2(1+\frac{\omega}{2|\vec{\mathbf{q}}|}\ln|\frac{\omega-|\vec{\mathbf{q}}|}{\omega+|\vec{\mathbf{q}}|}|), \tag{4.3}$$

$$\Delta_T^{-1}(T) = \omega^2 - \vec{\mathbf{q}}^2 - \frac{\mu_E(T)^2}{2} - \frac{(\omega^2-\vec{\mathbf{q}}^2)\mu_E(T)^2}{2\vec{\mathbf{q}}^2}(1+\frac{\omega}{2|\vec{\mathbf{q}}|}\ln|\frac{\omega-|\vec{\mathbf{q}}|}{\omega+|\vec{\mathbf{q}}|}|), \tag{4.4}$$

while the electric screening (the Debye mass) $\mu_E(T)$ can be obtained by self-consistently solving the expression [145] (n_f is number of the effective degrees of freedom, Λ_{QCD} is perturbative QCD scale):

$$\frac{\mu_E(T)^2}{\Lambda_{QCD}^2}\ln\left(\frac{\mu_E(T)^2}{\Lambda_{QCD}^2}\right) = \frac{1+n_f/6}{11-2/3\,n_f}\left(\frac{4\pi T}{\Lambda_{QCD}}\right)^2. \tag{4.5}$$

The gluon radiation spectrum takes the following form:

$$\frac{dN_{rad}}{dxd\tau} = \frac{C_2(G)C_R}{\pi}\frac{1}{x}\int\frac{d^2\mathbf{q}}{\pi}\frac{d^2\mathbf{k}}{\pi}\frac{\mu_E^2(T)-\mu_M^2(T)}{[\mathbf{q}^2+\mu_E^2(T)][\mathbf{q}^2+\mu_M^2(T)]}T\alpha_s(ET)\alpha_s\big(\frac{\mathbf{k}^2+\chi(T)}{x}\big) \times\Big[1-\cos\big(\frac{(\mathbf{k}+\mathbf{q})^2+\chi(T)}{xE^+}\tau\big)\Big]\frac{2(\mathbf{k}+\mathbf{q})}{(\mathbf{k}+\mathbf{q})^2+\chi(T)}\Big[\frac{\mathbf{k}+\mathbf{q}}{(\mathbf{k}+\mathbf{q})^2+\chi(T)} - \frac{\mathbf{k}}{\mathbf{k}^2+\chi(T)}\Big], \tag{4.6}$$

where $C_2(G) = 3$ and $\mu_M(T)$ is magnetic screening. $\mathbf{k}$ and $\mathbf{q}$ are transverse momenta of radiated and exchanged (virtual) gluon, respectively. $\chi(T) \equiv M^2x^2 + m_E(T)^2/2$, where x is the longitudinal momentum fraction of the jet carried away by the emitted gluon, M is the mass of the quark of gluon jet and $m_g(T) = \mu_E(T)/\sqrt{2}$ is effective gluon mass in finite temperature QCD medium [65]. We also recently abolished the soft-gluon approximation [67], for which we however showed that it does not significantly affect the model results; consequently, this improvement is not included in DREENA-B, but can be straightforwardly implemented in the future DREENA developments, if needed.

Note that, as a result of introducing medium evolution, we got that the dynamical energy loss formalism now explicitly contains changing temperature in the energy loss expression. This is contrary to most of the other models, in which temperature evolution is introduced indirectly, through $\hat{q}$ or $\frac{dNg}{dy}$ (see [79] and references therein). This then makes the dynamical energy loss a natural framework to incorporate diverse temperature profiles as a starting point for QGP tomography. As a first (major) step, we will below numerically implement this possibility through Bjorken 1D expansion [138].

Regarding the numerical procedure, computation efficiency of the algorithm implemented in DREENA-C framework [75] was already two orders of magnitude higher with respect to the basic (unoptimized) brute-force approach applied in [49]. However, straightforward adaptation of the DREENA-C code to the case of the Bjorken type evolving medium was not sufficient. This was dominantly due to additional integration over proper time τ, which increased the calculation time for more than two orders of magnitude. The computation of e.g. the radiative energy losses alone, for a single probe, took around 10 hours on the available computer resources (a high performance workstation). Taking into account that it requires $\sim 10^3$ such runs to produce the results presented in this paper, it is evident that a substantial computational speedup was necessary.

The main algorithmic tool that we used to optimize the calculation was a combination of sampling and tabulating various intermediate computation values and their subsequent interpolation. We used nonuniform adaptive grids of the sampling points, denser in the parts of the parameter volume where the sampled function changed rapidly. Similarly, the parameters used for the numerical integration (the number of Quasi Monte Carlo sampling points and the required accuracy) were also suitably varied throughout the parameter space. Finally, while the computation in DREENA-C was performed in a software for symbolic computation, the new algorithm was redeveloped in C programming language. The combined effect of all these improvements was a computational speedup of almost three orders of magnitude, which was a necessary prerequisite for both current practical applicability and future developments of DREENA framework.

Regarding the parameters, we implement Bjorken 1D expansion [138], with commonly used $\tau_0 = 0.6$ fm [146, 147], and initial temperatures for different centralities calculated according to $T_0 \sim (dN_{ch}/dy/A_\perp)^{1/3}$ [148], where dN_{ch}/dy is charged multiplicity and $A_\perp$ is overlap area for specific collision system and centrality. We use this equation, starting from $T_0 = 500$ MeV in 5.02 TeV $Pb+Pb$ most central collisions at the LHC, which is estimated based on average medium temperature of 348 MeV in these collisions, and QCD transition temperature of $T_c \approx 150$ MeV [149]. Note that the average medium temperature of 348 MeV in most central 5.02 TeV $Pb+Pb$ collisions comes from [133] the effective temperature (T_{eff}) of 304 MeV for 0-40% centrality 2.76 TeV Pb+Pb collisions at the LHC [150] experiments (as extracted by ALICE). Once T_0s for most central $Pb+Pb$ collisions is fixed, T_0 for both different centralities and different collision systems ($Xe+Xe$ and $Pb+Pb$) are obtained from the expression above.

Other parameters used in the calculation remain the same as in DREENA-C [75]. In particular, the path-length distributions for both $Xe+Xe$ and $Pb+Pb$ are calculated following the procedure described in [99], with an additional hard sphere restriction $r < R_A$ in the Woods-Saxon nuclear density distribution to regulate the path lengths in the peripheral collisions. Note that the path-length distributions for $Pb+Pb$ are explicitly provided in [75]; we have also checked that, for each centrality, our obtained eccentricities remain within the standard deviation of the corresponding Glauber Monte Carlo results [108] (results not shown). For $Xe+Xe$, it is straightforward to show that $Xe+Xe$ and $Pb+Pb$ distributions are the same up to recalling factor ($A^{1/3}$, where A is atomic number), as we discussed in Section 3. Furthermore,the path-length distributions correspond to geometric quantity, and are therefore the same for all types of partons (light and heavy). For QGP, we take $\Lambda_{QCD} = 0.2$ GeV and $n_f = 3$. As noted above, temperature dependent Debye mass $\mu_E(T)$ is obtained from [145]. For light quarks and gluons, we, respectively, assume that their effective masses are $M \approx \mu_E(T)/\sqrt{6}$ and $m_g \approx \mu_E(T)/\sqrt{2}$ [65]. The charm and bottom masses are $M = 1.2$ GeV and $M = 4.75$ GeV, respectively. Magnetic to electric mass ratio is extracted from non-perturbative calculations [73, 74], leading to $0.4 < \mu_M/\mu_E < 0.6$ - this range of screening masses lead to presented uncertainty in the predictions. We note that no fitting parameters are used in the calculations, that is, all the parameters correspond to standard literature values.

4.2 Results and discussion

In this section, we will present joint R_{AA} and v_2 predictions for light (charged hadrons) and heavy (D and B mesons) flavor in $Pb+Pb$ and $Xe+Xe$ collisions at the LHC, obtained by DREENA-B framework. Based on the path-length distributions from Figure 1 in [75], we will, in Figures 4.1 to 4.2, show R_{AA} and v_2 predictions for light and heavy flavor, in 5.02 TeV $Pb+Pb$ and 5.44 TeV $Xe+Xe$ collisions, at different centralities. We start by presenting charged hadrons predictions, where R_{AA} data are available for both $Pb+Pb$ and $Xe+Xe$, while v_2 data exist for $Pb+Pb$ collisions. Comparison of our joint predictions with experimental data is shown in Figure 4.1, where 1^{st} and 2^{nd} columns correspond, respectively, to R_{AA} and v_2 predictions at $Pb+Pb$, while 3^{rd} and 4^{th} columns present equivalent predictions/data for $Xe+Xe$ collisions at the LHC. From this figure, we see that DREENA-B is able to well explain joint R_{AA} and v_2 predictions. For 5.44 TeV $Xe+Xe$ collisions at the LHC, we observe good agreement of our predictions with preliminary R_{AA} data from ALICE, ATLAS and CMS data (where we note that these predictions were generated, and posted on arXiv, before the data became available), except for high centrality regions, where our predictions do not agree with ALICE (and partially with ATLAS) data; however, note that in these regions ALICE, ATLAS and CMS data also do not agree with eachother.

Furthermore, comparison of predictions obtained with DREENA-B and DREENA-C frameworks in Fig. 4.1, allow directly assessing the importance of inclusion of medium evolution on different observables, as the main difference between these two frameworks is that DREENA-B contains Bjorken evolution, while DREENA-C accounts for evolution in simplest form (through constant mean temperature). We see that inclusion of Bjorken evolution has negligible effect on R_{AA}, while significant effect on v_2. That is, it keeps R_{AA} almost unchanged, while significantly decreasing v_2. Consequently, small effect on R_{AA}, supports the fact that R_{AA} is weekly sensitive to medium evolution, making R_{AA} an excellent probe of jet-medium interactions in QGP; i.e. in QGP tomography, R_{AA} can be used to calibrate parton medium interaction models. On the other hand, medium evolution has significant influence on v_2 predictions, in line with previous conclusions [93, 94, 95]; this sensitivity makes v_2 an ideal probe to constrain QGP medium parameters also from the point of high $p_\perp$ measurements (in addition to constraining them from low $p_\perp$ predictions and data).

In Figure 4.2, we provide joint predictions for D and B meson R_{AA} (left panel) and v_2 (right panel) predictions for both 5.02 TeV $Pb+Pb$ and 5.44 TeV $Xe+Xe$ collisions at the LHC. Predictions are compared with the available experimental data. For D mesons, we again observe good joint agreement with the available R_{AA} and v_2 data. For B mesons (where the experimental data are yet to become available), we predict notably large suppression (see also [49, 154]), which is consistent with non-prompt J/Ψ R_{AA} measurements [155] (indirect probe od b quark suppression). Additionally, we predict non-zero v_2 for higher centrality regions. This does not necessarily mean that heavy B meson flows, since we here show predictions for high $p_\perp$, and flow is inherently connected with *low* $p_\perp$ v_2. On the other hand, high $p_\perp$ v_2 is connected with the difference in the B meson suppression for different (in-plane and out-of-plane) directions, leading to our predictions of non zero v_2 for *high* $p_\perp$ B mesons. Additionally, by comparing D and B meson v_2s in Fig. 4.2, we observe that their difference is large and that it qualitatively exhibits the same dependence on $p_\perp$ as R_{AA}. This v_2 comparison therefore presents additional important prediction of the heavy flavor dead-cone effect in QGP, where a strikingly similar signature of this effect is observed for R_{AA} and v_2.

The predicted similarity between R_{AA} and v_2 dead-cone effects can be analytically understood by using simple scaling arguments. Fractional energy loss can be estimated as [75]:

$$\Delta E/E \sim \eta T^a L^b, \tag{4.7}$$

where a, b are proportionality factors, T and L are, respectively, the average temperature of the

medium and the average path-length traversed by the jet. η is a proportionality factor that depends on initial jet mass M and transverse momentum $p_\perp$.

Under the assumption of small fractional energy loss, we can make the following estimate [75]:

$$\begin{aligned} R_{AA} &\approx 1-\xi(M,p_\perp)T^aL^b, \\ v_2 &\approx \xi(M,p_\perp)\frac{(T^aL^{b-1}\Delta L - T^{a-1}L^b\Delta T)}{2}, \end{aligned} \tag{4.8}$$

where ΔL and ΔT are, respectively, changes in average path-lengths and average temperatures along out-of-plane and in-plane directions. $\xi = (n-2)\eta/2$, where n is the steepness of the initial momentum distribution function.

The difference between R_{AA} and v_2 for D and B mesons then becomes:

$$\begin{aligned} R_{AA}^B - R_{AA}^D &\approx (\xi(M_c,p_\perp) - \xi(M_b,p_\perp))\, T^aL^b, \\ v_2^D - v_2^B &\approx (\xi(M_c,p_\perp) - \xi(M_b,p_\perp))\, \frac{(T^aL^{b-1}\Delta L - T^{a-1}L^b\Delta T)}{2}, \end{aligned} \tag{4.9}$$

where M_c and M_b are charm and bottom quark masses respectively. From Eq. 4.9, we see the same mass dependent prefactor for both R_{AA} and v_2 comparison, intuitively explaining our predicted dead-cone effect similarity for high-$p_\perp$ R_{AA} and v_2.

4.3 Summary

Overall, we see that comprehensive joint R_{AA} and v_2 predictions, obtained with our DREENA-B framework, lead to a good agreement with all available light and heavy flavor data. This is, to our knowledge, the first study to provide such comprehensive predictions for high $p_\perp$ observables. In the context of v_2 puzzle, this study presents a significant development, as the other models were not able to achieve this agreement without introducing new phenomena [156, 157]. However, for more definite conclusions, the inclusion of more complex QGP evolution within DREENA framework is needed, which is highly non-trivial task, due to the complexity of underlying energy loss formalism.

As an outlook, for $Xe+Xe$, we also showed an extensive set of predictions for both R_{AA} and v_2, for different flavors and centralities, to be compared with the upcoming experimental data. Reasonable agreement with these data would present a strong argument that the dynamical energy loss formalism can provide a reliable tool for precision QGP tomography. Moreover, such comparison between predictions and experimental data can also confirm interesting new patterns in suppression data, such as our prediction of strikingly similar signature of the dead-cone effect between R_{AA} and v_2 data.

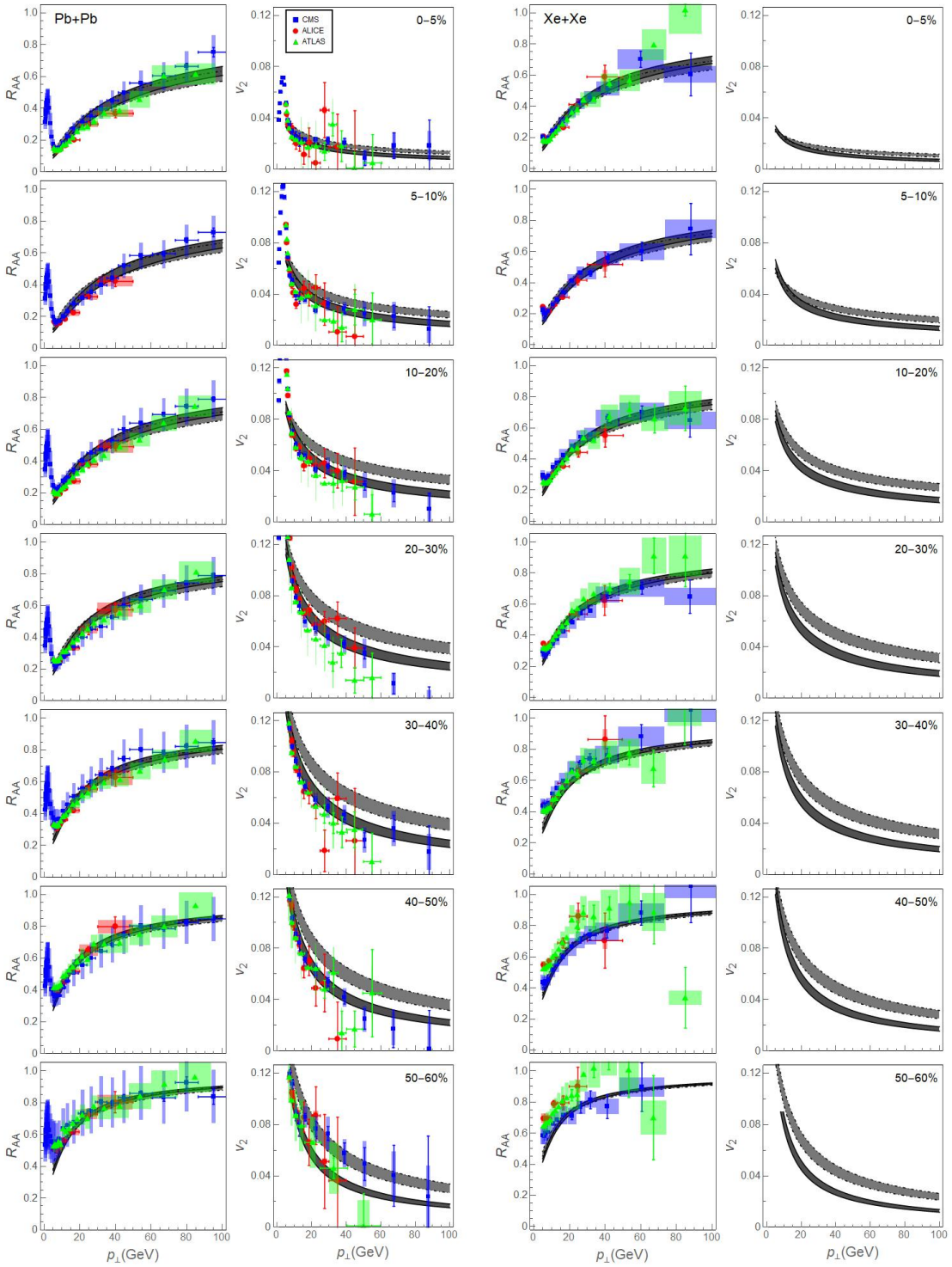

Figure 4.1: *First column:* R_{AA} *vs.* $p_\perp$ predictions are compared with 5.02 TeV $Pb+Pb$ ALICE [104], ATLAS [119] and CMS [105] $h^\pm$ experimental data. *Second column:* v_2 *vs.* $p_\perp$ predictions are compared with 5.02 TeV $Pb+Pb$ ALICE [125], ATLAS [126] and CMS [127] data. *Third column:* R_{AA} *vs.* $p_\perp$ predictions are compared with 5.44 TeV $Xe+Xe$ ALICE [151], ATLAS [152] and CMS [153] preliminary data. *Fourth column:* v_2 *vs.* $p_\perp$ predictions are shown for 5.44 TeV $Xe+Xe$ collisions. Rows 1-7 correspond to $0-5\%$, $5-10\%$, $10-20\%$,..., $50-60\%$ centrality regions. ALICE, ATLAS and CMS data are respectively represented by red circles, green triangles and blue squares. Full and dashed curves correspond, respectively, to the predictions obtained with DREENA-B and DREENA-C frameworks. In each panel, the upper (lower) boundary of each gray band corresponds to $\mu_M/\mu_E = 0.6$ ($\mu_M/\mu_E = 0.4$). Figure adapted from [2].

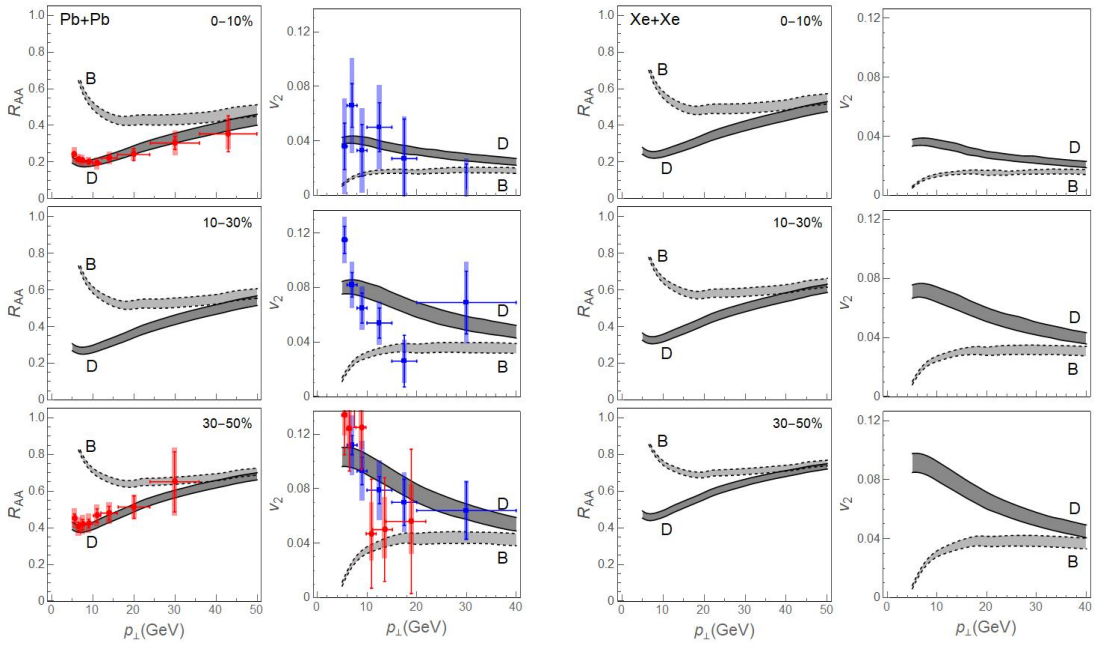

Figure 4.2: *First column:* Theoretical predictions for D and B meson R_{AA} *vs.* $p_\perp$ are compared with the available 5.02 TeV $Pb+Pb$ ALICE [120] (red circles) D meson experimental data. *Second column:* v_2 *vs.* $p_\perp$ predictions are compared with 5.02 TeV $Pb+Pb$ ALICE [130] (red circles) and CMS [129] (blue squares) D meson experimental data. *Third and fourth column:* Heavy flavor R_{AA} and v_2 *vs.* $p_\perp$ predictions are, respectively, provided for 5.44 TeV $Xe+Xe$ collisions at the LHC. First to third row, respectively, correspond to $0-10\%$, $10-30\%$ and $30-50\%$ centrality regions. On each panel, the upper (lower) boundary of each gray band corresponds to $\mu_M/\mu_E = 0.6$ ($\mu_M/\mu_E = 0.4$). Figure adapted from [2].

Chapter 5

Exploring the initial stages in heavy-ion collisions

It is by now firmly confirmed that a new state of matter − the quark-gluon plasma (QGP) [76, 114], in which quarks, antiquarks and gluons are deconfined, is formed at the Relativistic Heavy Ion Collider (RHIC) and the Large Hadron Collider (LHC). Rare high transverse momentum (high-$p_\perp$) particles, which are created immediately upon the collision, are sensitive to all stages of QGP evolution, and are considered to be excellent probes [77, 115, 116, 117, 118] of this extreme form of matter. As these probes traverse QGP, they lose energy, which is commonly assessed through high-$p_\perp$ angular averaged (R_{AA}) [104, 105, 119, 120, 158, 159, 122, 123, 124] and high-$p_\perp$ angular differential (v_2) [125, 127, 126, 130, 129] nuclear modification factors.

Commonly, the high-$p_\perp$ particles are used to study the nature of jet-medium interactions, while the low-$p_\perp$ particles are used to infer the bulk QGP properties. Accordingly, the scarce knowledge of the features of initial stages before QGP thermalization ($\tau < \tau_0$) was mostly inferred by utilizing data from low-$p_\perp$ sector [160, 161, 162] ($p_\perp \lesssim 5$ GeV). However, since high-$p_\perp$ partons effectively probe QGP properties, which in turn depend on initial stages, the idea of utilizing high-$p_\perp$ theory and data in exploring the initial stages emerged. This idea acquired an additional boost, since a wealth of precision high-$p_\perp$ R_{AA} [104, 105, 119, 120, 158, 159] and v_2 [125, 127, 126, 130, 129] data have recently became available. Thus, the main goal of this paper is to assess to what extent and through what observables, the initial stages of QGP evolution can be restrained by exploiting the energy loss of high-$p_\perp$ particles in evolving medium.

While clarifying these issues is clearly intriguing, the results of current theoretical studies on this subject are either inconclusive or questionable [113, 163, 106], as e.g., the energy loss parameters are fitted to reproduce the experimentally observed high-$p_\perp$ R_{AA} data, individually for different analyzed initial stages. The energy loss parametrization should, however, clearly be a property of high-$p_\perp$ parton interactions with the medium, rather than of individual temperature profiles. Consequently, to more rigorously study this issue, one needs a high control on both the energy loss and the analyzed temperature (T) profiles. To achieve this, we here use our state-of-the-art dynamical energy loss formalism, embedded in Bjorken 1D medium evolution [138] (DREENA-B framework from Section 4). Bjorken 1D medium evolution has a major advantage for this study, as it allows to analytically introduce different evolutions before thermalization, with the same evolution after thermalization, which therefore allows to clearly isolate only the effects of different initial stages. Consequently, we will here consider the effects on high-$p_\perp$ R_{AA} and v_2 predictions of four common initial-stage

cases [106], which have the same T profiles after, but differ in T profiles before the thermalization.

Furthermore, we demonstrated that DREENA-B framework (see Section 4) is able to accurately reproduce both high-$p_\perp$ R_{AA} and v_2 data for diverse colliding systems and energies ($Pb+Pb$ at 2.76 TeV and 5.02 TeV and $Xe+Xe$ at 5.44 TeV), for both light and heavy flavors ($h^\pm$, B, D) and all available centralities, without introducing new phenomena [156, 157, 164]. This is in distinction to many other formalisms, which employ more advanced medium evolution models, but contain simplified energy loss models, which have a tendency to underestimate v_2 relative to the experimental data, which is widely known as the v_2 puzzle [135, 165]. Moreover, we obtained that going from 1D Bjorken to full 3+1D hydrodynamics evolution (see Section 6), does not significantly change the agreement between our predictions and experimental data, strongly suggesting that, for high-$p_\perp$ data, accurate energy loss description is more important than the medium evolution. Consequently, for this study, using 1D Bjorken evolution has a major advantage of a tight control over the temperature profiles used to mimic different initial states, while, at the same time, providing a reasonably realistic description of the data within our model.

The chapter is organized as follows. In Section 5.1, theoretical and computational frameworks are outlined. In Section 5.2, we first assess the sensitivity of R_{AA} and v_2 to the aforementioned initial stages. We then adopt the approach of fitting initial temperature (T_0) to reproduce the same R_{AA} in all cases, and then assess the effect of thus obtained "modified" temperature profiles on R_{AA} and v_2. We finally reexamine the validity of widely-used procedure [113, 163, 106] of fitting the energy loss parameters for different initial-stage cases to reproduce the same R_{AA}. For all these studies, we analytically pinpoint the origin of the obtained results. Our conclusions are presented in Section 5.3.

5.1 Theoretical and computational frameworks

To obtain the medium modified distribution of high-$p_\perp$ light and heavy flavor particles, the generic pQCD convolution formula [49, 141] is utilized:

$$\frac{E_f d^3\sigma}{dp_f^3} = \frac{E_i d^3\sigma(Q)}{dp_i^3} \otimes P(E_i \to E_f) \otimes D(Q \to H_Q), \tag{5.1}$$

where indexes f and i refer to the final hadron (H_Q) and initial parton (Q), respectively. $\frac{E_i d^3\sigma(Q)}{dp_i^3}$ denotes the parton initial momentum distribution, calculated according to [97, 98]. $P(E_i \to E_f)$ presents the energy loss probability based on our dynamical energy loss formalism (see below). $D(Q \to H_Q)$ stands for fragmentation function of parton into the hadron (H_Q), where for the light hadrons, D and B mesons we apply DSS [100], BCFY [101, 139] and KLP [140] fragmentation functions, respectively.

The dynamical energy loss formalism [58, 64, 59] includes several unique features in modeling jet-medium interactions: *i)* The finite size QCD medium consisting of dynamical (moving) as opposed to static scattering centers, which allows the longitudinal momentum exchange with the medium constituents. *ii)* The calculations within the finite temperature generalized Hard-Thermal-Loop approach [62], so that infrared divergences are naturally regulated in a highly non-trivial manner, contrary to many models which apply tree-level (vacuum-like) propagators [43, 166, 60, 50, 61]. *iii)* Both radiative [58, 64] and collisional [59] contributions are calculated within the same theoretical framework. *iv)* The generalization to a finite magnetic mass [66], running coupling [49] and beyond the soft-gluon approximation [67] is performed. In this chapter for magnetic to electric mass ratio we assume value $\mu_M/\mu_E = 0.5$, since various non-perturbative [73, 74] approaches reported it to be in the range $0.4 - 0.6$. *v)* The energy loss probability comprises also multigluon [54] and

path-length [141] fluctuations. The path-length fluctuations are calculated according to the procedure presented in [99], and are provided in Ref. [75].

As outlined in Section 4, the analytical expression for single gluon radiation spectrum, in evolving medium, reads:

$$\frac{dN_{rad}}{dxd\tau} = \frac{C_2(G)C_R}{\pi}\frac{1}{x}\int\frac{d^2\mathbf{q}}{\pi}\frac{d^2\mathbf{k}}{\pi}\frac{\mu_E^2(T)-\mu_M^2(T)}{[\mathbf{q}^2+\mu_E^2(T)][\mathbf{q}^2+\mu_M^2(T)]}T\alpha_s(ET)\alpha_s\Big(\frac{\mathbf{k}^2+\chi(T)}{x}\Big) \\ \times\Big[1-\cos\Big(\frac{(\mathbf{k}+\mathbf{q})^2+\chi(T)}{xE^+}\tau\Big)\Big]\frac{2(\mathbf{k}+\mathbf{q})}{(\mathbf{k}+\mathbf{q})^2+\chi(T)}\Big[\frac{\mathbf{k}+\mathbf{q}}{(\mathbf{k}+\mathbf{q})^2+\chi(T)}-\frac{\mathbf{k}}{\mathbf{k}^2+\chi(T)}\Big], \tag{5.2}$$

where $\mathbf{k}$ and $\mathbf{q}$ denote transverse momenta of radiated and exchanged gluons, respectively, $C_2(G) = 3$, $C_R = 4/3$ ($C_R = 3$) for quark (gluon) jet, while $\mu_E(T)$ and $\mu_M(T)$ are electric (Debye) and magnetic screening masses, respectively. Temperature dependent Debye mass [145] is obtained by self-consistently solving Eq. (5) from Ref. [2]. α_s is the (temperature dependent) running coupling [167], E is the initial jet energy, while $\chi(T) = M^2x^2 + m_g^2(T)$, where x is the longitudinal momentum fraction of the jet carried away by the emitted gluon, M is the mass of the quark ($M_{u,d,s} \approx \mu_E(T)/\sqrt{6}$ i.e., the thermal mass, whereas $M_c = 1.2$ GeV and $M_b = 4.75$ GeV) or gluon jet and $m_g(T) = \mu_E(T)/\sqrt{2}$ [65] is the effective gluon mass in finite temperature QCD medium. Note that for all parameters we use standard literature values, i.e., we do not include additional fitting parameters when comparing our predictions with experimental data.

The analytical expression for the collisional energy loss per unit length in the evolving medium is given by [59]:

$$\frac{dE_{coll}}{d\tau} = \frac{2C_R}{\pi v^2}\alpha_s(ET)\alpha_s(\mu_E^2(T))\int_0^\infty n_{eq}(|\vec{\mathbf{k}}|,T)d|\vec{\mathbf{k}}| \\ \times\Big[\int_0^{|\vec{\mathbf{k}}|/(1+v)}d|\vec{\mathbf{q}}|\int_{-v|\vec{\mathbf{q}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega+\int_{|\vec{\mathbf{k}}|/(1+v)}^{|\vec{\mathbf{q}}|_{max}}d|\vec{\mathbf{q}}|\int_{|\vec{\mathbf{q}}|-2|\vec{\mathbf{k}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega\Big] \\ \times\Big[|\Delta_L(q,T)|^2\frac{(2|\vec{\mathbf{k}}|+\omega)^2-|\vec{\mathbf{q}}|^2}{2}+\Delta_T(q,T)|^2\frac{(|\vec{\mathbf{q}}|^2-\omega^2)((2|\vec{\mathbf{k}}|+\omega)^2+|\vec{\mathbf{q}}|^2)}{4|\vec{\mathbf{q}}|^4}(v^2|\vec{\mathbf{q}}|^2-\omega^2)\Big], \tag{5.3}$$

where $n_{eq}(|\vec{\mathbf{k}}|,T) = \frac{N}{e^{|\vec{\mathbf{k}}|/T}-1}+\frac{N_f}{e^{|\vec{\mathbf{k}}|/T}+1}$ is the equilibrium momentum distribution [69] comprising gluons, quarks and antiquarks ($N = 3$ and $N_f = 3$ are the number of colors and flavors, respectively). k is the 4-momentum of the incoming medium parton, v is velocity of the initial jet and $q = (\omega, \vec{\mathbf{q}})$ is the 4-momentum of the exchanged gluon. $|\vec{\mathbf{q}}|_{max}$ is provided in Ref. [59], while $\Delta_T(T)$ and $\Delta_L(T)$ are effective transverse and longitudinal gluon propagators given by Eqs. (3) and (4) in Ref. [2].

One of the assets of our energy loss formalism is the fact that energy loss explicitly depends on T, which makes it naturally suited for examining the QGP properties via implementation of various temperature profiles. In this paper, the temperature dependence on proper time (τ) is taken according to the ideal hydrodynamical 1D Bjorken expansion [138] $T(\tau) \sim \sqrt[3]{(\tau_0/\tau)}$, with thermalization time $\tau_0 = 0.6$ fm [146, 147]. The initial QGP temperature T0 for the chosen centrality bin is not a free parameter, i.e., it is constrained starting from the ALICE effective temperature [150] and following the numerical procedure outlined in Ref. [148]. In this paper, we will concentrate on mid central $30-40\%$ centrality region at 5.02 TeV $Pb+Pb$ at the LHC, which corresponds to $T_0 = 391$ MeV [2]. We however performed the extensive study on all centrality regions (as in Section 4), and checked that the results/conclusions obtained here are the same irrespectively of the centrality region. The QGP transition temperature is considered to be $T_C \approx 160$MeV [149].

DREENA-B framework is applied for generating predictions for two main high-$p_\perp$ observables – R_{AA} and v_2. The angular averaged nuclear modification factor R_{AA} is defined as the ratio of the

quenched $A+A$ spectrum to the $p+p$ spectrum, scaled by the number of binary collisions N_{bin}:

$$R_{AA}(p_T) = \frac{dN_{AA}/dp_T}{N_{\text{bin}} dN_{pp}/dp_T}, \tag{5.4}$$

while for intuitive understanding of the underlying effects we also use [75]:

$$R_{AA} \approx \frac{R^{in}_{AA} + R^{out}_{AA}}{2}, \tag{5.5}$$

where R^{in}_{AA} and R^{out}_{AA} denote in-plane and out-of-plane nuclear modification factors, respectively. The expression for the high-$p_\perp$ elliptic flow is derived in [106] (see also [75, 168, 91]), under the assumption of negligible higher harmonics at high-$p_\perp \gtrsim 10$ GeV, leading to:

$$v_2 \approx \frac{1}{2} \frac{R^{in}_{AA} - R^{out}_{AA}}{R^{in}_{AA} + R^{out}_{AA}}. \tag{5.6}$$

The advantage of using Eq. 5.6 for high-$p_\perp$ predictions is that it is computationally significantly less demanding than the commonly used v_2 expression (see, e.g., Eq. (1) from [125]). However, to explicitly verify its applicability, we checked that, for average temperature profiles, Eq. 5.6 will lead to the same result (up to less than 1% difference) as the commonly used azimuthally dependent expression. We also note that the approach to experimentally infer v_2 (see, e.g., Eq. (16) in [125]) is different from the abovementioned theoretical approaches. However, that approach could lead to different v_2 predictions only if event-by-event fluctuations are considered (which we do not do in this study). We also note that the importance of event-by-event fluctuations in adequately addressing high-$p_\perp$ v_2 is currently an open question; i.e., in [135], it was proposed that event-by-event fluctuations may increase the high-$p_\perp$ v_2, while this was not supported by two subsequent independent studies [157, 164, 169].

5.2 Results and discussion

In the first part of this section we address how different initial stages (before the thermalization time τ_0) affect our predictions of high-$p_\perp$ R_{AA} and v_2. To this end, we consider the following four common cases of initial stages [106], which assume the same 1D Bjorken hydro temperature (T) profile [138] upon thermalization (for $\tau \geq \tau_0$), but have different T profiles before the thermalization (for $\tau < \tau_0$):

a $T = 0$, the so-called *free-streaming case*, which corresponds to neglecting interactions (i.e., energy loss) before the QGP thermalization.

b The *linear case*, corresponding to linearly increasing T with time from transition temperature ($T_C = 160$ MeV at $\tau_C = 0.25$ fm) to the initial temperature T_0.

c The *constant case* $T = T_0$, and

d The *divergent case*, corresponding to 1D Bjorken expansion from $\tau = 0$.

These initial stages are depicted in Fig. 5.1, and it is clear that (a)-(d) case ordering corresponds to gradually increasing pre-thermal interactions. Note that we use this classification (a)-(d) consistently throughout the chapter to denote initial stages (for $\tau < \tau_0$), as well as for the entire evolution. Also, note that in this part of the study, we will include experimental data for comparison with our predictions. However, to allow better visualization of our obtained numerical results, in the other two parts

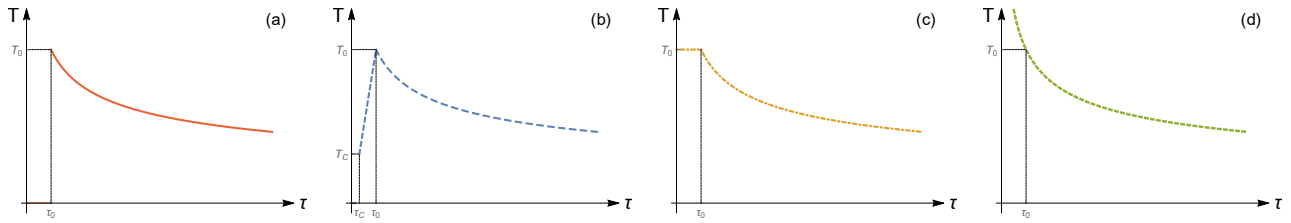

Figure 5.1: Four temperature evolution profiles, which differ at the initial stages. At $\tau \geq \tau_0$, all profiles assume the same temperature dependence on the proper time (1D Bjorken [138]). At the initial stage, i.e., for $0 < \tau < \tau_0$, the temperature is considered to be: (a) equal to zero; (b) increasing linearly from T_C to T_0 between τ_C and τ_0, otherwise zero; (c) constant and equal to T_0; and (d) a continuous function of τ matching the dependence for $\tau \geq \tau_0$. Note that, in each panel, T_0 has the same value at τ_0. Figure adapted from [3].

of the study we will omit the comparison with the data, as the error bars are large and the data remain the same.

Intuitively, one would expect that introducing these pre-thermal interactions would increase the energy loss compared to the commonly considered free-streaming case, and consequently lead to smaller R_{AA}. In Fig. 5.2 we indeed observe that R_{AA} is sensitive to the initial stages. That is, as expected, we see that the suppression progressively increases from case (a) to case (d). However, these differences are not very large, and the current errorbars at the LHC do not allow distinguishing between these scenarios, as can be seen in Fig. 5.2 (left).

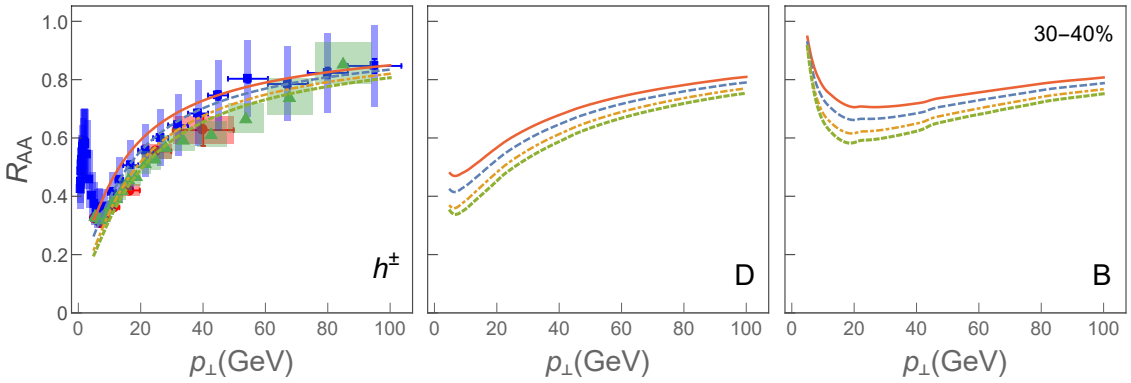

Figure 5.2: R_{AA} dependence on $p_\perp$ for four different initial stages depicted in Fig. 5.1 is shown for *charged hadrons* (left panel), D mesons (*central panel*) and *B mesons* (right panel). For charged hadrons, the predictions are compared with 5.02 TeV $Pb + Pb$ ALICE [104] (red circles), ATLAS [119] (green triangles) and CMS [105] (blue squares) $h^\pm$ R_{AA} experimental data. In each panel, temperature profile from Fig. 5.1 are presented by full red curve (case (a)), by dashed blue curve (case (b)), by dot-dashed orange curve (case (c)) and by dotted green curve (case (d)). The results correspond to the centrality bin $30 - 40\%$, and $\mu_M/\mu_E = 0.5$. Figure adapted from [3].

In contrast to R_{AA}, the effect of initial stages on v_2 is intuitively less clear, as this observable non-trivially depends on the energy loss or R_{AA}s (see Eq. 5.6). From Fig. 5.3, we surprisingly infer that v_2 is insensitive to the presumed initial stage for all types of particles (in distinction to the results obtained in [113]), so that v_2 is unable to distinguish between different initial-stage scenarios.

To quantitatively understand this unexpected observation, in Fig. 5.4 we show transverse momentum dependence of R_{AA}^{in}, R_{AA}^{out} and R_{AA} in $i = b, c, d$ cases relative to the baseline case (a) for charged hadrons. The conclusions for heavy particles are the same and therefore omitted. We distinguish three

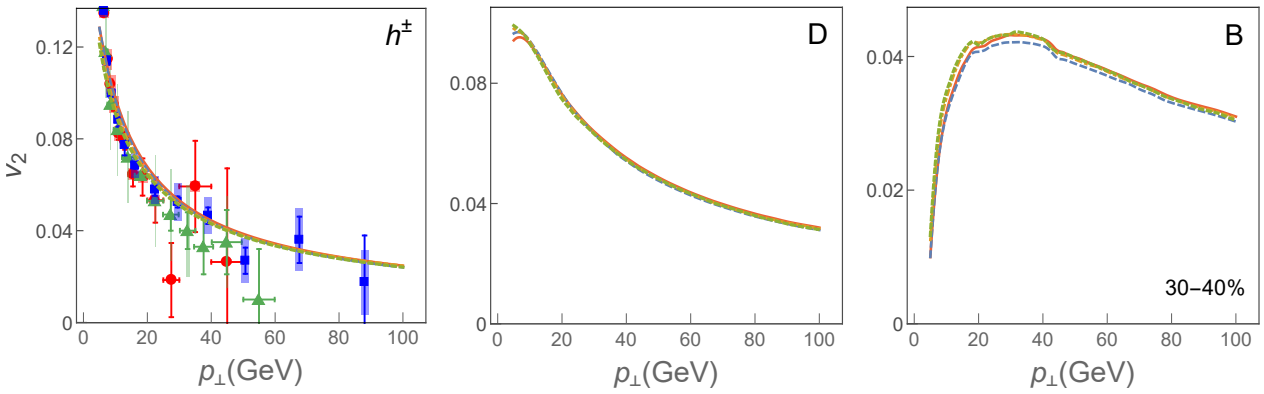

Figure 5.3: v_2 dependence on $p_\perp$ for four different initial stages depicted in Fig. 5.1. *Left*, *central* and *right panels* correspond to charged hadrons, D mesons and B mesons, respectively. For charged hadrons, the predictions are compared with 30-40% centrality 5.02 TeV $Pb+Pb$ ALICE [125] (red circles), ATLAS [126] (green triangles) and CMS [127] (blue squares) $h^\pm$ v_2 experimental data. The labeling and remaining parameters are the same as in Fig. 5.2. Figure adapted from [3].

sets of curves, which corresponds to the ratio of R_{AA}s in linear (b), constant (c), and divergent (d) cases relative to free-streaming (a) case. Note that the free-streaming case is used as a baseline, as it corresponds to the most commonly used scenario, both in low and high-$p_\perp$ calculations.

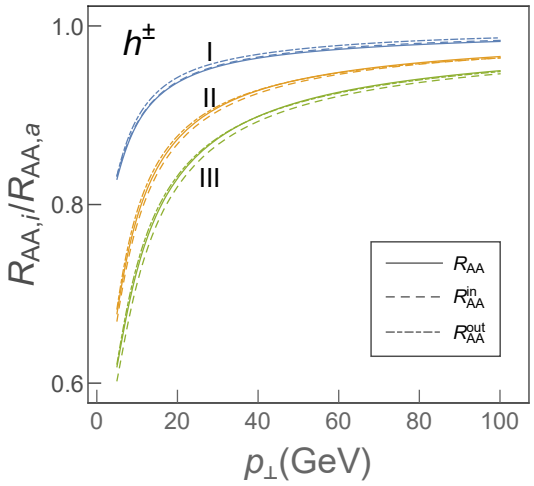

Figure 5.4: Transverse momentum dependence of in-plane (dashed), out-of plane (dot-dashed) and angular averaged (full curves) R_{AA} relative to the free-streaming case for charged hadrons. Blue (upper), orange (middle) and green (lower) set of curves correspond, respectively, to (b), (c) and (d) cases. The remaining parameters are the same as in Fig. 5.2. Figure adapted from [3].

Each set of curves in Fig. 5.4 contains three lines, representing proportionality functions $\gamma(p_\perp)$s, which are defined as follows:

$$\gamma_{ia}^{in} = \frac{R_{AA,i}^{in}}{R_{AA,a}^{in}}, \quad \gamma_{ia}^{out} = \frac{R_{AA,i}^{out}}{R_{AA,a}^{out}}, \quad \gamma_{ia} = \frac{R_{AA,i}}{R_{AA,a}}, \tag{5.7}$$

where $i = b, c, d$ denotes the corresponding cases from Fig. 5.1. From Fig. 5.4 we see that for the same i (i.e., within the same set of curves (b), (c) or (d)) the proportionality functions $\gamma_{ia}(p_\perp)$ are practically identical for the relations involving in-plane, out-of-plane and angular averaged R_{AA}s:

$$\gamma_{ia}^{in} \approx \gamma_{ia}^{out} \approx \gamma_{ia}. \tag{5.8}$$

Note also that $\gamma_{ia} < 1$, while γ_{ia}s from distinct sets significantly differ from one another (i.e., for $i \neq j \rightarrow \gamma_{ia}(p_\perp) \neq \gamma_{ja}(p_\perp)$).

Consequently, by implementing Eq. (5.7) in Eq. 5.6 and acknowledging Eq. 5.8, we obtain:

$$v_{2,i} \approx \frac{1}{2}\frac{\gamma_{ia}(R^{in}_{AA,a} - R^{out}_{AA,a})}{\gamma_{ia}(R^{in}_{AA,a} + R^{out}_{AA,a})} = v_{2,a}, \tag{5.9}$$

for any choice of $i = b, c, d$, as observed in 5.3. Therefore, we here showed that initial stages alone do not affect v_2, i.e., they affect only R_{AA}. R_{AA} susceptibility to the initial stages is in a qualitative agreement with papers [93, 94, 95, 2], where R_{AA} is shown to be only sensitive to the averaged properties of the evolving medium, i.e., average temperature ($\overline{T}$). Since R_{AA} is proportional to the $\overline{T}$, and since for all four initial-stage cases (a)-(d) the $\overline{T}$ is different ($\overline{T}_a < \overline{T}_b < \overline{T}_c < \overline{T}_d$), it is evident that R_{AA} will be different in these cases.

The fact that R_{AA} depends on the average temperature of the medium, motivate us to further explore the case in which we modify the above temperature profiles to reproduce the same average temperature. This is equivalent to re-evaluating the initial temperatures for different cases from Fig. 5.1, and based on the reasoning above, it is evident that new initial temperatures should satisfy the following ordering: $T_{0,d'} < T_{0,c'} < T_{0,b'} < T_{0,a'}$. This leads to T profiles, which do not differ only at early times ($\tau < \tau_0$), but represent *different evolutions altogether*. These new evolutions, that are illustrated in Fig. 5.5 (which is a counterpart of Fig. 5.1 for the second part of this section), are denoted as (a')-(d') and referred to as "modified" T profiles ((a)≡ (a')).

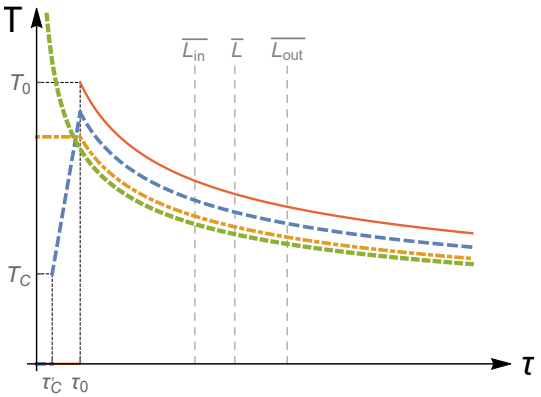

Figure 5.5: Temperature dependence on the proper time in the setup with the same average temperatures. The labeling is the same as in Fig. 5.1, apart from the fact that initial temperatures (T_0's) now differ in these four cases. As in Fig. 5.1, $T_C = 160$ MeV, $\tau_0 = 0.6$ fm and $\tau'_C = 0.27$ fm. Vertical gray dashed lines correspond to average in-medium path length ($\overline{L}$), and to the path lengths along in-plane ($\overline{L_{in}}$) and out-of-plane ($\overline{L_{out}}$) directions, as labeled in the figure. Figure adapted from [3].

In this second T-profiles setup, we first verify from Fig. 5.6 that R_{AA}s in all four cases practically overlap, as expected. We next address how these modified evolution cases $(a') - (d')$ affect v_2. From Fig. 5.7 we see that v_2 is now very sensitive to the transition from free-streaming case to other modified T profiles. More accurately, for all types of particles, the lowest v_2 is observed in modified divergent case, while the highest v_2 is observed in the free-streaming case.

The observation from Fig. 5.7 leads to the following two questions: *i)* Why is v_2 altered by these modified T profiles $(a') - (d')$? and *ii)* Are these discrepancies a consequence of different initial stages? The answer to these questions, we first note that, within this setup, the differences between v_2 (observed in Fig. 5.7) are proportional to $R^{in}_{AA} - R^{out}_{AA}$, as the denominator in Eq. 5.6 (as a starting premise) remains practically unchanged (see Fig. 5.6). The transverse momentum dependence of

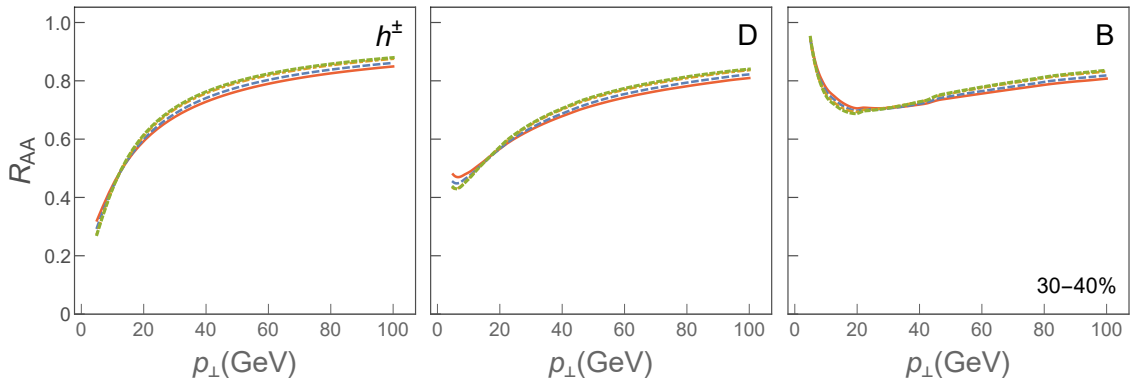

Figure 5.6: R_{AA} dependence on $p_\perp$ for four different medium evolutions depicted in Fig. 5.5. *Left*, *central* and *right panels* correspond to charged hadrons, D mesons and B mesons, respectively. In each panel, T profile corresponding to the case: (a') from Fig. 5.5 is presented by full red curve, (b') dashed blue curve, (c') dot-dashed orange curve and (d') dotted green curve. The results correspond to the centrality bin $30-40\%$, and $\mu_M/\mu_E = 0.5$. Figure adapted from [3].

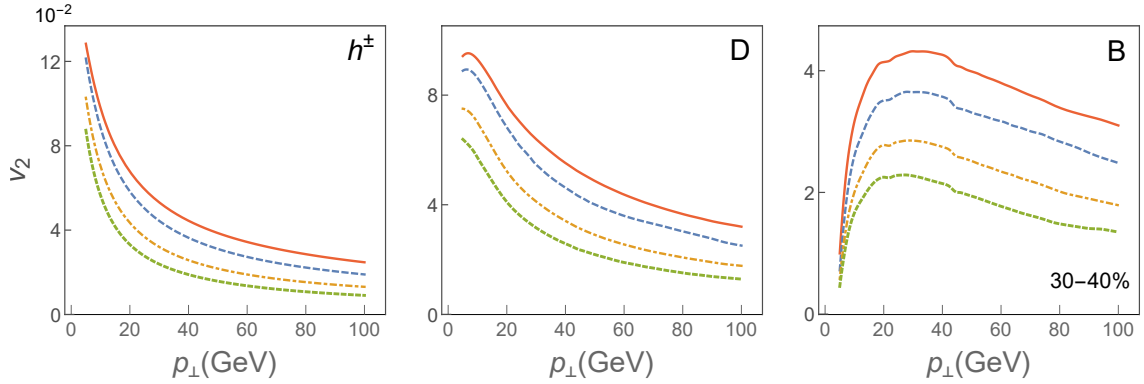

Figure 5.7: v_2 dependence on $p_\perp$ for four different medium evolutions depicted in Fig. 5.5. *Left*, *central* and *right panels* correspond to charged hadrons, D mesons and B mesons, respectively. The labeling and remaining parameters are the same as in Fig 5.6. Figure adapted from [3].

$R^{in}_{AA} - R^{out}_{AA}$ is further shown in Fig. 5.8 for charged hadrons (as results for D and B mesons will lead to the same conclusion). We see a clear hierarchy, i.e., the largest $R^{in}_{AA} - R^{out}_{AA}$ for free-streaming, descending towards divergent case. To quantitatively understand this observation, we note that for R^{in}_{AA}, the high-$p_\perp$ probes traverse, on the average, the medium up to $\overline{L}_{in}$, while for R^{out}_{AA}, the medium is traversed up to $\overline{L}_{out}$. Consequently, if we refer to Fig. 5.5, $R^{in}_{AA} - R^{out}_{AA}$ comes from T-profile difference in the time region between $\overline{L}_{in}$ and $\overline{L}_{out}$, i.e., *upon thermalization*. Since in this region $\overline{T}_{d'} < \overline{T}_{c'} < \overline{T}_{b'} < \overline{T}_{a'}$ holds, $R^{in}_{AA} - R^{out}_{AA}$ is the largest for free-streaming case and the smallest for the divergent case, as observed in Fig. 5.8, and in agreement with v_2 ordering in Fig. 5.7. This therefore provides clarification of why $R^{in}_{AA} - R^{out}_{AA}$, and consequently v_2, is affected by these four different QGP evolution profiles, and that this difference originates primarily from the interactions of high-$p_\perp$ parton with *thermalized QGP*, and *not the initial stages*. This agrees with the first part of this section (Figs. 5.2 and 5.3), where we showed and explained insensitivity of v_2 to different initial stages. It is worth emphasizing that, contrary to the first part of this section, in the second part we tested the effects on R_{AA} and v_2 not from distinctive initial stages, but instead from four entirely different evolutions of the QCD medium (related by the same global property, i.e., average temperature).

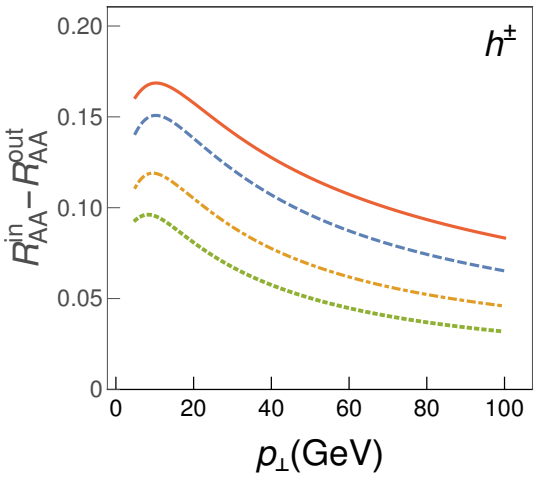

Figure 5.8: $R_{AA}^{in} - R_{AA}^{out}$ dependence on $p_{\perp}$ for charged hadrons. The labeling and remaining parameters are the same as in Fig. 5.6. Figure adapted from [3].

In the final, third, part of this section we adopt a commonly used approach, in which the energy loss is fitted through change of multiplicative fitting factor in the energy loss, to reproduce the desired high-$p_{\perp}$ R_{AA}, e.g., the one that best fits the experimental data (see e.g., [113, 91, 135, 92, 89, 170]). To this end, we use the same four T-profiles from the first part of this section (Fig. 5.1), while, in our full-fledged calculations (see Sec. 5.1) we introduce an additional multiplicative fitting factor (free parameter) C_i^{fit}, $i = b, c, d$. C_i^{fit} is then estimated for each initial-stage case as a best fit to the free-streaming R_{AA} (see Table 5.1). Thus obtained R_{AA}s are shown in the left panel of Fig. 5.9 only for the representative case of $h^{\pm}$, as the same conclusions stand for both light and heavy flavor hadrons. From the left panel of this figure we observe practically overlapping R_{AA}s in all (a)-(d) cases, as anticipated, which is obtained by decreasing C_i^{fit} consistently from the free-streaming to the divergent case (each $C_i^{fit} \leq 1$) in order to *compensate* for the higher energy losses in the corresponding cases compared to the case (a).

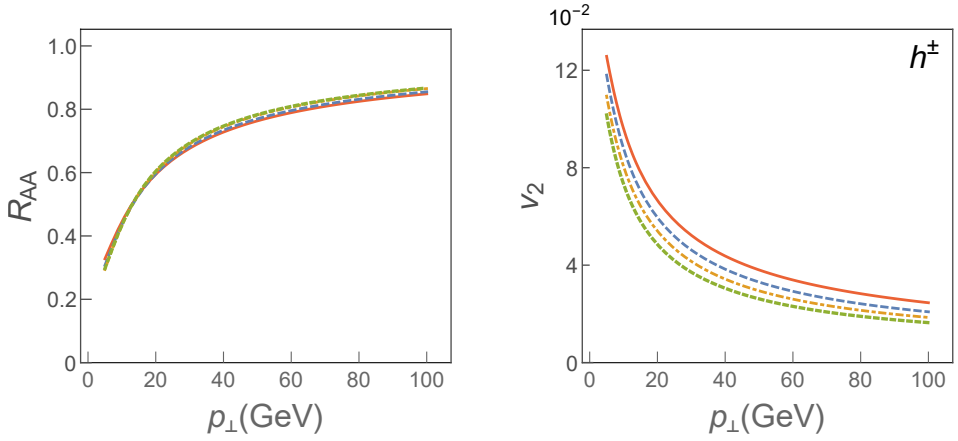

Figure 5.9: R_{AA} (*left panel*) and v_2 (*right panel*) dependence on $p_{\perp}$ for charged hadrons, when additional energy loss multiplicative factor is introduced to reproduce the free-streaming R_{AA}, in four different initial-stage cases depicted in Fig. 5.1. The labeling and remaining parameters are the same as in Figs. 5.2 and 5.3. Figure adapted from [3].

The effect of different T-profiles from Fig. 5.1 after introduction of multiplicative fitting factor C_i^{fit} in full-fledged numerical procedure on v_2 is depicted on the right panel of Fig. 5.9, where we see that elliptic flow in (a)-(d) cases notably differs, i.e., is the highest in the free-streaming case, while

T profile case	C_i^{fit}
Free-streaming case (a)	1
Linear case (b)	0.87
Constant case (c)	0.74
Divergent case (d)	0.67

Table 5.1: Fitting factors values. Table adapted from [3].

the lowest in the divergent case. Based on this observation, one could naively infer that initial stages, i.e., $\tau < \tau_0$ region (the only region in which T profiles differ), have a significant effect on v_2, as recently observed by alternative approach [113].

However, this kind of reasoning is inconsistent with our analysis outlined in the first two parts of this section, as well as with intuitive expectation that introduction of the energy loss at the initial stage affects R_{AA}. To quantitatively understand this result, we introduce asymptotic scaling behavior [75, 2, 1]. That is, for higher $p_\perp$ of the initial jet, and for higher centralities (where fractional energy loss is expected to be small), we can make the following estimates:

$$\Delta E/E \approx \chi \overline{T}^m \overline{L}^n, R_{AA} \approx 1 - \frac{l-2}{2}\frac{\Delta E}{E} = 1 - \xi \overline{T}^m \overline{L}^n, \tag{5.10}$$

where m, n are proportionality factors, $\overline{T}$ is the average temperature of the QGP, $\overline{L}$ denotes the average path length traversed by the jet, χ is a proportionality factor (that depends on $p_\perp$ and flavor of the jet). $\xi = \frac{l-2}{2}\chi$, where l is the steepness of a power law fit to the transverse momentum distribution.

If $\Delta E/E$ is fitted by additional multiplicative factor C, the new R_{AA}^{fit} becomes:

$$R_{AA,i}^{fit} \approx 1 - C_i \xi \overline{T}_i^m \overline{L}_i^n \approx 1 - C_i(1 - R_{AA,i}), \tag{5.11}$$

where $i = b, c, d$ and C_i ($C_i < 1, \forall i$) denotes the fitting factor, and the last part of Eq. (5.11) is obtained by using Eq. (5.10), leading to:

$$C_i \approx \frac{1 - R_{AA,i}^{fit}}{1 - R_{AA,i}}. \tag{5.12}$$

We note that Eq. (5.12) is applicable to the average, in-plane and out-of-plane R_{AA}s, since the same fitting factor is consistently applied in all three cases. By imposing the condition (which quantifies the equivalence of fitted R_{AA} in (b)-(d) cases to the free-streaming case):

$$R_{AA,i}^{fit} = R_{AA,a}, \tag{5.13}$$

and by applying Eqs. (5.5)-(5.8) and (5.13), together with Eqs. (5.10, 5.11) and their in-plane and out-of-plane analogons, we obtain:

$$v_{2,i}^{fit} \approx \frac{1}{2}\frac{C_i(R_{AA,i}^{in} - R_{AA,i}^{out})}{2R_{AA,a}} = \frac{1}{2}\frac{C_i\gamma_i(R_{AA,a}^{in} - R_{AA,a}^{out})}{R_{AA,a}^{in} + R_{AA,a}^{out}} = C_i\gamma_{ia}v_{2,a}, \tag{5.14}$$

which can also be written as:

$$C_i \approx \frac{v_{2,i}^{fit}}{\gamma_{ia}v_{2,a}}. \tag{5.15}$$

From Eq. (5.14), we see that decrease of v_2^{fit} in (b)-(d) cases compared to (a) is a result of a fitting factor $C_i(p_\perp)$ (which is smaller than 1), as well as the proportionality functions $\gamma_i(p_\perp)$ (also

smaller than 1). However, note that Eq. (5.14) describes asymptotic behavior at very high $p_\perp$, where, as mentioned earlier, γs approach 1. Consequently, the diminishing of elliptic flow compared to the case (a) is predominantly due to a decrease of the *artificially imposed fitting factor* C. Therefore, we obtain that, contrary to [113], *initial stages are not* mainly responsible for the obtained differences (the right panel of Fig. 5.9) in the v_2^{fit} curves for different T profiles. Moreover, this argument, as well as the obtained inconsistency of the results in this and the first two parts of the paper, implies that application of multiple fitting procedure for each different initial stage may result in incorrect energy loss estimates and in misinterpreting the underlying physics.

To asses if this qualitative conclusion indeed holds, i.e. that v_2 succesibility observed in Fig. 5.9 (as well as in [113]) is indeed a consequence of a fitting factor in the energy loss, in Fig. 5.10 we check the consistency of Eqs. (5.12) and (5.15) with the full-fledged numerical calculations. That is, a non-trivial consequence of Eqs. (5.12) and (5.15), is that C_i factors for the average, in-plane and out-of-plane R_{AA}s (Eq. 5.12) and v_2 (Eq. 5.15), should be the same in high-$p_\perp$ limit, and moreover overlap with C_i^{fit} in this limit. To this end, we define the following C factors (originating from Eqs. (5.12, 5.15)):

$$\begin{aligned} C_i^{in} &= \frac{1-R_{AA,i}^{in,fit}}{1-R_{AA,i}^{in}}, \\ C_i^{out} &= \frac{1-R_{AA,i}^{out,fit}}{1-R_{AA,i}^{out}}, \\ C_i^{av} &= \frac{1-R_{AA,i}^{fit}}{1-R_{AA,i}}, \\ C_i^{v_2} &= \frac{1}{\gamma_{ia}}\frac{v_{2,i}^{fit}}{v_{2,a}}, \end{aligned} \tag{5.16}$$

and compare them with C_i^{fit}, for each separate initial-stages case, $i = b, c, d$. Note that, while expression themselves on the right-hand side of each expression in Eq. (5.16) are obtained in high-$p_\perp$ limit (and consequently are expected to overlap in this limit, if our analytical estimate is valid), we calculate C_i^{fit}, and the terms on the the right-hand side of each expression in Eq. (5.16), through full-fledged numerical procedure. We indeed observe that, for each i and at high-$p_\perp$: C_i^{in}, C_i^{out}, C_i^{av} and $C_i^{v_2}$ factors are practically overlapping, and approach the value C_i^{fit}. Consequently, this highly non-trivial observation confirms that our qualitative conclusion is valid, and that v_2 susceptibility in this case is indeed a consequence of an additionally introduced fitting factor.

5.3 Conclusion

Traditionally, the features of initial stages before QGP thermalization are explored through comparison of bulk medium simulations and low-$p_\perp$ data. On the other hand, recent abundance of high-$p_\perp$ experimental data, motivates exploiting the high-$p_\perp$ energy loss in studying the initial stages. We here utilized state-of-the-art dynamical energy loss embedded in analytical 1D Bjorken medium expansion (DREENA-B framework), which allowed to tightly control the analyzed temperature profiles. In particular, we considered four temperature profiles, which are identical after, but are different before, thermalization, which correspond to four commonly considered initial-stage cases. This allowed to study the effects of different initial-stage cases on high-$p_\perp$ R_{AA} and v_2 predictions, under highly controlled conditions, by combining full-fledged numerical results and analytical estimates used to interpret the experimental results.

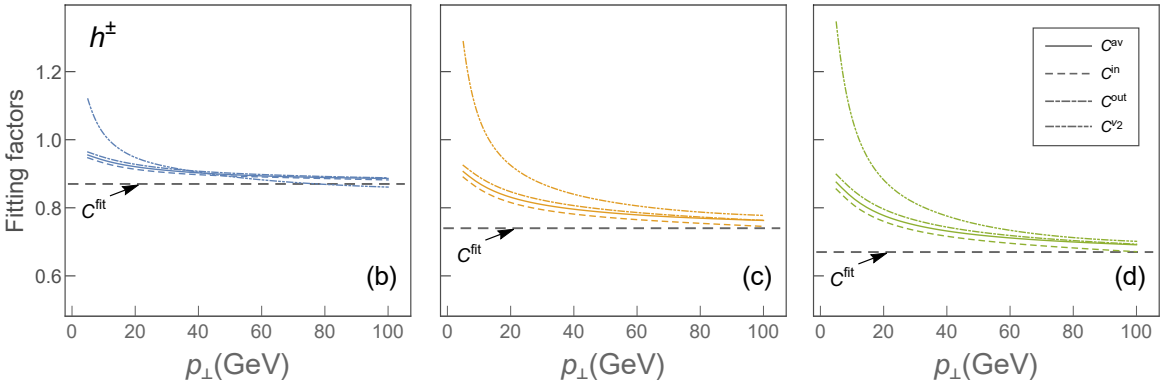

Figure 5.10: Comparison of four fitting factors defined by Eq. 5.16 with C_i^{fit} value, obtained from full-fledged numerical procedure, in *linear* (b) (left), *constant* (c) (central) and *divergent* (d) (right panel) cases. C factors presented by full, long dashed, dot-dashed and dot-dot-dashed curves correspond to $h^{\pm}$ angular averaged, in-plane, out-of-plane R_{AA} and v_2 cases, respectively. The horizontal gray dashed line presents energy loss fitted value C_i^{fit}. The results correspond to the centrality bin $30-40\%$, and $\mu_M/\mu_E = 0.5$. Figure adapted from [3].

We found that high-$p_\perp$ R_{AA} is sensitive to the pretermalized stages of the medium evolution, however, within the current errorbars, the senistivity is not sufficient to distinguish between different scenarios. On the other hand, the high-$p_\perp$ v_2 is unexpectedly insensitive to the initial stages. We furthermore found that previously reported sensitivity [113] of high-$p_\perp$ v_2 to initial stages is mainly a consequence of the fitting procedure in which the parameters in the energy loss are adjusted to reproduce experimentally observed R_{AA}, individually for different initial-stage cases. On the other hand, if the same global property, in particular the same average temperature, is imposed to tested temperature profiles, high sensitivity of high-$p_\perp$ v_2 is again obtained. This sensitivity is, however, a consequence of differences in final, rather than initial, stages. Overall, our results underscore that the simultaneous study of high-$p_\perp$ R_{AA} and v_2, with consistent/fixed energy loss parameters across the entire study and controlled temperature profiles (reflecting only the differences in the initial stages), is crucial to impose accurate constraints on the initial stages.

Chapter 6

DREENA-A framework as a QGP tomography tool

QCD predicted that a new form of matter [76, 114]— consisting of quarks, antiquarks, and gluons that are no longer confined—is created at extremely high energy densities. According to the current cosmology, this new state of matter, called Quark-Gluon Plasma (QGP) [115, 116, 77, 117, 118], existed immediately after the Big Bang [171]. Today, QGP is created in 'Little Bangs', when heavy ions collide at ultra-relativistic energies [77, 117]. Such collisions lead to an expanding fireball of quarks and gluons, which thermalises to form QGP; the QGP then cools down, and when the temperature reaches a critical point, quarks and gluons hadronise.

Successful production of this exotic state of matter at the Relativistic Heavy Ion Collider (RHIC) and the Large Hadron Collider (LHC) allowed systematical testing of different models of QGP evolution against experimental data. Up to now, it has been established that QGP is formed at the LHC and RHIC experiments through two main lines [77, 117, 172] of evidence: *i)* by comparison of low momentum ($p_\perp$) measurements with relativistic hydrodynamic predictions, which implied that created QGP is consistent with the description of a nearly perfect fluid [146, 173, 174], *ii)* by comparison of high-$p_\perp$ data [175, 176, 177, 178, 179] with pQCD predictions, which showed that high-$p_\perp$ partons (jets) significantly interact with an opaque medium. Beyond this discovery phase, the current challenge is to investigate the properties of this extreme form of matter.

While high-$p_\perp$ physics had a decisive role in the QGP discovery [77], it was rarely used for understanding the bulk medium properties. On the other hand, low-$p_\perp$ observables do not provide stringent constraints to all parameters of the models used to describe the evolution of QGP, and thus leave some properties of QGP badly constrained [180, 181, 182, 183]. Thus, it is desirable to explore QGP properties through independent theory and data set. We argue that this is provided by jet energy loss and high-$p_\perp$ data, complementing the low-$p_\perp$ constraints to QGP.

To use high-$p_\perp$ theory and data as a QGP tomography tool, it is necessary to have a realistic high-$p_\perp$ parton energy loss model. We use our dynamical energy loss formalism, which has the following properties: *i)* It is based on finite size, finite temperature field theory [62, 137], and takes into account that QGP constituents are dynamical (moving) particles. Consequently, all divergences are naturally regulated in the model. *ii)* Both collisional [59] and radiative [58, 64] energy losses are calculated in the same theoretical framework. In radiative energy loss, finite size effects induce a non-linear path length dependence of the energy loss, recovering both the incoherent Gunion Bertsch

and destructive Landau-Pomeanchuk-Migdal limit [58, 64]. For collisional energy loss, we show that finite size effects can be neglected [59], i.e., path-length dependence is close to linear. *iii)* It is applicable to both light and heavy flavors, so it can provide predictions for an extensive set of probes. *iv)* Temperature is a natural variable in the framework [184], so that the T profiles resulting from bulk medium simulations are a direct input in the model. *v)* The non-perturbative effects related to screening of the chromo-magnetic and chromo-electric fields are included [66] through the generalized hard-thermal-loop (HTL) approach. For radiative energy loss, the effective cross-section is handled through sum-rules [185], which allows consistent inclusion of non-perturbative medium-related interactions captured by lattice QCD (see [66] for more details). For collisional energy loss, the correction was done at the leading order through modification of the running coupling, following the procedure from [186] (see also [49]). *vi)* No parameters are adjusted when comparing the dynamical energy loss predictions with high-$p_\perp$ data [187, 188], i.e., all parameters are fixed to the standard literature values (specified in Subsection 2.1). The formalism explained a wide range of high-$p_\perp$ data [49, 133, 102, 134, 103], including puzzling data [103] and generating predictions for future experiments [102]. This suggests that the model realistically describes high-$p_\perp$ parton-medium interactions. While other available energy loss models (see e.g. [166, 83, 189, 60, 50, 84, 190]) have some of the above properties, none have all (or even most of them), making the dynamical energy loss an advanced framework for QGP tomography. As the temperature is the only input in the energy loss model, this allows further exploiting different temperature profiles that agree with low-$p_\perp$ data by testing their agreement with high-$p_\perp$ data. Consequently, a systematic comparison of data and predictions obtained by the same formalism and parameter set allows constraining the QGP parameters from both low and high-$p_\perp$ theory and data.

Including full medium evolution in the dynamical energy loss is, however, a highly non-trivial task, as all the model properties have to be preserved [68], without additional simplifications in the numerical procedure. Furthermore, to be effectively used as a precision QGP tomography tool, the framework needs to efficiently (timewise) generate a comprehensive set of light and heavy flavor suppression predictions through the same numerical framework and the same parameter set. Such predictions can then be compared with the available experimental data, sometimes even repeatedly (i.e., iteratively) – for different combinations of QGP medium parameters – to extract medium properties that are consistent with both low and high-$p_\perp$ data.

To introduce the medium evolution in the dynamical energy loss, we took a step-by-step approach, allowing us to check the consistency of each consecutive step by comparing its results with the previous (simpler) framework versions. Consequently, we first developed the DREENA-C framework [75], continuing to DREENA-B (details in Section 4). In this chapter, we present a fully optimised DREENA-A framework, where 'A' stands for 'adaptive' (i.e., arbitrary) temperature evolution. The convergence speed of the developed numerical procedure is analysed, as well as consistency with other (earlier) versions of the framework, as necessary for the reliable and efficient QGP tomography tool. Finally, as a utility check of the DREENA-A framework, the sensitivity of high-$p_\perp$ observables to different temperature profiles is presented.

The link to the software code implementing the DREENA-A framework (with usage instructions and example data) is provided [191]. Using this software, researchers can generate high-$p_\perp$ predictions for their own (different) models of medium evolution and compare the results with experimental data.

6.1 Methods

6.1.1 Theoretical outline

The calculation of the final hadron spectrum includes initial high-$p_\perp$ parton (quark and gluon) distributions from perturbative QCD, energy loss (if the QCD medium is formed), and fragmentation into hadrons. The cross section for quenched spectra is schematically written as [141, 148]:

$$\frac{E_f d^3\sigma_q(H_Q)}{dp_f^3} = \frac{E_i d^3\sigma(Q)}{dp_i^3} \otimes P(E_i \to E_f) \otimes D(Q \to H_Q), \tag{6.1}$$

where $\otimes$ is a generic convolution, and the change in the initial spectra due to energy loss in QGP is denoted $P(E_i \to E_f)$. If the medium is not created, then Eq. (6.1) reduces to cross section for unquenched spectra:

$$\frac{E_f d^3\sigma_u(H_Q)}{dp_f^3} = \frac{E_i d^3\sigma(Q)}{dp_i^3} \otimes D(Q \to H_Q)\,. \tag{6.2}$$

More specifically, $\frac{E_f d^3\sigma_q(H_Q)}{dp_f^3}$ is the final hadron spectrum in the presence of QGP, while $\frac{E_f d^3\sigma_u(H_Q)}{dp_f^3}$ is the spectrum in the absence of QGP. 'i' and 'f' correspond to 'initial' and 'final', respectively. Q denotes quarks and gluons, while H_Q denotes hadrons. Initial parton spectrum is denoted by $E_i d^3\sigma(Q)/dp_i^3$, and computed at next to leading order [97, 98, 192] for light and heavy partons. $P(E_i \to E_f)$ is the probability for energy transfer, which includes medium induced radiative [58, 64] and collisional [59] contributions in a finite size dynamical QCD medium with running coupling [49]. Both contributions include multi-gluon fluctuations, introduced according to Refs. [54, 49] for radiative and [142, 141] for collisional energy loss (for more details, see below). Q to hadron H_Q fragmentation is denoted by $D(Q \to H_Q)$. For charged hadrons we use DSS [100], for D mesons BCFY [101, 139] and for B mesons KLP [140] fragmentation functions, respectively.

In DREENA-A, the medium temperature needed to calculate $P(E_i \to E_f)$ depends on the position of the parton according to a temperature profile given as an input. Therefore, the temperature that the parton experiences along its path, becomes a function of the coordinates of its origin $(\mathrm{x}_0, \mathrm{y}_0)$, the angle of its trajectory ϕ, and the proper time τ:

$$T(\mathrm{x}_0, \mathrm{y}_0, \phi, \tau) = T_{profile}(\mathrm{x}_0 + \tau\cos\phi, \mathrm{y}_0 + \tau\sin\phi, \tau), \tag{6.3}$$

where $T_{profile}$ is, in principle, arbitrary. This temperature then appears in the expressions below.

The collisional energy loss is given by the following analytical expression [59]:

$$\begin{aligned}\frac{dE_{coll}}{d\tau} = {}& \frac{2C_R}{\pi v^2}\alpha_s(ET)\alpha_s(\mu_E^2(T))\int_0^\infty n_{eq}(|\vec{\mathbf{k}}|,T)d|\vec{\mathbf{k}}| \\ & \times\Big[\int_0^{|\vec{\mathbf{k}}|/(1+v)} d|\vec{\mathbf{q}}|\int_{-v|\vec{\mathbf{q}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega + \int_{|\vec{\mathbf{k}}|/(1+v)}^{|\vec{\mathbf{q}}|_{max}} d|\vec{\mathbf{q}}|\int_{|\vec{\mathbf{q}}|-2|\vec{\mathbf{k}}|}^{v|\vec{\mathbf{q}}|}\omega d\omega\Big] \\ & \times\Big[|\Delta_L(q,T)|^2\frac{(2|\vec{\mathbf{k}}|+\omega)^2-|\vec{\mathbf{q}}|^2}{2} + \Delta_T(q,T)|^2\frac{(|\vec{\mathbf{q}}|^2-\omega^2)((2|\vec{\mathbf{k}}|+\omega)^2+|\vec{\mathbf{q}}|^2)}{4|\vec{\mathbf{q}}|^4}(v^2|\vec{\mathbf{q}}|^2-\omega^2)\Big],\end{aligned} \tag{6.4}$$

Here we used the following notation: k is the 4-momentum of the incoming medium parton; T is the current temperature along the path, given by Eq. (6.3); $n_{eq}(|\vec{\mathbf{k}}|,T) = \frac{N}{e^{|\vec{\mathbf{k}}|/T}-1} + \frac{N_f}{e^{|\vec{\mathbf{k}}|/T}+1}$ is the equilibrium momentum distribution [69] at temperature T including quarks and gluons. $N = 3$

and N_f represent, respectively, the number of colors and flavors, where we assume $N_f = 3$ for the LHC and $N_f = 2.5$ for RHIC; $q = (\omega, \vec{\mathbf{q}})$ is the 4-momentum of the exchanged gluon; $E^2 = p^2 + M^2$ denotes the initial jet energy, p is the jet momentum, while M is the mass (specified below) of the quark or gluon jet; $v = p/\sqrt{p^2 + M^2}$ denotes velocity of the incoming jet; $C_R = \frac{4}{3}$ for quark jet and 3 for gluon jet; $\Delta_L(T)$ and $\Delta_T(T)$ are effective longitudinal and transverse gluon propagators [143, 144], while the electric screening (the Debye mass) $\mu_E(T)$ is obtained by self-consistently solving the expression from [145] (Λ_{QCD} is perturbative QCD scale):

$$\frac{\mu_E(T)^2}{\Lambda_{QCD}^2} \ln\left(\frac{\mu_E(T)^2}{\Lambda_{QCD}^2}\right) = \frac{1 + N_f/6}{11 - 2/3\, N_f} \left(\frac{4\pi T}{\Lambda_{QCD}}\right)^2. \tag{6.5}$$

Note that such solution leads to the Debye mass consistent with lattice QCD results [145, 74].

Running coupling $\alpha_S(Q^2)$ is defined as [167]

$$\alpha_S(Q^2) = \frac{4\pi}{(11 - 2/3N_f)\ln(Q^2/\Lambda_{QCD}^2)}, \tag{6.6}$$

where, in the collisional energy loss case, the coupling appears through the term α_S^2 [59], which can be factorised to $\alpha_S(\mu_E^2)\,\alpha_S(ET)$ [186] (see also [49]).

The radiation spectrum, as outlined in Section 4, is:

$$\begin{aligned}\frac{dN_{rad}}{dxd\tau} = \frac{C_2(G)C_R}{\pi}\frac{1}{x}\int \frac{d^2\mathbf{q}}{\pi}\frac{d^2\mathbf{k}}{\pi}\frac{\mu_E^2(T) - \mu_M^2(T)}{[\mathbf{q}^2 + \mu_E^2(T)][\mathbf{q}^2 + \mu_M^2(T)]} T\alpha_s(ET)\alpha_s\Big(\frac{\mathbf{k}^2 + \chi(T)}{x}\Big) \\ \times\Big[1 - \cos\Big(\frac{(\mathbf{k}+\mathbf{q})^2 + \chi(T)}{xE^+}\tau\Big)\Big]\frac{2(\mathbf{k}+\mathbf{q})}{(\mathbf{k}+\mathbf{q})^2 + \chi(T)}\Big[\frac{\mathbf{k}+\mathbf{q}}{(\mathbf{k}+\mathbf{q})^2 + \chi(T)} - \frac{\mathbf{k}}{\mathbf{k}^2 + \chi(T)}\Big],\end{aligned} \tag{6.7}$$

Here $C_2(G) = 3$; $\chi(T) \equiv M^2x^2 + m_g(T)^2$, where x is the longitudinal momentum fraction of the jet carried away by the emitted gluon, and $m_g(T) = \mu_E(T)/\sqrt{2}$ is the effective gluon mass in finite temperature QCD medium [65]; $M = 1.2$ GeV for charm, 4.75 GeV for bottom and $\mu_E(T)/\sqrt{6}$ for light quarks; $\mu_M(T)$ is magnetic screening, where different non-perturbative approaches suggest $0.4 < \mu_M(T)/\mu_E(T) < 0.6$ [73, 74]; $\mathbf{q}$ and $\mathbf{k}$ are transverse momenta of exchanged (virtual) and radiated gluon, respectively. $Q_k^2 = \frac{\mathbf{k}^2+\chi(T)}{x}$ in $\alpha_S(\frac{\mathbf{k}^2+\chi(T)}{x})$ corresponds to the off-shellness of the jet prior to the gluon radiation [58]. Note that, all α_S terms in Eqs. (6.4) and (6.7) are infrared safe (and moreover of a moderate value) [49]. Thus, contrary to majority of other approaches, we do not need to introduce a cut-off in $\alpha_S(Q^2)$.

We further assume that radiative and collisional energy losses can be separately treated in $P(E_i \to E_f)$, i.e., jet quenching is performed via two independent branching processes [49, 141]. We first calculate the modification of the quark and gluon spectrum due to radiative energy loss, then collisional energy loss (we checked that change of order is unimportant within our model). This is a reasonable approximation when the radiative and collisional energy losses can be considered small (which is in the essence of the soft-gluon, soft-rescattering approximation widely used in energy loss calculations) and when radiative and collisional energy loss processes are decoupled, as is the case in the generalized HTL approach [193] used in our energy loss calculations.

To obtain the radiative energy loss contribution to the suppression [54], we start with Eq. (6.7) and, for a given trajectory, we first compute the mean number of gluons emitted due to induced radiation (further denoted as $\overline{N}_{tr}(E)$), as well as the mean number of gluons emitted per fractional energy loss x (i.e., $\frac{d\overline{N}_{tr}(E)}{dx}$, for compactness further denoted as $\overline{N}'_{tr}(E,x)$):

$$\overline{N}_{tr}(E) = \int_{tr}\left(\int \frac{d^2N_{\rm rad}}{dxd\tau}dx\right)d\tau, \qquad \overline{N}'_{tr}(E,x) = \int_{tr}\frac{d^2N_{\rm rad}}{dxd\tau}d\tau, \tag{6.8}$$

where the subscript tr indicates that the value depends on the trajectory. Radiative energy loss suppression takes multi-gluon fluctuations into account, where we assume that the fluctuations of gluon number are uncorrelated. Such assumption is reasonable, as Ref. [194] studied full splitting cascade and found that independent branchings reasonably well approximate a full branching. The radiative energy loss probability can then be expressed via Poisson expansion [54, 49]:

$$P^{tr}_{rad}(E_i \to E_f) = \frac{\delta(E_i - E_f)}{e^{\overline{N}_{tr}(E_i)}} + \frac{\overline{N}'_{tr}(E_i, 1 - \frac{E_f}{E_i})}{E_i\, e^{\overline{N}_{tr}(E_i)}} + \\ + \sum_{n=2}^{\infty} \frac{e^{-\overline{N}_{tr}(E_i)}}{n! E_i} \int dx_1 \cdots dx_n \overline{N}'_{tr}(E_i, x_1) \cdots \\ \overline{N}'_{tr}(E_i, x_{n-1}) \overline{N}'_{tr}(E_i, 1 - \frac{E_f}{E_i} - x_1 - \cdots - x_{n-1}), \tag{6.9}$$

E_i and E_f are initial and final jet energy (before and after) radiative process.

To calculate the parton spectrum after radiative energy loss, we apply

$$\frac{E_{f,R} d^3\sigma}{dp^3_{f,R}} = \frac{E_i d^3\sigma(Q)}{dp^3_i} \otimes P^{tr}_{rad}(E_i \to E_{f,R}), \tag{6.10}$$

where the final spectra is obtained after integrating over $p_i > p_{f,R}$.

To find collisional energy loss contribution, Eq. (6.4) is first integrated over the given trajectory:

$$\overline{E}^{tr}_{\rm col}(E) = \int_{tr} \frac{dE_{col}}{d\tau} d\tau. \tag{6.11}$$

For collisional energy loss, the full fluctuation spectrum is approximated by a Gaussian centered at the average energy loss $\overline{E}^{tr}_{\rm col}(E)$ [142, 141]:

$$P^{tr}_{\rm col}(E_i, E_f) = \frac{1}{\sqrt{2\pi}\sigma^{tr}_{\rm col}(E_i)} \exp\Big(- \frac{(E_i - E_f - \overline{E}^{tr}_{\rm col}(E_i))^2}{2\sigma^{tr}_{\rm col}(E_i)^2} \Big), \tag{6.12}$$

with a variance

$$\sigma^{tr}_{\rm col}(E) = \sqrt{2\,\overline{T^{tr}}\,\overline{E}^{tr}_{\rm col}(E)}, \tag{6.13}$$

where $\overline{T^{tr}}$ is the average temperature along the trajectory, E_i and E_f are initial and final energy (before and after collisional processes).

To calculate the quenched hadron spectrum after collisional energy loss, we apply

$$\frac{E_f d^3\sigma_q(H_Q)}{dp^3_f} = \frac{E_{i,C} d^3\sigma(Q)}{dp^3_{i,C}} \otimes P^{tr}_{col}(E_{i,C} \to E_f) \otimes D(Q \to H_Q)\,, \tag{6.14}$$

where we assume $E_{i,C} = E_{f,R}$, i.e. the final jet energy after radiative quenching corresponds to the initial jet energy for collisional quenching. Since both collisional energy loss and gain contribute to the final spectra [59, 141], both $E_{i,C} > E_f$ and $E_{i,C} < E_f$ have to be taken into account in Eq. (6.14). Finally, the hadron suppression $R^{tr}_{AA}(p_f, H_Q)$ for the single trajectory, after radiative and collisional energy loss, is equal to the ratio of quenched and unquenched momentum spectra:

$$R^{tr}_{AA}(p_f, H_Q) = \frac{E_f d^3\sigma_q(H_Q)}{dp^3_f} \Big/ \frac{E_f d^3\sigma_u(H_Q)}{dp^3_f}\,, \tag{6.15}$$

where $\frac{E_f d^3\sigma_u(H_Q)}{dp_f^3}$ is given by Eq. (6.2). $R_{AA}^{tr}(p_f, H_Q)$ then needs to be averaged over trajectories with the same direction angle ϕ to obtain the suppression as a function of angle, $R_{AA}(p_f, \phi, H_Q)$. This is an important intermediary step since, depending on the details of QGP temperature evolution and the spatial variations in the temperature profile, energy loss may significantly depend on the parton's direction of motion. In earlier DREENA frameworks, this dependence was also present but was solely a consequence of the path-length distribution dependence on the angle. Once we have calculated $R_{AA}(p_f, \phi, H_Q)$, we can easily evaluate R_{AA} and v_2 observables as [195] (we here omit H_Q in the expressions, and denote $p_f = p_\perp$).

$$R_{AA}(p_\perp) = \frac{1}{2\pi}\int_0^{2\pi} R_{AA}(p_\perp, \phi)d\phi\,, \tag{6.16}$$

$$v_2(p_\perp) = \frac{\frac{1}{2\pi}\int_0^{2\pi}\cos(2\phi)R_{AA}(p_\perp, \phi)d\phi}{R_{AA}(p_\perp)}\,. \tag{6.17}$$

Note that, in Eqs. 6.16 and 6.17, using $R_{AA}(p_\perp, \phi)$, instead of the hadron $p_\perp$ spectrum, is computationally more efficient since $R_{AA}(p_\perp, \phi)$ is a well-behaved function, and the number of $p_\perp$ points where we need to evaluate $R_{AA}(p_\perp, \phi)$ is significantly smaller.

While the general expressions of the dynamical energy loss formalism are the same as in the DREENA-B framework [2], the fact that, in DREENA-A, the temperature entering the Eqs. (6.4 - 6.7) explicitly depends on the current parton position, notably complicates the implementation of these formulas, as we discuss in the following section.

6.1.2 Framework outline

Our previous DREENA-C and DREENA-B frameworks were based on computationally useful, but rough, approximations of the medium evolution: while in DREENA-C, there was no evolution, and the temperature remained constant both in time and along spatial dimensions, in DREENA-B, the medium was assumed to evolve according to 1D Bjorken approximation [138]. Due to these approximations, parton energy loss depended on its path length independently of its direction or production point. This allowed to analytically integrate energy-loss formulas to a significant extent, which notably reduced the number of required numerical integrations. Furthermore R_{AA} only needed to be averaged out over precalculated path-length distributions. Thus, these approximations of the medium evolution straightforwardly led to efficient computational algorithms for DREENA-C and DREENA-B.

DREENA-A framework, on the other hand, addresses fully general medium dynamics, with arbitrary spatio-temporal temperature distribution. The main input to the algorithm is the temperature profile $T_{profile}$ given as a three-dimensional matrix of temperature values at points with coordinates $(\mathrm{x}, \mathrm{y}, \tau)$ (in the input file, the values should be arranged in an array of quartets of the form $(\tau, \mathrm{x}, \mathrm{y}, T_{profile})$, and the lowest value of τ appearing in the data is taken to be τ_0). In addition to the temperature profile, the DREENA-A algorithm also takes, as inputs, the initial parton $p_\perp$ distributions $\frac{d^2\sigma}{dp_\perp^2}$ (each as an array of $(p_\perp, \frac{d^2\sigma}{dp_\perp^2})$ pairs) and the jet production probability distribution (as a matrix of probability density values in the transversal plane, formatted analogously as the profile temperature values). This level of generality requires a different approach than in previous frameworks. Since the DREENA-A algorithm takes arbitrary medium temperature evolution as the input, the energy loss has to be individually calculated for each parton trajectory.

This means that for each trajectory – given by the coordinates x_0 and y_0 of the parton origin (in the transversal plane) and the direction angle ϕ – we must first numerically evaluate integrals (6.8)

and (6.11). Since the current parton position – for a given trajectory – becomes a function of the proper time τ, integrands in (6.8) and (6.11) also become functions of τ, either through an explicit dependence, or via position and time dependent medium temperature (6.3). We numerically integrate these functions along the trajectory (parametrized by τ as $\mathrm{x} = \mathrm{x}_0 + \tau\cos\phi$, $\mathrm{y} = \mathrm{y}_0 + \tau\sin\phi$), starting from the origin at $(\mathrm{x}_0, \mathrm{y}_0)$ and moving in small integration steps along the direction ϕ (in practice, 0.1 fm step is sufficiently small for most of the profiles). The integration is terminated when the medium temperature at the current parton's position drops below $T_c = 155$ MeV [149], i.e., when the parton leaves the QGP phase. Also, we approximate that there are no energy losses before the initial time τ_0 (which is a parameter of the temperature evolution) and thus the first part of the trajectory, corresponding to $\tau < \tau_0$, is effectively skipped (i.e., τ_0 is taken as the lower limit of integration in (6.8) and (6.11)).

Once we, for a given trajectory, compute the integrals (6.8) and (6.11), we then perform the rest of procedure laid out by Eqs. (6.8-6.15). Most of the computation time is spent on numerical integrations, in particular for evaluating integrals in Eqs. (6.9,6.10). While, in principle, $n \to \infty$ in Eq. (6.9), in practice we show that $n = 5$ is sufficient for convergence in the case of quark jets, while for gluon jets $n = 7$ is needed. In general, the Quasi-Monte Carlo integration method turned out to be the most efficient and is used for all these integrals (as quasirandom numbers, we use precalculated and stored Halton sequences). The result of the integration, 6.15, is the final hadron suppression $R^{tr}_{AA}(p_\perp, H_Q)$ for the jet moving along the chosen trajectory, given as the function of its transversal momentum.

To obtain $R_{AA}(p_\perp, \phi, H_Q)$, we have to average this result over all production points (taking into account the provided jet production probability distribution) and repeat the procedure for many angles ϕ. In practice, this means that we must evaluate energy loss along a very large number of trajectories. This has significantly increased the computational complexity of the problem compared to DREENA-C and DREENA-B and required a number of optimisations.

6.1.3 Numerical optimisations of DREENA-A

We started by adapting optimisation methods that we successfully implemented in earlier versions. One useful approach was a tabulation and consequent interpolation of values for computationally expensive functions. In particular, this is crucial for the complicated integrals (6.4-6.7): while a two dimensional array is sufficient to tabulate $\frac{dE_{col}}{d\tau}$ (which is a function of T and p), values of $\frac{d^2N_{\mathrm{rad}}}{dxd\tau}$ (depending on τ, T, p and x) must be stored in a four-dimensional array. Tabulating such functions is done adaptively, with the density of evaluated points varying, depending on the function behaviour (i.e., using a denser grid where the functions change rapidly and sparser where the behaviour is smooth). In the case of these two functions, not only that the consequent interpolation can significantly reduce the overall number of integral evaluations, but the corresponding tables (for each particle type) can be evaluated only once and then permanently stored and reused for all trajectories and even for different temperature profiles. To further optimise the algorithm, we also precalculate the integral $\int \frac{d^2N_{\mathrm{rad}}}{dxd\tau} dx$ values and store a corresponding three-dimensional array (since it is a function of τ, T, and p).

When using this table-interpolation method, it is often necessary to make a function transformation before tabulation: e.g., it is more efficient and accurate to sample and later interpolate logarithm of a rapidly (nearly or approximately-exponentially) increasing function than the function itself (similarly, it is sometimes more optimal to tabulate ratio, or a product of functions than each of the functions separately). For example, it is much more optimal to tabulate and consecutively interpolate R_{AA}s (and other similarly behaving expressions) than the corresponding momentum distributions. This methodology is now extensively applied throughout DREENA-A (from some intermediate-level energy loss results to evaluating multi-dimensional integrals in the calculation of radiated gluon rates). Given the

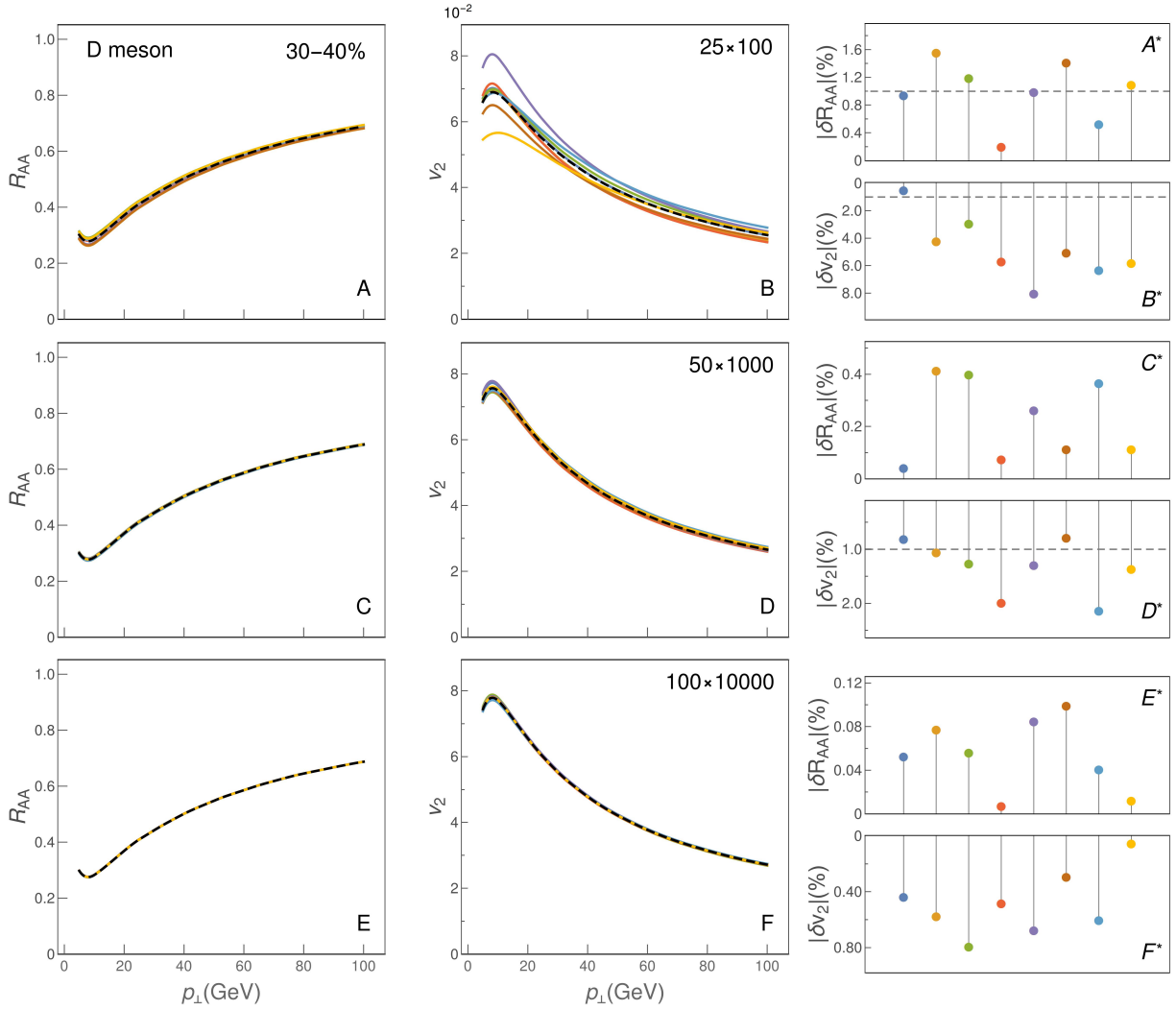

Figure 6.1: D meson R_{AA} (left) and v_2 (middle) at 30-40% centrality computed using different numbers of randomly generated trajectories (Monte Carlo approach), together with their deviations (right, scaled 1-norm was used as a metric) from the results averaged over the same ensemble of trajectories. The dashed horizontal line in rightmost panels indicates the threshold of 1% deviation. The top row depicts results obtained from sampling 25 trajectories at different angles originating from each of 100 randomly selected jet-production points; the middle row—50 angles from 1000 points; the bottom row—100 angles from 10000 points. Each panel shows the results of eight repeated computations (each with an independent ensemble of randomly generated trajectories), the dashed line representing the mean. $M = 1.2$ GeV. We use a single value $\mu_M/\mu_E = 0.5$ [73, 74] to make the figure clearer. Figure adapted from [4].

size of some of these tables and that many interpolations are needed, we ensured that the table lookup and interpolation algorithm are efficient.

As we encounter multiple numerical integrations at different stages of the computation, modifying their order was another type of optimisation, where the natural order (from the theoretical viewpoint) is not necessarily followed but is instead adapted to the particular function behaviour. Specifically, it turned out that a different order of integration (for radiative contribution) is optimal for heavy flavor particles compared to gluons. I.e., while it is natural, from the physical perspective, to start with the initial momentum distributions of partons and integrate over the radiative energy loss (see Eqs. (6.9,6.10)), it turned out that (for heavy flavor) the shape of the initial distributions necessitates a very high number of integration points to achieve the required computation precision. Reorganising

the formulas and postponing the integration over initial distributions to the very end turned much more computationally optimal for heavy flavor. A similar procedure in the case of light quarks allowed much of the integration to be carried out jointly for all quarks, since their effective masses are the same, but initial $p_\perp$ distributions differ.

The crucial optimisation in DREENA-A is the method used for averaging over the particle trajectories. In suppression calculations, it is common to carry out the averaging over production points and directions by Monte Carlo (MC) sampling, but it turned out that the equidistant sampling of both jet production points and direction angles was here significantly more efficient. We initially implemented the Monte Carlo approach, randomly selecting both the origin coordinates and the angles of particle trajectories. The binary collision density was used as the probability density for coordinates of origins, while the angles were generated from a uniform distribution. Convergence of the results by using this method required a large number of sampled trajectories, as illustrated in Fig. 6.1. The figure shows R_{AA} and v_2 results obtained by the DREENA-A algorithm for a different total number of trajectories (the computation was done for D meson traversing the temperature evolution generated using a Glauber initialised viscous hydrodynamic code [196], at 30-40% centrality class). The plots in the right column of Fig. 6.1 show the magnitude of the deviation of the particular curve from the median curve, where the latter is the arithmetic mean of all curves in the plot (as the measure of deviation of a function $f(p)$ from a reference function $\overline{f}(p)$ we use: $|\delta f| = \frac{\int |f(p)-\overline{f}(p)|dp}{\int |\overline{f}(p)|dp}$). We see that R_{AA} convergence is easily achieved, where relative deviations of the order of 1% are obtained by taking into account only 2500 trajectories (see Fig. 6.1-A and Fig. 6.1-A^*). Computing the v_2 value requires much more trajectories, i.e., we see a substantial scattering of the Monte Carlo results with 2500 trajectories, while $\sim 10^6$ trajectories are needed to reduce relative deviation below 1%. Note that a small number of sampled trajectories also causes a systematic error: the smaller the number of trajectories, the lower the averaged v_2.

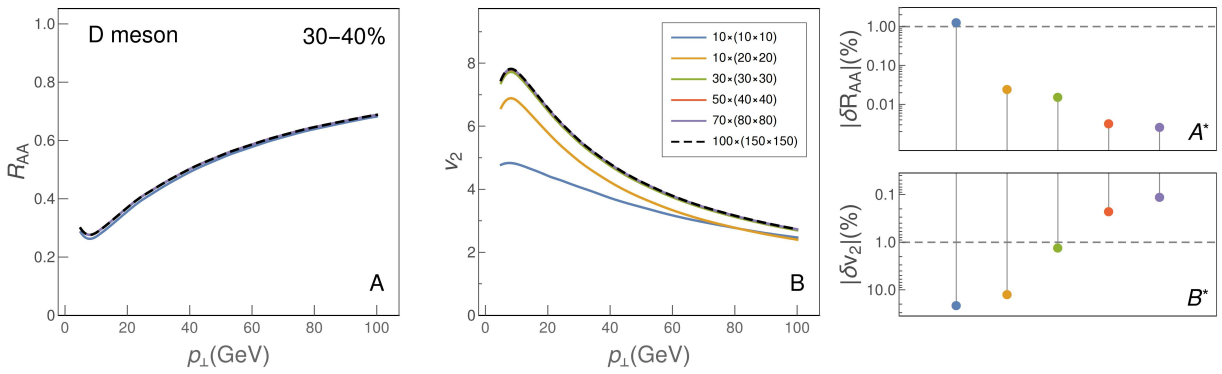

Figure 6.2: D meson R_{AA} (left) and v_2 (middle) at 30-40% centrality computed using different numbers of trajectories originating from equidistant points. Results are labeled by numbers $n_\phi \times (n_x \times n_y)$: jet directions are along n_ϕ uniformly distributed angles (from 0 to 2π) originating from each point of the n_x-by-n_y equidistant grid in the transversal plane. Deviation of each line from the baseline result (chosen as the outcome for $100 \times (150 \times 150)$ trajectories, dashed line) is shown in right panels. $M = 1.2$ GeV, $\mu_M/\mu_E = 0.5$. Figure adapted from [4].

When using the equidistant sampling method instead of Monte Carlo, we divide the transverse plane into an equidistant grid, whose points are used as jet origins. Energy loss for each trajectory is then weighted with the jet production probability at each point, and summed up. As production probability, we used the binary collision density evaluated using the optical Glauber model. In Fig. 6.2, we see that, for already ~ 10.000 evaluated trajectories, the integral has converged within 1% of the estimated 'proper' value. This modification resulted in a more than two orders of magnitude reduction

of the execution time. We also tested two hybrid variants: *i)* where trajectory origins were randomly selected but directions equidistantly, and *ii)* where production points were equidistantly selected, but directions randomly sampled. The convergence of the two variants interpolated between the MC sampling and the equidistant sampling (Figs. 6.1 and 6.2, respectively).

6.1.4 Convergence test of different DREENA methods

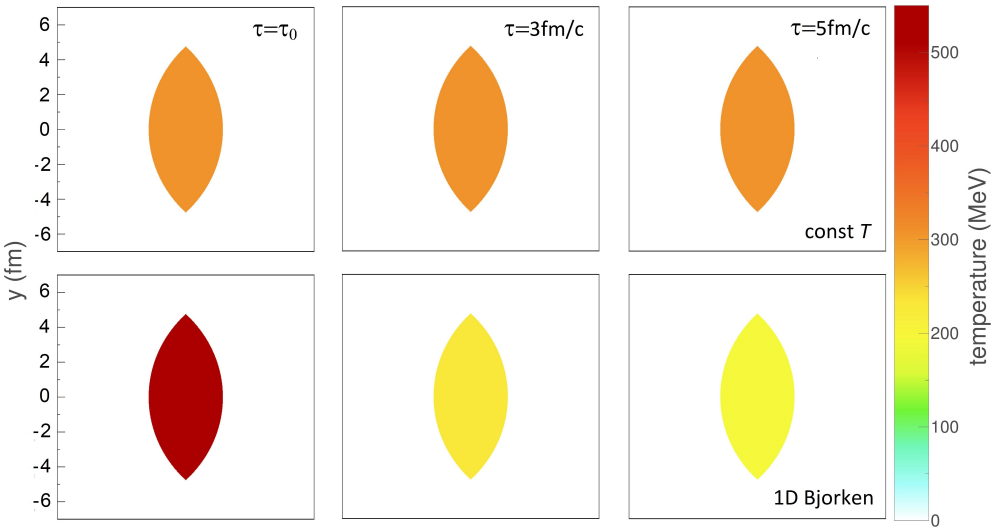

Figure 6.3: Temperature distribution (Pb + Pb collision, 30-40% centrality, mid-rapidity) for constant temperature [75] (first row) and 1D Bjorken evolution [2] (second row), at time (from left to right) $\tau = \tau_0$, 3, and 5 fm/c, represented by colour mapping. For constant temperature approximation, $\tau_0 = 0$ fm. For 1D Bjorken approximation, $\tau_0 = 0.6$ fm. Figure adapted from [4].

Finally, as a consistency check for DREENA-A, we compared its predictions with DREENA-C and DREENA-B results. For this purpose, we generated artificial T profiles suitable for this comparison, illustrated in Figure 6.3. The results of the DREENA-A and DREENA-B comparison, for R_{AA} and v_2, are shown in the upper panels of Figure 6.4, respectively. Lower panels of Figure 6.4 show the comparison of all three frameworks on the hard-cylinder collision profile constant in time (for this comparison, we modified the DREENA-B code to remove temperature dependence on time). We see that all frameworks lead to consistent results (up to computational precision), supporting the reliability of the DREENA-A.

6.2 Results and discussion

To demostrate the utility of the DREENA-A approach, we generated temperature profiles for Pb+Pb collisions at the full LHC energy ($\sqrt{s_{\rm NN}} = 5.02$ TeV) and Au+Au collisions at the full RHIC energy ($\sqrt{s_{\rm NN}} = 200$ GeV) using three different initialisations of the fluid-dynamical expansion.

First, we used optical Glauber initialisation at initial time $\tau_0 = 1.0$ fm without initial transverse flow. The evolution of the fluid was calculated using a 3+1D viscous fluid code from Ref. [196]. The parameters to describe collisions at the LHC energy were tuned to reproduce the low-$p_\perp$ data obtained in Pb+Pb collisions at $\sqrt{s_{\rm NN}} = 5.02$ TeV [187]. In particular, shear viscosity over entropy density ratio was constant $\eta/s = 0.12$, there was no bulk viscosity, and the equation of state (EoS) parametrisation was $s95p$-PCE-v1 [197]. For RHIC energy we used 'LH-LQ' parameters from Ref. [196],

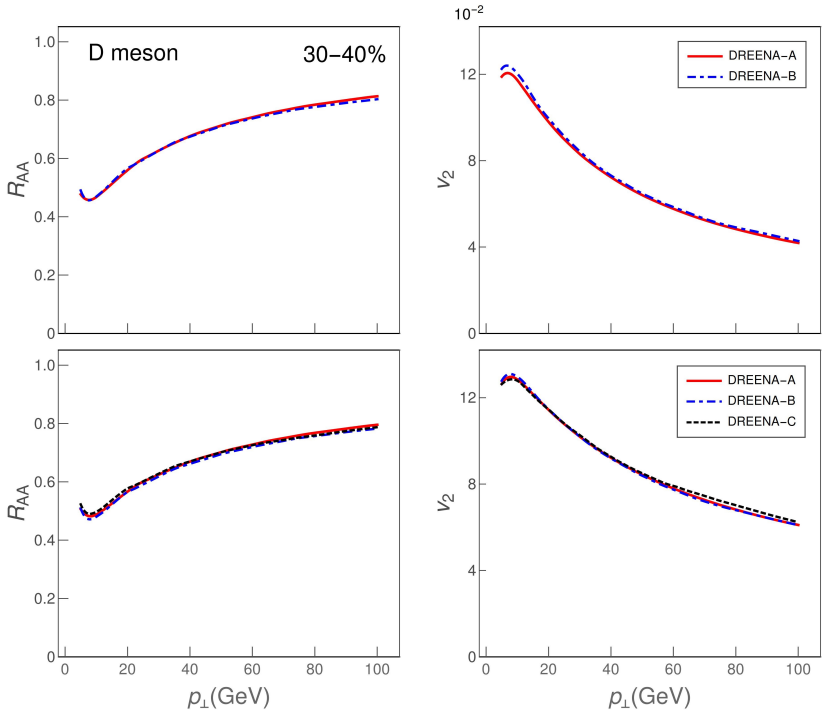

Figure 6.4: Comparison of different DREENA frameworks, for Bjorken medium evolution (upper panels) and for constant medium temperature approximation (lower panels), demonstrating inter-framework consistency. Upper panels show D meson R_{AA} (left) and v_2 (right) at 30-40% centrality computed using DREENA-A (supplied with temperature profiles representing Bjorken evolution) and DREENA-B. Lower panels show the same observables, computed using all three DREENA frameworks, when applied to the same constant temperature medium. $M = 1.2$ GeV, $\mu_M/\mu_E = 0.5$. Figure adapted from [4].

except that we used constant $\eta/s = 0.16$. Binary collision density from Glauber model was used as the probability distribution for the initial points of jets, while their directions were sampled from a uniform angular distribution.

Second, we used the EKRT initialisation [29, 30, 31], and evolved it using the same code we used to evolve the Glauber initialisation, but restricted to a boost-invariant expansion. In this case, the initial time was $\tau_0 = 0.2$ fm, and parameters were the favoured values of a Bayesian analysis of the data from Pb+Pb collisions at $\sqrt{s_{\rm NN}} = 2.76$ and 5.02 GeV, and from Au+Au collisions at $\sqrt{s_{\rm NN}} = 200$ GeV using the EoS parametrisation $s83s_{18}$ [181]. In particular, there was no bulk viscosity and the minimum value of temperature-dependent η/s was 0.18. Origins of the high-$p_\perp$ particles were sampled using the binary collision density of Glauber model, while the distribution of their directions was uniform.

Our third option was the T$_{\rm R}$ENTo initialisation [198] evolved using the VISH2+1 code [199] as described in [200, 201]. To describe collisions at LHC, parameters were based on a Bayesian analysis of the data at the above mentioned two LHC collision energies [201], although the analysis was done event-by-event, whereas we carried out the calculations using simple event-averaged initial states. In particular, the calculation included free streaming stage until $\tau_0 = 1.16$ fm, EoS based on the lattice results by the HotQCD collaboration [149], and temperature-dependent shear and bulk viscosity coefficients with the minimum value of $(\eta/s)_{\rm min} = 0.081$ and maximum of $(\zeta/s)_{\rm max} = 0.052$. For RHIC, we used the 'PTB' maximum a posteriori parameter values from Ref. [202], but changed the temperature-dependent shear viscosity coefficient $(\eta/s)(T)$ to a constant $\eta/s = 0.16$. The initial event-by-event collision points were used to generate the spatial probability distribution

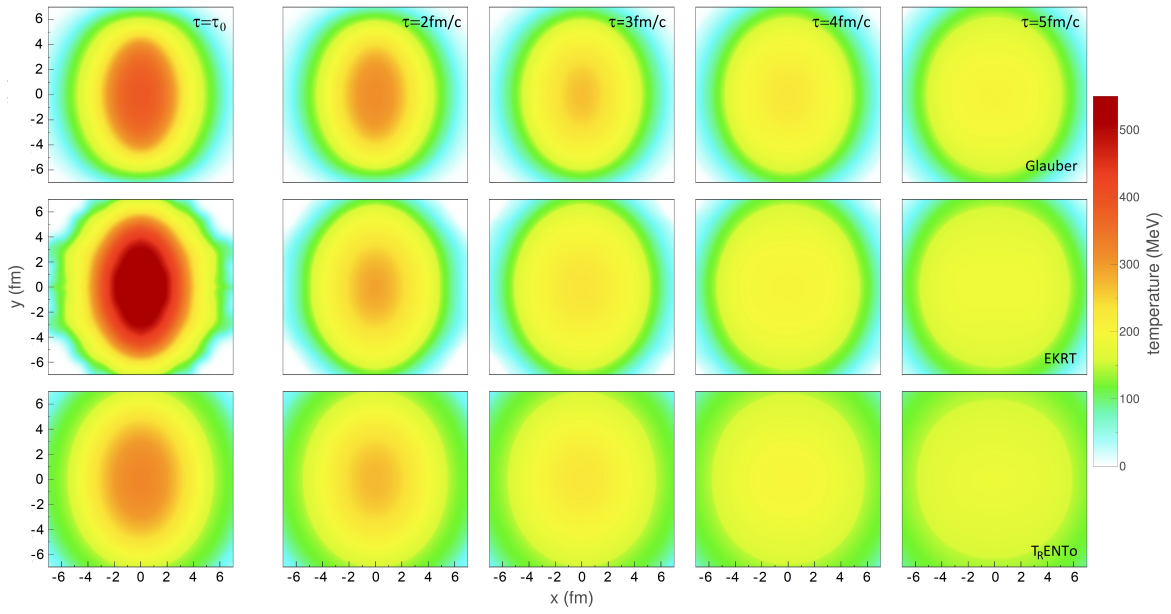

Figure 6.5: Temperature distribution (Pb + Pb $\sqrt{s_{\rm NN}} = 5.02$ TeV collision for 30-40% centrality at mid-rapidity) for different medium evolution models, at time (from left to right) $\tau = \tau_0$, 2, 3, 4 and 5 fm/c, represented by colour mapping. First row: 'Glauber', $\tau_0 = 1$ fm; second row: 'EKRT', $\tau_0 = 0.2$ fm; third row: 'T$_{\rm R}$ENTo', $\tau_0 = 1.16$ fm. Note that distributions in the first column correspond to different times. Figure adapted from [4].

for the initial coordinates of the high- $p_\perp$ particles, while their angular distribution was uniform.

All these calculations lead to an acceptable fit to measured charged hadron multiplicities, low-$p_\perp$ spectra, and $p_\perp$-differential v_2 in $10-20\%$, $20-30\%$, $30-40\%$, and $40-50\%$ centrality classes. As we may expect, different initialisations and initial times lead to a visibly different temperature evolution. This is demonstrated in Fig. 6.5 where we show the calculated temperature distributions in collisions at the LHC energy at various times. Looking at the profiles, it is easily noticeable that they evolve differently in space and time. Even if the initial anisotropy of the Glauber initialisation is lowest, later in time, its anisotropy is largest, since the very early start of EKRT initialisation, or the early free streaming of T$_{\rm R}$ENTo, dilute the spatial anisotropy very fast. That is, 'Glauber' exhibits larger asymmetry throughout the QGP evolution compared to the other two profiles (though 'EKRT' has larger asymmetry than 'Trento'), which might accordingly translate to differences in high-$p_\perp$ v_2. Similarly, the early start of EKRT leads to a large initial temperature, which is expected to result in a smaller R_{AA} than the other two profiles.

To test if these visual differences can be quantified through high-$p_\perp$ data at the LHC and RHIC, we used these profiles as an input to the DREENA-A to generate high-$p_\perp$ R_{AA} and v_2 predictions for charged hadrons, D and B mesons. As can be seen in Figs. 6.6 and 6.7, both R_{AA} and v_2 show notable differences for both experiments and all types of flavor. For example, 'EKRT' leads to the smallest R_{AA}, i.e., largest suppression, as can be expected based on the largest temperature. Similarly, the calculated high-$p_\perp$ v_2 depicts the same ordering as the system anisotropy during the evolution: 'Glauber' leads to the largest, followed by 'EKRT', while T$_{\rm R}$ENTo leads to the lowest v_2. Consequently, the DREENA-A framework can differentiate between temperature profiles by corresponding differences in high-$p_\perp$ observables, where these differences agree with the qualitative observations from Fig. 6.5. Since the differences in evolution are due to different initialisations, and different properties of the fluid (EoS and/or dissipative coefficients), R_{AA} and v_2 observables can be used to provide further constraints to the fluid properties. We note here that even low-$p_\perp$ data could be used to differentiate our three evolution scenarios, but such analysis would require evaluating χ^2 or a similar measure of

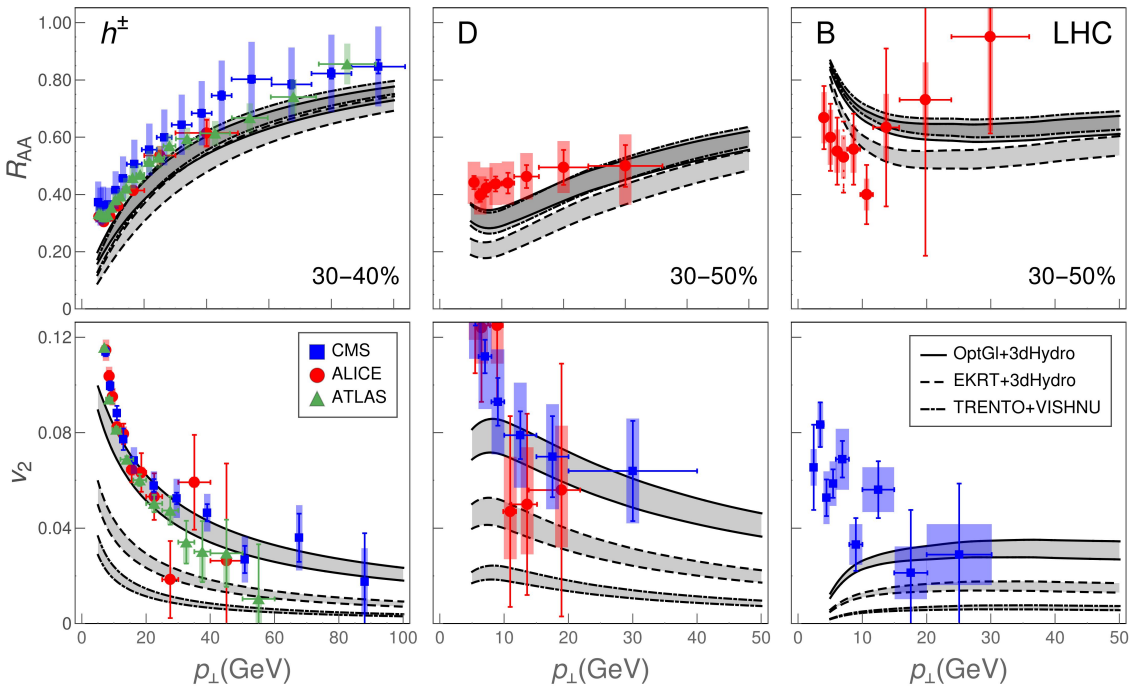

Figure 6.6: DREENA-A R_{AA} (top panels) and v_2 (bottom panels) predictions in Pb+Pb collisions at $\sqrt{s_{\rm NN}} = 5.02$ TeV are generated for different models of QGP medium evolution (indicated in the legend). Charged hadron (left) predictions are generated for 30-40% centrality, while D (middle) and B (right) meson predictions are generated for 30-50% centrality region. For charged hadrons, the predictions are compared with the experimental data from CMS [105, 127], ALICE [104, 125] and ATLAS [119, 126] experiments. For D mesons, the predictions are compared with ALICE [203, 130] and CMS [129] data. For B mesons predictions are compared with preliminary ALICE [204] and CMS [205] data. The boundary of each gray band corresponds to $0.4 < \mu_M/\mu_E < 0.6$ [73, 74]. Figure adapted from [4].

the quality of the fit, or computing Bayes factors [202]. The high-$p_\perp$ observables, on the other hand, show clear differences visible by the naked eye.

Moreover, from Figs. 6.6 and 6.6, we see that all types of flavor, at both RHIC and LHC, show apparent sensitivity to differences in medium evolution, making them equally suitable for exploring the bulk QGP properties with high-$p_\perp$ data. With the expected availability of precision data from the upcoming high-luminosity experiments at RHIC and LHC (see e.g., [212, 213, 214]), the DREENA-A framework provides a unique opportunity for exploring the bulk QGP properties. We propose that the adequate medium evolution should be able to reproduce high-$p_\perp$ observables in both RHIC and LHC experiments for different collision energies and collision systems, with reasonable accuracy. As demonstrated in this study, an equal emphasis should be given to light and heavy flavor, as they provide a valuable independent constraint for bulk medium evolution. Overall, DREENA-A provides a versatile tool to put large amounts of data generated at RHIC and LHC experiments to optimal use.

6.3 Summary

We presented the DREENA-A computational framework for tomography of quark-gluon plasma created in heavy-ion collisions at RHIC and the LHC. The tool is based on state-of-the-art energy loss calculation and can include arbitrary temperature profiles. This feature allows fully exploiting differ-

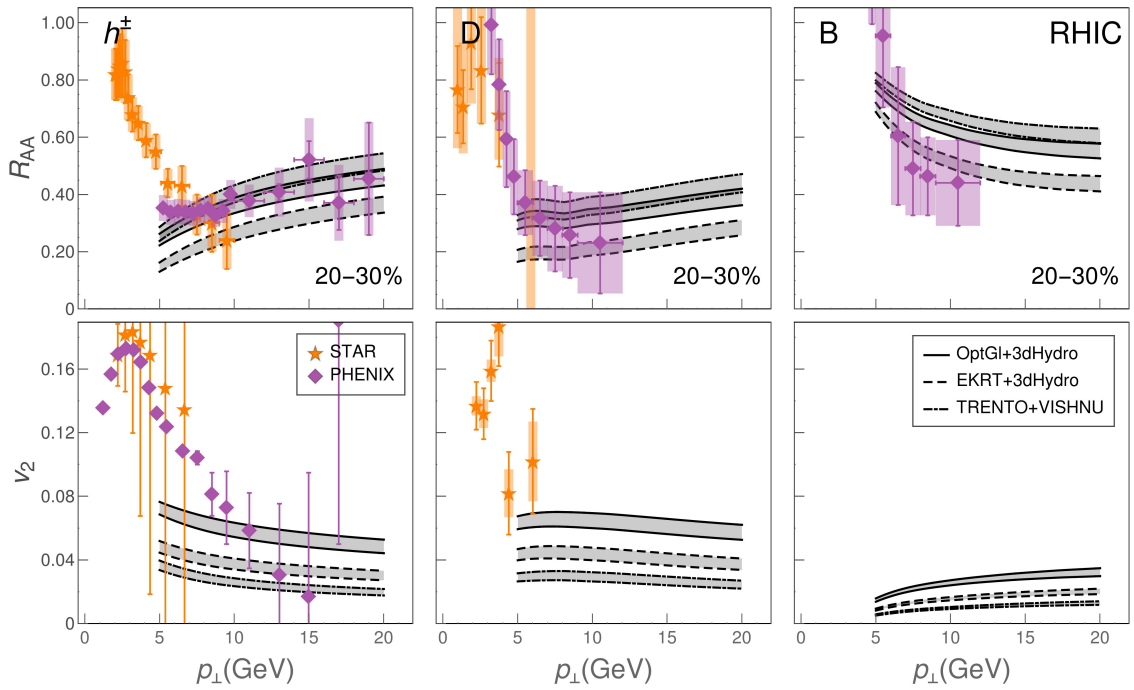

Figure 6.7: DREENA-A R_{AA} (top panels) and v_2 (bottom panels) predictions in Au+Au collisions at $\sqrt{s_{\rm NN}} = 200$ GeV are generated for different models of QGP medium evolution (indicated in the legend). Charged hadron (left), D meson (middle) and B meson (right) predictions are generated for 20-30% centrality region. The $h^{\pm}$ predictions are compared with π^0 data from PHENIX [123, 206] and $h^{\pm}$ data from STAR [207, 208] - note that for v_2 10-40% centrality data is shown for STAR. For D mesons, the predictions are compared with STAR [209, 210] data at 10-40% centrality and with PHENIX [211] data at 20-40%. B mesons predictions are compared with PHENIX [211] data at 20-40%. The boundary of each gray band corresponds to $0.4 < \mu_M/\mu_E < 0.6$ [73, 74]. Figure adapted from [4].

ent temperature profiles as the only input in the framework. We showed that the calculated high-$p_\perp$ R_{AA} and v_2 exhibit notable sensitivity to the details of the temperature profiles, consistent with intuitive expectations based on the profile visualisation. The DREENA-A framework applies to different types of flavor, collision systems, and collision energies. It can, consequently, provide an efficient and versatile QGP tomography tool for further constraining the bulk properties of this extreme form of matter. To facilitate this, we also provided the fully optimized, publicly available software for generating DREENA-A predictions. The code allows straightforwardly generating high-$p_\perp$ predictions for diverse models of QGP evolution.

Chapter 7

Importance of higher harmonics in quark-gluon plasma tomography

During the past two decades, an impressive experimental and theoretical effort has been invested in generating and exploring a new form of matter called Quark-Gluon Plasma (QGP) [115, 116, 77, 117, 118]. This form of matter consists of interacting and no longer confined quarks, antiquarks, and gluons [76, 114] and is created at extremely high energy densities achieved in ultra-relativistic heavy ion collisions at the Relativistic Heavy Ion Collider (RHIC) and the Large Hadron Collider (LHC) experiments. An unprecedented amount of data for different collision systems (large and small), collision energies, types of particles, momentum regions, centralities, etc., are generated in these experiments, and one of the major current goals is to optimally use these data to investigate the properties of this exciting form of matter.

As one of the latest experimental achievements, the high momentum (high-$p_\perp$) higher harmonics have recently become available at RHIC and the LHC. For example, for charged hadrons, the data are available up to the 7^{th} harmonic (for ATLAS [126]) and cover the $p_\perp$ region up to 100 GeV (for CMS [127]). For heavy flavor, the coverage is not that extensive (for both harmonics and momentum region)—still, the upcoming experimental data at high-luminosity LHC Run3 should provide these data for both light and heavy flavor with much higher precision. In the upcoming RHIC (sPHENIX and STAR) experiments, similar quality data is expected, with $p_\perp$ coverage up to 20 GeV. Even if the $p_\perp$ range accessible at RHIC is narrower than at the LHC, it is particularly useful for QGP tomography due to the pronounced difference between light and heavy flavor in that region. While these data (will) represent the state-of-the-art in the experimental sector, theoretically the higher harmonics at high-$p_\perp$ have not been well explored.

To use these data for QGP tomography, i.e., for exploring the bulk QGP properties through high-$p_\perp$ theory and data, one should first identify and address potential limitations, in particular related to coverage and design of different experiments. For example, four different methods are commonly used in the literature to evaluate v_n: two-particle cumulant $v_n\{2\}$, four-particle cumulant $v_n\{4\}$, event plane $v_n\{\mathrm{EP}\}$, and scalar product $v_n\{\mathrm{SP}\}$ methods (see section 7.1.3 for more details). Do these methods provide consistent results, especially when different experimental collaborations even define $v_n\{\mathrm{SP}\}$ in different ways?

Furthermore, in experimental analysis, the scalar product method correlates the particle of interest at midrapidity with the bulk medium constituents at higher rapidity regions to avoid non-flow

effects on measured v_n [126, 127]. From theoretical perspective, this means the use of the experimental definition for $v_n\{\mathrm{SP}\}$ necessitates 3+1D hydrodynamic modeling for event-by-event simulations. However, 3+1D simulations are computationally several orders of magnitude more demanding than 2+1D simulations and consequently time-wise impractical for high precision QGP tomography. Thus, the question arises whether it would be plausible to compare $v_n\{\mathrm{SP}\}$ obtained in boost-invariant 2+1D simulations to experimental data in a model where the high- and low-$p_\perp$ particles have separate sources (fragmenting jets and a thermal fireball, respectively), and are thus uncorrelated.

Next, for the second harmonic, v_2, event-by-event fluctuations are expected to either have a significant effect on v_2 values [135], or to be small enough to be considered negligible [215, 164]. However, these studies were done in limited and different centrality regions. It is expected [92] that the effects of event-by-event fluctuations increase with decreasing centrality. Thus, it is important to systematically investigate and quantify these effects for the high-$p_\perp$ region at different centralities.

Therefore, the study presented in this manuscript has the following main goals:

(i) Explore to what extent the different methods for calculating higher harmonics are compatible with each other.

(ii) Explore the importance of event-by-event fluctuations and correlations to high-$p_\perp$ v_2 and R_{AA}.

(iii) Explore the qualitative and quantitative effects of different medium evolution scenarios on high-$p_\perp$ higher harmonics, and how well the existing high-$p_\perp$ data can be reproduced without further tuning of parameters.

Overall, this study explores whether and how high-$p_\perp$ higher harmonics, with an adequate theoretical framework, can provide further constraints to the bulk QGP properties.

7.1 Methods

7.1.1 Outline of DREENA-A framework

To use the high-$p_\perp$ particles to explore the bulk properties, we developed a fully optimized modular framework DREENA-A (for more details see Section 6). We further optimized the framework for this study to efficiently incorporate any, arbitrary, event-by-event fluctuating temperature profile within the dynamical energy loss formalism. Due to the very large amount of temperature profile data processed in event-by-event calculations, we optimized file handling and formats. Also, we reorganized the parallelization of computation, as well as ensured that spatio-temporal resolution and calculation precision are optimal and adjusted to the event-by-event type of profiles.

The framework does not have fitting parameters within the energy loss model (i.e., all parameters used in the model correspond to standard literature values), which allows to systematically compare the data and predictions obtained by the same formalism and parameter set. Therefore different temperature profiles (which are the only input in the DREENA-A framework) resulting from different initial states, and QGP properties, can be distinguished by the high-$p_\perp$ observables they lead to, and the bulk QGP properties can be further constrained by studying low and high-$p_\perp$ theory and data jointly.

The dynamical energy loss formalism [58, 64, 59] has several important features, all of which are needed for accurate predictions [68]: *i)* QCD medium of *finite* size and temperature consisting of dynamical (i.e., moving) partons. *ii)* Calculations are based on generalized Hard-Thermal-Loop approach [62], with naturally regulated infrared divergences [58, 59, 65]. *iii)* Both radiative [58,

64] and collisional [59] energy losses are calculated in the same theoretical framework and apply to both light and heavy flavors. *iv)* The framework is generalized toward running coupling [49] and finite magnetic mass [66]. We have also investigated the validity of the widely used soft-gluon approximation [67], but found it a very good approximation which does not need to be relaxed.

The initial quark spectrum, for light and heavy partons, is computed at next to leading order [97, 98]. We use DSS [100] fragmentation functions to generate charged hadrons, and BCFY [101, 139] and KLP [140] fragmentation functions for D and B mesons, respectively. To generate high-$p_\perp$ predictions, we use the same parameter set as in DREENA-A from Section 6. Specifically, we assume effective light quark flavors $n_f = 3$ and $\Lambda_{QCD} = 0.2$ GeV. The temperature-dependent Debye mass μ_E is obtained by applying the procedure from [145] and leads to results compatible with the lattice QCD [216, 217]. For the gluon mass, we assume $m_g = \mu_E/\sqrt{2}$ [65], and for light quark mass $M = \mu_E/\sqrt{6}$. The charm mass is $M = 1.2$ GeV and the bottom mass is $M = 4.75$ GeV. For magnetic to electric mass ratio, we use $\mu_M/\mu_E = 0.5$ [73, 74].

7.1.2 Modeling the bulk evolution

We investigate three different event-by-event initializations for the bulk evolution. The first is Monte Carlo Glauber (MC-Glauber) initialization at initial time $\tau_0 = 1.0$ fm without initial transverse flow. We assign the binary collision points at halfway between the two colliding nucleons and convert these points to a continuous binary collision density using 2-D Gaussian distributions:

$$n_{BC}(x,y) = \frac{1}{2\pi\sigma_{BC}^2} \sum_{i=1}^{N_{BC}} \exp\left(-\frac{(x-x_i)^2+(y-y_i)^2}{2\sigma_{BC}^2}\right), \tag{7.1}$$

with a width parameter $\sigma_{BC} = 0.35$ fm. The binary collision density is then converted to energy density with the formula:

$$\epsilon(x,y) = C_0(n_{BC} + c_1 n_{BC}^2 + c_2 n_{BC}^3), \tag{7.2}$$

and further extended in the longitudinal direction using the LHC parametrization from Ref. [196]. The evolution of the fluid is calculated using a 3+1D viscous fluid code [196], with a constant shear viscosity over entropy density ratio $\eta/s = 0.03$ and no bulk viscosity. The equation of state (EoS) parametrization is $s95p$-PCE-v1 [197]. The model parameters were tuned to ALICE charged particle multiplicity [218] and $v_n(p_\perp)$ data [219] for 10-20%, 20-30% and 30-40% centrality classes in Pb+Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV.

The second model is the T$_\mathrm{R}$ENTo initialization [198] with a free streaming stage until $\tau_0 = 1.16$ fm, further evolved using the VISH2+1 code [199] as described in [200, 201]. The parameters in this calculation are based on a Bayesian analysis of the data at Pb+Pb collisions at $\sqrt{s_{NN}} = 2.76$ and 5.02 TeV [201]. In particular the calculation includes temperature dependent shear and bulk viscosity coefficients with the minimum value of $\eta/s = 0.081$ and maximum of $\zeta/s = 0.052$. The EoS [200] is based on the lattice results by the HotQCD collaboration [149].

The third investigated initialization model is IP-Glasma [27, 28]. The calculated event-by-event fluctuating initial states [220] are further evolved [221] using the MUSIC code [222, 223, 224] constrained to boost-invariant expansion. In these calculations, the switch from Yang-Mills to fluid-dynamical evolution takes place at $\tau_{\mathrm{switch}} = 0.4$ fm, shear viscosity over entropy density ratio is constant $\eta/s = 0.12$, and the temperature-dependent bulk viscosity coefficient over entropy density ratio has the maximum value $\zeta/s = 0.13$. The equation of state is based on the HotQCD lattice results [149] as presented in Ref. [225].

7.1.3 Flow analysis

Scalar product and event plane methods

We start by defining the low-$p_\perp$ normalized flow vector for n-th harmonic based on M particles as:

$$Q_n = \frac{1}{M}\sum_{j=1}^{M} e^{in\phi_j} \equiv |v_n| e^{in\Psi_n}\,, \tag{7.3}$$

where Ψ_n is the event plane angle: $\Psi_n = \arctan(\frac{\mathrm{Im}\, Q_n}{\mathrm{Re}\, Q_n})/n$.

Similarly to low $p_\perp$, we can define the flow vector for a high-$p_\perp$ bin as:

$$q_n^{\mathrm{hard}} = \frac{\frac{1}{2\pi}\int_0^{2\pi} e^{in\phi} R_{AA}(p_\perp,\phi)\, d\phi}{R_{AA}(p_\perp)}\,, \tag{7.4}$$

and single-event high-$p_\perp$ flow coefficients v_n^{hard} as [135]

$$v_n^{\mathrm{hard}} = \frac{\frac{1}{2\pi}\int_0^{2\pi} \cos[n(\phi - \Psi_n^{\mathrm{hard}}(p_\perp))] R_{AA}(p_\perp,\phi)\, d\phi}{R_{AA}(p_\perp)}\,, \tag{7.5}$$

where $R_{AA}(p_\perp)$ is defined as:

$$R_{AA}(p_\perp) = \frac{1}{2\pi}\int_0^{2\pi} R_{AA}(p_\perp,\phi)\, d\phi\,. \tag{7.6}$$

and the event plane angle $\Psi_n^{\mathrm{hard}}(p_\perp)$ is defined as:

$$\Psi_n^{\mathrm{hard}}(p_\perp) = \frac{1}{n}\arctan\left(\frac{\int_0^{2\pi}\sin(n\phi) R_{AA}(p_\perp,\phi)\, d\phi}{\int_0^{2\pi}\cos(n\phi) R_{AA}(p_\perp,\phi)\, d\phi}\right)\,. \tag{7.7}$$

The high-$p_\perp$ v_n is then calculated by correlating q_n with Q_n [135, 113, 215]:

$$\begin{aligned} v_n^{\mathrm{hard}}\{\mathrm{SP}\} &= \frac{\langle \mathrm{Re}\,(q_n^{\mathrm{hard}}(Q_n)^*)\rangle_{\mathrm{ev}}}{\sqrt{\langle Q_n (Q_n)^*\rangle_{\mathrm{ev}}}} \\ &= \frac{\langle |v_n^{\mathrm{hard}}||v_n| \cos[n(\Psi_n^{\mathrm{hard}}(p_\perp) - \Psi_n)]\rangle_{\mathrm{ev}}}{\sqrt{\langle |v_n|^2\rangle_{\mathrm{ev}}}}\,. \end{aligned} \tag{7.8}$$

We may also simply calculate the high-$p_\perp$ anisotropy with respect to the event plane Ψ_n, which we shall denote as the "event plane" v_n [215]:

$$\begin{aligned} v_n\{\mathrm{EP}\} &= \langle\langle \cos[n(\phi^{\mathrm{hard}} - \Psi_n)]\rangle\rangle_{ev} \\ &= \langle v_n^{\mathrm{hard}} \cos[n(\Psi_n^{\mathrm{hard}} - \Psi_n)]\rangle_{ev}\,. \end{aligned} \tag{7.9}$$

For our theoretical $v_n\{\mathrm{SP}\}$, the reference flow vector Q_n is calculated using only midrapidity particles. In order to reduce non-flow effects, it is common in experiments to introduce a rapidity gap between the particles of interest and the reference flow particles. ATLAS defines the scalar product v_n as [126]:

$$v_n\{\mathrm{SP_{ATLAS}}\} = \frac{\mathrm{Re}\,\langle\langle e^{in\phi}(Q_n^{-|+})^*\rangle\rangle_{\mathrm{ev}}}{\sqrt{\langle Q_n^-(Q_n^+)^*\rangle_{\mathrm{ev}}}}\,, \tag{7.10}$$

where $Q_n^- = \frac{1}{M^-}\sum_{j=1}^{M^-} e^{in\phi_j}$ refers to particles in the rapidity interval $-4.9 < \eta < -3.2$ and Q_n^+ similarly to particles in the interval $3.2 < \eta < 4.9$, while $e^{in\phi}$ is associated with particles in midrapidity $|\eta| < 2.5$. $Q_n^{-|+}$ indicates that particle of interest with $\eta < 0$ are coupled to Q_n^+ and particles with $\eta > 0$ to Q_n^- to maximize the rapidity gap. Since our high-$p_\perp$ particles are produced at $\eta = 0$, the choice of Q_n^+ or Q_n^- for the correlation is arbitrary.

CMS definition for the scalar product is [127]

$$v_n\{\mathrm{SP}_{\mathrm{CMS}}\} = \frac{\mathrm{Re}\,\langle Q_n Q_{nA}^*\rangle_{\mathrm{ev}}}{\sqrt{\frac{\langle Q_{nA}Q_{nB}^*\rangle_{\mathrm{ev}}\langle Q_{nA}Q_{nC}^*\rangle_{\mathrm{ev}}}{\langle Q_{nB}Q_{nC}^*\rangle_{\mathrm{ev}}}}}\,, \tag{7.11}$$

where the flow vector $Q_n = \sum_{j=1}^{M} e^{in\phi_j}$ consists of particles of interest in midrapidity $|\eta| < 1.0$, vectors $Q_{nA}, Q_{nB} = \sum_{j=1}^{M_{A,B}} E_T\, e^{in\phi_j}$ are measured from the HF calorimeters at $2.9 < |\eta| < 5.2$, one at the negative and the other at the positive rapidity, and the third reference vector $Q_{nC} = \sum_{j=1}^{M_C} p_\perp\, e^{in\phi_j}$ is obtained from tracks with $|\eta| < 0.75$. If the particle of interest comes from the positive-η side of the tracker, then Q_{nA} is calculated using the negative-η side of HF, and vice versa.

Cumulant method

For 2- and 4-particle cumulant analysis, we use the unnormalized flow vector:

$$\tilde{Q}_n = \sum_{j=1}^{M} e^{in\phi_j}\,. \tag{7.12}$$

The low-$p_\perp$ integrated reference flow is calculated using Eqs. (7)-(18) from Ref. [226]: The 2-particle cumulant v_n is defined as:

$$v_n\{2\} = \sqrt{c_n\{2\}}\,, \tag{7.13}$$

where the second order cumulant $c_n\{2\}$ equals the event-averaged 2-particle correlation $\langle\langle 2\rangle\rangle_{\mathrm{ev}}$. The 4-particle cumulant v_n is:

$$v_n\{4\} = \sqrt[4]{-c_n\{4\}}\,, \tag{7.14}$$

where $c_n\{4\}$ is the 4th order cumulant $\langle\langle 4\rangle\rangle_{\mathrm{ev}} - 2\langle\langle 2\rangle\rangle_{\mathrm{ev}}^2$.

For a single event, the 2-particle correlation is

$$\langle 2\rangle = \frac{|\tilde{Q}_n|^2 - M}{W_2}\,, \tag{7.15}$$

with a combinatorial weight factor $W_2 = M(M-1)$ and the single-event 4-particle correlation is:

$$\begin{aligned}\langle 4\rangle =& \frac{|\tilde{Q}_n|^4 + |\tilde{Q}_{2n}|^2 - 2\mathrm{Re}|\tilde{Q}_{2n}\tilde{Q}_n^*\tilde{Q}_n^*|}{W_4} \\ & - 2\frac{2(M-2)|\tilde{Q}_n|^2 - M(M-3)}{W_4}\,,\end{aligned} \tag{7.16}$$

with $W_4 = M(M-1)(M-2)(M-3)$.

Using the weight factors defined above, the weighted average of a k-particle correlation over multiple events is then

$$\langle\langle k\rangle\rangle_{\text{ev}} = \frac{\sum_{i=1}^{N_{\text{events}}} W_{k,i}\,\langle k\rangle_i}{\sum_{i=1}^{N_{\text{events}}} W_{k,i}}\,. \tag{7.17}$$

Once the reference flow has been determined, the p_T-differential flow can be calculated using Eqs. (20)-(35) of [226]. Here we denote the flow vector in a $p_\perp$ bin with m_q particles as:

$$q_n = \sum_{j=1}^{m_q} e^{in\phi_j}\,. \tag{7.18}$$

For high-$p_\perp$ particles, q_n is calculated from the distribution

$$q_n = \int_0^{2\pi} e^{in\phi}\frac{dN}{dp_\perp d\phi}d\phi\,, \tag{7.19}$$

with the associated multiplicity:

$$m_q = \int_0^{2\pi} \frac{dN}{dp_\perp d\phi}d\phi\,. \tag{7.20}$$

For high-$p_\perp$ differential flow, none of the particles in a $p_\perp$ bin are included in the calculation of the reference flow, so the weight factors are $W_2' = m_q M$ and $W_4' = m_q M(M-1)(M-2)$, and the 2-particle correlation is simply:

$$\langle 2'\rangle = \frac{q_n\tilde{Q}_n^*}{W_2'}, \tag{7.21}$$

while the 4-particle correlation is

$$\langle 4'\rangle = \frac{q_n\tilde{Q}_n\tilde{Q}_n^*\tilde{Q}_n^* - q_n\tilde{Q}_n\tilde{Q}_{2n}^* - 2Mq_n\tilde{Q}_n^* + 2q_n\tilde{Q}_n^*}{W_4'}\,. \tag{7.22}$$

With the knowledge of the correlations, we can calculate the differential cumulants:

$$\begin{aligned} d_n\{2\} &= \langle\langle 2'\rangle\rangle_{\text{ev}}\,,\\ d_n\{4\} &= \langle\langle 4'\rangle\rangle_{\text{ev}} - 2\langle\langle 2'\rangle\rangle_{\text{ev}}\langle\langle 2\rangle\rangle_{\text{ev}} \end{aligned} \tag{7.23}$$

and the differential flow:

$$\begin{aligned} v_n'\{2\} &= \frac{d_n\{2\}}{\sqrt{c_n\{2\}}}\,,\\ v_n'\{4\} &= -\frac{d_n\{4\}}{(-c_n\{4\})^{3/4}}. \end{aligned} \tag{7.24}$$

7.2 Results and discussion

7.2.1 Compatibility of analysis methods

In Fig. 7.1, we compare $v_n(p_\perp)$ for high-$p_\perp$ particles obtained using six different methods: 2- and 4-particle cumulants $v_n\{2\}$ and $v_n\{4\}$ given by Eq. (7.24), event plane $v_n\{\text{EP}\}$ defined by Eq. (7.9),

midrapidity scalar product $v_n\{\mathrm{SP}\}$ calculated using Eq. (7.8), scalar product $v_n\{\mathrm{SP}_{\mathrm{ATLAS}}\}$ as defined by the ATLAS collaboration (Eq. (7.10)), and scalar product $v_n\{\mathrm{SP}_{\mathrm{CMS}}\}$ as defined by the CMS collaboration (Eq. (7.11)). High-$p_\perp$ R_{AA} and v_n predictions were obtained using generalized DREENA-A framework with the temperature profiles calculated using the combination of 3+1D viscous fluid code and MC-Glauber initial conditions (i.e., the first bulk model described in the section 7.1.2).

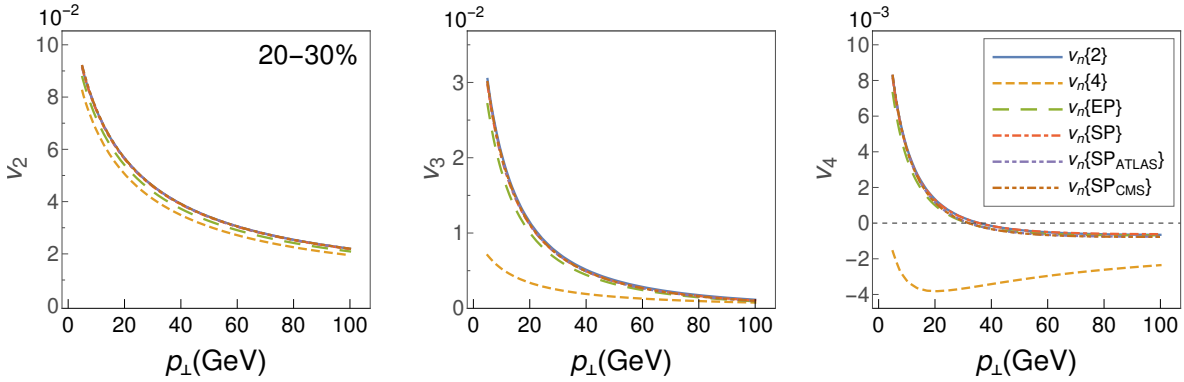

Figure 7.1: Charged hadron v_2 (left), v_3 (middle) and v_4 (right) in Pb+Pb collisions at $\sqrt{s_{\mathrm{NN}}} = 5.02$ TeV for 20-30% centrality class, computed using different analysis methods: 2-particle cumulant, 4-particle cumulant, event plane, midrapidity scalar product, ATLAS-defined scalar product, and CMS defined scalar product, each described in the section 7.1.3. Energy loss calculation was performed on MC-Glauber+3d-hydro temperature profiles, with $\mu_M/\mu_E = 0.5$.

As illustrated in Fig. 7.1, different scalar product methods for evaluating the v_n coefficients, and the 2-particle cumulant method, lead to the same results with $\approx 5\%$ level accuracy. In agreement with Refs. [215, 227, 164], the event plane results are also comparable to the scalar product results deviating only $\approx 10\%$, i.e., less than the current experimental uncertainty. The only method with significantly different results is the four-particle cumulant method $v_n\{4\}$, which is expected to differ from $v_n\{2\}$ in the presence of event-by-event fluctuations [92, 228]. The equivalence of different approaches simplifies comparison between theoretical predictions and experimental results, since a theoretical prediction calculated using any method (with the exception of the 4-particle cumulant method) can be directly compared to experimental data analyzed using any method. We have also checked that, in the scalar product method, the rapidity of particles used to calculate the reference flow vector has a negligible impact on high-$p_\perp$ particle v_n in our framework and setup, allowing us to make meaningful $v_n\{\mathrm{SP}\}$ data comparisons using the boost-invariant hydro simulations. However, it must be remembered that the scalar product method with large rapidity gap can be affected by the event plane decorrelation at different rapidities [229, 230]. In our approach the event plane is the same independent of rapidity, and thus the effect of decorrelation is not included. How the event plane depends on rapidity depends on the model used to create the longitudinal structure of the initial state, and since there are very few theoretical constraints for it, we leave these studies for a later work.

7.2.2 Event-by-event fluctuations

To investigate the influence of event-by-event fluctuations on high-$p_\perp$ observables, MC-Glauber initial conditions for all events within a single centrality class were averaged (we kept reaction planes aligned, and averaged binary collision densities before converting to energy density, (Eq. 7.2)) and then evolved using the 3+1D viscous fluid code (in a single run, instead of one run for each event). Obtained smooth temperature profile was used to calculate high-$p_\perp$ predictions, and R_{AA} as well as $v_2\{2\}$ and $v_2\{4\}$ results were compared to those obtained using full event-by-event calculations (evolved separately for each event), see Fig 7.2.

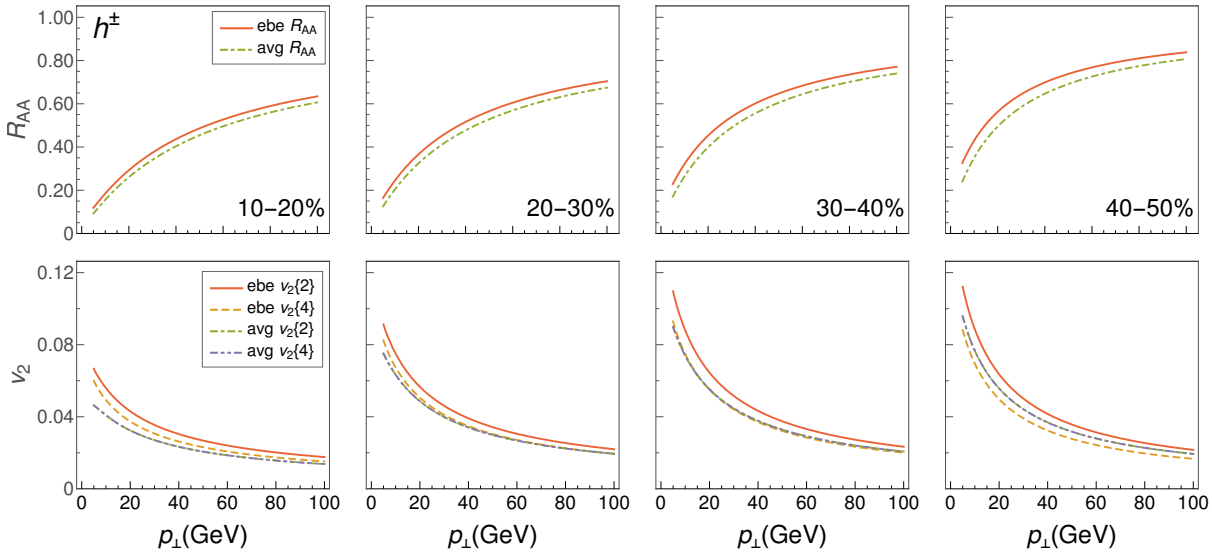

Figure 7.2: *Upper panels:* charged hadron R_{AA} calculated using event-by-event (ebe) fluctuating temperature profiles compared to R_{AA} calculated using a smooth temperature profile (avg). *Lower panels:* charged hadron $v_n\{2\}$ and $v_n\{4\}$ calculated using event-by-event (ebe) fluctuating temperature profiles compared to $v_n\{2\}$ and $v_n\{4\}$ calculated using a smooth temperature profile (avg). Calculation was done for Pb+Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV, $\mu_M/\mu_E = 0.5$, using MC-Glauber+3d-hydro bulk evolution. Each column represents different centrality class (from left to right: 10-20%, 20-30%, 30-40% and 40-50%).

We see that event-by-event fluctuations increase both R_{AA} and v_2. While the effect on the R_{AA} values is rather small ($\approx 7\%$) and does not have clear centrality dependence, the effect on $v_2\{2\}$ is more pronounced and increases with decreasing centrality. Quantitatively, we obtain that the average difference between event-by-event $v_2\{2\}$ and $v_2\{2\}$ calculated using smooth temperature profile goes from 14% for the 40-50% centrality class to 32% in the 10-20% centrality class. The observed centrality dependence can be explained by the fact that with the increase in centrality, the influence of geometry on $v_2\{2\}$ becomes larger, while at low centralities, event-by-event fluctuations have the dominant impact on $v_2\{2\}$. We also observe a $p_\perp$ dependence of these differences (generally decreasing with increasing $p_\perp$) and no notable difference between $v_2\{2\}$ and $v_2\{4\}$ when calculated on the smooth temperature profile, where initial state eccentricity fluctuations are absent.

7.2.3 Effects of initial state

To demonstrate the applicability of high-$p_\perp$ theoretical predictions as a QGP tomography tool, we generated three different sets of temperature profiles using three different initial conditions and hydrodynamics codes (see Section 7.1.2). Generalized DREENA-A from Section 6 was then used to calculate high-$p_\perp$ predictions, which are compared to experimental data and, for charged hadrons, presented in Fig. 7.3, and for D and B mesons in Fig. 7.4. As can be seen, different initializations of fluid-dynamical evolution lead to different high-$p_\perp$ predictions for both R_{AA} and v_2, v_3 and v_4, even though they all provide good agreement with low-$p_\perp$ data. Specific differences are visible already on the level of R_{AA} values, where the IP-Glasma model results in discernibly stronger suppression. The differences in predictions become even higher when we consider the v_2 observable, with T$_\mathrm{R}$ENTo leading to lower v_2 than IP-Glasma, while MC-Glauber predictions are far above the two. A similar magnitude of relative differences is also obtained for v_3 and v_4 predictions, with an additional qualitative signature appearing for these observables: we notice that some initializations lead to negative

values of high-$p_\perp$ v_3 and v_4, i.e., models can differ even in the expected sign of the flow coefficients.

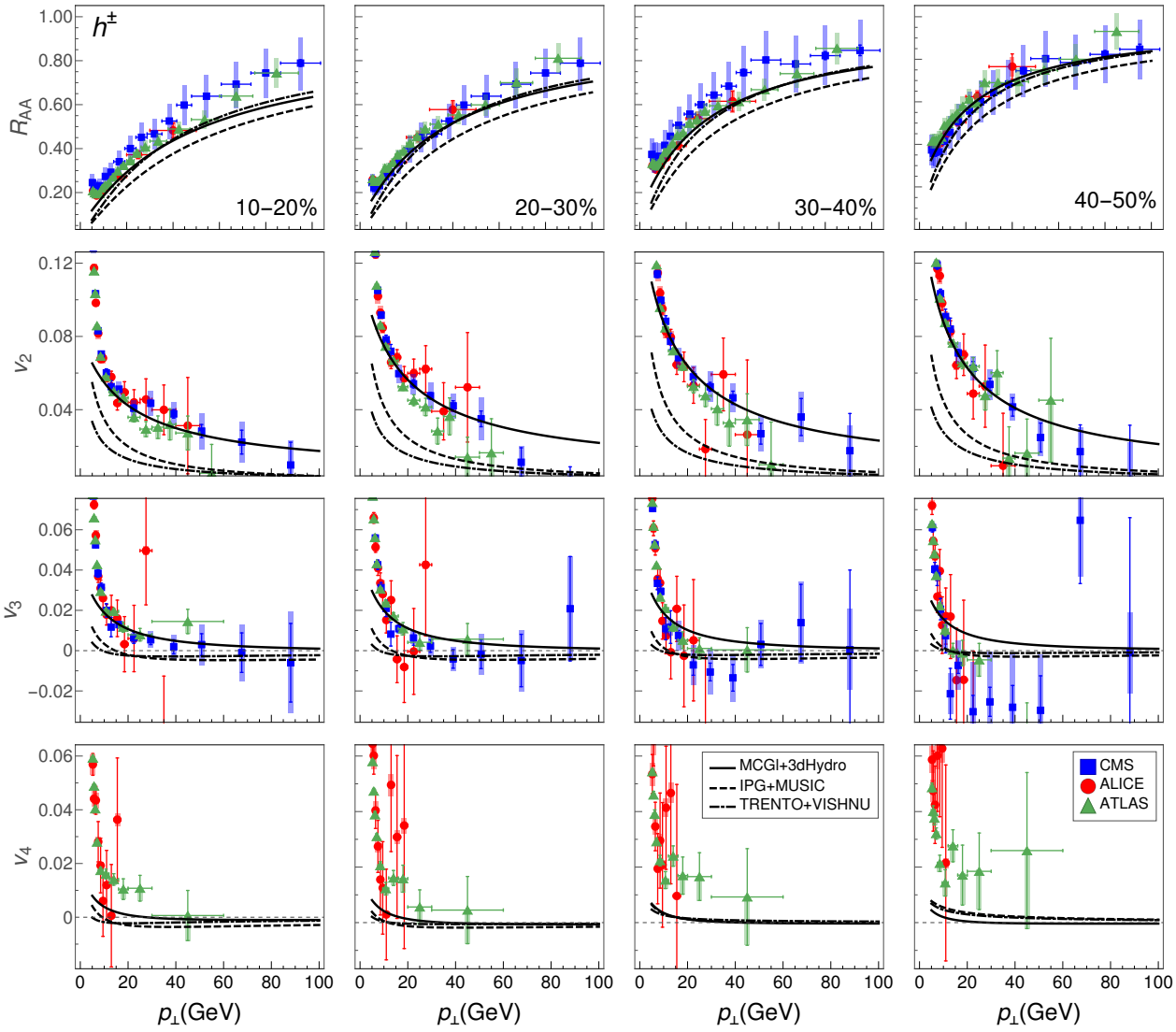

Figure 7.3: Charged hadron R_{AA} (first row) v_2 (second row), v_3 (third row) and v_4 (fourth row) in Pb+Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV for different initializations of the QGP evolution (indicated in the legend). Theoretical predictions, obtained using SP method, are compared to CMS [105, 127] (blue squares), ALICE [104, 125] (red circles) and ATLAS [119, 126] (green triangles) data. Columns 1-4 correspond to, respectively, 10-20%, 20-30%, 30-40% and 40-50% centrality classes. $\mu_M/\mu_E = 0.5$.

Since DREENA-A does not have fitting parameters in the energy loss (the only inputs are the temperature profile and binary collisions, which come as a direct output from fluid-dynamical calculation and the initial state model), Figs. 7.3 and 7.4 demonstrate that high-$p_\perp$ R_{AA} and higher harmonics can distinguish between different initializations and temperature profiles, and subsequently further constrain their parameters. Furthermore, Fig. 7.4 suggests that heavy flavor high-$p_\perp$ observables are even more sensitive to different temperature profiles than the light flavor. We also see that predictions for high-$p_\perp$ higher harmonics can be either positive or negative. Thus, the high-$p_\perp$ sector can provide both quantitative and qualitative constraints for different initial states.

Presently, of the considered models, the best agreement is observed for MC-Glauber. This result is compatible with our earlier findings [187], where the best agreement with high-$p_\perp$ data was found by delaying the start of transverse expansion and energy loss to time $\tau_0 \approx 1.0$ fm. However, all

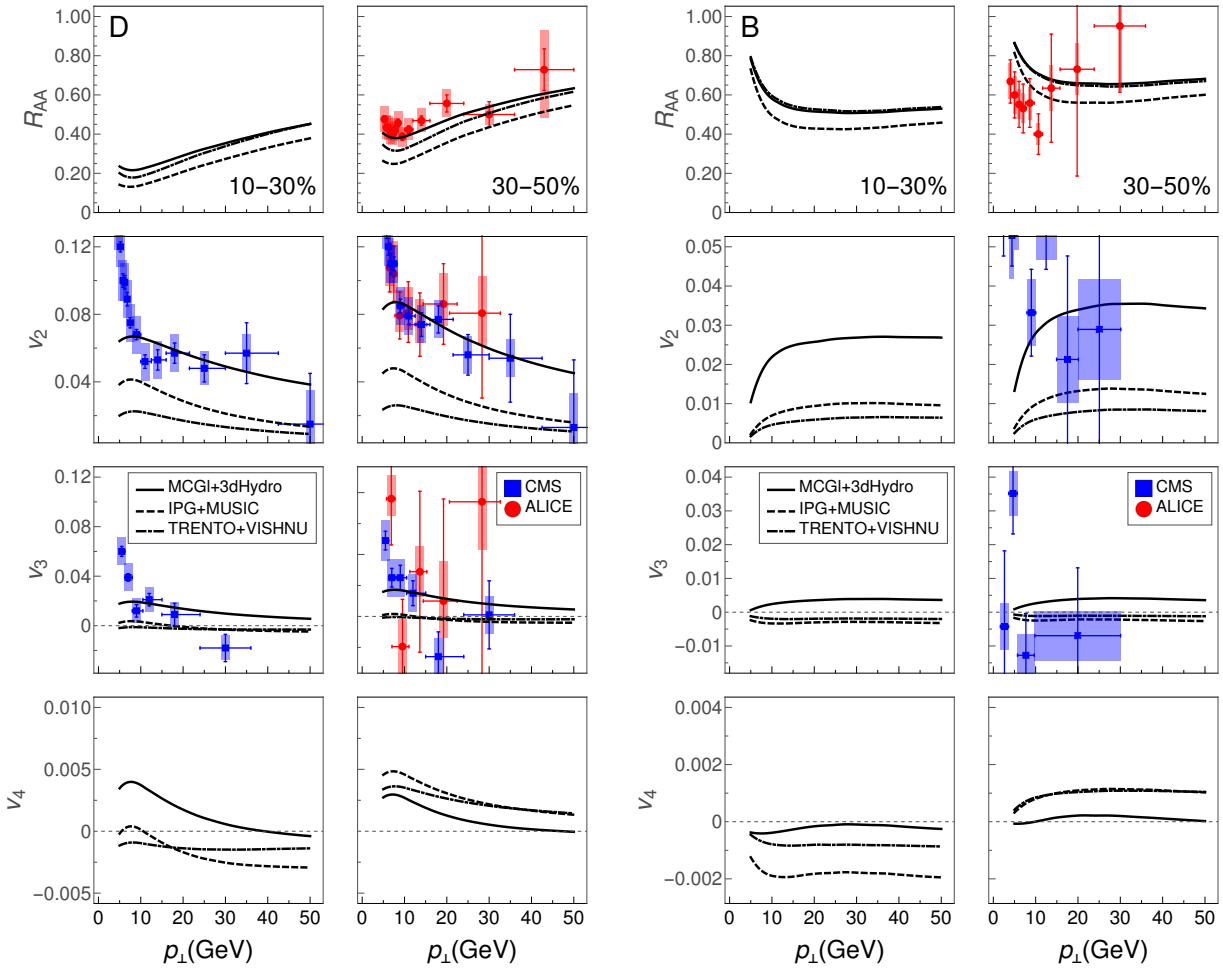

Figure 7.4: D meson (left 4×2 panel) and B meson (right 4×2 panel) predictions in Pb+Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV for different initializations of QGP evolution (indicated in the legend). In each 4×2 panel, first row corresponds to R_{AA}, the second, third, fourth to v_2, v_3, v_4, respectively, while the left (right) column corresponds to 10-30% (30-50%) centrality class. D meson theoretical predictions are compared to CMS [231] (blue squares) and ALICE [232, 233] (red circles) data, while B meson predictions are compared to preliminary CMS [205] (blue squares) and preliminary ALICE [204] (red circles) data for non-prompt D meson from b decay. $\mu_M/\mu_E = 0.5$.

models seem to vastly underestimate the v_4 values, though the error bars for the available v_4 data are quite large. If this tendency is preserved in future high luminosity experiments (e.g., in LHC run 3), it will present a new "high-$p_\perp$ v_4 puzzle", whose solution will require modifications to the present initial state models and/or energy loss mechanisms. Additionally, better quality heavy flavor data are needed, especially D and B-meson data, as they present valuable constraint to the evolution of the medium.

7.3 Summary

We obtained four main conclusions in this work: *i*) We found that different methods to calculate higher harmonics at high-$p_\perp$ are compatible with each other within ≈ 5–10% accuracy, which is less than the current experimental uncertainties. *ii*) Event-by-event calculations are particularly important

for high-$p_\perp$ v_2 in mid-central collisions. *iii*) Predictions for high-$p_\perp$ observables, and especially for higher harmonics, are sensitive to the initial state of fluid-dynamical evolution, and can distinguish between different initial state models. *iv*) All initial state models lead to way smaller high-$p_\perp$ v_4 than experimentally observed, and this disparity deserves to be called a "v_4 puzzle". Overall, the higher harmonics provide an exciting opportunity to obtain further constraints to the QGP properties and its evolution in heavy-ion collisions by combining new theoretical developments (with the corresponding predictions) and upcoming higher luminosity experimental measurements.

Chapter 8

Conclusions

This thesis presents a comprehensive study of the quark-gluon plasma (QGP), utilizing the dynamical energy loss formalism to investigate high-$p_\perp$ parton-medium interactions, QGP properties, and experimental observables from heavy-ion collisions. Through detailed theoretical modeling, numerical simulations, and comparisons with experimental data, the research advances our understanding of strongly interacting matter under extreme conditions. Below, the major findings and setups of each chapter are summarized.

Chapters 1 and 2 provide the theoretical foundations for the study, outlining the principles of Quantum Chromodynamics (QCD) and the QCD phase diagram. These chapters discusses the critical transitions between confined and deconfined phases, highlighting high-$p_\perp$ experimental observables such as nuclear modification factors (R_{AA}) and anisotropic flow coefficients (v_2). The introduction of the dynamical energy loss formalism, which unifies collisional and radiative energy loss mechanisms, sets the stage for subsequent analyses.

Chapter 3 explores the path-length dependence of energy loss mechanisms of high-energy partons traversing the quark-gluon plasma. The objective is to address how different energy loss mechanisms, such as collisional and radiative, behave in terms of their sensitivity to the distance traversed in a QCD medium. Path-length dependence is a crucial signature to distinguish between different energy loss models and mechanisms and to understand the underlying physics of parton-medium interactions.

The chapter begins by identifying suitable observables for studying path-length dependence. The nuclear modification factor (R_{AA}) is utilized due to its sensitivity to jet-medium interactions while being relatively insensitive to the details of medium evolution. However, R_{AA} alone cannot directly extract path-length dependence. To overcome this, the study introduces $1 - R_{AA}$ ratio between two systems - path-length sensitive suppression ratio, R^L_{AB}, which enhances sensitivity to the path length and reduce centrality and $p_\perp$ dependence.

The systems analyzed include Pb+Pb collisions at LHC energies and smaller systems such as Xe+Xe, Kr+Kr, Ar+Ar and O+O, with collision energies set to minimize differences in medium temperature and density. This allows for the clean extraction of path-length dependencies by focusing on the system size as the primary variable.

The study finds that radiative energy loss exhibits a quadratic dependence on path length, while collisional energy loss is closer to linear dependence. The chapter identifies path-length sensitive suppression ratio as a reliable and robust observable for extracting path-length dependence, highlighting

its utility in precision QGP tomography.

Chapter 4 of the thesis introduces and elaborates on the DREENA-B framework, a computational model that incorporates $1 + 1D$ Bjorken expansion to study partonic energy loss in a dynamically evolving QCD medium. The chapter focuses on extending the applicability of the dynamical energy loss formalism by introducing a more realistic treatment of medium evolution while maintaining the rigor of high-$p_\perp$ parton-medium interaction modeling. The DREENA-B framework addresses limitations in earlier energy loss models by combining a sophisticated treatment of parton-medium interactions with a dynamically evolving QCD medium. Existing models either oversimplify medium dynamics or rely on static approximations, while DREENA-B uses ideal hydrodynamic Bjorken expansion to model temperature evolution, bridging the gap between the constant temperature DREENA-C framework and more complex full-evolution models.

The framework produces comprehensive predictions for nuclear modification factor (R_{AA}) and eliptic flow coefficient (v_2) for light and heavy flavor particles. Predictions align closely with experimental data from ALICE, ATLAS, and CMS for a range of centralities and multiple collision system. v_2 predictions resolve the "v_2 puzzle" where earlier models systematically underestimated data, while the optimized implementation allows for large-scale calculations, making the framework suitable for future QGP tomography studies.

The chapter concludes with the potential for dynamical energy loss formalism to be extended to more complex medium evolution models. DREENA-B provides a critical step towards a full hydrodynamic treatment. The agreement of predictions with experimental data strengthens the role of the dynamical energy loss formalism as a reliable tool for precision QGP studies.

Chapter 5 delves into the initial stages of heavy-ion collisions and their impact on the evolution of the quark-gluon plasma (QGP). The chapter emphasizes the role of initial dynamics, particularly the effects of the early-stage temperature profiles and their influence on high-$p_\perp$ observables.

The analysis relies on the dynamical energy loss formalism embedded within the $1 + 1D$ Bjorken medium expansion model. This approach allows precise control over the temperature profiles during the initial stages. Four different initial-stage scenarios are considered: *i) free-streaming profile* with non-interacting medium prior to thermalization, *ii) linear increase profile* where the temperature rises linearly until thermalization, *iii) constant profile* in which the medium maintains a constant temperature prior to thermalization and *iv) divergent profile* where the temperature evolves continuously, matching the thermalized profile. All scenarios converge to the same hydrodynamic evolution after thermalization, isolating the impact of early-stage dynamics.

The results show that R_{AA} is moderately sensitive to the initial stages of QGP evolution, with suppression increasing progressively from free-streaming to divergent cases. This sensitivity arises because high-$p_\perp$ partons interact with the medium as it thermalizes, and the energy loss reflects differences in the early-time temperature profiles. Contrary to expectations, v_2 demonstrates insensitivity to initial stages, irrespective of the particle type (charged hadrons, D-mesons and B-mesons). This contrasts with earlier studies that suggested significant sensitivity. The findings indicate that v_2 is more influenced by differences in the final stages of medium evolution rather than the early stages.

A commonly used approach involves fitting energy loss parameters to reproduce experimental R_{AA} data for different initial-stage scenarios. This chapter critiques such methods, arguing that energy loss parameters should reflect intrinsic medium properties, not depend on specific initial-stage assumptions. The chapter concludes that high-$p_\perp$ R_{AA} is a viable observable for probing early-stage dynamics, but v_2 may not be as effective in distinguishing between initial-stage scenarios. The findings underscore the importance of consistent energy loss modeling and precise control over temperature profiles to disentangle the effects of initial stages from those of later evolution.

Chapter 6 introduces the DREENA-A framework, a computational model designed for precision

tomography of the QGP. By incorporating arbitrary temperature profiles into the dynamical energy loss formalism, this framework extends beyond its predecessors, DREENA-C (constant temperature) and DREENA-B (Bjorken expansion), to account for the full $3D$ evolution of the QGP medium. The chapter systematically develops the framework, demonstrates its reliability, and evaluates its sensitivity to medium properties.

Numerical optimizations, including trajectory averaging and integration order adjustments, ensure computational efficiency. Equidistant sampling of jet trajectories reduces execution time by orders of magnitude while maintaining precision.

The framework uses temperature profiles derived from various hydrodynamic models that evolve different initial conditions, including Glauber, EKRT and TRENTo initial conditions. Energy loss was calculated for temperature profiles in $\sqrt{s_{NN}} = 5.02$ TeV $Pb + Pb$ and $\sqrt{s_{NN}} = 200$ GeV $Au + Au$ collisions with the goal of understanding their influence on R_{AA} and v_2.

Both R_{AA} and v_2 exhibit notable sensitivity to the initial temperature and its evolution. Glauber profiles show higher anisotropy throughout the QGP lifetime, influencing v_2, while EKRT profiles result in smaller R_{AA} due to higher initial temperatures.

By addressing the limitations of previous frameworks and integrating state-of-the-art energy loss mechanisms, DREENA-A establishes itself as a cornerstone in the study of QGP dynamics and properties. This chapter solidifies the framework's role as a critical instrument for future theoretical investigations in high-energy nuclear physics.

Chapter 7 examines the significance of higher harmonics in the study of the quark-gluon plasma (QGP) using high transverse momentum observables. The focus is on utilizing the anisotropic flow coefficients, particularly higher-order harmonics (v_3 and v_4) as tools for constraining QGP properties and probing its bulk evolution. The chapter also addresses methodological challenges and evaluates the consistency of various experimental and theoretical approaches.

Anisotropic flow coefficients (v_n) capture the asymmetry in particle momentum distributions relative to the reaction plane, offering clues about the QGP's initial state and its hydrodynamic evolution. Higher harmonics, such as triangular (v_3) and quadrangular flow (v_4), are particularly sensitive to event-by-event fluctuations in the initial state and transport properties.

Simulations demonstrate that fluctuations in the initial geometry significantly enhance eliptic flow, particularly in central collisions. The effect becomes smaller at higher centralities, where geometric anisotropy dominates. Using various initial condition models (e.g., Monte Carlo Glauber, TRENTo, IP-Glasma), the study evaluates their impact on high-$p_\perp$ v_n predictions. Models with stronger initial fluctuations yield higher harmonics, underscoring their importance for QGP tomography. v_4 predictions are systematically lower than experimental data, revealing a potential "v_4 puzzle". This discrepancy emphasizes the need for refined medium modeling and better constraints on initial conditions.

The results establish higher harmonics as a critical component of QGP tomography. By incorporating event-by-event fluctuations and comparing theoretical predictions with high-$p_\perp$ data from LHC experiments, this chapter advances the understanding of QGP properties. The findings also highlight the sensitivity of higher harmonics to temperature profiles, transport coefficients, and initial-state fluctuations, providing avenues for future research.

Overall, this thesis demonstrates that the developed DREENA framework is a powerful tool for exploring the properties of the QGP using both low-$p_\perp$ and high-$p_\perp$ theory and data. By integrating state-of-the-art dynamical energy loss models with realistic medium evolution, the DREENA framework bridges the complementary insights offered by low- and high-$p_\perp$ observables. Its ability to unify these datasets within a single theoretical framework marks a significant advancement, enabling a more comprehensive characterization of the QGP across a wide range of temperatures and scales.

These findings emphasize the importance of integrating diverse datasets and highlight the transformative potential of the DREENA framework for future studies in heavy-ion physics. As experimental facilities like the LHC and RHIC continue to produce increasingly precise datasets, the DREENA framework is well-equipped to extract deeper insights into the fundamental properties of the QGP. This thesis thus establishes a robust foundation for advancing our understanding of the QGP and sets the stage for further theoretical and computational developments to study this extraordinary state of matter.

Appendix: DREENA-A code

DREENA-A is publicly available on GitHub [191]. It is written in C++ and is organized in two main classes: *i) lTables* class that contains code used for calculation of radiated gluon and collisional rates; these are calcuated only once and can be reused for energy loss calculations on different hydrodynamical backgrounds; *ii) enrergyLoss* class that contains code used for high-$p_\perp$ parton energy loss calculation along a path that uses precalculated rates.

Content of the header file for *energyLoss* class - *energyLoss.hpp* is:

```
#ifndef HEADERFILE_ELOSSHEADER
#define HEADERFILE_ELOSSHEADER

#include "grids.hpp"
#include "linearinterpolation.hpp"

#include <string>
#include <vector>
#include <map>

class energyLoss {

public:
    energyLoss(int argc, const char *argv[]);
    ~energyLoss();
    void runEnergyLoss();

private:
    bool m_error; //flag that checks if previous calculation is done properly

    std::string m_collsys;      // collision system
    std::string m_sNN;          // collision energy
    std::string m_pName;        // particle name
    std::string m_centrality;   // centrality class
    double m_xB;                // xB value
    size_t m_xGridN;            // initial position grid points and angle number
    long m_yGridN;              // initial position grid points and angle number
    size_t m_phiGridN;          // initial position grid points and angle number
    double m_TIMESTEP, m_TCRIT; // time step and critical temperature

    double m_nf;           // effective number of flavours
    const double m_lambda = 0.2; // QCD scale
    double m_mgC, m_MC;          // constant particle and gluon masses used for dA
    integrals
    double m_TCollConst;         // constant temperature used for Gauss filter
    integration
    double m_tau0;               // thermalization time

    gridPoints m_Grids; // grid points


    int loadInputsFromFile(const std::string &filePath, std::map<std::string, std::string
    > &inputParamsFile);
```

```
41
42      double productLog(double x) const;
43
44      interpolationF<double> m_dsdpti2; // initial pT distribution interpolated function
45      int loaddsdpti2(const std::string &pname, interpolationF<double> &dsdpti2int) const;
46
47      interpolationF<double> m_LNorm, m_Ldndx, m_LColl; // interpolated L tables
48      int loadLdndx();
49      int loadLNorm();
50      int loadLColl();
51
52      interpolationF<double> m_tempEvol; // temperature evolution interpolated function
53      int loadTempEvol();
54
55      interpolationF<double> m_binCollDensity;                    // binary collision density
         interpolated function
56      int loadBinCollDensity(interpolationF<double> &binCollDensity);
57      int loadPhiPoints(std::vector<double> &phipoints);
58      int loadBinCollPoints(std::vector<std::vector<double>> &bcpoints);
59      std::vector<double> m_xGridPts, m_yGridPts, m_phiGridPts; // vectors that store
        initial position points and angles
60      int generateInitPosPoints();
61
62
63      double haltonSequence(int index, int base) const;
64
65      size_t m_FdAMaxPoints2, m_FdAMaxPoints3, m_FdAMaxPoints4, m_FdAMaxPoints5; //number
        of points for FdA integration
66      std::vector<double> m_FdAHS2, m_FdAHS3, m_FdAHS4, m_FdAHS5;                 //vectors
        that store Halton sequences for FdA integrals
67      void FdAHaltonSeqInit(size_t FdAMaxPts);
68      double dAp410(double ph, const interpolationF<double> &norm) const;
69      double FdA411(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const;
70      double FdA412(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const;
71      double FdA413(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const;
72      double FdA414(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const;
73      double FdA415(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const;
74      double FdA(double ph, double dp, const interpolationF<double> &currnorm, const
        interpolationF<double> &currdndx) const;
75
76      size_t m_dAMaxPoints1, m_dAMaxPoints2, m_dAMaxPoints3, m_dAMaxPoints4, m_dAMaxPoints5
        , m_dAMaxPoints6, m_dAMaxPoints7; //number of points for dA integration
77      std::vector<double> m_dAHS1, m_dAHS2, m_dAHS3, m_dAHS4, m_dAHS5, m_dAHS6, m_dAHS7;
                          //vectors that store Halton sequences for dA integrals
78      void dAHaltonSeqInit(size_t dAMaxPts);
79      double dA410(double ph, const interpolationF<double> &norm) const;
80      double dA411(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
81      double dA412(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
82      double dA413(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
83      double dA414(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
84      double dA415(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
85      double dA416(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
86      double dA417(double ph, const interpolationF<double> &norm, const interpolationF<
        double> &dndx) const;
87      double dA41(double ph, interpolationF<double> &currnorm, interpolationF<double> &
        currdndx) const;
88
89      void radCollEnergyLoss(double x, double y, double phi, std::vector<double> &radRAA1,
        std::vector<std::vector<double>> &radRAA2, std::vector<double> &collEL, double &
        pathLength, double &temperature) const;
90      void radCollEnergyLoss(double x, double y, double phi, std::vector<double> &radRAA,
```

```
    std::vector<double> &collEL, double &pathLenght, double &temperature) const;
91
92      void generateGaussTab(std::vector<double> &qGTab, std::vector<double> &fGTab) const;
93      void gaussFilterIntegrate(const std::vector<double> &radiativeRAA1, const std::vector
        <std::vector<double>> &radiativeRAA2, const std::vector<double> &collisionalEL, std::
        vector<double> &singRAA1, std::vector<std::vector<double>> &singRAA2) const;
94      void gaussFilterIntegrate(const interpolationF<double> &dsdpti2lquark, const std::
        vector<double> &radiativeRAA1, const std::vector<std::vector<double>> &radiativeRAA2,
         const std::vector<double> &collisionalEL, std::vector<double> &singRAA1, std::vector
        <std::vector<double>> &singRAA2) const;
95      void gaussFilterIntegrate(const std::vector<double> &radiativeRAA, const std::vector<
        double> &collisionalEL, std::vector<double> &singRAA) const;
96
97      void calculateAvgPathlenTemps(const std::vector<double> &pathLenghDist, const std::
        vector<double> &temperatureDist, std::vector<double> &avgPathLength, std::vector<
        double> &avgTemp) const;
98
99      int exportResults(const std::string &pName, const std::vector<std::vector<double>> &
        RAADist, const std::vector<double> avgPathLength, const std::vector<double> avgTemp);
100
101     void runELossHeavyFlavour();
102     void runELossLightQuarks();
103     void runELossLightFlavour();
104 };
105
106 #endif
```

energyLoss class methods are separated into two files - one containing integrals in the Poisson expansion of the radiative energy loss, and the other one containing everything else. Content of the main source file for *energyLoss* class, *energyLoss.cpp* follows:

```
1  #include "energyloss.hpp"
2  #include "grids.hpp"
3  #include "linearinterpolation.hpp"
4  #include "polyintegration.hpp"
5
6  #include <iostream>
7  #include <string>
8  #include <sstream>
9  #include <vector>
10 #include <algorithm>
11 #include <random>
12 #include <map>
13 #include <tuple>
14 #include <fstream>
15 #include <cmath>
16 #include <limits>
17 #include <iomanip>
18
19 energyLoss::energyLoss(int argc, const char *argv[])
20 {
21   m_error = false;
22
23   std::vector<std::string> inputs; for (int i=2; i<argc; i++) inputs.push_back(argv[i]);
24
25   if ((inputs.size() == 1) && (inputs[0] == "-h")) {
26     std::cout << "default values: --collsys=PbPb --sNN=5020GeV --pName=Charm --centrality
       =30-40% --xB=0.6 --xGridN=50 --yGridN=50 --phiGridN=25 --TIMESTEP=0.1 --TCRIT=0.155"
       << std::endl;
27     m_error = true;
28   }
29
30   std::map<std::string, std::string> inputParams;
31   for (const auto &in : inputs) {
32       std::string key = in.substr(0, in.find("="));
33       std::string::size_type n = 0; while ((n = key.find("-", n)) != std::string::npos) {
       key.replace(n, 1, ""); n += 0;} //replacing all '-'
34     std::string val = in.substr(in.find("=")+1, in.length());
35     inputParams[key] = val;
36   }
37
```

```
38  std::vector<std::string> arguments = {"collsys", "sNN", "pName", "centrality", "xB", "
      xGridN", "yGridN", "phiGridN", "TIMESTEP", "TCRIT", "config", "h"};
39  for (const auto &inputParam : inputParams) {
40    if(std::find(arguments.begin(), arguments.end(), inputParam.first) == arguments.end()
      ) {
41      std::cerr << "Error: provide argument flag: " << inputParam.first << " is not an
      option." << std::endl;
42      std::cerr << "Valid parameters and default values are: ";
43      std::cerr << "--collsys=PbPb --sNN=5020GeV --pName=Charm --centrality=30-40% --xB
      =0.6 --xGridN=50 --yGridN=50 --phiGridN=25 --TIMESTEP=0.1 --TCRIT=0.155" << std::endl
      ;
44      std::cerr << "For congiguration file use: --config=[pathToConfFile]" << std::endl;
45      m_error = true;
46    }
47  }
48
49  //checking if configuration file is provided:
50  std::map<std::string, std::string> inputParamsFile;
51  if (inputParams.count("config") > 0) {
52    if (loadInputsFromFile(inputParams.at("config"), inputParamsFile) != 1) {
53      m_error = true;
54    }
55  }
56  std::vector<std::string> argumentsFile = {"collsys", "sNN", "pName", "centrality", "xB"
      , "xGridN", "yGridN", "phiGridN", "TIMESTEP", "TCRIT"};
57  for (const auto &inputParam : inputParamsFile) {
58    if(std::find(argumentsFile.begin(), argumentsFile.end(), inputParam.first) ==
      argumentsFile.end()) {
59      std::cerr << "Error: in configration file provided argument: '" << inputParam.first
       << "' is not an option." << std::endl;
60      std::cerr << "Valid parameters and default values are: \n";
61      std::cerr << "collsys = PbPb\nsNN = 5020GeV\npName = Charm\ncentrality = 30-40%\nxB
       = 0.6\nxGridN = 50\nyGridN = 50\nphiGridN = 25\nTIMESTEP = 0.1\nTCRIT = 0.155\n" <<
      std::endl;
62      m_error = true;
63    }
64  }
65
66  //setting parameter values based on config file values and overwriting with command
      line values:
67  //
68  m_collsys = "PbPb"; if (inputParamsFile.count("collsys") > 0) m_collsys =
      inputParamsFile.at("collsys");
69            if (    inputParams.count("collsys") > 0) m_collsys =     inputParams.at("
      collsys");
70
71  m_sNN = "5020GeV"; if (inputParamsFile.count("sNN") > 0) m_sNN = inputParamsFile.at("
      sNN");
72             if (    inputParams.count("sNN") > 0) m_sNN =     inputParams.at("sNN");
73
74  m_pName = "Charm"; if (inputParamsFile.count("pName") > 0) m_pName = inputParamsFile.at
      ("pName");
75             if (    inputParams.count("pName") > 0) m_pName =     inputParams.at("pName"
      );
76
77  m_centrality = "30-40%"; if (inputParamsFile.count("centrality") > 0) m_centrality =
      inputParamsFile.at("centrality");
78                if (    inputParams.count("centrality") > 0) m_centrality =
      inputParams.at("centrality");
79
80  m_xB = 0.6; if (inputParamsFile.count("xB") > 0) m_xB = stod(inputParamsFile.at("xB"));
81        if (    inputParams.count("xB") > 0) m_xB = stod(    inputParams.at("xB"));
82
83    m_xGridN = 25; if (inputParamsFile.count("xGridN") > 0) m_xGridN = stoi(
      inputParamsFile.at("xGridN"));
84           if (    inputParams.count("xGridN") > 0) m_xGridN = stoi(    inputParams.at("
      xGridN"));
85
86    m_yGridN = 25; if (inputParamsFile.count("yGridN") > 0) m_yGridN = stoi(
      inputParamsFile.at("yGridN"));
87           if (    inputParams.count("yGridN") > 0) m_yGridN = stoi(    inputParams.at("
      yGridN"));
```

```
88
89    m_phiGridN = 25; if (inputParamsFile.count("phiGridN") > 0) m_phiGridN = stoi(
        inputParamsFile.at("phiGridN"));
90              if (    inputParams.count("phiGridN") > 0) m_phiGridN = stoi(    inputParams.
        at("phiGridN"));
91
92    m_TIMESTEP = 0.1; if (inputParamsFile.count("TIMESTEP") > 0) m_TIMESTEP = stod(
        inputParamsFile.at("TIMESTEP"));
93              if (    inputParams.count("TIMESTEP") > 0) m_TIMESTEP = stod(    inputParams.
        at("TIMESTEP"));
94
95    m_TCRIT = 0.155; if (inputParamsFile.count("TCRIT") > 0) m_TCRIT = stod(inputParamsFile
        .at("TCRIT"));
96              if (    inputParams.count("TCRIT") > 0) m_TCRIT = stod(    inputParams.at("
        TCRIT"));
97
98    //checking if provided value of sNN is an option:
99    if ((m_sNN != "5440GeV") && (m_sNN != "5020GeV") && (m_sNN != "2760GeV") && (m_sNN != "
        200GeV")) {
100     std::cerr << "Error: provided sNN parameter not an option, please try 5440GeV, 5020
        GeV, 2760GeV or 200GeV. Aborting..." << std::endl;
101     m_error = true;
102   }
103
104   m_nf = m_sNN == "200GeV" ? 2.5 : 3.0;
105   double T = 3.0 / 2.0*m_TCRIT;
106   double mu = 0.197*std::sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
        m_lambda/productLog((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/m_lambda
        ));
107   m_mgC = mu / std::sqrt(2.0);
108   if (m_pName == "Bottom") m_MC = 4.75;
109   else if (m_pName == "Charm") m_MC = 1.2;
110   else if (m_pName == "Gluon") m_MC = mu/std::sqrt(2.0);
111   else m_MC = mu/sqrt(6.0);
112   m_TCollConst = T;
113 }
114
115 int energyLoss::loadInputsFromFile(const std::string &filePath, std::map<std::string, std
        ::string> &inputParamsFile)
116 {
117   std::ifstream file_in(filePath);
118   if (!file_in.is_open()) {
119     std::cerr << "Error: unable to open configuration file. Aborting..." << std::endl;
120     return -1;
121   }
122   std::string line, key, sep, val;
123   while (std::getline(file_in, line))
124   {
125     std::stringstream ss(line);
126     ss >> key; ss >> sep; ss >> val;
127     inputParamsFile[key] = val;
128   }
129   file_in.close();
130   return 1;
131 }
132
133 energyLoss::~energyLoss() {}
134
135 void energyLoss::runEnergyLoss()
136 {
137     if (m_error) return;
138
139   m_Grids.setGridPoints(m_sNN, m_pName, m_TCRIT);
140
141   if (loadLdndx() != 1) return;
142   if (loadLNorm() != 1) return;
143   if (loadLColl() != 1) return;
144
145     if (generateInitPosPoints() != 1) return;
146     if (loadTempEvol()          != 1) return;
147
148   if ((m_pName == "Bottom") || (m_pName == "Charm")) {
```

```
149     runELossHeavyFlavour();
150   }
151   else if (m_pName == "LQuarks") {
152     runELossLightQuarks();
153   }
154   else {
155     runELossLightFlavour();
156   }
157 }
158
159 double energyLoss::productLog(double x) const
160 {
161   if (x == 0.0) {
162     return 0.0;
163   }
164
165   double w0, w1;
166   if (x > 0.0) {
167     w0 = std::log(1.2 * x / std::log(2.4 * x / std::log1p(2.4 * x)));
168   }
169   else {
170     double v = 1.4142135623730950488 * std::sqrt(1.0 + 2.7182818284590452354 * x);
171     double N2 = 10.242640687119285146 + 1.9797586132081854940 * v;
172     double N1 = 0.29289321881345247560 * (1.4142135623730950488 + N2);
173     w0 = -1 + v * (N2 + v) / (N2 + v + N1 * v);
174   }
175
176   while (true) {
177     double e = std::exp(w0);
178     double f = w0 * e - x;
179     w1 = w0 - f / ((e * (w0 + 1.0) - (w0 + 2.0) * f / (w0 + w0 + 2.0)));
180     if (std::abs(w0 / w1 - 1.0) < 1.4901161193847656e-8) {
181       break;
182     }
183     w0 = w1;
184   }
185   return w1;
186 }
187
188
189 int energyLoss::loaddsdpti2(const std::string &pname, interpolationF<double> &dsdpti2int)
      const
190 {
191   const std::string path_in = "./ptDists/ptDist" + m_sNN + "/ptDist_" + m_sNN + "_" +
      pname + ".dat";
192
193   std::ifstream file_in(path_in);
194   if (!file_in.is_open()) {
195     std::cerr << "Error: unable to open initial pT distribution file. Aborting..." << std
      ::endl;
196     return -1;
197   }
198
199   std::vector<double> pTdistX, pTdistF;
200
201   std::string line; double buffer;
202
203   while (std::getline(file_in, line))
204   {
205         if (line.at(0) == '#')
206             continue;
207
208     std::stringstream ss(line);
209     ss >> buffer; pTdistX.push_back(buffer);
210     ss >> buffer; pTdistF.push_back(buffer);
211   }
212
213   dsdpti2int.setData(pTdistX, pTdistF);
214
215   file_in.close();
216
217   return 1;
```

```
}

int energyLoss::loadLdndx()
{
  std::string partName;
  if (m_pName == "Bottom") partName = "Bottom";
  else if (m_pName == "Charm") partName = "Charm";
  else if (m_pName == "Gluon") partName = "Gluon";
  else partName = "LQuarks";

  std::stringstream xBss; xBss << std::fixed << std::setprecision(1) << m_xB;
  std::stringstream nfss; nfss << std::fixed << std::setprecision(1) << m_nf;

  const std::string path_in = "./ltables/ldndx_nf=" + nfss.str() + "_" + partName + "_xB=
    " + xBss.str() + ".dat";

  std::ifstream file_in(path_in);
  if (!file_in.is_open()) {
    std::cerr << "Error: unable to open Ldndx table file. Aborting..." << std::endl;
    return -1;
  }

  std::vector<double> Ldndx_tau, Ldndx_p, Ldndx_T, Ldndx_x, Ldndx_f;

  std::string line; double buffer;

  while (std::getline(file_in, line))
  {
        if (line.at(0) == '#')
            continue;

    std::stringstream ss(line);
    ss >> buffer; Ldndx_tau.push_back(buffer);
    ss >> buffer; Ldndx_p.push_back(buffer);
    ss >> buffer; Ldndx_T.push_back(buffer);
    ss >> buffer; Ldndx_x.push_back(buffer);
    ss >> buffer; Ldndx_f.push_back(buffer);
  }

  file_in.close();

  m_Ldndx.setData(Ldndx_tau, Ldndx_p, Ldndx_T, Ldndx_x, Ldndx_f);

  std::vector<std::vector<double>> domain = m_Ldndx.domain();
  if (m_Grids.tauPts(0)  < domain[0][0]) {std::cerr << "Error: tau grid point(s) out of
    lower bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.tauPts(-1) > domain[0][1]) {std::cerr << "Error: tau grid point(s) out of
    upper bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.pPts(0)    < domain[1][0]) {std::cerr << "Error:   p grid point(s) out of
    lower bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.pPts(-1)   > domain[1][1]) {std::cerr << "Error:   p grid point(s) out of
    upper bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.TPts(0)    < domain[2][0]) {std::cerr << "Error:   T grid point(s) out of
    lower bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.TPts(-1)   > domain[2][1]) {std::cerr << "Error:   T grid point(s) out of
    upper bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.xPts(0)    < domain[3][0]) {std::cerr << "Error:   x grid point(s) out of
    lower bound of Ldndx domain. Aborting..." << std::endl; return -1;}
  if (m_Grids.xPts(-1)   > domain[3][1]) {std::cerr << "Error:   x grid point(s) out of
    upper bound of Ldndx domain. Aborting..." << std::endl; return -1;}

  return 1;
}

int energyLoss::loadLNorm()
{
  std::string partName;
  if (m_pName == "Bottom") partName = "Bottom";
  else if (m_pName == "Charm") partName = "Charm";
  else if (m_pName == "Gluon") partName = "Gluon";
  else partName = "LQuarks";

```

```
281   std::stringstream xBss; xBss << std::fixed << std::setprecision(1) << m_xB;
282   std::stringstream nfss; nfss << std::fixed << std::setprecision(1) << m_nf;
283
284   const std::string path_in = "./ltables/lnorm_nf=" + nfss.str() + "_" + partName + "_xB=
        " + xBss.str() + ".dat";
285
286   std::ifstream file_in(path_in);
287   if (!file_in.is_open()) {
288     std::cerr << "Error: unable to open LNorm table file. Aborting..." << std::endl;
289     return -1;
290   }
291
292   std::vector<double> LNorm_tau, LNorm_p, LNorm_T, LNorm_f; //defining vectors that store
        LNorm table values
293
294   std::string line; double buffer;
295
296   while (std::getline(file_in, line))
297   {
298         if (line.at(0) == '#')
299             continue;
300
301     std::stringstream ss(line);
302     ss >> buffer; LNorm_tau.push_back(buffer);
303     ss >> buffer; LNorm_p.push_back(buffer);
304     ss >> buffer; LNorm_T.push_back(buffer);
305     ss >> buffer; LNorm_f.push_back(buffer);
306   }
307
308   file_in.close();
309
310   m_LNorm.setData(LNorm_tau, LNorm_p, LNorm_T, LNorm_f);
311
312   std::vector<std::vector<double>> domain = m_LNorm.domain();
313   if (m_Grids.tauPts(0)  < domain[0][0]) {std::cerr << "Error: tau grid point(s) out of
        lower bound of LNorm domain. Aborting..." << std::endl; return -1;}
314   if (m_Grids.tauPts(-1) > domain[0][1]) {std::cerr << "Error: tau grid point(s) out of
        upeer bound of LNorm domain. Aborting..." << std::endl; return -1;}
315   if (m_Grids.pPts(0)    < domain[1][0]) {std::cerr << "Error:   p grid point(s) out of
        lower bound of LNorm domain. Aborting..." << std::endl; return -1;}
316   if (m_Grids.pPts(-1)   > domain[1][1]) {std::cerr << "Error:   p grid point(s) out of
        upeer bound of LNorm domain. Aborting..." << std::endl; return -1;}
317   if (m_Grids.TPts(0)    < domain[2][0]) {std::cerr << "Error:   T grid point(s) out of
        lower bound of LNorm domain. Aborting..." << std::endl; return -1;}
318   if (m_Grids.TPts(-1)   > domain[2][1]) {std::cerr << "Error:   T grid point(s) out of
        upeer bound of LNorm domain. Aborting..." << std::endl; return -1;}
319
320   return 1;
321 }
322
323 int energyLoss::loadLColl()
324 {
325   std::string partName;
326   if (m_pName == "Bottom") partName = "Bottom";
327   else if (m_pName == "Charm") partName = "Charm";
328   else if (m_pName == "Gluon") partName = "Gluon";
329   else partName = "LQuarks";
330
331   std::stringstream nfss; nfss << std::fixed << std::setprecision(1) << m_nf;
332
333   const std::string path_in = "./ltables/lcoll_nf=" + nfss.str() + "_" + partName + ".dat
        ";
334
335   std::ifstream file_in(path_in);
336   if (!file_in.is_open()) {
337     std::cerr << "Error: unable to open LColl table file. Aborting..." << std::endl;
338     return -1;
339   }
340
341   std::vector<double> LColl_p, LColl_T, LColl_f;
342
343   std::string line; double buffer;
```

```
344
345   while (std::getline(file_in, line))
346   {
347         if (line.at(0) == '#')
348             continue;
349
350     std::stringstream ss(line);
351     ss >> buffer; LColl_p.push_back(buffer);
352     ss >> buffer; LColl_T.push_back(buffer);
353     ss >> buffer; LColl_f.push_back(buffer);
354   }
355
356   file_in.close();
357
358   m_LColl.setData(LColl_p, LColl_T, LColl_f);
359
360   std::vector<std::vector<double>> domain = m_LColl.domain();
361   if (m_Grids.pCollPts(0)  < domain[0][0]) {std::cerr << "Error: p grid point(s) out of
        lower bound of LColl domain. Aborting..." << std::endl; return -1;}
362   if (m_Grids.pCollPts(-1) > domain[0][1]) {std::cerr << "Error: p grid point(s) out of
        upper bound of LColl domain. Aborting..." << std::endl; return -1;}
363   if (m_Grids.TCollPts(0)  < domain[1][0]) {std::cerr << "Error: T grid point(s) out of
        lower bound of LColl domain. Aborting..." << std::endl; return -1;}
364   if (m_Grids.TCollPts(-1) > domain[1][1]) {std::cerr << "Error: T grid point(s) out of
        upper bound of LColl domain. Aborting..." << std::endl; return -1;}
365
366   return 1;
367 }
368
369
370 int energyLoss::loadBinCollDensity(interpolationF<double> &binCollDensity)
371 {
372   std::string path_in = "binarycolldensities/binarycolldensity_cent=" + m_centrality + ".
        dat";
373   std::ifstream file_in(path_in, std::ios_base::in);
374   if (!file_in.is_open()) {
375     std::cerr << "Error: unable to open binary collision density file." << std::endl;
376     return -1;
377   }
378
379   std::string line; double buffer;
380
381   std::vector<double> bcdX, bcdY, bcdData;
382
383   while (std::getline(file_in, line))
384   {
385     if (line.at(0) == '#')
386       continue;
387
388     std::stringstream ss(line);
389     ss >> buffer;    bcdX.push_back(buffer);
390     ss >> buffer;    bcdY.push_back(buffer);
391     ss >> buffer; bcdData.push_back(buffer);
392   }
393
394   file_in.close();
395
396     double bcdXMin = *std::min_element(bcdX.begin(), bcdX.end());
397     double bcdYMin = *std::min_element(bcdY.begin(), bcdY.end());
398
399   if ((bcdXMin >= 0.0) && (bcdYMin >= 0.0)) {// if binary collision density is defined
        only in the first quadrant:
400
401     //creating full x grid:
402     std::vector<double> bcdXGrid(bcdX.begin(), bcdX.end());
403         size_t sizeX = bcdXGrid.size();
404         bcdXGrid.reserve(sizeX * 2);
405         for (size_t i=0; i<sizeX; ++i)
406             bcdXGrid.push_back(-1.0*bcdXGrid[i]);
407     sort(bcdXGrid.begin(), bcdXGrid.end());
408     bcdXGrid.erase(unique(bcdXGrid.begin(), bcdXGrid.end()), bcdXGrid.end());
409
```

```
410     //creating full y grid:
411     std::vector<double> bcdYGrid(bcdY.begin(), bcdY.end());
412         size_t sizeY = bcdYGrid.size();
413         bcdYGrid.reserve(sizeY * 2);
414         for (size_t i=0; i<sizeY; ++i)
415             bcdYGrid.push_back(-1.0*bcdYGrid[i]);
416     sort(bcdYGrid.begin(), bcdYGrid.end());
417     bcdYGrid.erase(unique(bcdYGrid.begin(), bcdYGrid.end()), bcdYGrid.end());
418 
419         // creating interpolated binary collision density defined in first quadrant:
420     interpolationF<double> binCollDensityFirstQuadrant(bcdX, bcdY, bcdData);
421 
422     //creating full binary collision density table:
423     std::vector<double> bcdXFull, bcdYFull, bcdDataFull;
424     for (const auto &x : bcdXGrid) {
425       for (const auto &y : bcdYGrid) {
426         bcdXFull.push_back(x);
427         bcdYFull.push_back(y);
428         bcdDataFull.push_back(binCollDensityFirstQuadrant.interpolation(std::abs(x), std
    ::abs(y)));
429       }
430         }
431 
432     binCollDensity.setData(bcdXFull, bcdYFull, bcdDataFull);
433   }
434   else { // if not, creating interpolated function with values from file:
435 
436     binCollDensity.setData(bcdX, bcdY, bcdData);
437   }
438 
439   return 1;
440 }
441 
442 int energyLoss::loadPhiPoints(std::vector<double> &phiPoints)
443 {
444   std::string path_in = "./phiGaussPts/phiptsgauss" + std::to_string(m_phiGridN) + ".dat"
      ;
445   std::ifstream file_in(path_in, std::ios_base::in);
446   if (!file_in.is_open()) {
447     std::cerr << "Error: unable to open phi points file. Aborting..." << std::endl;
448     return -1;
449   }
450 
451   phiPoints.resize(0);
452 
453   std::string line; double buffer;
454 
455   while (std::getline(file_in, line)) {
456     if (line.at(0) == '#')
457             continue;
458 
459     std::stringstream ss(line);
460     ss >> buffer; phiPoints.push_back(buffer);
461   }
462 
463   file_in.close();
464 
465   return 1;
466 }
467 
468 int energyLoss::loadBinCollPoints(std::vector<std::vector<double>> &bcPoints)
469 {
470   std::string path_in = "binarycollpoints/binarycollpoints_cent=" + m_centrality + ".dat"
      ;
471   std::ifstream file_in(path_in, std::ios_base::in);
472   if (!file_in.is_open()) {
473     std::cerr << "Error: unable to open binary collision points file." << std::endl;
474     return -1;
475   }
476 
477   bcPoints.resize(0);
478 
```

```
479   std::string line; double bufferX, bufferY;
480
481   while (std::getline(file_in, line)) {
482     std::stringstream ss(line);
483     ss >> bufferX;
484     ss >> bufferY;
485         bcPoints.push_back({bufferX, bufferY});
486   }
487
488   file_in.close();
489
490   return 1;
491 }
492
493 int energyLoss::generateInitPosPoints()
494 {
495   m_xGridPts.resize(0); m_yGridPts.resize(0); m_phiGridPts.resize(0);
496
497   if (m_yGridN == -2) {
498     //if yGridN is set to -2, MonteCarlo method is used to generate initial position
      points and angles
499     //number of x-y initial position points is equal to m_xGridN
500
501     interpolationF<double> binCollDensity; if (loadBinCollDensity(binCollDensity) != 1)
      return -1;
502
503     std::vector<std::vector<double>> bcDensDomain = binCollDensity.domain();
504     std::vector<double> bcDensCoDomain = binCollDensity.codomain();
505
506     //generating x and y points:
507     std::random_device rdX; std::mt19937 mtX(rdX());
508         std::uniform_real_distribution<double> distX(bcDensDomain[0][0], std::nextafter(
      bcDensDomain[0][1], std::numeric_limits<double>::max()));
509         std::random_device rdY; std::mt19937 mtY(rdY());
510         std::uniform_real_distribution<double> distY(bcDensDomain[1][0], std::nextafter(
      bcDensDomain[1][1], std::numeric_limits<double>::max()));
511         std::random_device rdZ; std::mt19937 mtZ(rdZ());
512         std::uniform_real_distribution<double> distZ(bcDensCoDomain[0],  std::nextafter(
      bcDensCoDomain[1],  std::numeric_limits<double>::max()));
513
514     double x, y, z;
515
516     for (size_t iXY=0; iXY<m_xGridN; iXY++) {
517       do {
518         x = distX(mtX);
519         y = distY(mtY);
520         z = distZ(mtZ);
521       }
522       while (z > binCollDensity.interpolation(x, y));
523
524       m_xGridPts.push_back(x); m_yGridPts.push_back(y);
525     }
526
527     std::random_device rdPhi; std::mt19937 mtPhi(rdPhi());
528         std::uniform_real_distribution<double> distPhi(0.0, std::nextafter(2.0*M_PI, std
      ::numeric_limits<double>::max()));
529
530     double phi;
531
532         // artificially adding 0 and 2Pi to the list:
533     phi = 0.0;    m_phiGridPts.push_back(phi); std::sort(m_phiGridPts.begin(),
      m_phiGridPts.end());
534     phi = 2.0*M_PI; m_phiGridPts.push_back(phi); std::sort(m_phiGridPts.begin(),
      m_phiGridPts.end());
535
536     //generating other points:
537     for (size_t iPhi=2; iPhi<m_phiGridN; iPhi++) {
538       do {
539                 phi = distPhi(mtPhi);
540             }
541             while (std::binary_search(m_phiGridPts.begin(), m_phiGridPts.end(), phi));
542       m_phiGridPts.push_back(phi); std::sort(m_phiGridPts.begin(), m_phiGridPts.end());
```

```
543     }
544
545     //generating binary collision density with function values 1:
546     std::vector<double> bcdX, bcdY, bcdData;
547     for (size_t iX=0; iX<=10; iX++){
548       for (size_t iY=0; iY<=10; iY++){
549           bcdX.push_back(bcDensDomain[0][0] + (bcDensDomain[0][1]-bcDensDomain[0][0])*
      static_cast<double>(iX)/10.0);
550           bcdY.push_back(bcDensDomain[1][0] + (bcDensDomain[1][1]-bcDensDomain[1][0])*
      static_cast<double>(iY)/10.0);
551         bcdData.push_back(1.0);
552       }
553     }
554     m_binCollDensity.setData(bcdX, bcdY, bcdData);
555
556   }
557   else if (m_yGridN == -1) {
558     //if yGridN is set to -1, MonteCarlo method is used to generate initial position
      points, while angles are on equidistant grid
559     //number of x-y initial position points is equal to xGridN
560
561     interpolationF<double> binCollDensity; if (loadBinCollDensity(binCollDensity) != 1)
      return -1;
562
563     std::vector<std::vector<double>> bcDensDomain = binCollDensity.domain();
564     std::vector<double> bcDensCoDomain = binCollDensity.codomain();
565
566     //generating x and y points:
567     std::random_device rdX; std::mt19937 mtX(rdX());
568         std::uniform_real_distribution<double> distX(bcDensDomain[0][0], std::nextafter(
      bcDensDomain[0][1], std::numeric_limits<double>::max()));
569         std::random_device rdY; std::mt19937 mtY(rdY());
570         std::uniform_real_distribution<double> distY(bcDensDomain[1][0], std::nextafter(
      bcDensDomain[1][1], std::numeric_limits<double>::max()));
571         std::random_device rdZ; std::mt19937 mtZ(rdZ());
572         std::uniform_real_distribution<double> distZ(bcDensCoDomain[0],  std::nextafter(
      bcDensCoDomain[1],  std::numeric_limits<double>::max()));
573
574     double x, y, z;
575
576     for (size_t iXY=0; iXY<m_xGridN; iXY++) {
577       do {
578         x = distX(mtX);
579         y = distY(mtY);
580         z = distZ(mtZ);
581       }
582       while (z > binCollDensity.interpolation(x, y));
583
584       m_xGridPts.push_back(x); m_yGridPts.push_back(y);
585     }
586
587     if (loadPhiPoints(m_phiGridPts) != 1) return -3;
588
589     //generating binary collision density with function values 1:
590     std::vector<double> bcdX, bcdY, bcdData;
591     for (size_t iX=0; iX<=10; iX++){
592       for (size_t iY=0; iY<=10; iY++){
593           bcdX.push_back(bcDensDomain[0][0] + (bcDensDomain[0][1]-bcDensDomain[0][0])*
      static_cast<double>(iX)/10.0);
594           bcdY.push_back(bcDensDomain[1][0] + (bcDensDomain[1][1]-bcDensDomain[1][0])*
      static_cast<double>(iY)/10.0);
595         bcdData.push_back(1.0);
596       }
597     }
598     m_binCollDensity.setData(bcdX, bcdY, bcdData);
599
600   }
601   else if (m_yGridN == 0) {
602     //if yGridN is set to 0, initial position points are randomly selected from a list,
      while angles are on equidistant grid
603     //number of x-y initial position points is equal to xGridN
604
```

```
605     std::vector<std::vector<double>> bcPoints; if (loadBinCollPoints(bcPoints) != 1)
        return -4;
606
607     if ((m_xGridN == bcPoints.size()) || (m_xGridN == 0)) {// take all points if xGridN
        is equal to total number of points or 0
608       for (size_t iXY=0; iXY<bcPoints.size(); iXY++) {
609         m_xGridPts.push_back(bcPoints[iXY][0]);
610                 m_yGridPts.push_back(bcPoints[iXY][1]);
611       }
612     }
613         else {// randomly select from imported points
614             std::random_device rd; auto rng = std::default_random_engine{rd()};
615           std::shuffle(bcPoints.begin(), bcPoints.end(), rng);
616             for (size_t iXY=0; iXY<m_xGridN; iXY++) {
617         m_xGridPts.push_back(bcPoints[iXY][0]);
618                 m_yGridPts.push_back(bcPoints[iXY][1]);
619       }
620         }
621
622     if (loadPhiPoints(m_phiGridPts) != 1) return -5;
623
624         //generating binary collision density with function values 1 in domain [-20, 20]:
625     std::vector<double> bcdX, bcdY, bcdData;
626     for (size_t iX=0; iX<=10; iX++) {
627       for (size_t iY=0; iY<=10; iY++) {
628           bcdX.push_back(-20.0 + 40.0*static_cast<double>(iX)/10.0);
629           bcdY.push_back(-20.0 + 40.0*static_cast<double>(iY)/10.0);
630         bcdData.push_back(1.0);
631       }
632     }
633
634     m_binCollDensity.setData(bcdX, bcdY, bcdData);
635
636   }
637   else {
638     //if yGridN is larger than 0, initial position points and angles are generated on
      equidistant grid
639     //number of x-y initial position points is equal to (xGridN+1)*(yGridN+1)
640
641     if (loadBinCollDensity(m_binCollDensity) != 1) return -6; //loading binary collision
      density
642
643     std::vector<std::vector<double>> bcDensDomain = m_binCollDensity.domain();
644
645     double initGridRange = 0.0;
646     if (std::abs(bcDensDomain[0][0]) > bcDensDomain[0][1])
647             initGridRange = std::abs(bcDensDomain[0][0]) - 0.5;
648     else
649             initGridRange = std::abs(bcDensDomain[0][1]) - 0.5;
650
651     for (size_t iX=0; iX<=m_xGridN; iX++) {
652       for (long iY=0; iY<=m_yGridN; iY++) {
653                 m_xGridPts.push_back(-1.0*initGridRange + 2.0*iX*initGridRange/
      static_cast<double>(m_xGridN));
654                 m_yGridPts.push_back(-1.0*initGridRange + 2.0*iY*initGridRange/
      static_cast<double>(m_yGridN));
655             }
656         }
657
658     if (loadPhiPoints(m_phiGridPts) != 1) return -7;
659   }
660
661   return 1;
662 }
663
664 int energyLoss::loadTempEvol()
665 {
666   std::string path_in = "./evols/tempevol_cent=" + m_centrality + ".dat";
667   std::ifstream file_in(path_in, std::ios_base::in);
668   if (!file_in.is_open()) {
669     std::cerr << "Error: unable to open temperature evolution file. Aborting..." << std::
      endl;
```

```
670     return -1;
671   }
672
673   std::string line; double buffer;
674
675     while (std::getline(file_in, line)) { // skiping header lines that start with '#'
676         if (line.at(0) == '#')
677             continue;
678         break;
679     }
680
681   //checking how many columns evolution file has:
682   size_t columnCnt = 0;
683   std::stringstream lineSStr(line); while (lineSStr >> buffer) columnCnt++;
684
685   file_in.clear(); file_in.seekg(0); //return to the begining of file
686
687   std::vector<double> tempTau, tempX, tempY, tempDataA, tempDataB, tempT;
688
689   if (columnCnt == 4) { //evolution file has 4 columns (just temperature)
690
691     while (std::getline(file_in, line)) {
692       if (line.at(0) == '#')
693         continue;
694
695       std::stringstream ss(line);
696       ss >> buffer; tempTau.push_back(buffer);
697       ss >> buffer; tempX.push_back(buffer);
698       ss >> buffer; tempY.push_back(buffer);
699       ss >> buffer; tempT.push_back(buffer);
700     }
701   }
702   else if (columnCnt == 5) { //evolution file has 5 columns (energy density and
       temperature)
703
704     while (std::getline(file_in, line)) {
705       if (line.at(0) == '#')
706         continue;
707
708       std::stringstream ss(line);
709       ss >> buffer; tempTau.push_back(buffer);
710       ss >> buffer; tempX.push_back(buffer);
711       ss >> buffer; tempY.push_back(buffer);
712       ss >> buffer; tempDataA.push_back(buffer);
713       ss >> buffer; tempDataB.push_back(buffer);
714     }
715
716     double tempDataAMax = *std::max_element(tempDataA.begin(), tempDataA.end());
717     double tempDataBMax = *std::max_element(tempDataB.begin(), tempDataB.end());
718
719         if (tempDataAMax < tempDataBMax) {
720             tempT.assign(tempDataA.begin(), tempDataA.end()); // 4th column is
       temperature
721     } else {
722             tempT.assign(tempDataB.begin(), tempDataB.end()); // 5th column is
       temperature
723     }
724
725   }
726   else { //evolution file is not suitable for interpolation
727
728     std::cerr << "Error: number of columns is not appropriate for temperature evolution
       interpolation. Aborting..." << std::endl;
729     return -2;
730   }
731
732     file_in.close();
733
734     double tempXMin = *std::min_element(tempX.begin(), tempX.end());
735     double tempYMin = *std::min_element(tempY.begin(), tempY.end());
736
737   if ((tempXMin >= 0.0) && (tempYMin >= 0.0)) {// if temperature evolution is defined
```

```
    only in the first quadrant

    //creating tau grid:
    std::vector<double> tempTauGrid(tempTau.begin(), tempTau.end());
    std::sort(tempTauGrid.begin(), tempTauGrid.end());
    tempTauGrid.erase(unique(tempTauGrid.begin(), tempTauGrid.end()), tempTauGrid.end());

    //creating full x grid:
    std::vector<double> tempXGrid(tempX.begin(), tempX.end());
        size_t sizeX = tempXGrid.size();
        tempXGrid.reserve(sizeX * 2);
        for (size_t i=0; i<sizeX; ++i)
            tempXGrid.push_back(-1.0*tempXGrid[i]);
    sort(tempXGrid.begin(), tempXGrid.end());
    tempXGrid.erase(unique(tempXGrid.begin(), tempXGrid.end()), tempXGrid.end());

    //creating full y grid:
    std::vector<double> tempYGrid(tempY.begin(), tempY.end());
        size_t sizeY = tempYGrid.size();
        tempYGrid.reserve(sizeY * 2);
        for (size_t i=0; i<sizeY; ++i)
            tempYGrid.push_back(-1.0*tempYGrid[i]);
    sort(tempYGrid.begin(), tempYGrid.end());
    tempYGrid.erase(unique(tempYGrid.begin(), tempYGrid.end()), tempYGrid.end());

        // temperature volution interpolated function in first quadtrant:
    interpolationF<double> tempEvolFirstQuadrant(tempTau, tempX, tempY, tempT);

    //creating full temperature evolution table:
    std::vector<double> tempTauFull, tempXFull, tempYFull, tempTFull;
    for (const auto &tau : tempTauGrid) {
      for (const auto &x : tempXGrid) {
        for (const auto &y : tempYGrid) {
          tempTauFull.push_back(tau);
            tempXFull.push_back(x);
            tempYFull.push_back(y);
            tempTFull.push_back(tempEvolFirstQuadrant.interpolation(tau, std::abs(x), std
    ::abs(y)));
        }
            }
        }

    m_tempEvol.setData(tempTauFull, tempXFull, tempYFull, tempTFull);
  }
  else {// if not, creating interpolated function with values from file:
    m_tempEvol.setData(tempTau, tempX, tempY, tempT);
  }

  m_tau0 = m_tempEvol.domain()[0][0];

  return 1;
}


void energyLoss::radCollEnergyLoss(double x, double y, double phi, std::vector<double> &
    radRAA1, std::vector<std::vector<double>> &radRAA2, std::vector<double> &collEL,
    double &pathLength, double &temperature) const
//function that calculates radiative and collisional EL for particles created in (X0, Y0)
     with direction phi0 (modefied pT integration algorithm)
//x, y, phi     - inital position and angle                        <- input
//radiativeRAA1 - radiative RAA for single trajectory (dA410)            <- output
//radiativeRAA2 - radiative RAA for single trajectory (rest of dA integrals) <- output
//collisionalEL - collisional energy loss for single trajectory            <- output
//pathLength    - path-length for single trajectory              <- output
//temperature   - temperature for single trajectory              <- output
{
  std::vector<double> currLTTabL, currLTTabT; //defining arrays that will store current
    path-lengths and temperatures

  double t = m_tau0, currTemp; // defining current path-length (time) and temperature

  while ((currTemp = m_tempEvol.interpolation(t, x + t*std::cos(phi), y + t*std::sin(phi)
```

```
    )) > m_TCRIT) {// calculating current path-length and temp table
804     currLTTabL.push_back(t);
805     currLTTabT.push_back(currTemp);
806     t += m_TIMESTEP;
807   }
808
809   if (currLTTabL.size() > 1) {// calculating energy loss if path-length is longer than
        thermalization time
810
811     //Radiative EnergyLoss calculation:
812
813     std::vector<double> currNormTabTau(currLTTabL.size()), currNormTabVal(currLTTabL.size
        ()); // LNorm table to be integrated over tau
814     std::vector<double> NormSparseP, NormSparseV;                        // table for
        currNormInterp
815
816     std::vector<double> currDndxTabTau(currLTTabL.size()), currDndxTabVal(currLTTabL.size
        ()); // Ldndx table to be integrated over tau
817     std::vector<double> dndxSparseP, dndxSparseX, dndxSparseV;                    //
        table for currDndxInterp
818
819     for (const auto &p : m_Grids.pPts()) //loop over ppts
820     {
821       for (size_t l=0; l<currLTTabL.size(); l++) {// loop over current path-length and
        temperature table
822         currNormTabTau[l] = currLTTabL[l];                       //setting path-lengths
823         currNormTabVal[l] = m_LNorm.interpolation(currLTTabL[l], p, currLTTabT[l]); //
        setting current norm values by integrating over time
824       }
825
826       NormSparseP.push_back(p);                              //setting p of current norm
        table
827       NormSparseV.push_back(poly::linearIntegrate(currNormTabTau, currNormTabVal)); //
        setting value of current norm table
828
829       for (const auto &x : m_Grids.xPts()) {// loop over xpts
830         for (size_t l=0; l<currLTTabL.size(); l++) {// loop over current path-length and
        temperature table
831           currDndxTabTau[l] = currLTTabL[l];                          //setting path-
        lengths
832           currDndxTabVal[l] = m_Ldndx.interpolation(currLTTabL[l], p, currLTTabT[l], x);
        //setting Ldndx values
833         }
834
835         dndxSparseP.push_back(p);                              //setting p of current dndx
        table
836         dndxSparseX.push_back(x);                              //setting x of current dndx
        table
837         dndxSparseV.push_back(poly::linearIntegrate(currDndxTabTau, currDndxTabVal)); //
        setting curernt dndx values by integrating over time
838       }
839     }
840
841     interpolationF<double> currNorm(NormSparseP, NormSparseV);        //constructing
        interpolated current norm
842     interpolationF<double> currDndx(dndxSparseP, dndxSparseX, dndxSparseV); //
        constructing interpolated current dndx
843
844
845     for (const auto &ph : m_Grids.RadPts()) {// loop over Radpts
846       radRAA1.push_back(dAp410(ph, currNorm));
847
848       radRAA2.push_back(std::vector<double>());
849       for (const auto &Fdp : m_Grids.FdpPts())
850         radRAA2.back().push_back(FdA(ph, Fdp, currNorm, currDndx));
851
852     }
853
854     //Collisional EnergyLoss calculation:
855
856     std::vector<double> currCollTabTau(currLTTabL.size()), currCollTabVal(currLTTabL.size
        ()); //collisional table to be integrated over tau
```

```
857
858     for (const auto &p : m_Grids.pCollPts()) {// loop over pCollPts
859       for (size_t l=0; l<currLTTabL.size(); l++) {// loop over current path-length and
        temperature table
860         currCollTabTau[l] = currLTTabL[l];               //setting path-lengths
861         currCollTabVal[l] = m_LColl.interpolation(p, currLTTabT[l]); //setting LColl
        values
862       }
863
864       collEL.push_back(poly::linearIntegrate(currCollTabTau, currCollTabVal)); //
        calculating collisional energy loss by integrating over time
865     }
866
867     pathLength = currLTTabL.back(); //setting value of path-length for single trajectory
868
869     //calculating mean temperature along path
870     temperature = 0.0;
871     for (size_t l=0; l<currLTTabL.size(); l++) temperature += currLTTabT[l];
872     temperature /= static_cast<double>(currLTTabL.size());
873   }
874   else { //if path-length is smaller than thermalization time:
875
876     pathLength  = 0.0; //setting path-length and temperature
877     temperature = 0.0;
878   }
879 }
880
881 void energyLoss::radCollEnergyLoss(double x, double y, double phi, std::vector<double> &
        radRAA, std::vector<double> &collEL, double &pathLenght, double &temperature) const
882 //function that calculates radiative and collisional EL for particles created in (X0, Y0)
         with direction phi0 (standard algorithm)
883 //x, y, phi    - inital position and angle             <- input
884 //radRAA       - radiative RAA for single trajectory       <- output
885 //collEL       - collisional energy loss for single trajectory <- output
886 //pathLenght   - path-length for single trajectory       <- output
887 //temperature - temperature for single trajectory         <- output
888 {
889   std::vector<double> currLTTabL, currLTTabT; //defining arrays that will store current
      path-lengths and temperatures
890
891   double t = m_tau0, currTemp; //defining current path-length (time) and temperature
892
893   while ((currTemp = m_tempEvol.interpolation(t, x + t*std::cos(phi), y + t*std::sin(phi)
      )) > m_TCRIT) { //calculating current path-length and temp table
894     currLTTabL.push_back(t);
895     currLTTabT.push_back(currTemp);
896     t += m_TIMESTEP;
897   }
898
899   if (currLTTabL.size() > 1) { //calculating energy loss if path-length is longer than
      thermalization time
900
901     //Radiative EnergyLoss calculation:
902
903     std::vector<double> currNormTabTau(currLTTabL.size()), currNormTabVal(currLTTabL.size
      ()); //LNorm table to be integrated over tau
904     std::vector<double> NormSparseP, NormSparseV;                        //table for
      currNormInterp
905
906     std::vector<double> currDndxTabTau(currLTTabL.size()), currDndxTabVal(currLTTabL.size
      ()); //Ldndx table to be integrated over tau
907     std::vector<double> dndxSparseP, dndxSparseX, dndxSparseV;                    //table
       for currDndxInterp
908
909     for (const auto &p : m_Grids.pPts()) //loop over ppts
910     {
911       for (size_t iL=0; iL<currLTTabL.size(); iL++) //loop over current path-length and
      temperature table
912       {
913         currNormTabTau[iL] = currLTTabL[iL];                        //setting path-lengths
914         currNormTabVal[iL] = m_LNorm.interpolation(currLTTabL[iL], p, currLTTabT[iL]); //
      setting current norm values by integrating over time
```

```
915       }
916
917       NormSparseP.push_back(p);                              //setting p of current norm
      table
918       NormSparseV.push_back(poly::linearIntegrate(currNormTabTau, currNormTabVal)); //
      setting value of current norm table
919
920       for (const auto &x : m_Grids.xPts()) //loop over xpts
921       {
922         for (size_t iL=0; iL<currLTTabL.size(); iL++) //loop over current path-length and
       temperature table
923         {
924           currDndxTabTau[iL] = currLTTabL[iL];                                  //setting path-
      lengths
925           currDndxTabVal[iL] = m_Ldndx.interpolation(currLTTabL[iL], p, currLTTabT[iL], x
      ); //setting Ldndx values
926         }
927
928         dndxSparseP.push_back(p);                              //setting p of current dndx
       table
929         dndxSparseX.push_back(x);                             //setting x of current dndx
      table
930         dndxSparseV.push_back(poly::linearIntegrate(currDndxTabTau, currDndxTabVal)); //
      setting curernt dndx values by integrating over time
931       }
932     }
933
934     interpolationF<double> currNorm(NormSparseP, NormSparseV);          //constructing
      interpolated current norm
935     interpolationF<double> currDndx(dndxSparseP, dndxSparseX, dndxSparseV); //
      constructing interpolated current dndx
936
937     for (const auto &p : m_Grids.RadPts())
938       radRAA.push_back(dA41(p, currNorm, currDndx)/m_dsdpti2.interpolation(p)); //
      calculating radiative RAA
939
940     //Collisional EnergyLoss calculation:
941
942     std::vector<double> currCollTabTau(currLTTabL.size()), currCollTabVal(currLTTabL.size
      ()); //collisional table to be integrated over tau
943
944     for (const auto &p : m_Grids.pCollPts()) //loop over pCollPts
945     {
946       for (size_t iL=0; iL<currLTTabL.size(); iL++) //loop over current path-length and
      temperature table
947       {
948         currCollTabTau[iL] = currLTTabL[iL];                    //setting path-lengths
949         currCollTabVal[iL] = m_LColl.interpolation(p, currLTTabT[iL]); //setting LColl
      values
950       }
951
952       collEL.push_back(poly::linearIntegrate(currCollTabTau, currCollTabVal)); //
      calculating collisional energy loss by integrating over time
953     }
954
955     pathLenght = currLTTabL.back(); //setting value of path-length for single trajectory
956
957     //calculating mean temperature along path
958     temperature = 0.0;
959     for (size_t iL=0; iL<currLTTabL.size(); iL++) temperature += currLTTabT[iL];
960     temperature /= static_cast<double>(currLTTabL.size());
961   }
962   else { //if path-length is smaller than thermalization time:
963
964     pathLenght    = 0.0; //setting path-length and temperature
965     temperature   = 0.0;
966   }
967 }
968
969
970 void energyLoss::generateGaussTab(std::vector<double> &qGTab, std::vector<double> &fGTab)
       const
```

```
971 //function that generates sampling points for Gaussian integration
972 //qGTab, fGTab - vectors that store sampling point <- output
973 {
974   double sigmaNum = 3.5; //setting sigma
975   double sigmaStep = 0.25; //setting step
976   size_t GTabLen = 2 * static_cast<size_t>(sigmaNum / sigmaStep) + 1; //setting length of
        sampling points
977
978   double GaussTabSum = 0.0; //setting normalization sum to zero
979
980   for (size_t iG=0; iG<GTabLen; iG++) //calculating sampling points
981   {
982     qGTab.push_back(-1.0*sigmaNum + static_cast<double>(iG)*sigmaStep); //setting
      qGaussTab values
983     fGTab.push_back(std::exp(-qGTab.back()*qGTab.back()/2.0));          //setting
      fGaussTab values
984     GaussTabSum += fGTab.back();                                        //adding to
      normalization sum
985   }
986
987   for (size_t iG=0; iG<GTabLen; iG++)  //normalizing
988   {
989     fGTab[iG] /= GaussTabSum; //dividing fGaussTab values with total sum
990   }
991 }
992
993 void energyLoss::gaussFilterIntegrate(const std::vector<double> &radiativeRAA1, const std
      ::vector<std::vector<double>> &radiativeRAA2, const std::vector<double> &
      collisionalEL, std::vector<double> &singRAA1, std::vector<std::vector<double>> &
      singRAA2) const
994 //function that performs Gauss filter integration - modefied pT integration algorithm
995 //radiativeRAA1 - raditive RAA (dA410)                        <- input
996 //radiativeRAA2 - raditive RAA (rest of dA integrals)               <- input
997 //collisionalEL - collisional energy loss                     <- input
998 //singRAA1    - RAA array after Gauss filter integration (dA410)          <- output
999 //singRAA2    - RAA array after Gauss filter integration (rest of dA integrals) <- output
1000 {
1001     interpolationF<double> muCollInt(m_Grids.pCollPts(), collisionalEL); //creating
      collisional energy loss interpolated function
1002
1003   std::vector<double> qGaussTabOG, fGaussTabOG; //defining vectors that will store
        original Gauss filter sampling points
1004   generateGaussTab(qGaussTabOG, fGaussTabOG);   //generating sampling points and settin
        number of sampling poins
1005
1006   std::vector<double> qGaussTab, fGaussTab; //defining vectors that will store Gauss
        filter sampling points
1007
1008   ////////////////////////////////////////////////////////////////////////////
1009   //Gauss integration of dAp410:
1010   {
1011         interpolationF<double> RadRelInt(m_Grids.RadPts(), radiativeRAA1); //creating
      radiative RAA1 interpolated function
1012
1013     double GFSum; //defining sum variable for Gauss filter
1014     double dppT;  //defining integration variable
1015
1016     double muCollCurrVal; //defining variable that stores value of interpolated muColl
      for specific pT, ie current value
1017     double sigmaColl;     //defining variable for collisional sigma
1018
1019     for (const auto &pT : m_Grids.finPts())
1020     {
1021       GFSum = 0.0;
1022
1023       muCollCurrVal = muCollInt.interpolation(pT);
1024
1025       sigmaColl = std::sqrt(2.0*m_TCollConst*muCollCurrVal);
1026
1027       qGaussTab = qGaussTabOG; fGaussTab = fGaussTabOG; //setting Gauss filter
1028
1029       if ((muCollCurrVal + sigmaColl * qGaussTab.front()) < -3.0) {
```

```
    //checking if Gauss is out of bound on lower bound
1030        double resfac = ((-3.0 + 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.front();
              //setting rescaling factor
1031        std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
    resfac; }); //rescaling sampling points if they are out of bounds
1032      }
1033
1034      if ((muCollCurrVal + sigmaColl * qGaussTab.back()) > 20.0) {                    //
    checking if Gauss is out of bound on upper bound
1035        double resfac = ((20.0 - 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.back();
             //setting rescaling factor
1036        std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
    resfac; }); //rescaling sampling points if they are out of bounds
1037      }
1038
1039      //calculating Gauss filter
1040      for (size_t iG=0; iG<qGaussTab.size(); iG++)
1041      {
1042        dppT = muCollCurrVal + sigmaColl * qGaussTab[iG];
1043        GFSum += (m_dsdpti2.interpolation(pT + dppT)*RadRelInt.interpolation(pT + dppT)*(
    pT + dppT) / pT * fGaussTab[iG]);
1044      }
1045
1046      singRAA1.push_back(1.0 / m_dsdpti2.interpolation(pT) * GFSum);
1047    }
1048  }
1049
1050  ////////////////////////////////////////////////////////////////////////////////
1051  //Gauss integration of FdA:
1052  {
1053    interpolationF<double> RadRelInt(m_Grids.RadPts(), m_Grids.FdpPts(), radiativeRAA2);
1054
1055    double GFSum; //defining sum variable for Gauss filter
1056    double dppT;  //defining integration variable
1057
1058    double muCollCurrVal; //defining variable that stores value of interpolated muColl
    for specific pT, ie current value
1059    double sigmaColl;     //defining variable for collisional sigma
1060
1061    for (const auto &pT : m_Grids.finPts())
1062    {
1063      singRAA2.push_back(std::vector<double>()); //resizing single RAA vector
1064
1065      muCollCurrVal = muCollInt.interpolation(pT);
1066
1067      sigmaColl = std::sqrt(2.0*m_TCollConst*muCollCurrVal);
1068
1069      qGaussTab = qGaussTabOG; fGaussTab = fGaussTabOG; //setting Gauss filter
1070
1071      if ((muCollCurrVal + sigmaColl * qGaussTab.front()) < -3.0) {
    //checking if Gauss is out of bound on lower bound
1072        double resfac = ((-3.0 + 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.front();
              //setting rescaling factor
1073        std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
    resfac; }); //rescaling sampling points if they are out of bounds
1074      }
1075
1076      if ((muCollCurrVal + sigmaColl * qGaussTab.back()) > 20.0) {                    //
    checking if Gauss is out of bound on upper bound
1077        double resfac = ((20.0 - 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.back();
              //setting rescaling factor
1078        std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
    resfac; }); //rescaling sampling points if they are out of bounds
1079      }
1080
1081      for (const auto &dpT : m_Grids.FdpPts()) //loop over FdpPts
1082      {
1083        GFSum = 0.0; //setting sum to 0
1084
1085        //calculating Gauss filter
1086        for (size_t iG=0; iG<qGaussTab.size(); iG++)
1087        {
```

```
1088            dppT = muCollCurrVal + sigmaColl * qGaussTab[iG];
1089            GFSum += (m_dsdpti2.interpolation(pT + dpT + dppT)*RadRelInt.interpolation(pT +
     dppT, dpT)*(pT + dppT)/(pT+ dpT + dppT)*fGaussTab[iG]);
1090          }
1091
1092          singRAA2.back().push_back(1.0 / m_dsdpti2.interpolation(pT) * GFSum);
1093        }
1094      }
1095    }
1096  }
1097
1098  void energyLoss::gaussFilterIntegrate(const interpolationF<double> &dsdpti2lquark, const
       std::vector<double> &radiativeRAA1, const std::vector<std::vector<double>> &
       radiativeRAA2, const std::vector<double> &collisionalEL, std::vector<double> &
       singRAA1, std::vector<std::vector<double>> &singRAA2) const
1099  //function that performs Gauss filter integration - modefied pT integration algorithm
       used in all lquarks algorithm
1100  //dsdpti2lquark - light quark initial pT distribution                    <- input
1101  //radiativeRAA1 - raditive RAA (dA410)                          <- input
1102  //radiativeRAA2 - raditive RAA (rest of dA integrals)                 <- input
1103  //collisionalEL - collisional energy loss                       <- input
1104  //singRAA1    - RAA array after Gauss filter integration (dA410)          <- output
1105  //singRAA2    - RAA array after Gauss filter integration (rest of dA integrals) <- output
1106  {
1107      interpolationF<double> muCollInt(m_Grids.pCollPts(), collisionalEL); //creating
       collisional energy loss interpolated function
1108
1109    std::vector<double> qGaussTabOG, fGaussTabOG; //defining vectors that will store
       original Gauss filter sampling points
1110    generateGaussTab(qGaussTabOG, fGaussTabOG);   //generating sampling points and settin
       number of sampling poins
1111
1112    std::vector<double> qGaussTab, fGaussTab; //defining vectors that will store Gauss
       filter sampling points
1113
1114    ////////////////////////////////////////////////////////////////////////////////
1115    //Gauss integration of dAp410:
1116    {
1117          interpolationF<double> RadRelInt(m_Grids.RadPts(), radiativeRAA1); //creating
       radiative RAA1 interpolated function
1118
1119      double GFSum; //defining sum variable for Gauss filter
1120      double dppT;  //defining integration variable
1121
1122      double muCollCurrVal; //defining variable that stores value of interpolated muColl
       for specific pT, ie current value
1123      double sigmaColl;     //defining variable for collisional sigma
1124
1125      for (const auto &pT : m_Grids.finPts())
1126      {
1127        GFSum = 0.0;
1128
1129        muCollCurrVal = muCollInt.interpolation(pT);
1130
1131        sigmaColl = std::sqrt(2.0*m_TCollConst*muCollCurrVal);
1132
1133        qGaussTab = qGaussTabOG; fGaussTab = fGaussTabOG; //setting Gauss filter
1134
1135        if ((muCollCurrVal + sigmaColl * qGaussTab.front()) < -3.0) {
       //checking if Gauss is out of bound on lower bound
1136          double resfac = ((-3.0 + 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.front();
             //setting rescaling factor
1137          std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
       resfac; }); //rescaling sampling points if they are out of bounds
1138        }
1139
1140        if ((muCollCurrVal + sigmaColl * qGaussTab.back()) > 20.0) {                    //
       checking if Gauss is out of bound on upper bound
1141          double resfac = ((20.0 - 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.back();
           //setting rescaling factor
1142          std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
       resfac; }); //rescaling sampling points if they are out of bounds
```

```
1143        }
1144
1145        //calculating Gauss filter
1146        for (size_t iG=0; iG<qGaussTab.size(); iG++)
1147        {
1148          dppT = muCollCurrVal + sigmaColl * qGaussTab[iG];
1149          GFSum += (dsdpti2lquark.interpolation(pT + dppT)*RadRelInt.interpolation(pT +
      dppT)*(pT + dppT) / pT * fGaussTab[iG]);
1150        }
1151
1152        singRAA1.push_back(1.0 / dsdpti2lquark.interpolation(pT) * GFSum);
1153      }
1154    }
1155
1156    ////////////////////////////////////////////////////////////////////////////
1157    //Gauss integration of FdA:
1158    {
1159      interpolationF<double> RadRelInt(m_Grids.RadPts(), m_Grids.FdpPts(), radiativeRAA2);
1160
1161      double GFSum; //defining sum variable for Gauss filter
1162      double dppT;  //defining integration variable
1163
1164      double muCollCurrVal; //defining variable that stores value of interpolated muColl
      for specific pT, ie current value
1165      double sigmaColl;     //defining variable for collisional sigma
1166
1167      for (const auto &pT : m_Grids.finPts())
1168      {
1169        singRAA2.push_back(std::vector<double>()); //resizing single RAA vector
1170
1171        muCollCurrVal = muCollInt.interpolation(pT);
1172
1173        sigmaColl = std::sqrt(2.0*m_TCollConst*muCollCurrVal);
1174
1175        qGaussTab = qGaussTabOG; fGaussTab = fGaussTabOG; //setting Gauss filter
1176
1177        if ((muCollCurrVal + sigmaColl * qGaussTab.front()) < -3.0) {
      //checking if Gauss is out of bound on lower bound
1178          double resfac = ((-3.0 + 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.front();
                //setting rescaling factor
1179          std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
      resfac; }); //rescaling sampling points if they are out of bounds
1180        }
1181
1182        if ((muCollCurrVal + sigmaColl * qGaussTab.back()) > 20.0) {                    //
      checking if Gauss is out of bound on upper bound
1183          double resfac = ((20.0 - 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.back();
           //setting rescaling factor
1184          std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *=
      resfac; }); //rescaling sampling points if they are out of bounds
1185        }
1186
1187        for (const auto &dpT : m_Grids.FdpPts()) //loop over FdpPts
1188        {
1189          GFSum = 0.0; //setting sum to 0
1190
1191          //calculating Gauss filter
1192          for (size_t iG=0; iG<qGaussTab.size(); iG++)
1193          {
1194            dppT = muCollCurrVal + sigmaColl * qGaussTab[iG];
1195            GFSum += (dsdpti2lquark.interpolation(pT + dpT + dppT)*RadRelInt.interpolation(
      pT + dppT, dpT)*(pT + dppT)/(pT+ dpT + dppT)*fGaussTab[iG]);
1196          }
1197
1198          singRAA2.back().push_back(1.0 / dsdpti2lquark.interpolation(pT) * GFSum);
1199        }
1200      }
1201    }
1202  }
1203
1204  void energyLoss::gaussFilterIntegrate(const std::vector<double> &radiativeRAA, const std
      ::vector<double> &collisionalEL, std::vector<double> &singRAA) const
```

```
1205 //function that performs Gauss filter integration - default algorithm
1206 //radiativeRAA  - raditive RAA                  <- input
1207 //collisionalEL - collisional energy loss          <- input
1208 //singRAA     - RAA array after Gauss filter integration <- output
1209 {
1210     interpolationF<double> RadRelInt(m_Grids.RadPts(),   radiativeRAA);  //creating
       radiative RAA interpolated function
1211     interpolationF<double> muCollInt(m_Grids.pCollPts(), collisionalEL); //creating
       collisional energy loss interpolated function
1212
1213   std::vector<double> qGaussTabOG, fGaussTabOG; //defining vectors that will store
       original Gauss filter sampling points
1214   generateGaussTab(qGaussTabOG, fGaussTabOG);   //generating sampling points and settin
       number of sampling poins
1215
1216   std::vector<double> qGaussTab, fGaussTab; //defining vectors that will store Gauss
       filter sampling points
1217
1218   double GFSum; //defining sum variable for Gauss filter
1219
1220   double dpT; //defining pT and dpT variables
1221
1222   double muCollCurrVal; //defining variable that stores value of interpolated muColl for
       specific pT, ie current value
1223
1224   double sigmaColl; //defining variable for collisional sigma
1225
1226   //Gauss filter
1227   for (const auto &pT : m_Grids.finPts())
1228   {
1229     GFSum = 0.0L;
1230
1231     muCollCurrVal = muCollInt.interpolation(pT);
1232
1233     sigmaColl = std::sqrt(2.0*m_TCollConst*muCollCurrVal);
1234
1235     qGaussTab = qGaussTabOG; fGaussTab = fGaussTabOG; //setting Gauss filter
1236
1237     if ((muCollCurrVal + sigmaColl * qGaussTab.front()) < -3.0) {                      //
       checking if Gauss is out of bound on lower bound
1238       double resfac = ((-3.0 + 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.front();
          //setting rescaling factor
1239       std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *= resfac;
        }); //rescaling sampling points if they are out of bounds
1240     }
1241
1242     if ((muCollCurrVal + sigmaColl * qGaussTab.back()) > 20.0) {                     //
       checking if Gauss is out of bound on upper bound
1243       double resfac = ((20.0 - 1e-12) - muCollCurrVal)/sigmaColl/qGaussTab.back();
         //setting rescaling factor
1244       std::for_each(qGaussTab.begin(), qGaussTab.end(), [resfac](double &c){ c *= resfac;
        }); //rescaling sampling points if they are out of bounds
1245     }
1246
1247     //calculating Gauss filter
1248     for (size_t iG=0; iG<qGaussTab.size(); iG++)
1249     {
1250       dpT = muCollCurrVal + sigmaColl * qGaussTab[iG];
1251       GFSum += (m_dsdpti2.interpolation(pT + dpT)*RadRelInt.interpolation(pT + dpT)*(pT +
        dpT) / pT * fGaussTab[iG]);
1252     }
1253
1254     singRAA.push_back(1.0 / m_dsdpti2.interpolation(pT) * GFSum);
1255   }
1256 }
1257
1258
1259 void energyLoss::calculateAvgPathlenTemps(const std::vector<double> &pathLenghDist, const
       std::vector<double> &temperatureDist, std::vector<double> &avgPathLength, std::
      vector<double> &avgTemp) const
1260 {
1261   interpolationF<double> pathLenghDistInt(m_phiGridPts, pathLenghDist);
```

```
1262   avgPathLength[0] = poly::cubicIntegrate(m_phiGridPts, pathLenghDist)/2.0/M_PI;
1263   avgPathLength[1] = (pathLenghDistInt.interpolation(m_phiGridPts.front()) +
         pathLenghDistInt.interpolation(m_phiGridPts.back()))/2.0;
1264   avgPathLength[2] = (pathLenghDistInt.interpolation(M_PI/2.0)             +
         pathLenghDistInt.interpolation(3.0*M_PI/2.0))       /2.0;
1265
1266   interpolationF<double> temperatureDistInt(m_phiGridPts, temperatureDist);
1267   avgTemp[0] = poly::cubicIntegrate(m_phiGridPts, temperatureDist)/2.0/M_PI;
1268   avgTemp[1] = (temperatureDistInt.interpolation(m_phiGridPts.front()) +
         temperatureDistInt.interpolation(m_phiGridPts.back()))/2.0;
1269   avgTemp[2] = (temperatureDistInt.interpolation(M_PI/2.0)             +
         temperatureDistInt.interpolation(3.0*M_PI/2.0))       /2.0;
1270 }
1271
1272 int energyLoss::exportResults(const std::string &pName, const std::vector<std::vector<
       double>> &RAADist, const std::vector<double> avgPathLength, const std::vector<double>
        avgTemp)
1273 {
1274   std::vector<std::string> header;
1275   header.push_back("#collision_system: " + m_collsys);
1276   header.push_back("#collision_energy: " + m_sNN);
1277   header.push_back("#particle_type: " + pName);
1278   header.push_back("#centrality: " + m_centrality);
1279
1280   std::stringstream xbSStr; xbSStr << std::fixed << std::setprecision(1) << m_xB;
1281   header.push_back("#xB = " + xbSStr.str());
1282
1283   std::stringstream avgPathLengthSStr[3];
1284     for (size_t i=0; i<3; i++) avgPathLengthSStr[i] << std::fixed << std::setprecision(6)
        << avgPathLength[i];
1285   header.push_back("#average_path-lengths: " + avgPathLengthSStr[0].str() + ", " +
       avgPathLengthSStr[1].str() + ", " + avgPathLengthSStr[2].str());
1286
1287   std::stringstream avgTempSStr[3];
1288     for (size_t i=0; i<3; i++) avgTempSStr[i] << std::fixed << std::setprecision(6) <<
       avgTemp[i];
1289   header.push_back("#average_temperatures: " + avgTempSStr[0].str() + ", " + avgTempSStr
       [1].str() + ", " + avgTempSStr[2].str());
1290
1291   if (m_yGridN <= 0) {
1292     header.push_back("#number_of_angles:    " + std::to_string(m_phiGridN));
1293     header.push_back("#number_of_xy_points: " + std::to_string(m_xGridN));
1294   }
1295   else {
1296     header.push_back("#number_of_angles:      " + std::to_string(m_phiGridN));
1297     header.push_back("#number_of_grid_points: " + std::to_string(m_xGridN) + ", " + std::
       to_string(m_yGridN));
1298   }
1299
1300   header.push_back("#-------------------------------------------------------");
1301   header.push_back("#   pT [GeV]       phi          R_AA   ");
1302
1303   const std::string path_out = "./results/results" + pName + "/" + pName + "_" +
       m_collsys + "_sNN=" + m_sNN + "_cent=" + m_centrality + "_xB=" + xbSStr.str() + "
       _dist.dat";
1304
1305   std::ofstream file_out(path_out, std::ios_base::out);
1306   if (!file_out.is_open()) {
1307     std::cerr << "Error: unable to open RAA(pT,phi) distribution file. Aborting..." <<
       std::endl;
1308     return -1;
1309   }
1310
1311   for (const auto &h : header) file_out << h << "\n";
1312
1313   for (size_t ipT= 0; ipT<m_Grids.finPtsLength(); ipT++)
1314     for (size_t iPhi=0; iPhi<m_phiGridN; iPhi++) {
1315       file_out << std::fixed << std::setw(14) << std::setprecision(10) <<   m_Grids.
       finPts(ipT) << " ";
1316       file_out << std::fixed << std::setw(12) << std::setprecision(10) <<    m_phiGridPts
       [iPhi] << " ";
1317       file_out << std::fixed << std::setw(12) << std::setprecision(10) << RAADist[ipT][
```

```
    iPhi] << "\n";
1318    }
1319
1320  file_out.close();
1321
1322  return 1;
1323 }
1324
1325
1326 void energyLoss::runELossHeavyFlavour()
1327 {
1328  if (loaddsdpti2(m_pName, m_dsdpti2) != 1) return;
1329
1330  FdAHaltonSeqInit(150);
1331
1332  std::vector<std::vector<double>> RAADist(m_Grids.finPtsLength(), std::vector<double>(
      m_phiGridN, 0.0));
1333
1334  std::vector<double> pathLengthDist(m_phiGridN, 0.0), temperatureDist(m_phiGridN, 0.0);
1335
1336  #pragma omp declare reduction(vectorDoublePlus : std::vector<double> : \
1337                                  std::transform(omp_out.begin(), omp_out.end(), omp_in.
      begin(), omp_out.begin(), std::plus<double>())) \
1338                          initializer(omp_priv = decltype(omp_orig)(omp_orig.size()))
1339
1340  #pragma omp parallel for reduction(vectorDoublePlus : pathLengthDist, temperatureDist)
      schedule(dynamic)
1341  for (size_t iPhi=0; iPhi<m_phiGridN; iPhi++) {
1342    double phi = m_phiGridPts[iPhi];
1343
1344    std::vector<double> sumRAA1(m_Grids.finPtsLength(), 0.0);
1345
1346    std::vector<std::vector<double>> sumRAA2(m_Grids.finPtsLength(), std::vector<double>(
      m_Grids.FdpPtsLength(), 0.0));
1347
1348    double weightsumEL = 0.0, weightsumPLT = 0.0; //energy and path-length and
      temperature loss weightsum
1349
1350    for (size_t iXY=0; iXY<m_xGridPts.size(); iXY++) {
1351      double x = m_xGridPts[iXY], y = m_yGridPts[iXY];
1352      double binCollDensity = m_binCollDensity.interpolation(x, y);
1353
1354      if (binCollDensity > 0) {
1355        weightsumEL += binCollDensity;
1356
1357        std::vector<double> radRAA1; std::vector<std::vector<double>> radRAA2; std::
      vector<double> collEL;
1358        double pathLength, temperature;
1359        radCollEnergyLoss(x, y, phi, radRAA1, radRAA2, collEL, pathLength, temperature);
1360
1361        if (pathLength > m_tau0) { //checking if path-length is larger than
      thermalization time
1362
1363          for (auto &cEL : collEL) cEL += 1e-12; //modifying collEL to prevent division
      by 0
1364
1365              weightsumPLT += binCollDensity;
1366           pathLengthDist[iPhi] += (pathLength*binCollDensity);
1367          temperatureDist[iPhi] += (temperature*binCollDensity);
1368
1369          std::vector<double> singleRAA1; std::vector<std::vector<double>> singleRAA2;
1370          gaussFilterIntegrate(radRAA1, radRAA2, collEL, singleRAA1, singleRAA2);
1371
1372          for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1373            sumRAA1[iFinPts] += singleRAA1[iFinPts]*binCollDensity;
1374            for (size_t iFdp=0; iFdp<m_Grids.FdpPtsLength(); iFdp++) {
1375              sumRAA2[iFinPts][iFdp] += singleRAA2[iFinPts][iFdp]*binCollDensity;
1376            }
1377          }
1378        }
1379        else {// if path length is smaller than tau0:
1380          for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {//
```

```
    multiplying RAA1 (which is 1) with binary collision function as weigth and adding to
    RAA sum; RAA2 is 0 in this case
1381            sumRAA1[iFinPts] += binCollDensity;
1382          }
1383        }
1384      }
1385    }
1386
1387    std::for_each(sumRAA1.begin(), sumRAA1.end(), [weightsumEL](double &c){c/=weightsumEL
    ;});
1388    for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1389      std::for_each(sumRAA2[iFinPts].begin(), sumRAA2[iFinPts].end(), [weightsumEL](
    double &c){c/=weightsumEL;});
1390    }
1391    for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1392      RAADist[iFinPts][iPhi] = sumRAA1[iFinPts] + poly::cubicIntegrate(m_Grids.FdpPts(),
    sumRAA2[iFinPts])/m_Grids.finPts(iFinPts);
1393    }
1394    pathLengthDist[iPhi] /= weightsumPLT; temperatureDist[iPhi] /= weightsumPLT;
1395  }
1396
1397  std::vector<double> avgPathLength(3, 0.0), avgTemp(3, 0.0);
1398  calculateAvgPathlenTemps(pathLengthDist, temperatureDist, avgPathLength, avgTemp);
1399
1400  if (exportResults(m_pName, RAADist, avgPathLength, avgTemp) != 1) return;
1401 }
1402
1403 void energyLoss::runELossLightQuarks()
1404 {
1405  const std::vector<std::string> lightQuarksList{"Down", "DownBar", "Strange", "Up", "
    UpBar"};
1406
1407  std::vector<interpolationF<double>> dsdpti2LightQuarks(lightQuarksList.size());
1408  for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++)
1409    if (loaddsdpti2(lightQuarksList[iLQ], dsdpti2LightQuarks[iLQ]) != 1) return;
1410
1411  FdAHaltonSeqInit(100);
1412
1413  std::vector<std::vector<std::vector<double>>> RAADist(lightQuarksList.size(), std::
    vector<std::vector<double>>(m_Grids.finPtsLength(), std::vector<double>(m_phiGridN)))
    ;
1414
1415  std::vector<double> pathLengthDist(m_phiGridN, 0.0), temperatureDist(m_phiGridN, 0.0);
1416
1417  #pragma omp declare reduction(vectorDoublePlus : std::vector<double> : \
1418                                std::transform(omp_out.begin(), omp_out.end(), omp_in.
    begin(), omp_out.begin(), std::plus<double>())) \
1419                          initializer(omp_priv = decltype(omp_orig)(omp_orig.size()))
1420
1421  #pragma omp parallel for reduction(vectorDoublePlus : pathLengthDist, temperatureDist)
    schedule(dynamic)
1422  for (size_t iPhi=0; iPhi<m_phiGridN; iPhi++) {
1423    double phi = m_phiGridPts[iPhi];
1424
1425    std::vector<std::vector<double>> sumRAA1(lightQuarksList.size(), std::vector<double>(
    m_Grids.finPtsLength(), 0.0));
1426    std::vector<std::vector<std::vector<double>>> sumRAA2(lightQuarksList.size(), std::
    vector<std::vector<double>>(m_Grids.finPtsLength(), std::vector<double>(m_Grids.
    FdpPtsLength(), 0.0)));
1427
1428    double weightsumEL = 0.0, weightsumPLT = 0.0; //energy and path-length and
    temperature loss weightsum
1429
1430    for (size_t iXY=0; iXY<m_xGridPts.size(); iXY++) {
1431      double x = m_xGridPts[iXY], y = m_yGridPts[iXY];
1432      double binCollDensity = m_binCollDensity.interpolation(x, y);
1433
1434      if (binCollDensity > 0) {
1435        weightsumEL += binCollDensity;
1436
1437        std::vector<double> radRAA1; std::vector<std::vector<double>> radRAA2; std::
    vector<double> collEL;
```

```
1438        double pathLength, temperature;
1439        radCollEnergyLoss(x, y, phi, radRAA1, radRAA2, collEL, pathLength, temperature);
1440
1441        if (pathLength > m_tau0) { //checking if path-length is larger than
    thermalization time
1442
1443          for (auto &cEL : collEL) cEL += 1e-12; //modifying collEL to prevent division
    by 0
1444
1445              weightsumPLT += binCollDensity;
1446           pathLengthDist[iPhi] += (pathLength*binCollDensity);
1447          temperatureDist[iPhi] += (temperature*binCollDensity);
1448
1449          std::vector<std::vector<double>> singleRAA1(lightQuarksList.size());
1450          std::vector<std::vector<std::vector<double>>> singleRAA2(lightQuarksList.size()
    );
1451          for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++)
1452            gaussFilterIntegrate(dsdpti2LightQuarks[iLQ], radRAA1, radRAA2, collEL,
    singleRAA1[iLQ], singleRAA2[iLQ]);
1453
1454          for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++) {
1455            for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1456              sumRAA1[iLQ][iFinPts] += singleRAA1[iLQ][iFinPts]*binCollDensity;
1457              for (size_t iFdp=0; iFdp<m_Grids.FdpPtsLength(); iFdp++)
1458                sumRAA2[iLQ][iFinPts][iFdp] += singleRAA2[iLQ][iFinPts][iFdp]*
    binCollDensity;
1459            }
1460          }
1461        }
1462        else {
1463          for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++)
1464            for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++)
1465              sumRAA1[iLQ][iFinPts] += binCollDensity;
1466        }
1467      }
1468    }
1469
1470    for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++) {
1471      std::for_each(sumRAA1[iLQ].begin(), sumRAA1[iLQ].end(), [weightsumEL](double &c){ c
    /=weightsumEL; });
1472      for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1473        std::for_each(sumRAA2[iLQ][iFinPts].begin(), sumRAA2[iLQ][iFinPts].end(), [
    weightsumEL](double &c){ c/=weightsumEL; });
1474      }
1475    }
1476
1477    //setting RAA(pT,phi) value by integrating over p:
1478    for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++) {
1479      for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1480        RAADist[iLQ][iFinPts][iPhi] = sumRAA1[iLQ][iFinPts] + poly::cubicIntegrate(
    m_Grids.FdpPts(), sumRAA2[iLQ][iFinPts])/m_Grids.finPts(iFinPts);
1481      }
1482    }
1483
1484    pathLengthDist[iPhi] /= weightsumPLT; temperatureDist[iPhi] /= weightsumPLT;
1485  }
1486
1487  std::vector<double> avgPathLength(3, 0.0), avgTemp(3, 0.0);
1488  calculateAvgPathlenTemps(pathLengthDist, temperatureDist, avgPathLength, avgTemp);
1489
1490  for (size_t iLQ=0; iLQ<lightQuarksList.size(); iLQ++) {
1491    if (exportResults(lightQuarksList[iLQ], RAADist[iLQ], avgPathLength, avgTemp) != 1)
    return;
1492  }
1493}
1494
1495void energyLoss::runELossLightFlavour()
1496{
1497  if (loaddsdpti2(m_pName, m_dsdpti2) != 1) return;
1498
1499  dAHaltonSeqInit(1000);
1500
```

```
1501  std::vector<std::vector<double>> RAADist(m_Grids.finPtsLength(), std::vector<double>(
        m_phiGridN, 0.0));
1502
1503  std::vector<double> pathLengthDist(m_phiGridN, 0.0), temperatureDist(m_phiGridN, 0.0);
1504
1505  #pragma omp declare reduction(vectorDoublePlus : std::vector<double> : \
1506                                std::transform(omp_out.begin(), omp_out.end(), omp_in.
        begin(), omp_out.begin(), std::plus<double>())) \
1507                            initializer(omp_priv = decltype(omp_orig)(omp_orig.size()))
1508
1509  #pragma omp parallel for reduction(vectorDoublePlus : pathLengthDist, temperatureDist)
        schedule(dynamic)
1510  for (size_t iPhi=0; iPhi<m_phiGridN; iPhi++) {
1511    double phi = m_phiGridPts[iPhi];
1512
1513    std::vector<double> sumRAA(m_Grids.finPtsLength(), 0.0);
1514
1515    double weightsumEL = 0.0, weightsumPLT = 0.0; //energy and path-length and
      temperature loss weightsum
1516
1517    for (size_t iXY=0; iXY<m_xGridPts.size(); iXY++) {
1518      double x = m_xGridPts[iXY], y = m_yGridPts[iXY];
1519      double binCollDensity = m_binCollDensity.interpolation(x, y);
1520
1521      if (binCollDensity > 0) {
1522        weightsumEL += binCollDensity;
1523
1524        std::vector<double> radRAA; std::vector<double> collEL;
1525        double pathLength, temperature;
1526        radCollEnergyLoss(x, y, phi, radRAA, collEL, pathLength, temperature);
1527
1528        if (pathLength > m_tau0) { //checking if path-length is larger than
      thermalization time
1529
1530          for (auto &cEL : collEL) cEL += 1e-12; //modifying collEL to prevent division
      by 0
1531
1532              weightsumPLT += binCollDensity;
1533           pathLengthDist[iPhi] += (pathLength*binCollDensity);
1534          temperatureDist[iPhi] += (temperature*binCollDensity);
1535
1536          std::vector<double> singleRAA;
1537          gaussFilterIntegrate(radRAA, collEL, singleRAA);
1538
1539          for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1540            sumRAA[iFinPts] += singleRAA[iFinPts]*binCollDensity;
1541          }
1542        }
1543        else {// if path length is smaller than tau0:
1544          for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {//
      multiplying RAA1 (which is 1) with binary collision function as weigth and adding to
      RAA sum
1545            sumRAA[iFinPts] += binCollDensity;
1546          }
1547        }
1548      }
1549    }
1550
1551    for (size_t iFinPts=0; iFinPts<m_Grids.finPtsLength(); iFinPts++) {
1552      RAADist[iFinPts][iPhi] = sumRAA[iFinPts]/weightsumEL;
1553    }
1554
1555    pathLengthDist[iPhi] /= weightsumPLT; temperatureDist[iPhi] /= weightsumPLT;
1556  }
1557
1558  std::vector<double> avgPathLength(3, 0.0), avgTemp(3, 0.0);
1559  calculateAvgPathlenTemps(pathLengthDist, temperatureDist, avgPathLength, avgTemp);
1560
1561  if (exportResults(m_pName, RAADist, avgPathLength, avgTemp) != 1) return;
1562 }
```

Content of the source file that contains integrals in the Poisson expansion of the radiative energy

loss, *daintegrals.cpp*, follows:

```
#include "energyloss.hpp"
#include "linearinterpolation.hpp"

#include <vector>
#include <cmath>

double energyLoss::haltonSequence(int index, int base) const
{
  double f = 1.0;
  double res = 0.0;

  while (index > 0) {
    f = f / static_cast<double>(base);
    res += f * static_cast<double>(index % base);
    index = index / base; // integer division
  }

  return res;
}


void energyLoss::FdAHaltonSeqInit(size_t FdAMaxPts)
{
  m_FdAMaxPoints2 = FdAMaxPts;    //setting values of dAMaxPoints
  m_FdAMaxPoints3 = FdAMaxPts-25;
  m_FdAMaxPoints4 = FdAMaxPts-50;
  m_FdAMaxPoints5 = FdAMaxPts-75;

  for (size_t i=0; i<FdAMaxPts; i++) //generating Halton sequences
  {
    m_FdAHS2.push_back(haltonSequence((i+1)*409, 2));
    m_FdAHS3.push_back(haltonSequence((i+1)*409, 3));
    m_FdAHS4.push_back(haltonSequence((i+1)*409, 5));
    m_FdAHS5.push_back(haltonSequence((i+1)*409, 7));
  }
}

double energyLoss::dAp410(double ph, const interpolationF<double> &norm) const {
  return (1.0 / std::exp(norm.interpolation(ph)));
}

double energyLoss::FdA411(double ph, double dp, const interpolationF<double> &norm, const
     interpolationF<double> &dndx) const {
  return (1.0 / std::exp(norm.interpolation(ph + dp))*dndx.interpolation(ph + dp, 1.0 -
    ph/(ph + dp)));
}

double energyLoss::FdA412(double ph, double dp, const interpolationF<double> &norm, const
     interpolationF<double> &dndx) const{
  if (dp < 2.0*m_mgC / 2.0) return 0.0;
  double p = ph + dp;
  double yl, yh, yq, y;
  double sum = 0.0;
  for (size_t i=0; i<m_FdAMaxPoints2; i++) {
    yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yh = 1.0 - ph/p - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yq = yh - yl;
    y = yl + m_FdAHS2[i]*yq;

    sum += 1.0 / std::exp(norm.interpolation(p))*(1.0 / 2.0)*dndx.interpolation(p, 1.0 -
    ph/p - y)*
      dndx.interpolation(p, y)*(yh - yl);
  }

  return (sum/static_cast<double>(m_FdAMaxPoints2));
}

double energyLoss::FdA413(double ph, double dp, const interpolationF<double> &norm, const
     interpolationF<double> &dndx) const {
  if (dp < 3.0*m_mgC / 2.0) return 0.0;
  double p = ph + dp;
```

```
67    double yl, yh, yq, y;
68    double zl, zh, zq, z;
69    double sum = 0.0;
70    for (size_t i=0; i<m_FdAMaxPoints3; i++) {
71      yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
72      yh = 1.0 - ph/p - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
73      yq = yh - yl;
74      y = yl + m_FdAHS2[i]*yq;
75
76      zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
77      zh = 1.0 - ph/p - y - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
78      zq = zh - zl;
79      z = zl + m_FdAHS3[i]*zq;
80
81      sum += 1.0 / std::exp(norm.interpolation(p))*(1.0 / 2.0 / 3.0)*dndx.interpolation(p,
        1.0 - ph/p - y - z)*
82        dndx.interpolation(p, y)*dndx.interpolation(p, z)*(yh - yl)*(zh - zl);
83    }
84
85    return (sum/static_cast<double>(m_FdAMaxPoints3));
86  }
87
88  double energyLoss::FdA414(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const {
89    if (dp < 4.0*m_mgC / 2.0) return 0.0;
90    double p = ph + dp;
91    double yl, yh, yq, y;
92    double zl, zh, zq, z;
93    double zzl, zzh, zzq, zz;
94    double sum = 0.0;
95    for (size_t i=0; i<m_FdAMaxPoints4; i++) {
96      yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
97      yh = 1.0 - ph/p - 3.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
98      yq = yh - yl;
99      y = yl + m_FdAHS2[i]*yq;
100
101     zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
102     zh = 1.0 - ph/p - y - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
103     zq = zh - zl;
104     z = zl + m_FdAHS3[i]*zq;
105
106     zzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
107     zzh = 1.0 - ph/p - y - z - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
108     zzq = zzh - zzl;
109     zz = zzl + m_FdAHS4[i]*zzq;
110
111     sum += 1.0 / std::exp(norm.interpolation(p))*(1.0 / 2.0 / 3.0 / 4.0)*dndx.
        interpolation(p, 1.0 - ph/p - y - z - zz)*
112       dndx.interpolation(p, y)*dndx.interpolation(p, z)*dndx.interpolation(p, zz)*(yh -
        yl)*(zh - zl)*(zzh - zzl);
113   }
114
115   return (sum/static_cast<double>(m_FdAMaxPoints4));
116 }
117
118 double energyLoss::FdA415(double ph, double dp, const interpolationF<double> &norm, const
        interpolationF<double> &dndx) const {
119   if (dp < 5.0*m_mgC / 2.0) return 0.0;
120   double p = ph + dp;
121   double yl, yh, yq, y;
122   double zl, zh, zq, z;
123   double zzl, zzh, zzq, zz;
124   double zzzl, zzzh, zzzq, zzz;
125   double sum = 0.0;
126   for (size_t i=0; i<m_FdAMaxPoints5; i++) {
127     yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
128     yh = 1.0 - ph/p - 4.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
129     yq = yh - yl;
130     y = yl + m_FdAHS2[i]*yq;
131
132     zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
133     zh = 1.0 - ph/p - y - 3.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
```

```
134     zq = zh - zl;
135     z = zl + m_FdAHS3[i]*zq;
136 
137     zzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
138     zzh = 1.0 - ph/p - y - z - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
139     zzq = zzh - zzl;
140     zz = zzl + m_FdAHS4[i]*zzq;
141 
142     zzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
143     zzzh = 1.0 - ph/p - y - z - zz - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
144     zzzq = zzzh - zzzl;
145     zzz = zzzl + m_FdAHS5[i]*zzzq;
146 
147     sum += 1.0 / std::exp(norm.interpolation(p))*(1.0 / 2.0 / 3.0 / 4.0 / 5.0)*dndx.
      interpolation(p, 1.0 - ph/p - y - z - zz - zzz)*
148       dndx.interpolation(p, y)*dndx.interpolation(p, z)*dndx.interpolation(p, zz)*dndx.
      interpolation(p, zzz)*(yh - yl)*(zh - zl)*(zzh - zzl)*(zzzh - zzzl);
149 
150   }
151 
152   return (sum/static_cast<double>(m_FdAMaxPoints5));
153 }
154 
155 double energyLoss::FdA(double ph, double dp, const interpolationF<double> &currnorm,
      const interpolationF<double> &currdndx) const {
156   return (FdA411(ph, dp, currnorm, currdndx) + FdA412(ph, dp, currnorm, currdndx) +
      FdA413(ph, dp, currnorm, currdndx) +
157       FdA414(ph, dp, currnorm, currdndx) + FdA415(ph, dp, currnorm, currdndx));
158 }
159 
160 
161 void energyLoss::dAHaltonSeqInit(size_t dAMaxPts)
162 {
163   m_dAMaxPoints1 = dAMaxPts;   //setting values of dAMaxPoints
164   m_dAMaxPoints2 = dAMaxPts-100;
165   m_dAMaxPoints3 = dAMaxPts-200;
166   m_dAMaxPoints4 = dAMaxPts-300;
167   m_dAMaxPoints5 = dAMaxPts-400;
168   m_dAMaxPoints6 = dAMaxPts-500;
169   m_dAMaxPoints7 = dAMaxPts-600;
170 
171   for (size_t i=0; i<dAMaxPts; i++) //generating Halton sequences
172   {
173     m_dAHS1.push_back(haltonSequence((i+1)*409, 2));
174     m_dAHS2.push_back(haltonSequence((i+1)*409, 3));
175     m_dAHS3.push_back(haltonSequence((i+1)*409, 5));
176     m_dAHS4.push_back(haltonSequence((i+1)*409, 7));
177     m_dAHS5.push_back(haltonSequence((i+1)*409, 11));
178     m_dAHS6.push_back(haltonSequence((i+1)*409, 13));
179     m_dAHS7.push_back(haltonSequence((i+1)*409, 17));
180   }
181 }
182 
183 double energyLoss::dA410(double ph, const interpolationF<double> &norm) const {
184   return (m_dsdpti2.interpolation(ph)/exp(norm.interpolation(ph)));
185 }
186 
187 double energyLoss::dA411(double ph, const interpolationF<double> &norm, const
      interpolationF<double> &dndx) const {
188   double p1 = ph + m_mgC / 2.0;
189   double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
190   double pq = p2 - p1;
191   double p;
192   double sum = 0.0;
193   for (size_t i=0; i<m_dAMaxPoints1; i++) {
194     p = p1 + m_dAHS1[i]*pq;
195 
196     sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*dndx.
      interpolation(p, 1.0 - ph/p);
197   }
198 
199   return (sum*pq/static_cast<double>(m_dAMaxPoints1));
```

```
}

double energyLoss::dA412(double ph, const interpolationF<double> &norm, const
    interpolationF<double> &dndx) const {
  double p1 = ph + 2.0*m_mgC / 2.0;
  double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
  double pq = p2 - p1;
  double p;
  double yl, yh, yq, y;
  double sum = 0.0;
  for (size_t i=0; i<m_dAMaxPoints2; i++) {
    p = p1 + m_dAHS1[i]*pq;

    yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yh = 1.0 - ph/p - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yq = yh - yl;
    y = yl + m_dAHS2[i]*yq;

    sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*(1.0 / 2.0)*
    dndx.interpolation(p, 1.0 - ph / p - y)*
      dndx.interpolation(p, y)*(yh - yl);
  }

  return (sum*pq/static_cast<double>(m_dAMaxPoints2));
}

double energyLoss::dA413(double ph, const interpolationF<double> &norm, const
    interpolationF<double> &dndx) const {
  double p1 = ph + 3.0*m_mgC / 2.0;
  double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
  double pq = p2 - p1;
  double p;
  double yl, yh, yq, y;
  double zl, zh, zq, z;
  double sum = 0.0;
  for (size_t i=0; i<m_dAMaxPoints3; i++) {
    p = p1 + m_dAHS1[i]*pq;

    yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yh = 1.0 - ph/p - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yq = yh - yl;
    y = yl + m_dAHS2[i]*yq;

    zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zh = 1.0 - ph/p - y - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zq = zh - zl;
    z = zl + m_dAHS3[i]*zq;

    sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*(1.0 / 2.0 /
    3.0)*dndx.interpolation(p, 1.0 - ph/p - y - z)*
      dndx.interpolation(p, y)*dndx.interpolation(p, z)*(yh - yl)*(zh - zl);
  }

  return (sum*pq/static_cast<double>(m_dAMaxPoints3));
}

double energyLoss::dA414(double ph, const interpolationF<double> &norm, const
    interpolationF<double> &dndx) const {
  double p1 = ph + 4.0*m_mgC / 2.0;
  double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
  double pq = p2 - p1;
  double p;
  double yl, yh, yq, y;
  double zl, zh, zq, z;
  double zzl, zzh, zzq, zz;
  double sum = 0.0;
  for (size_t i=0; i<m_dAMaxPoints4; i++) {
    p = p1 + m_dAHS1[i]*pq;

    yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yh = 1.0 - ph/p - 3.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yq = yh - yl;
```

```
267     y = yl + m_dAHS2[i]*yq;
268
269     zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
270     zh = 1.0 - ph/p - y - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
271     zq = zh - zl;
272     z = zl + m_dAHS3[i]*zq;
273
274     zzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
275     zzh = 1.0 - ph/p - y - z - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
276     zzq = zzh - zzl;
277     zz = zzl + m_dAHS4[i]*zzq;
278
279     sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*(1.0 / 2.0 /
        3.0 / 4.0)*dndx.interpolation(p, 1.0 - ph/p - y - z - zz)*
280       dndx.interpolation(p, y)*dndx.interpolation(p, z)*dndx.interpolation(p, zz)*(yh -
        yl)*(zh - zl)*(zzh - zzl);
281   }
282
283   return (sum*pq/static_cast<double>(m_dAMaxPoints4));
284 }
285
286 double energyLoss::dA415(double ph, const interpolationF<double> &norm, const
      interpolationF<double> &dndx) const {
287   double p1 = ph + 5.0*m_mgC / 2.0;
288   double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
289   double pq = p2 - p1;
290   double p;
291   double yl, yh, yq, y;
292   double zl, zh, zq, z;
293   double zzl, zzh, zzq, zz;
294   double zzzl, zzzh, zzzq, zzz;
295   double sum = 0.0;
296   for (size_t i=0; i<m_dAMaxPoints5; i++) {
297     p = p1 + m_dAHS1[i]*pq;
298
299     yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
300     yh = 1.0 - ph/p - 4.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
301     yq = yh - yl;
302     y = yl + m_dAHS2[i]*yq;
303
304     zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
305     zh = 1.0 - ph/p - y - 3.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
306     zq = zh - zl;
307     z = zl + m_dAHS3[i]*zq;
308
309     zzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
310     zzh = 1.0 - ph/p - y - z - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
311     zzq = zzh - zzl;
312     zz = zzl + m_dAHS4[i]*zzq;
313
314     zzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
315     zzzh = 1.0 - ph/p - y - z - zz - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
316     zzzq = zzzh - zzzl;
317     zzz = zzzl + m_dAHS5[i]*zzzq;
318
319     sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*(1.0 / 2.0 /
        3.0 / 4.0 / 5.0)*dndx.interpolation(p, 1.0 - ph/p - y - z - zz - zzz)*
320       dndx.interpolation(p, y)*dndx.interpolation(p, z)*dndx.interpolation(p, zz)*dndx.
        interpolation(p, zzz)*(yh - yl)*(zh - zl)*(zzh - zzl)*(zzzh - zzzl);
321   }
322
323   return (sum*pq/static_cast<double>(m_dAMaxPoints5));
324 }
325
326 double energyLoss::dA416(double ph, const interpolationF<double> &norm, const
      interpolationF<double> &dndx) const {
327   double p1 = ph + 6.0*m_mgC / 2.0;
328   double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
329   double pq = p2 - p1;
330   double p;
331   double yl, yh, yq, y;
332   double zl, zh, zq, z;
```

```
  double zzl, zzh, zzq, zz;
  double zzzl, zzzh, zzzq, zzz;
  double zzzzl, zzzzh, zzzzq, zzzz;
  double sum = 0.0;
  for (size_t i=0; i<m_dAMaxPoints6; i++) {
    p = p1 + m_dAHS1[i]*pq;

    yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yh = 1.0 - ph/p - 5.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yq = yh - yl;
    y = yl + m_dAHS2[i]*yq;

    zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zh = 1.0 - ph/p - y - 4.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zq = zh - zl;
    z = zl + m_dAHS3[i]*zq;

    zzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzh = 1.0 - ph/p - y - z - 3.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzq = zzh - zzl;
    zz = zzl + m_dAHS4[i]*zzq;

    zzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzzh = 1.0 - ph/p - y - z - zz - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzzq = zzzh - zzzl;
    zzz = zzzl + m_dAHS5[i]*zzzq;

    zzzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzzzh = 1.0 - ph/p - y - z - zz - zzz - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzzzq = zzzzh - zzzzl;
    zzzz = zzzzl + m_dAHS6[i]*zzzzq;

    sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*(1.0 / 2.0 /
    3.0 / 4.0 / 5.0 / 6.0)*dndx.interpolation(p, 1.0 - ph/p - y - z - zz - zzz - zzzz)*
      dndx.interpolation(p, y)*dndx.interpolation(p, z)*dndx.interpolation(p, zz)*dndx.
    interpolation(p, zzz)*dndx.interpolation(p, zzzz)*(yh - yl)*(zh - zl)*
      (zzh - zzl)*(zzzh - zzzl)*(zzzzh - zzzzl);
  }

  return (sum*pq/static_cast<double>(m_dAMaxPoints6));
}

double energyLoss::dA417(double ph, const interpolationF<double> &norm, const
    interpolationF<double> &dndx) const {
  double p1 = ph + 7.0*m_mgC / 2.0;
  double p2 = (((2.0*ph) < (ph + 30.0)) ? (2.0*ph) : (ph + 30.0));
  double pq = p2 - p1;
  double p;
  double yl, yh, yq, y;
  double zl, zh, zq, z;
  double zzl, zzh, zzq, zz;
  double zzzl, zzzh, zzzq, zzz;
  double zzzzl, zzzzh, zzzzq, zzzz;
  double zzzzzl, zzzzzh, zzzzzq, zzzzz;
  double sum = 0.0;
  for (size_t i=0; i<m_dAMaxPoints7; i++) {
    p = p1 + m_dAHS1[i]*pq;

    yl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yh = 1.0 - ph/p - 6.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    yq = yh - yl;
    y = yl + m_dAHS2[i]*yq;

    zl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zh = 1.0 - ph/p - y - 5.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zq = zh - zl;
    z = zl + m_dAHS3[i]*zq;

    zzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzh = 1.0 - ph/p - y - z - 4.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
    zzq = zzh - zzl;
    zz = zzl + m_dAHS4[i]*zzq;
```

```
402
403     zzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
404     zzzh = 1.0 - ph/p - y - z - zz - 3.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
405     zzzq = zzzh - zzzl;
406     zzz = zzzl + m_dAHS5[i]*zzzq;
407
408     zzzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
409     zzzzh = 1.0 - ph/p - y - z - zz - zzz - 2.0*m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
410     zzzzq = zzzzh - zzzzl;
411     zzzz = zzzzl + m_dAHS6[i]*zzzzq;
412
413     zzzzzl = m_mgC/(p + std::sqrt(m_MC*m_MC + p*p));
414     zzzzzh = 1.0 - ph/p - y - z - zz - zzz - zzzz - m_mgC/(p + std::sqrt(m_MC*m_MC + p*p)
        );
415     zzzzzq = zzzzzh - zzzzzl;
416     zzzzz = zzzzzl + m_dAHS7[i]*zzzzzq;
417
418     sum += m_dsdpti2.interpolation(p) / p / std::exp(norm.interpolation(p))*(1.0 / 2.0 /
        3.0 / 4.0 / 5.0 / 6.0 / 7.0)*dndx.interpolation(p, 1.0 - ph/p - y - z - zz - zzz -
        zzzz - zzzzz)*
419       dndx.interpolation(p, y)*dndx.interpolation(p, z)*dndx.interpolation(p, zz)*dndx.
        interpolation(p, zzz)*dndx.interpolation(p, zzzz)*dndx.interpolation(p, zzzzz)*
420       (yh - yl)*(zh - zl)*(zzh - zzl)*(zzzh - zzzl)*(zzzzh - zzzzl)*(zzzzzh - zzzzzl);
421   }
422
423   return (sum*pq/static_cast<double>(m_dAMaxPoints7));
424 }
425
426 double energyLoss::dA41(double ph, interpolationF<double> &currnorm, interpolationF<
        double> &currdndx) const {
427   if (m_pName == "Gluon") { //gluon needs 7 dA integrals
428     return (dA410(ph, currnorm) + dA411(ph, currnorm, currdndx) + dA412(ph, currnorm,
        currdndx) +dA413(ph, currnorm, currdndx) +
429           dA414(ph, currnorm, currdndx) + dA415(ph, currnorm, currdndx) + dA416(ph,
        currnorm, currdndx) +
430           dA417(ph, currnorm, currdndx));
431   }
432   else { //light quarks need 5 dA integrals
433     return (dA410(ph, currnorm) + dA411(ph, currnorm, currdndx) + dA412(ph, currnorm,
        currdndx) + dA413(ph, currnorm, currdndx) +
434           dA414(ph, currnorm, currdndx) + dA415(ph, currnorm, currdndx));
435   }
436 }
```

lTables class' header file, *ltables.hpp* is:

```
1  #ifndef HEADERFILE_LTABLESHEADER
2  #define HEADERFILE_LTABLESHEADER
3
4  #include "grids.hpp"
5
6  #include <string>
7  #include <vector>
8  #include <complex>
9
10 class lTables {
11
12 public:
13     lTables(int argc, const char *argv[]);
14     ~lTables();
15     void runLTables();
16
17 private:
18     bool m_error; //flag that checks if previous calculation is done properly
19
20     std::string m_sNN;       //collision energy
21     std::string m_pName;     //particle name
22     double m_xB;             //xB value
23     size_t m_LdndxMaxPoints; //maximal number of points for Ldndx integration
24     size_t m_LCollMaxPoints; //maximal number of points for collisional integration
25     double m_TCRIT;          //critical temperature
26
27     double m_nf;                    //effective number of flavours
```

```
    const double m_Ng = 3.0;     //effective number of gluons
    const double m_lambda = 0.2;   //QCD scale
    const double m_kmaxColl = 5.0; //kMaxColl value
          double m_CR;           //Casimir (3 for gluons, 4/3 for quakrs)

    gridPoints m_Grids; //grids

    double productLog(double x) const;
    double unitStep(double x) const;
    long double unitStep(long double x) const;

    std::vector<double> m_LdndxHSeq1, m_LdndxHSeq2, m_LdndxHSeq3;
    double haltonSequence(int index, int base) const;
    void LdndxHSeqInit();

    std::vector<std::vector<std::vector<std::vector<double>>>> m_LdndxTbl;
    std::vector<std::vector<std::vector<double>>> m_LNormTbl;
    double dElossDYN(double tau, double p, double x, double k, double q, double varphi,
    double T) const;
    double Ldndx(double tau, double p, double T, double x) const;
    void RadLTables();

    std::vector<double> m_LCollHSeq1, m_LCollHSeq2, m_LCollHSeq3;
    void LCollHSeqInit();

    std::vector<std::vector<double>> m_LCollTbl;
    std::complex<double> deltaL2(double q, double w, double T) const;
    std::complex<double> deltaT2(double q, double w, double T) const;
    double ENumFinite(double p, double T) const;
    void CollLTables();

    int exportLTables() const;

};

#endif
```

lTables class' source file, *ltables.cpp* is:

```
#include "ltables.hpp"
#include "grids.hpp"
#include "polyintegration.hpp"

#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <vector>
#include <map>
#include <cmath>
#include <complex>
#include <iomanip>

lTables::lTables(int argc, const char *argv[])
{
  m_error = false;

  std::vector<std::string> inputs; for (int i=2; i<argc; i++) inputs.push_back(argv[i]);

  if ((inputs.size() == 1) && (inputs[0] == "-h")) {
    std::cout << "default values: --sNN=5020GeV --pName=Charm --xB=0.6 --LdndxMaxPoints
    =500000 --LCollMaxPoints=10000 --TCRIT=0.155" << std::endl;
    m_error = true;
  }

  std::map<std::string, std::string> inputparams;
  for (const auto &in : inputs)
  {
      std::string key = in.substr(0, in.find("="));
      std::string::size_type n = 0; while ((n = key.find("-", n)) != std::string::npos) {
    key.replace(n, 1, ""); n += 0;} //replacing all '-'
    std::string val = in.substr(in.find("=")+1, in.length());
    inputparams[key] = val;
```

```
33   }
34
35   //checking if configuration file is provided:
36   std::map<std::string, std::string> inputparams_f;
37   if (inputparams.count("c") > 0) {
38     std::ifstream file_in(inputparams["c"]);
39     if (!file_in.is_open()) {
40       std::cerr << "Error: unable to open configuration file. Aborting..." << std::endl;
41       m_error = true;
42     }
43     std::string line, key, sep, val;
44     while (std::getline(file_in, line))
45     {
46       std::stringstream ss(line);
47       ss >> key; ss >>sep; ss >> val;
48       inputparams_f[key] = val;
49     }
50     file_in.close();
51   }
52
53   //setting parameter values based on config file values and overwriting with command
       line values:
54   //
55   m_sNN = "5020GeV"; if (inputparams_f.count("sNN") > 0) m_sNN = inputparams_f["sNN"];
56              if (inputparams.count("sNN")    > 0) m_sNN =    inputparams["sNN"];
57
58   m_pName = "Charm"; if (inputparams_f.count("pName") > 0) m_pName = inputparams_f["pName
       "];
59              if (inputparams.count("pName")    > 0) m_pName =    inputparams["pName"];
60
61   m_xB = 0.6; if (inputparams_f.count("xB") > 0) m_xB = stod(inputparams_f["xB"]);
62         if (inputparams.count("xB")    > 0) m_xB = stod(  inputparams["xB"]);
63
64   m_LdndxMaxPoints = 500000; if (inputparams_f.count("LdndxMaxPoints") > 0)
       m_LdndxMaxPoints = stoi(inputparams_f["LdndxMaxPoints"]);
65                   if (  inputparams.count("LdndxMaxPoints") > 0) m_LdndxMaxPoints = stoi
       (  inputparams["LdndxMaxPoints"]);
66
67   m_LCollMaxPoints = 10000; if (inputparams_f.count("LCollMaxPoints") > 0)
       m_LCollMaxPoints = stoi(inputparams_f["LCollMaxPoints"]);
68                  if (inputparams.count("LCollMaxPoints")    > 0) m_LCollMaxPoints = stoi(
         inputparams["LCollMaxPoints"]);
69
70   m_TCRIT = 0.155; if (inputparams_f.count("TCRIT") > 0) m_TCRIT = stod(inputparams_f["
       TCRIT"]);
71            if (  inputparams.count("TCRIT") > 0) m_TCRIT = stod(  inputparams["TCRIT"]);
72
73   //checking if provided value of sNN is an option:
74   if ((m_sNN != "5440GeV") && (m_sNN != "5020GeV") && (m_sNN != "2760GeV") && (m_sNN != "
       200GeV")) {
75     std::cerr << "Error: provided sNN parameter not an option, please try 5440GeV, 5020
       GeV, 2760GeV or 200GeV. Aborting..." << std::endl;
76     m_error = true;
77   }
78
79   m_nf = m_sNN   == "200GeV" ? 2.5 : 3.0;
80   m_CR = m_pName == "Gluon"  ? 3.0 : 4.0/3.0;
81 }
82
83 lTables::~lTables() {}
84
85 void lTables::runLTables()
86 {
87   if (m_error) return;
88
89   m_Grids.setGridPoints(m_sNN, m_pName, m_TCRIT);
90
91     RadLTables();
92
93     CollLTables();
94
95   if (exportLTables() != 1) return;
```

```
}

double lTables::haltonSequence(int index, int base) const
{
  double f = 1.0;
  double res = 0.0;

  while (index > 0) {
    f = f / static_cast<double>(base);
    res += f * static_cast<double>(index % base);
    index = index / base; // integer division
  }

  return res;
}

void lTables::LdndxHSeqInit()
{
  for (size_t i=0; i<m_LdndxMaxPoints; i++) {
    m_LdndxHSeq1.push_back(haltonSequence((i+1)*409, 2));
    m_LdndxHSeq2.push_back(haltonSequence((i+1)*409, 3));
    m_LdndxHSeq3.push_back(haltonSequence((i+1)*409, 5));
  }
}

double lTables::productLog(double x) const
{
  if (x == 0.0) {
    return 0.0;
  }

  double w0, w1;
  if (x > 0.0) {
    w0 = std::log(1.2 * x / std::log(2.4 * x / std::log1p(2.4 * x)));
  }
  else {
    double v = 1.4142135623730950488 * std::sqrt(1.0 + 2.7182818284590452354 * x);
    double N2 = 10.242640687119285146 + 1.9797586132081854940 * v;
    double N1 = 0.29289321881345247560 * (1.4142135623730950488 + N2);
    w0 = -1 + v * (N2 + v) / (N2 + v + N1 * v);
  }

  while (true) {
    double e = std::exp(w0);
    double f = w0 * e - x;
    w1 = w0 - f / ((e * (w0 + 1.0) - (w0 + 2.0) * f / (w0 + w0 + 2.0)));
    if (std::abs(w0 / w1 - 1.0) < 1.4901161193847656e-8) {
      break;
    }
    w0 = w1;
  }
  return w1;
}

double lTables::unitStep(double x) const {
    return (x < 0.0) ? 0.0 : 1.0;
}

long double lTables::unitStep(long double x) const {
    return (x < 0.0L) ? 0.0L : 1.0L;
}

double lTables::dElossDYN(double tau, double p, double x, double k, double q, double
    varphi, double T) const
{
  double mu = 0.197*std::sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
    m_lambda/productLog((-8.0*(6.0 + m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
    m_lambda));
  double mg = mu / std::sqrt(2.0);
  double M = 0.0;
  if (m_pName == "Bottom") M = 4.75;
  else if (m_pName == "Charm") M = 1.2;
```

```
165   else if (m_pName == "Gluon") M = mu/std::sqrt(2.0);
166   else M = mu/std::sqrt(6.0);
167
168   double b = std::sqrt(mg*mg + M * M*x*x);
169   double e = std::sqrt(p*p + M * M);
170   double alpha  = 4.0*M_PI/(11.0 - 2.0*m_nf/3.0)/std::log((k*k + mg*mg + M*M*x*x)/x/
        m_lambda/m_lambda);
171   double alpha1 = 4.0*M_PI/(11.0 - 2.0*m_nf/3.0)/std::log(e*T/0.2/0.2);
172
173   double fn = 1.0;
174   fn *= 1.0 / 0.197*m_CR*alpha/M_PI*3.0*alpha1*T*2.0*k*q/M_PI;
175   fn *= (mu*mu - mu*mu*m_xB*m_xB)/(q*q + mu*mu*m_xB*m_xB)/(q*q + mu*mu);
176
177   double psi = (k*k + q*q + 2.0*k*q*std::cos(varphi) + b*b)/2.0/x/e*tau/0.197;
178
179   fn *= (1 - std::cos(psi));
180   fn *= 2.0/(k*k + b*b)/(k*k + q*q + 2.0*k*q*cos(varphi) + b*b)/(k*k + q*q + 2.0*k*q*std
        ::cos(varphi) + b*b);
181   fn *= (-1.0*k*q*std::cos(varphi)*(k*k + q*q + 2.0*k*q*std::cos(varphi)) + b*b*(k*q*std
        ::cos(varphi) + q*q));
182
183   return fn;
184 }
185
186 double lTables::Ldndx(double tau, double p, double T, double x) const
187 {
188   double mu = 0.197*std::sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
        m_lambda/productLog((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/m_lambda
        ));
189   double mg = mu / std::sqrt(2.0);
190   double M = 0.0;
191   if (m_pName == "Bottom") M = 4.75;
192   else if (m_pName == "Charm") M = 1.2;
193   else if (m_pName == "Gluon") M = mg;
194   else M = mu/sqrt(6.0);
195   double e = sqrt(p*p + M * M);
196
197   double kl = 0.00000001;
198   double kh = 2.0*x*(1 - x)*e;
199   double kq = (kh - kl);
200   double ql = 0.000001;
201   double qh = sqrt(4.0*e*T);
202   double qq = qh - ql;
203   double phil = 0.0;
204   double phih = M_PI;
205   double phiq = (phih - phil);
206   double sum = 0.0; //integration sum
207   double k, q, phi; //integration variables
208
209   #pragma omp parallel for reduction(+:sum) private(k,q,phi)
210   for (size_t i = 0; i<m_LdndxMaxPoints; i++) {
211       k  =   kl + m_LdndxHSeq1[i]*kq;
212       q  =   ql + m_LdndxHSeq2[i]*qq;
213     phi  = phil + m_LdndxHSeq3[i]*phiq;
214     sum += 2*dElossDYN(tau, p, x, k, q, phi, T)/x;
215   }
216
217   return (sum*kq*qq*phiq/static_cast<double>(m_LdndxMaxPoints));
218 }
219
220 void lTables::RadLTables()
221 {
222   LdndxHSeqInit();
223
224   m_LdndxTbl.resize(m_Grids.tauPtsLength(), std::vector<std::vector<std::vector<double
        >>>(m_Grids.pPtsLength(), std::vector<std::vector<double>>(m_Grids.TPtsLength(), std
        ::vector<double>(m_Grids.xPtsLength(), 0.0))));;
225   m_LNormTbl.resize(m_Grids.tauPtsLength(), std::vector<std::vector<double>>(m_Grids.
        pPtsLength(), std::vector<double>(m_Grids.TPtsLength(), 0.0)));
226
227   double tau, p, T, x, mu, M, xIntegLimitLow, xIntegLimitHigh;
228
```

```
229   for (size_t itau=0; itau<m_Grids.tauPtsLength(); itau++) {
230     tau = m_Grids.tauPts(itau);
231 
232     for (size_t ip=0; ip<m_Grids.pPtsLength(); ip++) {
233       p = m_Grids.pPts(ip);
234 
235       for (size_t iT=0; iT<m_Grids.TPtsLength(); iT++) {
236         T = m_Grids.TPts(iT);
237 
238         for (size_t ix=0; ix<m_Grids.xPtsLength(); ix++) {
239           x = m_Grids.xPts(ix);
240           m_LdndxTbl[itau][ip][iT][ix] = Ldndx(tau, p, T, x);
241         }
242 
243         mu = 0.197*std::sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
    m_lambda/productLog((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/m_lambda
    ));
244         if (m_pName == "Bottom") M = 4.75;
245         else if (m_pName == "Charm") M = 1.2;
246         else if (m_pName == "Gluon") M = mu/sqrt(2.0);
247         else M = mu/sqrt(6.0);
248 
249         xIntegLimitLow = mu/std::sqrt(2.0)/(p + std::sqrt(p*p + M*M));
250         if (m_pName == "Gluon") xIntegLimitHigh = 0.5;
251         else xIntegLimitHigh = 1.0 - M/(std::sqrt(p*p + M*M) + p);
252 
253         m_LNormTbl[itau][ip][iT] = poly::cubicIntegrate(m_Grids.xPts(), m_LdndxTbl[itau][
    ip][iT], xIntegLimitLow, xIntegLimitHigh);
254       }
255     }
256   }
257 }
258 
259 void lTables::LCollHSeqInit()
260 {
261   for (size_t i=0; i<m_LCollMaxPoints; i++) {
262     m_LCollHSeq1.push_back(haltonSequence((i+1)*409, 2));
263     m_LCollHSeq2.push_back(haltonSequence((i+1)*409, 3));
264     m_LCollHSeq3.push_back(haltonSequence((i+1)*409, 5));
265   }
266 }
267 
268 std::complex<double> lTables::deltaL2(double q, double w, double T) const
269 {
270   double mu = 0.197*sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
    m_lambda/productLog((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/m_lambda
    ));
271 
272   std::complex<double> q_c = q, w_c = w;
273   std::complex<double> log_c = std::log((q_c + w_c)/(q_c - w_c));
274 
275   std::complex<double> fn = q*q + mu*mu*(1.0 - w/2.0/q*log_c);
276   fn  = fn*fn;
277   fn += (M_PI*M_PI*mu*mu*mu*mu/4.0*w*w/q/q);
278 
279   return (1.0/fn);
280 }
281 
282 std::complex<double> lTables::deltaT2(double q, double w, double T) const
283 {
284   double mu = 0.197*sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
    m_lambda/productLog((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/m_lambda
    ));
285 
286   std::complex<double> q_c = q, w_c = w;
287   std::complex<double> log_c = std::log((q_c + w_c)/(q_c - w_c));
288 
289   std::complex<double> fn = w*w/q/q + w*(q*q - w*w)/2.0/q/q/q*log_c;
290   fn *= (mu*mu/2.0);
291   fn += (q*q - w*w);
292   fn = fn*fn;
293   fn += (M_PI*M_PI*mu*mu*mu*mu/4.0*w*w/q/q*(q*q - w*w)*(q*q - w*w)/4.0/q/q/q/q);
```

```
294
295   return (1.0/fn);
296 }
297
298 double lTables::ENumFinite(double p, double T) const
299 {
300   double mu = 0.197*sqrt((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/
        m_lambda/productLog((-8.0*(6.0+m_nf)*M_PI*M_PI*T*T)/(2.0*m_nf-33.0)/m_lambda/m_lambda
        ));
301   double M = 1.0;
302   if (m_pName == "Bottom") M = 4.75;
303   else if (m_pName == "Charm") M = 1.2;
304   else if (m_pName == "Gluon") M = mu/std::sqrt(2.0);
305   else M = mu/std::sqrt(6.0);
306   double e = std::sqrt(p*p + M * M);
307   double v = p/e;
308   double alpha1 = 4.0*M_PI/(11.0 - 2.0/3.0*m_nf)/std::log(e*T/0.2/0.2);
309   double alpha2 = 2.0*M_PI/(11.0 - 2.0/m_nf*3.0)/std::log(mu/0.2);
310
311   //ENumFinite1 integral:
312   double ENumFiniteSum1 = 0.0;
313
314   double nfCol;
315
316   double k;
317   double kl = 0.0001;
318   double kh = m_kmaxColl;
319   double kq = kh - kl;
320
321   double ql = 0.0001;
322   double qh, qq, q, qmaxCol, qh1, qh2;
323
324   double wl, wh, wq, w;
325
326   #pragma omp parallel for reduction(+:ENumFiniteSum1) private(k,nfCol, qh,qq,q,qmaxCol,
        wl,wh,wq,w)
327   for (size_t i=0; i<m_LCollMaxPoints; i++) {
328     std::complex<double> fn_comp;
329
330     k = kl + m_LCollHSeq1[i]*kq;
331     nfCol = m_Ng/(std::exp(k/T) - 1.0) + m_nf/(std::exp(k/T) + 1.0);
332
333     qmaxCol = std::sqrt(6.0*e*T);
334     qh = ((qmaxCol < k) ? qmaxCol : k);
335     qq = qh - ql;
336     q = ql + m_LCollHSeq2[i]*qq;
337
338     wl = -q;
339     wh = q;
340     wq = wh - wl;
341     w = wl + m_LCollHSeq3[i]*wq;
342
343     fn_comp  = 2.0/0.197*m_CR*alpha1*alpha2/M_PI/v/v*nfCol*w*unitStep(v*v*q*q - w*w);
344     fn_comp *= (deltaL2(q, w, T)*((2.0*k + w)*(2.0*k + w) - q*q)/2.0 + deltaT2(q, w, T)*(
        q*q - w*w)/4.0/q/q/q/q*((2.0*k + w)*(2.0*k + w) + q*q)*(v*v*q*q - w*w));
345
346     ENumFiniteSum1 += fn_comp.real()*qq*wq;
347   }
348
349   ENumFiniteSum1 = ENumFiniteSum1*kq/static_cast<double>(m_LCollMaxPoints);
350
351   //ENumFinite2 integral:
352   double ENumFiniteSum2 = 0.0;
353
354   #pragma omp parallel for reduction(+:ENumFiniteSum2) private(k,nfCol, ql,qh,qh1,qh2,qq,
        q,qmaxCol, wl,wh,wq,w)
355   for (size_t i=0; i<m_LCollMaxPoints; i++) {
356     std::complex<double> fn_comp;
357
358     k = kl + m_LCollHSeq1[i]*kq;
359     nfCol = m_Ng/(std::exp(k/T) - 1.0) + m_nf/(std::exp(k/T) + 1.0);
360
```

```
361     qmaxCol = std::sqrt(6.0*e*T);
362     ql = ((qmaxCol < k) ? qmaxCol : k);
363     qh1 = 2.0*k*(1.0 - k/e)/(1.0 - v + 2.0*k/e);
364     qh2 = ((k > qh1) ? k : qh1);
365     qh = ((qmaxCol < qh2) ? qmaxCol : qh2);
366     qq = qh - ql;
367     q = ql + m_LCollHSeq2[i]*qq;
368 
369     wl = q - 2.0*k;
370     wh = q;
371     wq = wh - wl;
372     w = wl + m_LCollHSeq3[i] * wq;
373 
374     fn_comp  = 2.0/0.197*m_CR*alpha1*alpha2/M_PI/v/v*nfCol*w*unitStep(v*v*q*q - w*w);
375     fn_comp *= (deltaL2(q, w, T)*((2.0*k + w)*(2.0*k + w) - q*q)/2.0 + deltaT2(q, w, T)*(
    q*q - w*w)/4.0/q/q/q/q*((2.0*k + w)*(2.0*k + w) + q*q)*(v*v*q*q - w*w));
376 
377     ENumFiniteSum2 += fn_comp.real()*qq*wq;
378   }
379 
380   ENumFiniteSum2 = ENumFiniteSum2*kq/static_cast<double>(m_LCollMaxPoints);
381 
382   return (ENumFiniteSum1 + ENumFiniteSum2);
383 }
384 
385 void lTables::CollLTables()
386 {
387   LCollHSeqInit();
388 
389   m_LCollTbl.resize(m_Grids.pCollPtsLength(), std::vector<double>(m_Grids.TCollPtsLength
      (), 0.0));
390 
391   for (size_t ip=0; ip<m_Grids.pCollPtsLength(); ip++) {
392     for (size_t iT=0; iT<m_Grids.TCollPtsLength(); iT++) {
393       m_LCollTbl[ip][iT] = ENumFinite(m_Grids.pCollPts(ip), m_Grids.TCollPts(iT));
394     }
395   }
396 }
397 
398 int lTables::exportLTables() const
399 {
400   std::stringstream xBss; xBss << std::fixed << std::setprecision(1) << m_xB;
401   std::stringstream nfss; nfss << std::fixed << std::setprecision(1) << m_nf;
402 
403   {//exporting Ldndx table
404     const std::string path_out = "./ltables/ldndx_nf=" + nfss.str() + "_" + m_pName + "
      _xB=" + xBss.str() + ".dat";
405     std::ofstream file_out(path_out, std::ios_base::out);
406     if (!file_out.is_open()) {
407       std::cerr << "Error: unable to open Ldndx table export file." << std::endl;
408       return -1;
409     }
410 
411     file_out << "#";
412     file_out << std::fixed << std::setw(12) <<   "tau" << " ";
413     file_out << std::fixed << std::setw(14) <<     "p" << " ";
414     file_out << std::fixed << std::setw(12) <<     "T" << " ";
415     file_out << std::fixed << std::setw(12) <<     "x" << " ";
416     file_out << std::fixed << std::setw(17) << "Ldndx" << "\n";
417 
418     for (size_t itau=0; itau<m_Grids.tauPtsLength(); itau++) {
419       for (size_t ip=0; ip<m_Grids.pPtsLength(); ip++) {
420         for (size_t iT=0; iT<m_Grids.TPtsLength(); iT++) {
421           for (size_t ix=0; ix<m_Grids.xPtsLength(); ix++) {
422             file_out << std::fixed    << std::setw(13) << std::setprecision(10) <<
    m_Grids.tauPts(itau) << " ";
423             file_out << std::fixed    << std::setw(14) << std::setprecision(10) <<
    m_Grids.pPts(ip) << " ";
424             file_out << std::fixed    << std::setw(12) << std::setprecision(10) <<
    m_Grids.TPts(iT) << " ";
425             file_out << std::fixed    << std::setw(12) << std::setprecision(10) <<
    m_Grids.xPts(ix) << " ";
```

```
426             file_out << std::scientific << std::setw(17) << std::setprecision(10) <<
    m_LdndxTbl[itau][ip][iT][ix] << "\n";
427           }
428         }
429       }
430     }
431
432     file_out.close();
433   }
434
435   {//exporting LNorm table
436     const std::string path_out = "./ltables/lnorm_nf=" + nfss.str() + "_" + m_pName + "
    _xB=" + xBss.str() + ".dat";
437     std::ofstream file_out(path_out, std::ios_base::out);
438     if (!file_out.is_open()) {
439       std::cerr << "Error: unable to open LNorm table export file." << std::endl;
440       return -2;
441     }
442
443     file_out << "#";
444     file_out << std::fixed << std::setw(12) <<   "tau" << " ";
445     file_out << std::fixed << std::setw(14) <<     "p" << " ";
446     file_out << std::fixed << std::setw(12) <<     "T" << " ";
447     file_out << std::fixed << std::setw(17) << "LNorm" << "\n";
448
449     for (size_t itau=0; itau<m_Grids.tauPtsLength(); itau++) {
450       for (size_t ip=0; ip<m_Grids.pPtsLength(); ip++) {
451         for (size_t iT=0; iT<m_Grids.TPtsLength(); iT++) {
452           file_out << std::fixed    << std::setw(13) << std::setprecision(10) << m_Grids.
    tauPts(itau) << " ";
453           file_out << std::fixed    << std::setw(14) << std::setprecision(10) << m_Grids.
    pPts(ip) << " ";
454           file_out << std::fixed    << std::setw(12) << std::setprecision(10) << m_Grids.
    TPts(iT) << " ";
455           file_out << std::scientific << std::setw(17) << std::setprecision(10) <<
    m_LNormTbl[itau][ip][iT] << "\n";
456         }
457       }
458     }
459
460     file_out.close();
461   }
462
463   {//exporting LColl table
464     std::string path_out = "./ltables/lcoll_nf=" + nfss.str() + "_" + m_pName + ".dat";
465     std::ofstream file_out(path_out, std::ios_base::out);
466     if (!file_out.is_open()) {
467       std::cerr << "Error: unable to open LColl table export file." << std::endl;
468       return -3;
469     }
470
471     file_out << "#";
472     file_out << std::fixed << std::setw(13) <<     "p" << " ";
473     file_out << std::fixed << std::setw(12) <<     "T" << " ";
474     file_out << std::fixed << std::setw(17) << "LColl" << "\n";
475
476     for (size_t ip=0; ip<m_Grids.pCollPtsLength(); ip++) {
477       for (size_t iT=0; iT<m_Grids.TCollPtsLength(); iT++) {
478         file_out << std::fixed    << std::setw(14) << std::setprecision(10) << m_Grids.
    pCollPts(ip) << " ";
479         file_out << std::fixed    << std::setw(12) << std::setprecision(10) << m_Grids.
    TCollPts(iT) << " ";
480         file_out << std::scientific << std::setw(17) << std::setprecision(10) <<
    m_LCollTbl[ip][iT] << "\n";
481       }
482     }
483
484     file_out.close();
485   }
486
487   return 1;
488 }
```

All the calculations in DREENA-A are done on grids - equidistant or non-uniform sequences of points of variables sush as momentum, p, proper time, τ, temperature, T, and similar. With this in mind, another class, *grids* is introduced. Its header file, *grids.hpp* follows:

```
#ifndef HEADERFILE_GRIDPOINTS
#define HEADERFILE_GRIDPOINTS

#include <vector>
#include <string>

class gridPoints {

//public functions:
public:

  //CONSTRUCTORS:
  gridPoints();
  gridPoints(const std::string &sNN, const std::string &particleName, double tcrit);
  void setGridPoints(const std::string &sNN, const std::string &particleName, double
    tcrit);

  //DESTRUCTOR:
  ~gridPoints();

  //GRID FUNCTIONS:
  const std::vector<double> & tauPts() const;
  double tauPts(int i) const;
  size_t tauPtsLength() const;

  const std::vector<double> & pPts() const;
  double pPts(int i) const;
  size_t pPtsLength() const;

  const std::vector<double> & xPts() const;
  double xPts(int i) const;
  size_t xPtsLength() const;

  const std::vector<double> & TPts() const;
  double TPts(int i) const;
  size_t TPtsLength() const;

  const std::vector<double> & FdpPts() const;
  double FdpPts(int i) const;
  size_t FdpPtsLength() const;

  const std::vector<double> & RadPts() const;
  double RadPts(int i) const;
  size_t RadPtsLength() const;

  const std::vector<double> & pCollPts() const;
  double pCollPts(int i) const;
  size_t pCollPtsLength() const;

  const std::vector<double> & TCollPts() const;
  double TCollPts(int i) const;
  size_t TCollPtsLength() const;

  const std::vector<double> & finPts() const;
  double finPts(int i) const;
  size_t finPtsLength() const;

//private variables and functions:
private:

  double m_nf     = 3.0;
  double m_lambda = 0.2;
  double m_TCRIT  = 0.155;
  double productLog(double x);
  double muF(double temp);
  std::vector<double> m_tauPts, m_pPts, m_TPts, m_xPts, m_RadPts, m_FdpPts;
  std::vector<double> m_pCollPts, m_TCollPts, m_finPts;
  double linearIntegrate(const std::vector<double> &dataX, const std::vector<double> &
```

```
    dataF, double xH) const;
  void generateGrids(const std::vector<std::vector<double>> &density, size_t numpts, std
    ::vector<double> &gridpoints);
};

#endif
```

grids class source file is:

```
#include "grids.hpp"
#include "linearinterpolation.hpp"

#include <iostream>
#include <vector>
#include <string>
#include <cmath>

gridPoints::gridPoints() {}

gridPoints::gridPoints(const std::string &sNN, const std::string &particleName, double
    tcrit)
{
  setGridPoints(sNN, particleName, tcrit);
}

void gridPoints::setGridPoints(const std::string &sNN, const std::string &particleName,
    double tcrit)
{
  //setting nf value based on sNN (default value is 3.0 <-> LHC):
  if (sNN == "200GeV") m_nf = 2.5;
  //setting the value of critical temperature:
  m_TCRIT = tcrit;

  if (particleName == "Bottom") {

    //tauPts:
    size_t taugridn = 21;
    std::vector<std::vector<double>> tauden{{0.0, 10.0}, {20.0, 10.0}};
    generateGrids(tauden, taugridn, m_tauPts);

    //pPts:
    size_t pgridn = 25;
    double pgridmax = sNN == "200GeV" ? 100.0 : 200.0;
    std::vector<std::vector<double>> pden{{1.0, 8.0}, {20.0, 7.0}, {30.0, 3.0}, {60.0,
    5.0}, {pgridmax, 1.0}};
    generateGrids(pden, pgridn, m_pPts);

    //TPts:
    size_t Tgridn = 40;
    std::vector<std::vector<double>> Tden{{0.01, 10.0}, {2.0, 10.0}};
    generateGrids(Tden, Tgridn, m_TPts);

    //xPts:
    double mg = muF(m_TPts[0])/std::sqrt(2.0);
    double M = 4.75;
    double MAXP = sNN == "200GeV" ? 100.0 : 200.0;
    size_t xgridn = 30;
    double xmin = mg/(MAXP + std::sqrt(MAXP*MAXP + M*M));
    for (size_t i=0; i<xgridn; i++)
      m_xPts.push_back(std::exp(std::log(xmin) - std::log(xmin)/static_cast<double>(
    xgridn-1)*static_cast<double>(i)));

    //RadPts:
    size_t Radgridn = 20;
    double Radgridmax = sNN == "200GeV" ? 70.0 : 170.0;
    std::vector<std::vector<double>> Radden{{2.0, 10.0}, {21.8, 10.0}, {44.5, 1.050001},
    {Radgridmax, 1.0}};
    generateGrids(Radden, Radgridn, m_RadPts);

    //FdpPts:
    double mgC = muF(3.0/2.0*m_TCRIT)/std::sqrt(2.0);
    size_t Fdpgridn = 16;
```

```
    std::vector<std::vector<double>> Fdpden = {{5.0*mgC/2.0, 10.0}, {12.0, 5.0}, {30.0, 0.0}};
    generateGrids(Fdpden, Fdpgridn-4, m_FdpPts);
    m_FdpPts.insert(m_FdpPts.begin(), 4.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 3.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 2.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 1.0*mgC/2.0);

    //pCollPts:
    size_t pCollgridn = 20;
    double pCollgridmax = sNN == "200GeV" ? 70.0 : 170.0;
    std::vector<std::vector<double>> pCollden{{1.0, 10.0}, {4.0, 10.0}, {9.0, 2.5}, {30.0, 0.6}, {60.0, 0.5}, {pCollgridmax, 0.3}};
    generateGrids(pCollden, pCollgridn, m_pCollPts);

    //TCollPts:
    size_t TCollgridn = 40;
    std::vector<std::vector<double>> TCollden{{0.01, 10.0}, {2.0, 10.0}};
    generateGrids(TCollden, TCollgridn, m_TCollPts);

    //finpts:
    size_t fingridn = 30;
    double fingridmax = sNN == "200GeV" ? 50.0 : 150.0;
    std::vector<std::vector<double>> finden{{5.0, 10.0}, {50.0, 10.0}, {70.0, 5.0}, {fingridmax, 3.0}};
    generateGrids(finden, fingridn, m_finPts);
  }
  else if (particleName == "Charm") {

    //tauPts:
    size_t taugridn = 21;
    std::vector<std::vector<double>> tauden{{0.0, 10.0}, {5.0, 10.0}, {10.0, 10.0}, {15.0, 10.0}, {20.0, 10.0}};
    generateGrids(tauden, taugridn, m_tauPts);

    //pPts:
    size_t pgridn = 25;
    double pgridmax = sNN == "200GeV" ? 100.0 : 200.0;
    std::vector<std::vector<double>> pden{{1.0, 8.0}, {20.0, 7.0}, {30.0, 3.0}, {60.0, 5.0}, {pgridmax, 1.0}};
    generateGrids(pden, pgridn, m_pPts);

    //TPts:
    size_t Tgridn = 40;
    std::vector<std::vector<double>> Tden{{0.01, 10.0}, {2.0, 10.0}};
    generateGrids(Tden, Tgridn, m_TPts);

    //xPts:
    double mg = muF(m_TPts[0])/std::sqrt(2.0);
    double M = 1.2;
    double MAXP = sNN == "200GeV" ? 100.0 : 200.0;
    size_t xgridn = 30;
    double xmin = mg/(MAXP + std::sqrt(MAXP*MAXP + M*M));
    for (size_t i=0; i<xgridn; i++)
            m_xPts.push_back(std::exp(std::log(xmin) - std::log(xmin)/static_cast<double>(xgridn-1)*static_cast<double>(i)));

    //RadPts:
    size_t Radgridn = 20;
    double Radgridmax = sNN == "200GeV" ? 70.0 : 170.0;
    std::vector<std::vector<double>> Radden{{2.0, 10.0}, {21.8, 10.0}, {44.5, 1.05}, {Radgridmax, 1.0}};
    generateGrids(Radden, Radgridn, m_RadPts);

    //FdpPts:
    double mgC = muF(3.0/2.0*m_TCRIT)/std::sqrt(2.0);
    size_t Fdpgridn = 16;
    std::vector<std::vector<double>> Fdpden = {{5.0*mgC/2.0, 10.0}, {12.0, 5.0}, {30.0, 0.0}};
    generateGrids(Fdpden, Fdpgridn-4, m_FdpPts);
    m_FdpPts.insert(m_FdpPts.begin(), 4.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 3.0*mgC/2.0);
```

```
123     m_FdpPts.insert(m_FdpPts.begin(), 2.0*mgC/2.0);
124     m_FdpPts.insert(m_FdpPts.begin(), 1.0*mgC/2.0);
125
126     //pCollPts:
127     size_t pCollgridn = 20;
128     double pCollgridmax = sNN == "200GeV" ? 70.0 : 170.0;
129     std::vector<std::vector<double>> pCollden{{1.0, 10.0}, {4.0, 10.0}, {9.0, 2.5},
        {30.0, 0.6}, {60.0, 0.5}, {pCollgridmax, 0.3}};
130     generateGrids(pCollden, pCollgridn, m_pCollPts);
131
132     //TCollPts:
133     size_t TCollgridn = 40;
134     std::vector<std::vector<double>> TCollden{{0.01, 10.0}, {2.0, 10.0}};
135     generateGrids(TCollden, TCollgridn, m_TCollPts);
136
137     //finpts:
138     size_t fingridn = 30;
139     double fingridmax = sNN == "200GeV" ? 50.0 : 150.0;
140     std::vector<std::vector<double>> finden{{5.0, 10.0}, {50.0, 10.0}, {70.0, 5.0}, {
        fingridmax, 3.0}};
141     generateGrids(finden, fingridn, m_finPts);
142
143   }
144   else if (particleName == "Gluon") {
145
146     //tauPts:
147     size_t taugridn = 21;
148     std::vector<std::vector<double>> tauden{{0.0, 10.0}, {20.0, 10.0}};
149     generateGrids(tauden, taugridn, m_tauPts);
150
151     //pPts:
152     size_t    pgridn = sNN == "200GeV" ?    35 :    50;
153     double  pgridmax = sNN == "200GeV" ? 150.0 : 450.0;
154     double pgridmaxw = sNN == "200GeV" ?   0.5 :   1.0;
155     std::vector<std::vector<double>> pden{{1.0, 8.0}, {20.0, 7.0}, {40.0, 3.0}, {100.0,
        5.0}, {pgridmax, pgridmaxw}};
156     generateGrids(pden, pgridn, m_pPts);
157
158     //TPts:
159     size_t Tgridn = 40;
160     std::vector<std::vector<double>> Tden{{0.01, 10.0}, {2.0, 10.0}};
161     generateGrids(Tden, Tgridn, m_TPts);
162
163     //xPts:
164     double mg = muF(m_TPts[0])/std::sqrt(2.0);
165     double  M = muF(m_TPts[0])/std::sqrt(2.0);
166     double MAXP = sNN == "200GeV" ? 150.0 : 450.0;
167     size_t xgridn = 50;
168     double xmin = mg/(MAXP + std::sqrt(MAXP*MAXP + M*M));
169     for (size_t i=0; i<xgridn; i++)
170             m_xPts.push_back(std::exp(std::log(xmin) - std::log(xmin)/static_cast<double
        >(xgridn-1)*static_cast<double>(i)));
171
172
173     //RadPts:
174     size_t   Radgridn = sNN == "200GeV" ?    30 :    40;
175     double Radgridmax = sNN == "200GeV" ? 120.0 : 420.0;
176     std::vector<std::vector<double>> Radden{{2.0, 10.0}, {50.0, 10.0}, {70.0, 1.0}, {
        Radgridmax, 1.0}};
177     generateGrids(Radden, Radgridn, m_RadPts);
178
179     //FdpPts:
180     double mgC = muF(3.0/2.0*m_TCRIT)/std::sqrt(2.0);
181     size_t Fdpgridn = 22;
182     std::vector<std::vector<double>> Fdpden = {{5.0*mgC/2.0, 10.0}, {12.0, 5.0}, {30.0,
        0.0}};
183     generateGrids(Fdpden, Fdpgridn-4, m_FdpPts);
184     m_FdpPts.insert(m_FdpPts.begin(), 4.0*mgC/2.0);
185     m_FdpPts.insert(m_FdpPts.begin(), 3.0*mgC/2.0);
186     m_FdpPts.insert(m_FdpPts.begin(), 2.0*mgC/2.0);
187     m_FdpPts.insert(m_FdpPts.begin(), 1.0*mgC/2.0);
188
```

```
    //pCollPts:
    size_t   pCollgridn = sNN == "200GeV" ?    30 :    40;
    double pCollgridmax = sNN == "200GeV" ? 120.0 : 420.0;
    std::vector<std::vector<double>> pCollden{{1.0, 10.0}, {4.0, 10.0}, {9.0, 2.5},
    {30.0, 0.6}, {60.0, 0.5}, {pCollgridmax, 0.3}};
    generateGrids(pCollden, pCollgridn, m_pCollPts);

    //TCollPts:
    size_t TCollgridn = 40;
    std::vector<std::vector<double>> TCollden{{0.01, 10.0}, {2.0, 10.0}};
    generateGrids(TCollden, TCollgridn, m_TCollPts);

    //finpts:
    size_t   fingridn = sNN == "200GeV" ?    35 :    50;
    double fingridmax = sNN == "200GeV" ? 100.0 : 400.0;
    std::vector<std::vector<double>> finden{{5.0, 10.0}, {50.0, 10.0}, {70.0, 5.0}, {
    fingridmax, 3.0}};
    generateGrids(finden, fingridn, m_finPts);
  }
  else {

    //tauPts:
    size_t taugridn = 21;
    std::vector<std::vector<double>> tauden{{0.0, 10.0}, {20.0, 10.0}};
    generateGrids(tauden, taugridn, m_tauPts);

    //pPts:
    size_t    pgridn = sNN == "200GeV" ?    35 :    50;
    double  pgridmax = sNN == "200GeV" ? 150.0 : 450.0;
    double pgridmaxw = sNN == "200GeV" ?   0.5 :   1.0;
    std::vector<std::vector<double>> pden{{1.0, 8.0}, {20.0, 7.0}, {40.0, 3.0}, {100.0,
    5.0}, {pgridmax, pgridmaxw}};
    generateGrids(pden, pgridn, m_pPts);

    //TPts:
    size_t Tgridn = 40;
    std::vector<std::vector<double>> Tden{{0.01, 10.0}, {2.0, 10.0}};
    generateGrids(Tden, Tgridn, m_TPts);

    //xPts:
    double mg = muF(m_TPts[0])/std::sqrt(2.0);
    double M  = muF(m_TPts[0])/std::sqrt(6.0);
    double MAXP = sNN == "200GeV" ? 150.0 : 450.0;
    size_t xgridn = 50;
    double xmin = mg/(MAXP + std::sqrt(MAXP*MAXP + M*M));
    for (size_t i=0; i<xgridn; i++)
            m_xPts.push_back(std::exp(std::log(xmin) - std::log(xmin)/static_cast<double
    >(xgridn-1)*static_cast<double>(i)));

    //RadPts:
    size_t   Radgridn = sNN == "200GeV" ?    30 :    40;
    double Radgridmax = sNN == "200GeV" ? 120.0 : 420.0;
    std::vector<std::vector<double>> Radden{{2.0, 10.0}, {50.0, 10.0}, {70.0, 1.0}, {
    Radgridmax, 1.0}};
    generateGrids(Radden, Radgridn, m_RadPts);

    //FdpPts:
    double mgC = muF(3.0/2.0*m_TCRIT)/std::sqrt(2.0);
    size_t Fdpgridn = 22;
    std::vector<std::vector<double>> Fdpden = {{5.0*mgC/2.0, 10.0}, {12.0, 5.0}, {30,
    0.0}};
    generateGrids(Fdpden, Fdpgridn-4, m_FdpPts);
    m_FdpPts.insert(m_FdpPts.begin(), 4.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 3.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 2.0*mgC/2.0);
    m_FdpPts.insert(m_FdpPts.begin(), 1.0*mgC/2.0);

    //pCollPts:
    size_t   pCollgridn = sNN == "200GeV" ?    30 :    40;
    double pCollgridmax = sNN == "200GeV" ? 120.0 : 420.0;
    std::vector<std::vector<double>> pCollden{{1.0, 10.0}, {4.0, 10.0}, {9.0, 2.5},
    {30.0, 0.6}, {60.0, 0.5}, {pCollgridmax, 0.3}};
```

```
254     generateGrids(pCollden, pCollgridn, m_pCollPts);
255
256     //TCollPts:
257     size_t TCollgridn = 40;
258     std::vector<std::vector<double>> TCollden{{0.01, 10.0}, {2.0, 10.0}};
259     generateGrids(TCollden, TCollgridn, m_TCollPts);
260
261     //finpts:
262     size_t   fingridn = sNN == "200GeV" ?    35 :    50;
263     double fingridmax = sNN == "200GeV" ? 100.0 : 400.0;
264     std::vector<std::vector<double>> finden{{5.0, 10.0}, {50.0, 10.0}, {70.0, 5.0}, {
        fingridmax, 3.0}};
265     generateGrids(finden, fingridn, m_finPts);
266   }
267
268   //rounding grids to 10 decimal points:
269   for (size_t i=0; i<m_tauPts.size(); i++) m_tauPts[i] = std::round(m_tauPts[i]*1e10)/1
        e10;
270   for (size_t i=0; i<m_pPts.size();   i++)   m_pPts[i] = std::round(m_pPts[i]*1e10)/1e10;
271   for (size_t i=0; i<m_TPts.size();   i++)   m_TPts[i] = std::round(m_TPts[i]*1e10)/1e10;
272   for (size_t i=0; i<m_xPts.size();   i++)   m_xPts[i] = std::round(m_xPts[i]*1e10)/1e10;
273   for (size_t i=0; i<m_RadPts.size(); i++) m_RadPts[i] = std::round(m_RadPts[i]*1e10)/1
        e10;
274   for (size_t i=0; i<m_FdpPts.size(); i++) m_FdpPts[i] = std::round(m_FdpPts[i]*1e10)/1
        e10;
275
276   for (size_t i=0; i<m_pCollPts.size(); i++) m_pCollPts[i] = std::round(m_pCollPts[i]*1
        e10)/1e10;
277   for (size_t i=0; i<m_TCollPts.size(); i++) m_TCollPts[i] = std::round(m_TCollPts[i]*1
        e10)/1e10;
278
279   for (size_t i=0; i<m_finPts.size(); i++) m_finPts[i] = std::round(m_finPts[i]*1e10)/1
        e10;
280 }
281
282 gridPoints::~gridPoints() {}
283
284 const std::vector<double> & gridPoints::tauPts() const {
285     return m_tauPts;
286 }
287 double gridPoints::tauPts(int i) const {
288     if (i < 0)
289         return m_tauPts.at(m_tauPts.size() + i);
290     return m_tauPts.at(i);
291 }
292 size_t gridPoints::tauPtsLength() const {
293     return m_tauPts.size();
294 }
295
296 const std::vector<double> & gridPoints::pPts() const {
297     return m_pPts;
298 }
299 double gridPoints::pPts(int i) const {
300     if (i < 0) return m_pPts.at(m_pPts.size() + i);
301     return m_pPts.at(i);
302 }
303 size_t gridPoints::pPtsLength() const {
304     return m_pPts.size();
305 }
306
307 const std::vector<double> & gridPoints::TPts() const {
308     return m_TPts;
309 }
310 double gridPoints::TPts(int i) const {
311     if (i < 0)
312         return m_TPts.at(m_TPts.size() + i);
313     return m_TPts.at(i);
314 }
315 size_t gridPoints::TPtsLength() const {
316     return m_TPts.size();
317 }
318
```

```
const std::vector<double> & gridPoints::xPts() const {
    return m_xPts;
}
double gridPoints::xPts(int i) const {
    if (i < 0)
        return m_xPts.at(m_xPts.size() + i);
    return m_xPts.at(i);
}
size_t gridPoints::xPtsLength() const {
    return m_xPts.size();
}

const std::vector<double> & gridPoints::RadPts() const {
    return m_RadPts;
}
double gridPoints::RadPts(int i) const {
    if (i < 0)
        return m_RadPts.at(m_RadPts.size() + i);
    return m_RadPts.at(i);
}
size_t gridPoints::RadPtsLength() const {
    return m_RadPts.size();
}

const std::vector<double> & gridPoints::FdpPts() const {
    return m_FdpPts;
}
double gridPoints::FdpPts(int i) const {
    if (i < 0)
        return m_FdpPts.at(m_FdpPts.size() + i);
    return m_FdpPts.at(i);
}
size_t gridPoints::FdpPtsLength() const {
    return m_FdpPts.size();
}

const std::vector<double> & gridPoints::pCollPts() const {
    return m_pCollPts;
}
double gridPoints::pCollPts(int i) const {
    if (i < 0)
        return m_pCollPts.at(m_pCollPts.size() + i);
    return m_pCollPts.at(i);
}
size_t gridPoints::pCollPtsLength() const {
    return m_pCollPts.size();
}

const std::vector<double> & gridPoints::TCollPts() const {
    return m_TCollPts;
}
double gridPoints::TCollPts(int i) const {
    if (i < 0)
        return m_TCollPts.at(m_TCollPts.size() + i);
    return m_TCollPts.at(i);
}
size_t gridPoints::TCollPtsLength() const {
    return m_TCollPts.size();
}

const std::vector<double> & gridPoints::finPts() const {
    return m_finPts;
}
double gridPoints::finPts(int i) const {
    if (i < 0)
        return m_finPts.at(m_finPts.size() + i);
    return m_finPts.at(i);
}
size_t gridPoints::finPtsLength() const {
    return m_finPts.size();
}

```

```
391 double gridPoints::productLog(double x)
392 {
393   if (x == 0.0) {
394     return 0.0;
395   }
396
397   double w0, w1;
398   if (x > 0.0) {
399     w0 = std::log(1.2 * x / std::log(2.4 * x / std::log1p(2.4 * x)));
400   }
401   else {
402     double v = 1.4142135623730950488 * std::sqrt(1.0 + 2.7182818284590452354 * x);
403     double N2 = 10.242640687119285146 + 1.9797586132081854940 * v;
404     double N1 = 0.29289321881345247560 * (1.4142135623730950488 + N2);
405     w0 = -1 + v * (N2 + v) / (N2 + v + N1 * v);
406   }
407
408   while (true) {
409     double e = std::exp(w0);
410     double f = w0 * e - x;
411     w1 = w0 - f / ((e * (w0 + 1.0) - (w0 + 2.0) * f / (w0 + w0 + 2.0)));
412     if (std::abs(w0 / w1 - 1.0) < 1.4901161193847656e-8) {
413       break;
414     }
415     w0 = w1;
416   }
417   return w1;
418 }
419
420 double gridPoints::muF(double temp)
421 {
422   return (0.197*sqrt((-8.0*(6.0 + m_nf)*M_PI*M_PI*temp*temp)/(2.0*m_nf - 33.0)/m_lambda/
      m_lambda/productLog((-8.0*(6.0 + m_nf)*M_PI*M_PI*temp*temp)/(2.0*m_nf - 33.0)/
      m_lambda/m_lambda)));
423 }
424
425 double gridPoints::linearIntegrate(const std::vector<double> &dataX, const std::vector<
      double> &dataF, double xH) const
426 {
427   std::vector<double> k, c;
428   for (size_t i=0; i<(dataX.size()-1); i++)
429   {
430     k.push_back((dataF[i+1]-dataF[i])/(dataX[i+1]-dataX[i]));
431     c.push_back(dataF[i]-k.back()*dataX[i]);
432   }
433
434   int xHi = 0; while (xH > dataX[xHi]) xHi++; xHi--;
435
436   double sum = 0.0;
437
438   for (int i=0; i<xHi; i++)
439   {
440     sum += 0.5*k[i]*(dataX[i+1]*dataX[i+1] - dataX[i]*dataX[i]) + c[i]*(dataX[i+1] -
      dataX[i]);
441   }
442
443   sum += 0.5*k[xHi]*(xH*xH - dataX[xHi]*dataX[xHi]) + c[xHi]*(xH - dataX[xHi]);
444
445   return sum;
446 }
447
448 void gridPoints::generateGrids(const std::vector<std::vector<double>> &density, size_t
      numpts, std::vector<double> &gridpoints)
449 {
450   std::vector<double> densityX, densityF;
451   for (size_t i=0; i<density.size(); i++) {densityX.push_back(density[i][0]); densityF.
      push_back(density[i][1]);}
452
453   std::vector<double> inttabX, inttabF;
454
455   double xxx = densityX.front();
456   inttabX.push_back(0.0);
```

```
457   inttabF.push_back(xxx);
458
459   for (size_t i=1; i<19; i++)
460   {
461     xxx = densityX.front() + (densityX.back()-densityX.front())/static_cast<double>(19)*
      static_cast<double>(i);
462     inttabX.push_back(linearIntegrate(densityX, densityF, xxx));
463     inttabF.push_back(xxx);
464   }
465
466   xxx = densityX.back();
467   inttabX.push_back(linearIntegrate(densityX, densityF, xxx));
468   inttabF.push_back(xxx);
469
470   interpolationF<double> inttabInt(inttabX, inttabF);
471
472   gridpoints.resize(0);
473
474   gridpoints.push_back(densityX.front());
475
476   for (size_t i=1; i<numpts-1; i++)
477   {
478     double a = inttabX.front() + (inttabX.back()-inttabX.front())*static_cast<double>(i)/
      static_cast<double>(numpts-1);
479     gridpoints.push_back(inttabInt.interpolation(a));
480   }
481
482   gridpoints.push_back(densityX.back());
483 }
```

Used throughout the code is stand-alone linear interpolation class, *interpolationF*. This class performs linear interpolation of tabular functions given on the ordered grid and is templated for floar, double and long double types. It's header file, *linearinterpolation.hpp* is:

```
1  #ifndef HEADERFILE_LINEARINTERPOLATION
2  #define HEADERFILE_LINEARINTERPOLATION
3
4  #include <vector>
5
6  template<typename T>
7  class interpolationF {
8  public:
9    //CONSTRUCTORS:
10   interpolationF();
11
12   //input is 2 1D arrays:
13   interpolationF(const T *xData, const T *fData, size_t NofElements);
14   void setData(const T *xData, const T *fData, size_t NofElements);
15
16   //input is 2 1D vectors:
17   interpolationF(const std::vector<T> &xData, const std::vector<T> &fData);
18   void setData(const std::vector<T> &xData, const std::vector<T> &fData);
19
20   //input is 3 1D arrays:
21   interpolationF(const T *x1Data, const T *x2Data, const T *fData, size_t NofElements);
22   void setData(const T *x1Data, const T *x2Data, const T *fData, size_t NofElements);
23
24   //input is 3 1D vectors:
25   interpolationF(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
       vector<T> &fData);
26   void setData(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
       vector<T> &fData);
27
28   //input is 2 1D vectors (grids) and 1 2d vector (function values):
29   interpolationF(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
       vector<std::vector<T>> &fData);
30   void setData(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
       vector<std::vector<T>> &fData);
31
32   //input is 4 1D arrays:
33   interpolationF(const T *x1Data, const T *x2Data, const T *x3Data, const T *fData,
       size_t NofElements);
```

```
34    void setData(const T *x1Data, const T *x2Data, const T *x3Data, const T *fData, size_t
        NofElements);
35
36    //input is 4 1D vectors:
37    interpolationF(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
        vector<T> &x3Data, const std::vector<T> &fData);
38    void setData(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
        vector<T> &x3Data, const std::vector<T> &fData);
39
40    //input is 5 1D arrays:
41    interpolationF(const T *x1Data, const T *x2Data, const T *x3Data, const T *x4Data,
        const T *fData, size_t NofElements);
42    void setData(const T *x1Data, const T *x2Data, const T *x3Data, const T *x4Data, const
        T *fData, size_t NofElements);
43
44    //input is 5 1D vectors:
45    interpolationF(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
        vector<T> &x3Data, const std::vector<T> &x4Data, const std::vector<T> &fData);
46    void setData(const std::vector<T> &x1Data, const std::vector<T> &x2Data, const std::
        vector<T> &x3Data, const std::vector<T> &x4Data, const std::vector<T> &fData);
47
48    //DESTRUCTOR:
49    ~interpolationF();
50
51    //INTERPOLATION FUNCTIONS:
52    //1D interpolation
53    T interpolation(T pointValue) const;
54
55    //2D interpolation
56    T interpolation(T pointValue1, T pointValue2) const;
57
58    //3D interpolation
59    T interpolation(T pointValue1, T pointValue2, T pointValue3) const;
60
61    //4D interpolation
62    T interpolation(T pointValue1, T pointValue2, T pointValue3, T pointValue4) const;
63
64    //miscellaneous FUNCTIONS:
65    //function that returns domains:
66    const std::vector<std::vector<T>> & domain() const;
67
68    //function that returns codomain:
69    const std::vector<T> & codomain() const;
70
71  private:
72    size_t m_dataLength;
73    std::vector<std::vector<T>> m_data;
74    size_t m_variableN;
75    std::vector<size_t> m_gridLengths;
76    std::vector<size_t> m_relPosition;
77    std::vector<std::vector<T>> m_domain;
78    std::vector<T> m_codomain;
79
80    void createGrids();
81
82    //function that locates points
83    void locatePointF(const std::vector<T> &points, std::vector<size_t> &positions) const;
84
85    T lin1DInterpolation(const T x[2], const T f[2], T xx) const;
86
87    //1D interpolation (full function)
88    T interpolation1D(T pointValue) const;
89
90    //2D interpolation
91    T interpolation2D(T pt1, T pt2) const;
92
93    //3D interpolation
94    T interpolation3D(T pt1, T pt2, T pt3) const;
95
96    //4D interpolation
97    T interpolation4D(T pt1, T pt2, T pt3, T pt4) const;
98  };
```

```
99
100 #endif
```

Content of *interpolationF* class' source file, *linearinterpolation.cpp* follows:

```
1  #include "linearinterpolation.hpp"
2
3  #include <iostream>
4  #include <vector>
5  #include <algorithm>
6  #include <limits>
7
8  //CONSTRUCTORS:
9  template <typename T>
10 interpolationF<T>::interpolationF() {}
11
12 //input is 2 1D arrays:
13 template <typename T>
14 interpolationF<T>::interpolationF(const T *xData, const T *fData, size_t NofElements)
15 {
16   setData(xData, fData, NofElements);
17 }
18
19 template <typename T>
20 void interpolationF<T>::setData(const T *xData, const T *fData, size_t NofElements)
21 {
22   m_variableN = 1;
23   m_dataLength = NofElements;
24
25   m_data.resize(m_variableN+1);
26
27   m_data[0] = std::vector<T>(xData, xData + m_dataLength);
28     m_data[1] = std::vector<T>(fData, fData + m_dataLength);
29
30   createGrids();
31
32   for (size_t iv=0; iv<m_variableN; iv++)
33     if (m_data[iv].size() < 2)
34       std::cerr << "Error: not enough data for interplation for variable " + std::
     to_string(iv) + "." << std::endl;
35 }
36
37 //input is 2 1D vectors:
38 template <typename T>
39 interpolationF<T>::interpolationF(const std::vector<T> &xData, const std::vector<T> &
     fData)
40 {
41   setData(xData, fData);
42 }
43
44 template <typename T>
45 void interpolationF<T>::setData(const std::vector<T> &xData, const std::vector<T> &fData)
46 {
47   m_variableN = 1;
48   m_dataLength = fData.size();
49
50   m_data.resize(m_variableN+1);
51
52   m_data[0] = std::vector<T>(xData.begin(), xData.begin() + m_dataLength);
53     m_data[1] = std::vector<T>(fData.begin(), fData.begin() + m_dataLength);
54
55   createGrids();
56
57   for (size_t iv=0; iv<m_variableN; iv++)
58     if (m_data[iv].size() < 2)
59       std::cerr << "Error: not enough data for interplation for variable " + std::
     to_string(iv) + "." << std::endl;
60 }
61
62 //input is 3 1D arrays:
63 template <typename T>
64 interpolationF<T>::interpolationF(const T *x1Data, const T *x2Data, const T *fData,
     size_t NofElements)
```

```
65 {
66   setData(x1Data, x2Data, fData, NofElements);
67 }
68
69 template <typename T>
70 void interpolationF<T>::setData(const T *x1Data, const T *x2Data, const T *fData, size_t
       NofElements)
71 {
72   m_variableN = 2;
73   m_dataLength = NofElements;
74
75   m_data.resize(m_variableN+1);
76
77   m_data[0] = std::vector<T>(x1Data, x1Data + m_dataLength);
78   m_data[1] = std::vector<T>(x2Data, x2Data + m_dataLength);
79     m_data[2] = std::vector<T>( fData,  fData + m_dataLength);
80
81   createGrids();
82
83   for (size_t iv=0; iv<m_variableN; iv++)
84     if (m_data[iv].size() < 2)
85       std::cerr << "Error: not enough data for interplation for variable " + std::
       to_string(iv) + "." << std::endl;
86 }
87
88 //input is 3 1D vectors:
89 template <typename T>
90 interpolationF<T>::interpolationF(const std::vector<T> &x1Data, const std::vector<T> &
       x2Data, const std::vector<T> &fData)
91 {
92   setData(x1Data, x2Data, fData);
93 }
94
95 template <typename T>
96 void interpolationF<T>::setData(const std::vector<T> &x1Data, const std::vector<T> &
       x2Data, const std::vector<T> &fData)
97 {
98   m_variableN = 2;
99   m_dataLength = fData.size();
100
101   m_data.resize(m_variableN+1);
102
103   m_data[0] = std::vector<T>(x1Data.begin(), x1Data.begin() + m_dataLength);
104   m_data[1] = std::vector<T>(x2Data.begin(), x2Data.begin() + m_dataLength);
105     m_data[2] = std::vector<T>( fData.begin(),  fData.begin() + m_dataLength);
106
107   createGrids();
108
109   for (size_t iv=0; iv<m_variableN; iv++)
110     if (m_data[iv].size() < 2)
111       std::cerr << "Error: not enough data for interplation for variable " + std::
       to_string(iv) + "." << std::endl;
112 }
113
114 //input is 2 1D vectors (grids) and 1 2d vector (function values):
115 template <typename T>
116 interpolationF<T>::interpolationF(const std::vector<T> &x1Data, const std::vector<T> &
       x2Data, const std::vector<std::vector<T>> &fData)
117 {
118   setData(x1Data, x2Data, fData);
119 }
120
121 template <typename T>
122 void interpolationF<T>::setData(const std::vector<T> &x1Data, const std::vector<T> &
       x2Data, const std::vector<std::vector<T>> &fData)
123 {
124   m_variableN = 2;
125   m_dataLength = fData.size();
126
127   m_data.resize(m_variableN+1);
128
129   m_data[0] = std::vector<T>(x1Data.begin(), x1Data.end());
```

```
130   m_data[1] = std::vector<T>(x2Data.begin(), x2Data.end());
131   for (const auto &row : fData)
132     for (const auto &elem : row)
133       m_data[2].push_back(elem);
134 
135   createGrids();
136 
137   for (size_t iv=0; iv<m_variableN; iv++)
138     if (m_data[iv].size() < 2)
139       std::cerr << "Error: not enough data for interplation for variable " + std::
      to_string(iv) + "." << std::endl;
140 }
141 
142 //input is 4 1D arrays:
143 template <typename T>
144 interpolationF<T>::interpolationF(const T *x1Data, const T *x2Data, const T *x3Data,
      const T *fData, size_t NofElements)
145 {
146   setData(x1Data, x2Data, x3Data, fData, NofElements);
147 }
148 
149 template <typename T>
150 void interpolationF<T>::setData(const T *x1Data, const T *x2Data, const T *x3Data, const
      T *fData, size_t NofElements)
151 {
152   m_variableN = 3;
153   m_dataLength = NofElements;
154 
155   m_data.resize(m_variableN+1);
156 
157   m_data[0] = std::vector<T>(x1Data, x1Data + m_dataLength);
158   m_data[1] = std::vector<T>(x2Data, x2Data + m_dataLength);
159   m_data[2] = std::vector<T>(x3Data, x3Data + m_dataLength);
160     m_data[3] = std::vector<T>( fData,  fData + m_dataLength);
161 
162   createGrids();
163 
164   for (size_t iv=0; iv<m_variableN; iv++)
165     if (m_data[iv].size() < 2)
166       std::cerr << "Error: not enough data for interplation for variable " + std::
      to_string(iv) + "." << std::endl;
167 }
168 
169 //input is 4 1D vectors:
170 template <typename T>
171 interpolationF<T>::interpolationF(const std::vector<T> &x1Data, const std::vector<T> &
      x2Data, const std::vector<T> &x3Data, const std::vector<T> &fData)
172 {
173   setData(x1Data, x2Data, x3Data, fData);
174 }
175 
176 template <typename T>
177 void interpolationF<T>::setData(const std::vector<T> &x1Data, const std::vector<T> &
      x2Data, const std::vector<T> &x3Data, const std::vector<T> &fData)
178 {
179   m_variableN = 3;
180   m_dataLength = fData.size();
181 
182   m_data.resize(m_variableN+1);
183 
184   m_data[0] = std::vector<T>(x1Data.begin(), x1Data.begin() + m_dataLength);
185   m_data[1] = std::vector<T>(x2Data.begin(), x2Data.begin() + m_dataLength);
186   m_data[2] = std::vector<T>(x3Data.begin(), x3Data.begin() + m_dataLength);
187     m_data[3] = std::vector<T>( fData.begin(),  fData.begin() + m_dataLength);
188 
189   createGrids();
190 
191   for (size_t iv=0; iv<m_variableN; iv++)
192     if (m_data[iv].size() < 2)
193       std::cerr << "Error: not enough data for interplation for variable " + std::
      to_string(iv) + "." << std::endl;
194 }
```

```
195
196 //input is 5 1D arrays:
197 template <typename T>
198 interpolationF<T>::interpolationF(const T *x1Data, const T *x2Data, const T *x3Data,
        const T *x4Data, const T *fData, size_t NofElements)
199 {
200   setData(x1Data, x2Data, x3Data, x4Data, fData, NofElements);
201 }
202
203 template <typename T>
204 void interpolationF<T>::setData(const T *x1Data, const T *x2Data, const T *x3Data, const
        T *x4Data, const T *fData, size_t NofElements)
205 {
206   m_variableN = 4;
207   m_dataLength = NofElements;
208
209   m_data.resize(m_variableN+1);
210
211   m_data[0] = std::vector<T>(x1Data, x1Data + m_dataLength);
212   m_data[1] = std::vector<T>(x2Data, x2Data + m_dataLength);
213   m_data[2] = std::vector<T>(x3Data, x3Data + m_dataLength);
214   m_data[3] = std::vector<T>(x4Data, x4Data + m_dataLength);
215     m_data[4] = std::vector<T>( fData,  fData + m_dataLength);
216
217   createGrids();
218
219   for (size_t iv=0; iv<m_variableN; iv++)
220     if (m_data[iv].size() < 2)
221       std::cerr << "Error: not enough data for interplation for variable " + std::
        to_string(iv) + "." << std::endl;
222 }
223
224 //input is 5 1D vectors:
225 template <typename T>
226 interpolationF<T>::interpolationF(const std::vector<T> &x1Data, const std::vector<T> &
        x2Data, const std::vector<T> &x3Data, const std::vector<T> &x4Data, const std::vector
        <T> &fData)
227 {
228   setData(x1Data, x2Data, x3Data, x4Data, fData);
229 }
230
231 template <typename T>
232 void interpolationF<T>::setData(const std::vector<T> &x1Data, const std::vector<T> &
        x2Data, const std::vector<T> &x3Data, const std::vector<T> &x4Data, const std::vector
        <T> &fData)
233 {
234   m_variableN = 4;
235   m_dataLength = fData.size();
236
237   m_data.resize(m_variableN+1);
238
239   m_data[0] = std::vector<T>(x1Data.begin(), x1Data.begin() + m_dataLength);
240   m_data[1] = std::vector<T>(x2Data.begin(), x2Data.begin() + m_dataLength);
241   m_data[2] = std::vector<T>(x3Data.begin(), x3Data.begin() + m_dataLength);
242   m_data[3] = std::vector<T>(x4Data.begin(), x4Data.begin() + m_dataLength);
243     m_data[4] = std::vector<T>( fData.begin(),  fData.begin() + m_dataLength);
244
245   createGrids();
246
247   for (size_t iv=0; iv<m_variableN; iv++)
248     if (m_data[iv].size() < 2)
249       std::cerr << "Error: not enough data for interplation for variable " + std::
        to_string(iv) + "." << std::endl;
250 }
251
252 //DESTRUCTORS:
253 template <typename T>
254 interpolationF<T>::~interpolationF() {}
255
256 //INTERPOLATION FUNCTIONS:
257 //1D interpolation
258 template <typename T>
```

```
T interpolationF<T>::interpolation(T pointValue) const
{
  if (m_variableN > 1) {
    std::cerr << "Error: not enough points for interpolation." << std::endl;
    return std::numeric_limits<T>::quiet_NaN();
  }
  else {
    if (pointValue < m_domain[0][0]) {
      std::cerr << "Error: point value in dimension 1 smaller than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    if (pointValue > m_domain[0][1]) {
      std::cerr << "Error: point value in dimension 1 larger than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    return interpolation1D(pointValue);
  }
}

//2D interpolation
template <typename T>
T interpolationF<T>::interpolation(T pointValue1, T pointValue2) const
{
  if (m_variableN < 2) {
    std::cerr << "Error: too much points for interpolation." << std::endl;
    return std::numeric_limits<T>::quiet_NaN();
  }
  else if (m_variableN > 2) {
    std::cerr << "Error: not enough points for interpolation." << std::endl;
    return std::numeric_limits<T>::quiet_NaN();
  }
  else {
    if (pointValue1 < m_domain[0][0]) {
      std::cerr << "Error: point value in dimension 1 smaller than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    if (pointValue1 > m_domain[0][1]) {
      std::cerr << "Error: point value in dimension 1 larger than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    if (pointValue2 < m_domain[1][0]) {
      std::cerr << "Error: point value in dimension 2 smaller than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    if (pointValue2 > m_domain[1][1]) {
      std::cerr << "Error: point value in dimension 2 larger than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    return interpolation2D(pointValue1, pointValue2);
  }
}

//3D interpolation
template <typename T>
T interpolationF<T>::interpolation(T pointValue1, T pointValue2, T pointValue3) const
{
  if (m_variableN < 3) {
    std::cerr << "Error: too much points for interpolation." << std::endl;
    return std::numeric_limits<T>::quiet_NaN();
  }
  else if (m_variableN > 3) {
    std::cerr << "Error: not enough points for interpolation." << std::endl;
    return std::numeric_limits<T>::quiet_NaN();
  }
  else {
    if (pointValue1 < m_domain[0][0]) {
      std::cerr << "Error: point value in dimension 1 smaller than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
    }
    if (pointValue1 > m_domain[0][1]) {
      std::cerr << "Error: point value in dimension 1 larger than domain." << std::endl;
      return std::numeric_limits<T>::quiet_NaN();
```

```
331     }
332     if (pointValue2 < m_domain[1][0]) {
333       std::cerr << "Error: point value in dimension 2 smaller than domain." << std::endl;
334       return std::numeric_limits<T>::quiet_NaN();
335     }
336     if (pointValue2 > m_domain[1][1]) {
337       std::cerr << "Error: point value in dimension 2 larger than domain." << std::endl;
338       return std::numeric_limits<T>::quiet_NaN();
339     }
340     if (pointValue3 < m_domain[2][0]) {
341       std::cerr << "Error: point value in dimension 3 smaller than domain." << std::endl;
342       return std::numeric_limits<T>::quiet_NaN();
343     }
344     if (pointValue3 > m_domain[2][1]) {
345       std::cerr << "Error: point value in dimension 3 larger than domain." << std::endl;
346       return std::numeric_limits<T>::quiet_NaN();
347     }
348     return interpolation3D(pointValue1, pointValue2, pointValue3);
349   }
350   return 0.0;
351 }
352
353 //4D interpolation
354 template <typename T>
355 T interpolationF<T>::interpolation(T pointValue1, T pointValue2, T pointValue3, T
      pointValue4) const
356 {
357   if (m_variableN < 4) {
358     std::cerr << "Error: too much points for interpolation." << std::endl;
359     return std::numeric_limits<T>::quiet_NaN();
360   }
361   else if (m_variableN > 4) {
362     std::cerr << "Error: not enough points for interpolation." << std::endl;
363     return std::numeric_limits<T>::quiet_NaN();
364   }
365   else {
366     if (pointValue1 < m_domain[0][0]) {
367       std::cerr << "Error: point value in dimension 1 smaller than domain." << std::endl;
368       return std::numeric_limits<T>::quiet_NaN();
369     }
370     if (pointValue1 > m_domain[0][1]) {
371       std::cerr << "Error: point value in dimension 1 larger than domain." << std::endl;
372       return std::numeric_limits<T>::quiet_NaN();
373     }
374     if (pointValue2 < m_domain[1][0]) {
375       std::cerr << "Error: point value in dimension 2 smaller than domain." << std::endl;
376       return std::numeric_limits<T>::quiet_NaN();
377     }
378     if (pointValue2 > m_domain[1][1]) {
379       std::cerr << "Error: point value in dimension 2 larger than domain." << std::endl;
380       return std::numeric_limits<T>::quiet_NaN();
381     }
382     if (pointValue3 < m_domain[2][0]) {
383       std::cerr << "Error: point value in dimension 3 smaller than domain." << std::endl;
384       return std::numeric_limits<T>::quiet_NaN();
385     }
386     if (pointValue3 > m_domain[2][1]) {
387       std::cerr << "Error: point value in dimension 3 larger than domain." << std::endl;
388       return std::numeric_limits<T>::quiet_NaN();
389     }
390     if (pointValue4 < m_domain[3][0]) {
391       std::cerr << "Error: point value in dimension 4 smaller than domain." << std::endl;
392       return std::numeric_limits<T>::quiet_NaN();
393     }
394     if (pointValue4 > m_domain[3][1]) {
395       std::cerr << "Error: point value in dimension 4 larger than domain." << std::endl;
396       return std::numeric_limits<T>::quiet_NaN();
397     }
398     return interpolation4D(pointValue1, pointValue2, pointValue3, pointValue4);
399   }
400   return 0.0;
401 }
```

```
402
403 template <typename T>
404 const std::vector<std::vector<T>> & interpolationF<T>::domain() const
405 {
406   return m_domain;
407 }
408
409 template <typename T>
410 const std::vector<T> & interpolationF<T>::codomain() const
411 {
412   return m_codomain;
413 }
414
415 template <typename T>
416 void interpolationF<T>::createGrids()
417 {
418   for (size_t iv=0; iv<m_variableN; iv++) {
419     std::sort(m_data[iv].begin(), m_data[iv].end());
420     m_data[iv].erase(std::unique(m_data[iv].begin(), m_data[iv].end()), m_data[iv].end())
      ;
421     m_domain.push_back({m_data[iv].front(), m_data[iv].back()});
422   }
423   m_codomain.push_back(*std::min_element(m_data[m_variableN].begin(), m_data[m_variableN
      ].end()));
424   m_codomain.push_back(*std::max_element(m_data[m_variableN].begin(), m_data[m_variableN
      ].end()));
425 }
426
427 template <typename T>
428 void interpolationF<T>::locatePointF(const std::vector<T> &points, std::vector<size_t> &
    positions) const
429 {
430   positions.resize(points.size(), 0);
431     int ju, jm, jl, mm = 1 + 1;
432   bool ascnd;
433   for (size_t iv=0; iv<m_data.size()-1; iv++)
434   {
435     jl = 0;
436     ju = m_data[iv].size() - 1;
437     ascnd = (m_data[iv].back() >= m_data[iv][0]);
438
439     while ((ju - jl) >1) {
440       jm = (ju + jl) >> 1;
441       if ((points[iv] >= m_data[iv][jm]) == ascnd) {
442         jl = jm;
443       }
444       else {
445         ju = jm;
446       }
447     }
448     int n = static_cast<int>(m_data[iv].size());
449     positions[iv] = static_cast<size_t>(std::max(0, std::min(n - mm, jl - ((mm - 2) >> 1)
      )));
450   }
451 }
452
453 //1D linear interpolation
454 template <typename T>
455 T interpolationF<T>::lin1DInterpolation(const T x[2], const T f[2], T xx) const
456 {
457   return (f[0] + (xx - x[0])*(f[1] - f[0]) / (x[1] - x[0]));
458 }
459
460 //1D interpolation (full function)
461 template <typename T>
462 T interpolationF<T>::interpolation1D(T pointValue) const
463 {
464   //searching for position
465   const std::vector<T> points{pointValue};
466   std::vector<size_t> positions;
467   locatePointF(points, positions);
468
```

```
469   //setting x and Q values
470   T x[] = {m_data[0][positions[0]], m_data[0][positions[0] + 1]};
471   T Q[] = {m_data[1][positions[0]], m_data[1][positions[0] + 1]};
472
473   return lin1DInterpolation(x, Q, pointValue);
474 }
475
476 //2D interpolation
477 template <typename T>
478 T interpolationF<T>::interpolation2D(T pointValue1, T pointValue2) const
479 {
480   //searching for position
481   const std::vector<T> points{pointValue1, pointValue2};
482   std::vector<size_t> positions;
483   locatePointF(points, positions);
484
485   T x1[] = {m_data[0][positions[0]], m_data[0][positions[0] + 1]};
486   T x2[] = {m_data[1][positions[1]], m_data[1][positions[1] + 1]};
487
488   T Q2[2][2];
489   for (size_t i1=0; i1<2; i1++)
490     for (size_t i2=0; i2<2; i2++)
491       Q2[i1][i2] = m_data[2][(positions[0] + i1)*m_data[1].size() + (positions[1] + i2)];
492
493   T Q1[2];
494   for (int i1=0; i1<2; i1++)
495     Q1[i1] = lin1DInterpolation(x2, Q2[i1], pointValue2);
496
497   return lin1DInterpolation(x1, Q1, pointValue1);
498 }
499
500 //3D interpolation
501 template <typename T>
502 T interpolationF<T>::interpolation3D(T pointValue1, T pointValue2, T pointValue3) const
503 {
504   //searching for position
505   const std::vector<T> points{pointValue1, pointValue2, pointValue3};
506   std::vector<size_t> positions;
507   locatePointF(points, positions);
508
509   T x1[] = {m_data[0][positions[0]], m_data[0][positions[0] + 1]};
510   T x2[] = {m_data[1][positions[1]], m_data[1][positions[1] + 1]};
511   T x3[] = {m_data[2][positions[2]], m_data[2][positions[2] + 1]};
512
513   T Q3[2][2][2];
514   for (size_t i1=0; i1<2; i1++)
515     for (size_t i2=0; i2<2; i2++)
516       for (size_t i3=0; i3<2; i3++)
517         Q3[i1][i2][i3] = m_data[3][(positions[0] + i1)*m_data[2].size()*m_data[1].size()
    +
518                       (positions[1] + i2)*m_data[2].size() +
519                       (positions[2] + i3)];
520
521   T Q2[2][2];
522   for (size_t i1=0; i1<2; i1++)
523     for (size_t i2=0; i2<2; i2++)
524       Q2[i1][i2] = lin1DInterpolation(x3, Q3[i1][i2], pointValue3);
525
526   T Q1[2];
527   for (size_t i1=0; i1<2; i1++)
528     Q1[i1] = lin1DInterpolation(x2, Q2[i1], pointValue2);
529
530   return lin1DInterpolation(x1, Q1, pointValue1);
531 }
532
533 //4D interpolation
534 template <typename T>
535 T interpolationF<T>::interpolation4D(T pointValue1, T pointValue2, T pointValue3, T
    pointValue4) const
536 {
537   //searching for position
538   const std::vector<T> points{pointValue1, pointValue2, pointValue3, pointValue4};
```

```
  std::vector<size_t> positions;
  locatePointF(points, positions);

  T x1[] = {m_data[0][positions[0]], m_data[0][positions[0] + 1]};
  T x2[] = {m_data[1][positions[1]], m_data[1][positions[1] + 1]};
  T x3[] = {m_data[2][positions[2]], m_data[2][positions[2] + 1]};
  T x4[] = {m_data[3][positions[3]], m_data[3][positions[3] + 1]};

  T Q4[2][2][2][2];
  for (size_t i1=0; i1<2; i1++)
    for (size_t i2=0; i2<2; i2++)
      for (size_t i3=0; i3<2; i3++)
        for (size_t i4=0; i4<2; i4++)
          Q4[i1][i2][i3][i4] = m_data[4][(positions[0] + i1)*m_data[3].size()*m_data[2].
    size()*m_data[1].size() +
                           (positions[1] + i2)*m_data[3].size()*m_data[2].size() +
                           (positions[2] + i3)*m_data[3].size() +
                           (positions[3] + i4)];

  T Q3[2][2][2];
  for (size_t i1=0; i1<2; i1++)
    for (size_t i2=0; i2<2; i2++)
      for (size_t i3=0; i3<2; i3++)
        Q3[i1][i2][i3] = lin1DInterpolation(x4, Q4[i1][i2][i3], pointValue4);

  T Q2[2][2];
  for (size_t i1=0; i1<2; i1++)
    for (size_t i2=0; i2<2; i2++)
      Q2[i1][i2] = lin1DInterpolation(x3, Q3[i1][i2], pointValue3);


  T Q1[2];
  for (size_t i1=0; i1<2; i1++)
    Q1[i1] = lin1DInterpolation(x2, Q2[i1], pointValue2);

  return lin1DInterpolation(x1, Q1, pointValue1);
}

template class interpolationF<float>;
template class interpolationF<double>;
template class interpolationF<long double>;
```

Also used throughout the code are functions that perform analytical integration of linearly or cubically interpolated polynomials. These functions are templated for float, double and long double types and are embedded in *poly* namespace. These functions are defined in *polyintegration.hpp* file:

```
#ifndef HEADERFILE_POLYINTHEADER
#define HEADERFILE_POLYINTHEADER

#include <vector>

namespace poly {
    template <typename T>
    T linearIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata);

    template <typename T>
    T linearIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata, T
    lowLimit, T highLimit);

    template <typename T>
    T cubicIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata);

    template <typename T>
    T cubicIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata, T lowLimit
    , T highLimit);
}

#endif
```

Source for these functions is in *polyintegration.cpp* file:

```cpp
#include "polyintegration.hpp"

#include <vector>
#include <cmath>

template <typename T>
static size_t locatePoint(const std::vector<T> &data, T x, int interpolationOrder)
{
  int ju, jm, jl;
  int mm = interpolationOrder + 1;
  int n = data.size();
  bool ascnd = (data.back() >= data.front());
  jl = 0;
  ju = n - 1;
  while (ju - jl > 1)
  {
    jm = (ju + jl) >> 1;
    if ((x >= data[jm]) == ascnd) {
      jl = jm;
    }
    else {
      ju = jm;
    }
  }
  int pointLocation = std::max(0, std::min(n - mm, jl - ((mm - 2) >> 1)));
  return static_cast<size_t>(pointLocation);
}

template <typename T>
static void polynomialCoeff(const std::vector<T> &dataX, const std::vector<T> &dataF, std
    ::vector<double> &coeff)
{
  size_t n = dataX.size();
  coeff.resize(n, 0.0);
  double phi, ff, b;
  std::vector<double> s(n, 0.0);
  s[n-1] = -static_cast<double>(dataX[0]);

  for (size_t i=1; i<n; i++) {
    for (size_t j=n-1-i; j<n-1; j++)
      s[j] -= static_cast<double>(dataX[i]) * s[j+1];
    s[n-1] -= static_cast<double>(dataX[i]);
  }

  for (size_t j=0; j<n; j++) {
    phi = static_cast<double>(n);

    for (size_t k=n-1; k>0; k--)
      phi = static_cast<double>(k)*s[k] + static_cast<double>(dataX[j])*phi;

    ff = static_cast<double>(dataF[j])/phi;
    b = 1.0;

    for (int k=n-1; k>=0; k--) {
      coeff[k] += b * ff;
      b = s[k] + static_cast<double>(dataX[j]) * b;
    }
  }
}

template <typename T>
T poly::linearIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata)
{
  if (xdata.size() < 2) return 0.0;

  std::vector<double> k, c;
  for (size_t i=0; i<(xdata.size()-1); i++)
  {
    k.push_back(static_cast<double>(fdata[i+1]-fdata[i])/static_cast<double>(xdata[i+1]-
    xdata[i]));
    c.push_back(static_cast<double>(fdata[i])-static_cast<double>(k.back())*static_cast<
    double>(xdata[i]));
```

```
70    }
71
72    double res = 0.0;
73
74    for (size_t i=0; i<(xdata.size()-1); i++)
75      res += 0.5*k[i]*(static_cast<double>(xdata[i+1]*xdata[i+1]) - static_cast<double>(
      xdata[i]*xdata[i])) + c[i]*static_cast<double>(xdata[i+1] - xdata[i]);
76
77    return res;
78  }
79  template float poly::linearIntegrate<float>(const std::vector<float> &xdata, const std::
      vector<float> &fdata);
80  template double poly::linearIntegrate<double>(const std::vector<double> &xdata, const std
      ::vector<double> &fdata);
81  template long double poly::linearIntegrate<long double>(const std::vector<long double> &
      xdata, const std::vector<long double> &fdata);
82
83  template <typename T>
84  T poly::linearIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata, T
      lowLimit, T highLimit)
85  {
86    if (xdata.size() < 2) return 0.0;
87
88    std::vector<double> k, c;
89    for (size_t i=0; i<(xdata.size()-1); i++)
90    {
91      k.push_back(static_cast<double>(fdata[i+1]-fdata[i])/static_cast<double>(xdata[i+1]-
      xdata[i]));
92      c.push_back(static_cast<double>(fdata[i]-k.back()*xdata[i]));
93    }
94
95    //calculating value of full integral (in it's whole range):
96    double sum = 0.0L;
97
98    for (size_t i=0; i<(xdata.size()-1); i++)
99      sum += 0.5*k[i]*(static_cast<double>(xdata[i+1]*xdata[i+1]) - static_cast<double>(
      xdata[i]*xdata[i])) +
100               c[i]*static_cast<double>(xdata[i+1] - xdata[i]);
101
102   //calculating value of integral from lower range to lower limit:
103   size_t lowLimitPos = locatePoint(xdata, lowLimit, 1);
104
105   double lowSum = 0.0L;
106
107   for (size_t i=0; i<lowLimitPos; i++)
108     lowSum += 0.5*k[i]*(static_cast<double>(xdata[i+1]*xdata[i+1]) - static_cast<double>(
      xdata[i]*xdata[i])) +
109                 c[i]*static_cast<double>(xdata[i+1] - xdata[i]);
110
111   lowSum += 0.5*k[lowLimitPos]*(static_cast<double>(lowLimit*lowLimit) - static_cast<
      double>(xdata[lowLimitPos]*xdata[lowLimitPos])) +
112                   c[lowLimitPos]*static_cast<double>(lowLimit - xdata[lowLimitPos]);
113
114   //calculating value of integral from higher limit to higer range:
115   size_t highLimitPos = locatePoint(xdata, highLimit, 1);
116
117   double highSum = 0.0L;
118
119   highSum += 0.5*k[highLimitPos]*(static_cast<double>(xdata[highLimitPos+1]*xdata[
      highLimitPos+1]) - static_cast<double>(highLimit*highLimit)) +
120            c[highLimitPos]*static_cast<double>(xdata[highLimitPos+1] - highLimit);
121
122   for (size_t i=highLimitPos+1; i<xdata.size()-1; i++)
123     highSum += 0.5*k[i]*(static_cast<double>(xdata[i+1]*xdata[i+1]) - static_cast<double
      >(xdata[i]*xdata[i])) +
124                 c[i]*static_cast<double>(xdata[i+1] - xdata[i]);
125
126   //integral value is full-low-high
127   return (sum - highSum - lowSum);
128 }
129 template float poly::linearIntegrate<float>(const std::vector<float> &xdata, const std::
      vector<float> &fdata, float lowLimit, float highLimit);
```

```
130 template double poly::linearIntegrate<double>(const std::vector<double> &xdata, const std
        ::vector<double> &fdata, double lowLimit, double highLimit);
131 template long double poly::linearIntegrate<long double>(const std::vector<long double> &
        xdata, const std::vector<long double> &fdata, long double lowLimit, long double
        highLimit);
132
133 template <typename T>
134 T poly::cubicIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata)
135 {
136   if (xdata.size() < 2) return 0;
137
138   //calculating polynomial coefficients for each segment:
139   std::vector<std::vector<double>> coefficents; coefficents.resize(xdata.size() - 1);
140   for (size_t i=0; i<coefficents.size(); i++)
141   {
142     size_t pointLocation = locatePoint(xdata, xdata[i], 3);
143     std::vector<T> xdatatemp(xdata.begin()+pointLocation, xdata.begin()+pointLocation+4);
144     std::vector<T> fdatatemp(fdata.begin()+pointLocation, fdata.begin()+pointLocation+4);
145     polynomialCoeff(xdatatemp, fdatatemp, coefficents[i]);
146   }
147
148   //calculating value of integral:
149   double sum = 0.0L;
150   for (size_t i=0; i<xdata.size()-1; i++)
151     for (size_t j=0; j<coefficents[i].size(); j++)
152       sum += 1.0/static_cast<double>(j+1)*coefficents[i][j]*(std::pow(static_cast<double
        >(xdata[i+1]), static_cast<double>(j+1)) - std::pow(static_cast<double>(xdata[i]),
        static_cast<double>(j+1)));
153
154   return sum;
155 }
156 template float poly::cubicIntegrate<float>(const std::vector<float> &xdata, const std::
        vector<float> &fdata);
157 template double poly::cubicIntegrate<double>(const std::vector<double> &xdata, const std
        ::vector<double> &fdata);
158 template long double poly::cubicIntegrate<long double>(const std::vector<long double> &
        xdata, const std::vector<long double> &fdata);
159
160 template <typename T>
161 T poly::cubicIntegrate(const std::vector<T> &xdata, const std::vector<T> &fdata, T
        lowLimit, T highLimit)
162 {
163   if (xdata.size() < 2) return 0;
164
165   //calculating polynomial coefficients for each segment:
166   std::vector<std::vector<double>> coefficents; coefficents.resize(xdata.size() - 1);
167   for (size_t i=0; i<coefficents.size(); i++)
168   {
169     size_t pointLocation = locatePoint(xdata, xdata[i], 3);
170     std::vector<T> xdatatemp(xdata.begin()+pointLocation, xdata.begin()+pointLocation+4);
171     std::vector<T> fdatatemp(fdata.begin()+pointLocation, fdata.begin()+pointLocation+4);
172     polynomialCoeff(xdatatemp, fdatatemp, coefficents[i]);
173   }
174
175   //calculating value of full integral (in it's whole range):
176   double sum = 0.0L;
177   for (size_t i=0; i<xdata.size()-1; i++)
178     for (size_t j=0; j<coefficents[i].size(); j++)
179       sum += 1.0L/static_cast<double>(j+1)*coefficents[i][j]*
180                         (std::pow(static_cast<double>(xdata[i+1]), static_cast<double>(j
        +1)) - std::pow(static_cast<double>(xdata[i]), static_cast<double>(j+1)));
181
182   //calculating value of integral from lower range to lower limit:
183   size_t lowLimitPos = locatePoint(xdata, lowLimit, 1);
184
185   double lowSum = 0.0L;
186
187   for (size_t i=0; i<lowLimitPos; i++)
188     for (size_t j=0; j<coefficents[i].size(); j++)
189       lowSum += 1.0L/static_cast<double>(j+1)*coefficents[i][j]*
190                             (std::pow(static_cast<double>(xdata[i+1]), static_cast<double
        >(j+1)) - std::pow(static_cast<double>(xdata[i]), static_cast<double>(j+1)));
```

```
191
192   for (size_t j=0; j<coefficents[lowLimitPos].size(); j++)
193     lowSum += 1.0L/static_cast<double>(j+1)*coefficents[lowLimitPos][j]*
194                           (std::pow(static_cast<double>(lowLimit), static_cast<double>(j+1)
      ) - std::pow(static_cast<double>(xdata[lowLimitPos]), static_cast<double>(j+1)));
195
196   //calculating value of integral from higher limit to higer range:
197   size_t highLimitPos = locatePoint(xdata, highLimit, 1);
198
199   double highSum = 0.0L;
200
201   for (size_t j=0; j<coefficents[highLimitPos].size(); j++)
202     highSum += 1.0L/static_cast<double>(j+1)*coefficents[highLimitPos][j]*
203                           (std::pow(static_cast<double>(xdata[highLimitPos+1]), static_cast
      <double>(j+1)) - std::pow(static_cast<double>(highLimit), static_cast<double>(j+1)));
204
205   for (size_t i=highLimitPos+1; i<xdata.size()-1; i++)
206     for (size_t j=0; j<coefficents[i].size(); j++)
207       highSum += 1.0L/static_cast<double>(j+1)*coefficents[i][j]*
208                              (std::pow(static_cast<double>(xdata[i+1]), static_cast<double
      >(j+1)) - std::pow(static_cast<double>(xdata[i]), static_cast<double>(j+1)));
209
210   //integral value is full-low-high
211   return (sum - highSum - lowSum);
212 }
213 template float poly::cubicIntegrate<float>(const std::vector<float> &xdata, const std::
      vector<float> &fdata, float lowLimit, float highLimit);
214 template double poly::cubicIntegrate<double>(const std::vector<double> &xdata, const std
      ::vector<double> &fdata, double lowLimit, double highLimit);
215 template long double poly::cubicIntegrate<long double>(const std::vector<long double> &
      xdata, const std::vector<long double> &fdata, long double lowLimit, long double
      highLimit);
```

Bibliography

[1] Magdalena Djordjevic, Dusan Zigic, Marko Djordjevic, and Jussi Auvinen. How to test path-length dependence in energy loss mechanisms: analysis leading to a new observable. *Phys. Rev. C*, 99(6):061902, 2019.

[2] Dusan Zigic, Igor Salom, Jussi Auvinen, Marko Djordjevic, and Magdalena Djordjevic. DREENA-B framework: first predictions of R_{AA} and v_2 within dynamical energy loss formalism in evolving QCD medium. *Phys. Lett. B*, 791:236–241, 2019.

[3] Dusan Zigic, Bojana Ilic, Marko Djordjevic, and Magdalena Djordjevic. Exploring the initial stages in heavy-ion collisions with high-$p_\perp$ R_{AA} and v_2 theory and data. *Phys. Rev. C*, 101(6):064909, 2020.

[4] Dusan Zigic, Igor Salom, Jussi Auvinen, Pasi Huovinen, and Magdalena Djordjevic. DREENA-A framework as a QGP tomography tool. *Front. in Phys.*, 10:957019, 2022.

[5] Dusan Zigic, Jussi Auvinen, Igor Salom, Magdalena Djordjevic, and Pasi Huovinen. Importance of higher harmonics and v4 puzzle in quark-gluon plasma tomography. *Phys. Rev. C*, 106(4):044909, 2022.

[6] D. J. Gross and Frank Wilczek. Asymptotically Free Gauge Theories - I. *Phys. Rev. D*, 8:3633–3652, 1973.

[7] H. David Politzer. Reliable Perturbative Results for Strong Interactions? *Phys. Rev. Lett.*, 30:1346–1349, 1973.

[8] William J. Marciano and Heinz Pagels. Quantum Chromodynamics: A Review. *Phys. Rept.*, 36:137, 1978.

[9] E. Eichten, K. Gottfried, T. Kinoshita, John B. Kogut, K. D. Lane, and Tung-Mow Yan. The Spectrum of Charmonium. *Phys. Rev. Lett.*, 34:369–372, 1975. [Erratum: Phys.Rev.Lett. 36, 1276 (1976)].

[10] Alexandre Deur, Stanley J. Brodsky, and Guy F. de Teramond. The QCD Running Coupling. *Nucl. Phys.*, 90:1, 2016.

[11] Quark potential. https://www2.ph.ed.ac.uk/~playfer/PPlect8.pdf.

[12] Bo Andersson, G. Gustafson, G. Ingelman, and T. Sjostrand. Parton Fragmentation and String Dynamics. *Phys. Rept.*, 97:31–145, 1983.

[13] David J. Gross and Frank Wilczek. Ultraviolet Behavior of Nonabelian Gauge Theories. *Phys. Rev. Lett.*, 30:1343–1346, 1973.

[14] P. A. Zyla et al. Review of Particle Physics. *PTEP*, 2020(8):083C01, 2020.

[15] Szabolcs Borsanyi, Gergely Endrodi, Zoltan Fodor, Antal Jakovac, Sandor D. Katz, Stefan Krieg, Claudia Ratti, and Kalman K. Szabo. The QCD equation of state with dynamical quarks. *JHEP*, 11:077, 2010.

[16] Kenji Fukushima. Chiral effective model with the Polyakov loop. *Phys. Lett. B*, 591:277–284, 2004.

[17] Edward V. Shuryak. Quantum Chromodynamics and the Theory of Superdense Matter. *Phys. Rept.*, 61:71–158, 1980.

[18] Mark G. Alford, Andreas Schmitt, Krishna Rajagopal, and Thomas Schäfer. Color superconductivity in dense quark matter. *Rev. Mod. Phys.*, 80:1455–1515, 2008.

[19] Y. Aoki, G. Endrodi, Z. Fodor, S. D. Katz, and K. K. Szabo. The Order of the quantum chromodynamics transition predicted by the standard model of particle physics. *Nature*, 443:675–678, 2006.

[20] Ludwik Turko. Looking for the Phase Transition—Recent NA61/SHINE Results. *Universe*, 4(3):52, 2018.

[21] P. Kovtun, Dan T. Son, and Andrei O. Starinets. Viscosity in strongly interacting quantum field theories from black hole physics. *Phys. Rev. Lett.*, 94:111601, 2005.

[22] Mikhail A. Stephanov. QCD Phase Diagram and the Critical Point. *Prog. Theor. Phys. Suppl.*, 153:139–156, 2004.

[23] Zuzana Fecková, Jan Steinheimer, Boris Tomášik, and Marcus Bleicher. Net-proton number kurtosis and skewness in nuclear collisions: Influence of deuteron formation. *Phys. Rev. C*, 92(6):064908, 2015.

[24] L. Adamczyk et al. Energy Dependence of Moments of Net-proton Multiplicity Distributions at RHIC. *Phys. Rev. Lett.*, 112:032302, 2014.

[25] G. F. Burgio, H. J. Schulze, I. Vidana, and J. B. Wei. Neutron stars and the nuclear equation of state. *Prog. Part. Nucl. Phys.*, 120:103879, 2021.

[26] Philippe de Forcrand. Simulating QCD at finite density. *PoS*, LAT2009:010, 2009.

[27] Bjoern Schenke, Prithwish Tribedy, and Raju Venugopalan. Fluctuating Glasma initial conditions and flow in heavy ion collisions. *Phys. Rev. Lett.*, 108:252301, 2012.

[28] Bjoern Schenke, Prithwish Tribedy, and Raju Venugopalan. Event-by-event gluon multiplicity, energy density, and eccentricities in ultrarelativistic heavy-ion collisions. *Phys. Rev. C*, 86:034908, 2012.

[29] K. J. Eskola, K. Kajantie, P. V. Ruuskanen, and Kimmo Tuominen. Scaling of transverse energies and multiplicities with atomic number and energy in ultrarelativistic nuclear collisions. *Nucl. Phys. B*, 570:379–389, 2000.

[30] R. Paatelainen, K. J. Eskola, H. Holopainen, and K. Tuominen. Multiplicities and p_T spectra in ultrarelativistic heavy ion collisions from a next-to-leading order improved perturbative QCD + saturation + hydrodynamics model. *Phys. Rev. C*, 87(4):044904, 2013.

[31] R. Paatelainen, K. J. Eskola, H. Niemi, and K. Tuominen. Fluid dynamics with saturated minijet initial conditions in ultrarelativistic heavy-ion collisions. *Phys. Lett. B*, 731:126–130, 2014.

[32] Ulrich W. Heinz and Peter F. Kolb. Early thermalization at RHIC. *Nucl. Phys. A*, 702:269–280, 2002.

[33] Peter Braun-Munzinger and Benjamin Dönigus. Loosely-bound objects produced in nuclear collisions at the LHC. *Nucl. Phys. A*, 987:144–201, 2019.

[34] Jean-Yves Ollitrault. Anisotropy as a signature of transverse collective flow. *Phys. Rev. D*, 46:229–245, 1992.

[35] Raimond Snellings. Elliptic Flow: A Brief Review. *New J. Phys.*, 13:055008, 2011.

[36] Wojciech Florkowski. *Phenomenology of Ultra-Relativistic Heavy-Ion Collisions*. 3 2010.

[37] B. B. Back et al. Charged particle multiplicity near mid-rapidity in central Au + Au collisions at $S^{(1/2)}$ = 56-A/GeV and 130-A/GeV. *Phys. Rev. Lett.*, 85:3100–3104, 2000.

[38] Betty Abelev et al. Centrality Dependence of Charged Particle Production at Large Transverse Momentum in Pb–Pb Collisions at $\sqrt{s_{NN}} = 2.76$ TeV. *Phys. Lett. B*, 720:52–62, 2013.

[39] Michael L. Miller, Klaus Reygers, Stephen J. Sanders, and Peter Steinberg. Glauber modeling in high energy nuclear collisions. *Ann. Rev. Nucl. Part. Sci.*, 57:205–243, 2007.

[40] C. Adler et al. Centrality dependence of high p_T hadron suppression in Au+Au collisions at $\sqrt{s}_{NN}$ = 130-GeV. *Phys. Rev. Lett.*, 89:202301, 2002.

[41] Betty Abelev et al. Centrality determination of Pb-Pb collisions at $\sqrt{s_{NN}}$ = 2.76 TeV with ALICE. *Phys. Rev. C*, 88(4):044909, 2013.

[42] David G. d'Enterria. Quark-Gluon Matter. *J. Phys. G*, 34:S53–S82, 2007.

[43] R. Baier, Yuri L. Dokshitzer, Alfred H. Mueller, S. Peigne, and D. Schiff. Radiative energy loss of high-energy quarks and gluons in a finite volume quark - gluon plasma. *Nucl. Phys. B*, 483:291–320, 1997.

[44] B. Z. Kopeliovich, J. Nemchik, A. Schafer, and A. V. Tarasov. Cronin effect in hadron production off nuclei. *Phys. Rev. Lett.*, 88:232303, 2002.

[45] V. Greco, C. M. Ko, and P. Levai. Parton coalescence and anti-proton / pion anomaly at RHIC. *Phys. Rev. Lett.*, 90:202302, 2003.

[46] S. Voloshin and Y. Zhang. Flow study in relativistic nuclear collisions by Fourier expansion of Azimuthal particle distributions. *Z. Phys. C*, 70:665–672, 1996.

[47] B. Alver and G. Roland. Collision geometry fluctuations and triangular flow in heavy-ion collisions. *Phys. Rev. C*, 81:054905, 2010. [Erratum: Phys.Rev.C 82, 039903 (2010)].

[48] P. Huovinen, P. F. Kolb, Ulrich W. Heinz, P. V. Ruuskanen, and S. A. Voloshin. Radial and elliptic flow at RHIC: Further predictions. *Phys. Lett. B*, 503:58–64, 2001.

[49] Magdalena Djordjevic and Marko Djordjevic. LHC jet suppression of light and heavy flavor observables. *Phys. Lett. B*, 734:286–289, 2014.

[50] M. Gyulassy, P. Levai, and I. Vitev. Reaction operator approach to nonAbelian energy loss. *Nucl. Phys. B*, 594:371–419, 2001.

[51] Miklos Gyulassy, Peter Levai, and Ivan Vitev. Jet quenching in thin plasmas. *Nucl. Phys. A*, 661:637–640, 1999.

[52] Miklos Gyulassy, Peter Levai, and Ivan Vitev. Jet quenching in thin quark gluon plasmas. 1. Formalism. *Nucl. Phys. B*, 571:197–233, 2000.

[53] M. Gyulassy, P. Levai, and I. Vitev. NonAbelian energy loss at finite opacity. *Phys. Rev. Lett.*, 85:5535–5538, 2000.

[54] M. Gyulassy, P. Levai, and I. Vitev. Jet tomography of Au+Au reactions including multigluon fluctuations. *Phys. Lett. B*, 538:282–288, 2002.

[55] Magdalena Djordjevic and Miklos Gyulassy. Heavy quark radiative energy loss in QCD matter. *Nucl. Phys. A*, 733:265–298, 2004.

[56] S. S. Adler et al. Nuclear modification of electron spectra and implications for heavy quark energy loss in Au+Au collisions at s(NN)**(1/2) - 200-GeV. *Phys. Rev. Lett.*, 96:032301, 2006.

[57] Y. Akiba. Probing the properties of dense partonic matter at RHIC. *Nucl. Phys. A*, 774:403–408, 2006.

[58] Magdalena Djordjevic. Theoretical formalism of radiative jet energy loss in a finite size dynamical QCD medium. *Phys. Rev. C*, 80:064909, 2009.

[59] Magdalena Djordjevic. Collisional energy loss in a finite size QCD matter. *Phys. Rev. C*, 74:064907, 2006.

[60] Nestor Armesto, Carlos A. Salgado, and Urs Achim Wiedemann. Medium induced gluon radiation off massive quarks fills the dead cone. *Phys. Rev. D*, 69:114003, 2004.

[61] Xin-Nian Wang and Xiao-feng Guo. Multiple parton scattering in nuclei: Parton energy loss. *Nucl. Phys. A*, 696:788–832, 2001.

[62] Joseph I. Kapusta. *Finite Temperature Field Theory*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, Cambridge, 1989.

[63] Michel Le Bellac. *Thermal Field Theory*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 1996.

[64] Magdalena Djordjevic and Ulrich W. Heinz. Radiative energy loss in a finite dynamical QCD medium. *Phys. Rev. Lett.*, 101:022302, 2008.

[65] Magdalena Djordjevic and Miklos Gyulassy. The Ter-Mikayelian effect on QCD radiative energy loss. *Phys. Rev. C*, 68:034914, 2003.

[66] Magdalena Djordjevic and Marko Djordjevic. Generalization of radiative jet energy loss to non-zero magnetic mass. *Phys. Lett. B*, 709:229–233, 2012.

[67] Bojana Blagojevic, Magdalena Djordjevic, and Marko Djordjevic. Calculating hard probe radiative energy loss beyond the soft-gluon approximation: Examining the approximation validity. *Phys. Rev. C*, 99(2):024901, 2019.

[68] Bojana Blagojevic and Magdalena Djordjevic. Importance of different energy loss effects in jet suppression at the RHIC and the LHC. *J. Phys. G*, 42(7):075105, 2015.

[69] Eric Braaten and Markus H. Thoma. Energy loss of a heavy fermion in a hot plasma. *Phys. Rev. D*, 44:1298–1310, 1991.

[70] J. F. Gunion and G. Bertsch. HADRONIZATION BY COLOR BREMSSTRAHLUNG. *Phys. Rev. D*, 25:746, 1982.

[71] A. B. Migdal. Bremsstrahlung and pair production in condensed media at high-energies. *Phys. Rev.*, 103:1811–1820, 1956.

[72] L. D. Landau and I. Pomeranchuk. Limits of applicability of the theory of bremsstrahlung electrons and pair production at high-energies. *Dokl. Akad. Nauk Ser. Fiz.*, 92:535–536, 1953.

[73] Y. Maezawa, S. Aoki, S. Ejiri, T. Hatsuda, N. Ishii, K. Kanaya, N. Ukita, and T. Umeda. Electric and Magnetic Screening Masses at Finite Temperature from Generalized Polyakov-Line Correlations in Two-flavor Lattice QCD. *Phys. Rev. D*, 81:091501, 2010.

[74] A. Nakamura, T. Saito, and S. Sakai. Lattice calculation of gluon screening masses. *Phys. Rev. D*, 69:014506, 2004.

[75] Dusan Zigic, Igor Salom, Jussi Auvinen, Marko Djordjevic, and Magdalena Djordjevic. DREENA-C framework: joint R_{AA} and v_2 predictions and implications to QGP tomography. *J. Phys. G*, 46(8):085101, 2019.

[76] John C. Collins and M. J. Perry. Superdense Matter: Neutrons Or Asymptotically Free Quarks? *Phys. Rev. Lett.*, 34:1353, 1975.

[77] Miklos Gyulassy and Larry McLerran. New forms of QCD matter discovered at RHIC. *Nucl. Phys. A*, 750:30–63, 2005.

[78] J. D. Bjorken. Energy Loss of Energetic Partons in Quark - Gluon Plasma: Possible Extinction of High p(t) Jets in Hadron - Hadron Collisions. 8 1982.

[79] Karen M. Burke et al. Extracting the jet transport coefficient from jet quenching in high-energy heavy-ion collisions. *Phys. Rev. C*, 90(1):014909, 2014.

[80] Yingru Xu et al. Resolving discrepancies in the estimation of heavy quark transport coefficients in relativistic heavy-ion collisions. *Phys. Rev. C*, 99(1):014902, 2019.

[81] Markus H. Thoma and Miklos Gyulassy. Quark Damping and Energy Loss in the High Temperature QCD. *Nucl. Phys. B*, 351:491–506, 1991.

[82] Eric Braaten and Markus H. Thoma. Energy loss of a heavy quark in the quark - gluon plasma. *Phys. Rev. D*, 44(9):R2625, 1991.

[83] B. G. Zakharov. Fully quantum treatment of the Landau-Pomeranchuk-Migdal effect in QED and QCD. *JETP Lett.*, 63:952–957, 1996.

[84] Peter Brockway Arnold, Guy D. Moore, and Laurence G. Yaffe. Photon emission from ultra-relativistic plasmas. *JHEP*, 11:057, 2001.

[85] Peter Brockway Arnold, Guy D. Moore, and Laurence G. Yaffe. Effective kinetic theory for high temperature gauge theories. *JHEP*, 01:030, 2003.

[86] A. Majumder and M. Van Leeuwen. The Theory and Phenomenology of Perturbative QCD Based Jet Quenching. *Prog. Part. Nucl. Phys.*, 66:41–92, 2011.

[87] C. Marquet and T. Renk. Jet quenching in the strongly-interacting quark-gluon plasma. *Phys. Lett. B*, 685:270–276, 2010.

[88] Fabio Dominguez, C. Marquet, A. H. Mueller, Bin Wu, and Bo-Wen Xiao. Comparing energy loss and p-perpendicular - broadening in perturbative QCD with strong coupling N = 4 SYM theory. *Nucl. Phys. A*, 811:197–222, 2008.

[89] Caio A. G. Prado, Jacquelyn Noronha-Hostler, Roland Katz, Alexandre A. P. Suaide, Jorge Noronha, Marcelo G. Munhoz, and Mauro R. Cosentino. Event-by-event correlations between soft hadrons and D^0 mesons in 5.02 TeV PbPb collisions at the CERN Large Hadron Collider. *Phys. Rev. C*, 96(6):064903, 2017.

[90] Marlene Nahrgang, Jörg Aichelin, Pol Bernard Gossiaux, and Klaus Werner. Toward a consistent evolution of the quark-gluon plasma and heavy quarks. *Phys. Rev. C*, 93(4):044909, 2016.

[91] Barbara Betz and Miklos Gyulassy. Constraints on the Path-Length Dependence of Jet Quenching in Nuclear Collisions at RHIC and LHC. *JHEP*, 08:090, 2014. [Erratum: JHEP 10, 043 (2014)].

[92] Barbara Betz, Miklos Gyulassy, Matthew Luzum, Jorge Noronha, Jacquelyn Noronha-Hostler, Israel Portillo, and Claudia Ratti. Cumulants and nonlinear response of high p_T harmonic flow at $\sqrt{s_{NN}} = 5.02$ TeV. *Phys. Rev. C*, 95(4):044901, 2017.

[93] Thorsten Renk. Constraining the Physics of Jet Quenching. *Phys. Rev. C*, 85:044903, 2012.

[94] Denes Molnar and Deke Sun. Interplay between bulk medium evolution and (D)GLV energy loss. *Nucl. Phys. A*, 932:140–145, 2014.

[95] Denes Molnar and Deke Sun. Realistic medium-averaging in radiative energy loss. *Nucl. Phys. A*, 910-911:486–489, 2013.

[96] Thorsten Renk, Hannu Holopainen, Jussi Auvinen, and Kari J. Eskola. Energy Loss in a Fluctuating Hydrodynamical Background. *Phys. Rev. C*, 85:044915, 2012.

[97] Zhong-Bo Kang, Ivan Vitev, and Hongxi Xing. Nuclear modification of high transverse momentum particle production in p+A collisions at RHIC and LHC. *Phys. Lett. B*, 718:482–487, 2012.

[98] Rishi Sharma, Ivan Vitev, and Ben-Wei Zhang. Light-cone wave function approach to open heavy flavor dynamics in QCD matter. *Phys. Rev. C*, 80:054902, 2009.

[99] Andrea Dainese. Perspectives for the study of charm in-medium quenching at the LHC with ALICE. *Eur. Phys. J. C*, 33:495–503, 2004.

[100] Daniel de Florian, Rodolfo Sassot, and Marco Stratmann. Global analysis of fragmentation functions for pions and kaons and their uncertainties. *Phys. Rev. D*, 75:114010, 2007.

[101] Matteo Cacciari and Paolo Nason. Charm cross-sections for the Tevatron Run II. *JHEP*, 09:006, 2003.

[102] Magdalena Djordjevic and Marko Djordjevic. Predictions of heavy-flavor suppression at 5.1 TeV Pb + Pb collisions at the CERN Large Hadron Collider. *Phys. Rev. C*, 92(2):024918, 2015.

[103] Magdalena Djordjevic. Complex suppression patterns distinguish between major energy loss effects in Quark–Gluon Plasma. *Phys. Lett. B*, 763:439–444, 2016.

[104] S. Acharya et al. Transverse momentum spectra and nuclear modification factors of charged particles in pp, p-Pb and Pb-Pb collisions at the LHC. *JHEP*, 11:013, 2018.

[105] Vardan Khachatryan et al. Charged-particle nuclear modification factors in PbPb and pPb collisions at $\sqrt{s_{\mathrm{N\,N}}} = 5.02$ TeV. *JHEP*, 04:039, 2017.

[106] Jiechen Xu, Alessandro Buzzatti, and Miklos Gyulassy. Azimuthal jet flavor tomography with CUJET2.0 of nuclear collisions at RHIC and LHC. *JHEP*, 08:063, 2014.

[107] Giuliano Giacalone, Jacquelyn Noronha-Hostler, Matthew Luzum, and Jean-Yves Ollitrault. Hydrodynamic predictions for 5.44 TeV Xe+Xe collisions. *Phys. Rev. C*, 97(3):034904, 2018.

[108] Constantin Loizides, Jason Kamin, and David d'Enterria. Improved Monte Carlo Glauber predictions at present and future nuclear colliders. *Phys. Rev. C*, 97(5):054910, 2018. [Erratum: Phys.Rev.C 99, 019901 (2019)].

[109] K. J. Eskola, H. Niemi, R. Paatelainen, and K. Tuominen. Predictions for multiplicities and flow harmonics in 5.44 TeV Xe+Xe collisions at the CERN Large Hadron Collider. *Phys. Rev. C*, 97(3):034911, 2018.

[110] Shreyasi Acharya et al. Centrality and pseudorapidity dependence of the charged-particle multiplicity density in Xe–Xe collisions at $\sqrt{s_{NN}}$ =5.44TeV. *Phys. Lett. B*, 790:35–48, 2019.

[111] Albert M Sirunyan et al. Charged-particle nuclear modification factors in XeXe collisions at $\sqrt{s_{\mathrm{NN}}} = 5.44$ TeV. *JHEP*, 10:138, 2018.

[112] Magdalena Djordjevic, Stefan Stojku, Marko Djordjevic, and Pasi Huovinen. Shape of the quark gluon plasma droplet reflected in the high-$p_\perp$ data. *Phys. Rev. C*, 100(3):031901, 2019.

[113] Carlota Andres, Néstor Armesto, Harri Niemi, Risto Paatelainen, and Carlos A. Salgado. Jet quenching as a probe of the initial stages in heavy-ion collisions. *Phys. Lett. B*, 803:135318, 2020.

[114] G. Baym and S. A. Chin. Can a Neutron Star Be a Giant MIT Bag? *Phys. Lett. B*, 62:241–244, 1976.

[115] Edward V. Shuryak. What RHIC experiments and theory tell us about properties of quark-gluon plasma? *Nucl. Phys. A*, 750:64–83, 2005.

[116] Edward Shuryak. Strongly coupled quark-gluon plasma in heavy ion collisions. *Rev. Mod. Phys.*, 89:035001, 2017.

[117] Barbara Jacak and Peter Steinberg. Creating the perfect liquid in heavy-ion collisions. *Phys. Today*, 63N5:39–43, 2010.

[118] Berndt Muller, Jurgen Schukraft, and Boleslaw Wyslouch. First Results from Pb+Pb collisions at the LHC. *Ann. Rev. Nucl. Part. Sci.*, 62:361–386, 2012.

[119] Measurement of nuclear modification factor R_{AA} in Pb+Pb collisions at $\sqrt{s_{NN}} = 5.02$TeV with the ATLAS detector at the LHC. 2 2017.

[120] Syaefudin Jaelani. Measurement of the D-meson Nuclear Modification Factor and Elliptic Flow in Pb–Pb Collisions at $\sqrt{s}_{NN} = 5.02$ TeV with ALICE at the LHC. *Int. J. Mod. Phys. Conf. Ser.*, 46:1860018, 2018.

[121] Betty Abelev et al. Suppression of high transverse momentum D mesons in central Pb-Pb collisions at $\sqrt{s_{NN}} = 2.76$ TeV. *JHEP*, 09:112, 2012.

[122] A. Adare et al. Suppression pattern of neutral pions at high transverse momentum in Au+Au collisions at $\sqrt{s_{NN}} = 200$ GeV and constraints on medium transport coefficients. *Phys. Rev. Lett.*, 101:232301, 2008.

[123] A. Adare et al. Neutral pion production with respect to centrality and reaction plane in Au+Au collisions at $\sqrt{s_{NN}}$=200 GeV. *Phys. Rev. C*, 87(3):034911, 2013.

[124] B. I. Abelev et al. Energy dependence of pi+-, p and anti-p transverse momentum spectra for Au+Au collisions at s(NN)**(1/2) = 62.4 and 200-GeV. *Phys. Lett. B*, 655:104–113, 2007.

[125] S. Acharya et al. Energy dependence and fluctuations of anisotropic flow in Pb-Pb collisions at $\sqrt{s_{\rm NN}} = 5.02$ and 2.76 TeV. *JHEP*, 07:103, 2018.

[126] Morad Aaboud et al. Measurement of the azimuthal anisotropy of charged particles produced in $\sqrt{s_{\rm NN}}$ = 5.02 TeV Pb+Pb collisions with the ATLAS detector. *Eur. Phys. J. C*, 78(12):997, 2018.

[127] A. M. Sirunyan et al. Azimuthal anisotropy of charged particles with transverse momentum up to 100 GeV/ c in PbPb collisions at $\sqrt{s}_{NN}$=5.02 TeV. *Phys. Lett. B*, 776:195–216, 2018.

[128] Betty Bezverkhny Abelev et al. Azimuthal anisotropy of D meson production in Pb-Pb collisions at $\sqrt{s_{NN}} = 2.76$ TeV. *Phys. Rev. C*, 90(3):034904, 2014.

[129] Albert M Sirunyan et al. Measurement of prompt D^0 meson azimuthal anisotropy in Pb-Pb collisions at $\sqrt{s_{NN}}$ = 5.02 TeV. *Phys. Rev. Lett.*, 120(20):202301, 2018.

[130] Shreyasi Acharya et al. D-meson azimuthal anisotropy in midcentral Pb-Pb collisions at $\sqrt{\mathbf{s_{NN}}} = \mathbf{5.02}$ TeV. *Phys. Rev. Lett.*, 120(10):102301, 2018.

[131] Georges Aad et al. Measurement of the pseudorapidity and transverse momentum dependence of the elliptic flow of charged particles in lead-lead collisions at $\sqrt{s_{NN}} = 2.76$ TeV with the ATLAS detector. *Phys. Lett. B*, 707:330–348, 2012.

[132] Serguei Chatrchyan et al. Azimuthal Anisotropy of Charged Particles at High Transverse Momenta in PbPb Collisions at $\sqrt{s_{NN}} = 2.76$ TeV. *Phys. Rev. Lett.*, 109:022301, 2012.

[133] Magdalena Djordjevic, Marko Djordjevic, and Bojana Blagojevic. RHIC and LHC jet suppression in non-central collisions. *Phys. Lett. B*, 737:298–302, 2014.

[134] Magdalena Djordjevic. Heavy flavor puzzle at LHC: a serendipitous interplay of jet suppression and fragmentation. *Phys. Rev. Lett.*, 112(4):042302, 2014.

[135] Jacquelyn Noronha-Hostler, Barbara Betz, Jorge Noronha, and Miklos Gyulassy. Event-by-event hydrodynamics + jet energy loss: A solution to the $R_{AA} \otimes v_2$ puzzle. *Phys. Rev. Lett.*, 116(25):252301, 2016.

[136] Denes Molnar and Deke Sun. High-pT suppression and elliptic flow from radiative energy loss with realistic bulk medium expansion. 5 2013.

[137] Michel Le Bellac. *Thermal Field Theory*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 3 2011.

[138] J. D. Bjorken. Highly Relativistic Nucleus-Nucleus Collisions: The Central Rapidity Region. *Phys. Rev. D*, 27:140–151, 1983.

[139] Eric Braaten, King-man Cheung, Sean Fleming, and Tzu Chiang Yuan. Perturbative QCD fragmentation functions as a model for heavy quark fragmentation. *Phys. Rev. D*, 51:4819–4829, 1995.

[140] V. G. Kartvelishvili, A. K. Likhoded, and V. A. Petrov. On the Fragmentation Functions of Heavy Quarks Into Hadrons. *Phys. Lett. B*, 78:615–617, 1978.

[141] Simon Wicks, William Horowitz, Magdalena Djordjevic, and Miklos Gyulassy. Elastic, inelastic, and path length fluctuations in jet tomography. *Nucl. Phys. A*, 784:426–442, 2007.

[142] Guy D. Moore and Derek Teaney. How much do heavy quarks thermalize in a heavy ion collision? *Phys. Rev. C*, 71:064904, 2005.

[143] Alexei Selikhov and Miklos Gyulassy. Color diffusion and conductivity in a quark - gluon plasma. *Phys. Lett. B*, 316:373–380, 1993.

[144] A. V. Selikhov and M. Gyulassy. QCD Fokker-Planck equations with color diffusion. *Phys. Rev. C*, 49:1726–1729, 1994.

[145] A. Peshier. Running coupling and screening in the (s)QGP. 1 2006.

[146] Peter F. Kolb and Ulrich W. Heinz. Hydrodynamic description of ultrarelativistic heavy ion collisions. pages 634–714, 5 2003.

[147] Jonah E. Bernhard, J. Scott Moreland, and Steffen A. Bass. Characterization of the initial state and QGP medium from a combined Bayesian analysis of LHC data at 2.76 and 5.02 TeV. *Nucl. Phys. A*, 967:293–296, 2017.

[148] Magdalena Djordjevic, Miklos Gyulassy, Ramona Vogt, and Simon Wicks. Influence of bottom quark jet quenching on single electron tomography of Au + Au. *Phys. Lett. B*, 632:81–86, 2006.

[149] A. Bazavov et al. Equation of state in (2+1)-flavor QCD. *Phys. Rev. D*, 90:094503, 2014.

[150] Martin Wilde. Measurement of Direct Photons in pp and Pb-Pb Collisions with ALICE. *Nucl. Phys. A*, 904-905:573c–576c, 2013.

[151] Shreyasi Acharya et al. Transverse momentum spectra and nuclear modification factors of charged particles in Xe-Xe collisions at $\sqrt{s_{NN}}$ = 5.44 TeV. *Phys. Lett. B*, 788:166–179, 2019.

[152] Charged hadron spectra and dijet p_{T} correlations measured in Xe+Xe collisions at $\sqrt{s_{\mathrm{NN}}} = 5.44$TeV with the ATLAS detector. 5 2018.

[153] Charged-particle nuclear modification factors in XeXe collisions at $\sqrt{s_{\mathrm{NN}}} = 5.44$ TeV. 2018.

[154] Magdalena Djordjevic, Bojana Blagojevic, and Lidija Zivkovic. Mass tomography at different momentum ranges in quark-gluon plasma. *Phys. Rev. C*, 94(4):044908, 2016.

[155] Mihee Jo. Suppression of open bottom at high p_T via non-prompt J/ψ decays in PbPb collisions at 2.76 TeV with CMS. *Nucl. Phys. A*, 904-905:657c–660c, 2013.

[156] Jiechen Xu, Jinfeng Liao, and Miklos Gyulassy. Consistency of Perfect Fluidity and Jet Quenching in semi-Quark-Gluon Monopole Plasmas. *Chin. Phys. Lett.*, 32(9):092501, 2015.

[157] Shuzhe Shi, Jinfeng Liao, and Miklos Gyulassy. Probing the Color Structure of the Perfect QCD Fluids via Soft-Hard-Event-by-Event Azimuthal Correlations. *Chin. Phys. C*, 42(10):104104, 2018.

[158] Jing Wang. D meson nuclear modification factor in PbPb at 5.02 TeV with CMS. *Nucl. Part. Phys. Proc.*, 289-290:249–252, 2017.

[159] Ta-Wei Wang. B meson nuclear modification factor in Pb-Pb at 5.02 TeV with CMS. *Nucl. Part. Phys. Proc.*, 289-290:229–232, 2017.

[160] Francois Gelis and Bjoern Schenke. Initial State Quantum Fluctuations in the Little Bang. *Ann. Rev. Nucl. Part. Sci.*, 66:73–94, 2016.

[161] Georges Aad et al. Measurement of the distributions of event-by-event flow harmonics in lead-lead collisions at = 2.76 TeV with the ATLAS detector at the LHC. *JHEP*, 11:183, 2013.

[162] H. Niemi, G. S. Denicol, H. Holopainen, and P. Huovinen. Event-by-event distributions of azimuthal asymmetries in ultrarelativistic heavy-ion collisions. *Phys. Rev. C*, 87(5):054901, 2013.

[163] Roland Katz, Caio A. G. Prado, Jacquelyn Noronha-Hostler, Jorge Noronha, and Alexandre A. P. Suaide. Sensitivity study with a D and B mesons modular simulation code of heavy flavor RAA and azimuthal anisotropies based on beam energy, initial conditions, hadronization, and suppression mechanisms. *Phys. Rev. C*, 102(2):024906, 2020.

[164] Shuzhe Shi, Jinfeng Liao, and Miklos Gyulassy. Global constraints from RHIC and LHC on transport properties of QCD fluids in CUJET/CIBJET framework. *Chin. Phys. C*, 43(4):044101, 2019.

[165] Santosh K. Das, Francesco Scardina, Salvatore Plumari, and Vincenzo Greco. Toward a solution to the R_{AA} and v_2 puzzle for heavy quarks. *Phys. Lett. B*, 747:260–264, 2015.

[166] R. Baier, Yuri L. Dokshitzer, Alfred H. Mueller, S. Peigne, and D. Schiff. Radiative energy loss and p(T) broadening of high-energy partons in nuclei. *Nucl. Phys. B*, 484:265–282, 1997.

[167] R. D. Field. *Applications of Perturbative QCD*, volume 77. 1989.

[168] Peter Christiansen, Konrad Tywoniuk, and Vytautas Vislavicius. Universal scaling dependence of QCD energy loss from data driven studies. *Phys. Rev. C*, 89(3):034912, 2014.

[169] Shanshan Cao, Long-Gang Pang, Tan Luo, Yayun He, Guang-You Qin, and Xin-Nian Wang. R_{AA} vs. v_2 of heavy and light hadrons within a linear Boltzmann transport model. *Nucl. Part. Phys. Proc.*, 289-290:217–220, 2017.

[170] S. Cao et al. Multistage Monte-Carlo simulation of jet modification in a static medium. *Phys. Rev. C*, 96(2):024909, 2017.

[171] R. Stock. The Physics of Dense Nuclear Matter From Supernovae to Quark Gluon Plasma. *Nature*, 337:319–324, 1989.

[172] Johanna Stachel. Has the quark-gluon plasma been seen? *Int. J. Mod. Phys. A*, 21:1750–1763, 2006.

[173] Paul Romatschke and Ulrike Romatschke. Viscosity Information from Relativistic Nuclear Collisions: How Perfect is the Fluid Observed at RHIC? *Phys. Rev. Lett.*, 99:172301, 2007.

[174] Ulrich Heinz and Raimond Snellings. Collective flow and viscosity in relativistic heavy-ion collisions. *Ann. Rev. Nucl. Part. Sci.*, 63:123–151, 2013.

[175] John Adams et al. Experimental and theoretical challenges in the search for the quark gluon plasma: The STAR Collaboration's critical assessment of the evidence from RHIC collisions. *Nucl. Phys. A*, 757:102–183, 2005.

[176] K. Adcox et al. Formation of dense partonic matter in relativistic nucleus-nucleus collisions at RHIC: Experimental evaluation by the PHENIX collaboration. *Nucl. Phys. A*, 757:184–283, 2005.

[177] Georges Aad et al. Observation of a Centrality-Dependent Dijet Asymmetry in Lead-Lead Collisions at $\sqrt{s_{NN}} = 2.77$ TeV with the ATLAS Detector at the LHC. *Phys. Rev. Lett.*, 105:252303, 2010.

[178] K. Aamodt et al. Suppression of Charged Particle Production at Large Transverse Momentum in Central Pb-Pb Collisions at $\sqrt{s_{NN}} = 2.76$ TeV. *Phys. Lett. B*, 696:30–39, 2011.

[179] Serguei Chatrchyan et al. Observation and studies of jet quenching in PbPb collisions at nucleon-nucleon center-of-mass energy = 2.76 TeV. *Phys. Rev. C*, 84:024906, 2011.

[180] James L. Nagle, Ian G. Bearden, and William A. Zajc. Quark-Gluon Plasma at RHIC and the LHC: Perfect Fluid too Perfect? *New J. Phys.*, 13:075004, 2011.

[181] Jussi Auvinen, Kari J. Eskola, Pasi Huovinen, Harri Niemi, Risto Paatelainen, and Peter Petreczky. Temperature dependence of η/s of strongly interacting matter: Effects of the equation of state and the parametric form of $(\eta/s)(T)$. *Phys. Rev. C*, 102(4):044911, 2020.

[182] J. D. Orjuela Koop, A. Adare, D. McGlinchey, and J. L. Nagle. Azimuthal anisotropy relative to the participant plane from a multiphase transport model in central p + Au , d + Au , and ^{3}He + Au collisions at $\sqrt{s_{NN}} = 200$ GeV. *Phys. Rev. C*, 92(5):054903, 2015.

[183] Jussi Auvinen, Jonah E. Bernhard, Steffen A. Bass, and Iurii Karpenko. Investigating the collision energy dependence of η/s in the beam energy scan at the BNL Relativistic Heavy Ion Collider using Bayesian statistics. *Phys. Rev. C*, 97(4):044905, 2018.

[184] Stefan Stojku, Bojana Ilic, Marko Djordjevic, and Magdalena Djordjevic. Extracting the temperature dependence in high-$p_\perp$ particle energy loss. *Phys. Rev. C*, 103(2):024908, 2021.

[185] Patrick. Aurenche, Francois Gelis, and Haitham Zaraket. A Simple sum rule for the thermal gluon spectral function and applications. *JHEP*, 05:043, 2002.

[186] Stephane Peigne and Andre Peshier. Collisional energy loss of a fast heavy quark in a quark-gluon plasma. *Phys. Rev. D*, 77:114017, 2008.

[187] Stefan Stojku, Jussi Auvinen, Marko Djordjevic, Pasi Huovinen, and Magdalena Djordjevic. Early evolution constrained by high-p⊥ quark-gluon plasma tomography. *Phys. Rev. C*, 105(2):L021901, 2022.

[188] Stefan Stojku, Jussi Auvinen, Lidija Zivkovic, Pasi Huovinen, and Magdalena Djordjevic. Jet-perceived anisotropy revealed through high-p⊥ data. *Phys. Lett. B*, 835:137501, 2022.

[189] B. G. Zakharov. Radiative energy loss of high-energy quarks in finite size nuclear matter and quark - gluon plasma. *JETP Lett.*, 65:615–620, 1997.

[190] Peter Brockway Arnold, Guy D. Moore, and Laurence G. Yaffe. Photon emission from quark gluon plasma: Complete leading order results. *JHEP*, 12:009, 2001.

[191] Dusan Zigic. DREENA-A. https://github.com/DusanZigic/DREENA-A/, 2022.

[192] Matteo Cacciari, Stefano Frixione, Nicolas Houdeau, Michelangelo L. Mangano, Paolo Nason, and Giovanni Ridolfi. Theoretical predictions for charm and bottom production at the LHC. *JHEP*, 10:137, 2012.

[193] M. Djordjevic. Heavy quark energy loss: Collisional vs radiative. *Nucl. Phys. A*, 783:197–204, 2007.

[194] Michael Fickinger, Grigory Ovanesyan, and Ivan Vitev. Angular distributions of higher order splitting functions in the vacuum and in dense QCD matter. *JHEP*, 07:059, 2013.

[195] Matthew Luzum and Hannah Petersen. Initial State Fluctuations and Final State Correlations in Relativistic Heavy-Ion Collisions. *J. Phys. G*, 41:063102, 2014.

[196] Etele Molnar, Hannu Holopainen, Pasi Huovinen, and Harri Niemi. Influence of temperature-dependent shear viscosity on elliptic flow at backward and forward rapidities in ultrarelativistic heavy-ion collisions. *Phys. Rev. C*, 90(4):044904, 2014.

[197] Pasi Huovinen and Pter Petreczky. QCD Equation of State and Hadron Resonance Gas. *Nucl. Phys. A*, 837:26–53, 2010.

[198] J. Scott Moreland, Jonah E. Bernhard, and Steffen A. Bass. Alternative ansatz to wounded nucleon and binary collision scaling in high-energy nuclear collisions. *Phys. Rev. C*, 92(1):011901, 2015.

[199] Huichao Song and Ulrich W. Heinz. Causal viscous hydrodynamics in 2+1 dimensions for relativistic heavy-ion collisions. *Phys. Rev. C*, 77:064901, 2008.

[200] Jonah E. Bernhard. *Bayesian parameter estimation for relativistic heavy-ion collisions*. PhD thesis, Duke U., 4 2018.

[201] Jonah E. Bernhard, J. Scott Moreland, and Steffen A. Bass. Bayesian estimation of the specific shear and bulk viscosity of quark–gluon plasma. *Nature Phys.*, 15(11):1113–1117, 2019.

[202] D. Everett et al. Multisystem Bayesian constraints on the transport coefficients of QCD matter. *Phys. Rev. C*, 103(5):054904, 2021.

[203] S. Acharya et al. Measurement of D^0, D^+, D^{*+} and D_s^+ production in Pb-Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV. *JHEP*, 10:174, 2018.

[204] Xinye Peng. Beauty production in heavy-ion collisions with ALICE at the LHC. 7 2022.

[205] Yongsun Kim. Highlights from the CMS experiment. 2022.

[206] A. Adare et al. Azimuthal anisotropy of neutral pion production in Au+Au collisions at $\sqrt{(s_N N)}$ = 200 GeV: Path-length dependence of jet quenching and the role of initial geometry. *Phys. Rev. Lett.*, 105:142301, 2010.

[207] J. Adams et al. Transverse momentum and collision energy dependence of high p(T) hadron suppression in Au+Au collisions at ultrarelativistic energies. *Phys. Rev. Lett.*, 91:172302, 2003.

[208] B. I. Abelev et al. Centrality dependence of charged hadron and strange hadron elliptic flow from s(NN)**(1/2) = 200-GeV Au + Au collisions. *Phys. Rev. C*, 77:054901, 2008.

[209] L. Adamczyk et al. Observation of D^0 Meson Nuclear Modifications in Au+Au Collisions at $\sqrt{s_{NN}} = 200$ GeV. *Phys. Rev. Lett.*, 113(14):142301, 2014. [Erratum: Phys.Rev.Lett. 121, 229901 (2018)].

[210] L. Adamczyk et al. Measurement of D^0 Azimuthal Anisotropy at Midrapidity in Au+Au Collisions at $\sqrt{s_{NN}}$=200 GeV. *Phys. Rev. Lett.*, 118(21):212301, 2017.

[211] Takashi Hachiya. Charm and Bottom quark energy loss and flow measurements in Au+Au collisions by the PHENIX experiment. 2022.

[212] sPHENIX Beam Use Proposal. 2021.

[213] BUR Committee. The STAR Beam Use Request for Run-22 and data taking in 2023-25. 2021.

[214] Stephane Fartoukh et al. LHC Configuration and Operational Scenario for Run 3.

[215] Yayun He, Wei Chen, Tan Luo, Shanshan Cao, Long-Gang Pang, and Xin-Nian Wang. Event-by-event jet anisotropy and hard-soft tomography of the quark-gluon plasma. *Phys. Rev. C*, 106(4):044904, 2022.

[216] O. Kaczmarek, F. Karsch, F. Zantow, and P. Petreczky. Static quark anti-quark free energy and the running coupling at finite temperature. *Phys. Rev. D*, 70:074505, 2004. [Erratum: Phys.Rev.D 72, 059903 (2005)].

[217] Olaf Kaczmarek and Felix Zantow. Static quark anti-quark interactions in zero and finite temperature QCD. I. Heavy quark free energies, running coupling and quarkonium binding. *Phys. Rev. D*, 71:114510, 2005.

[218] Jaroslav Adam et al. Centrality Dependence of the Charged-Particle Multiplicity Density at Midrapidity in Pb-Pb Collisions at $\sqrt{\mathrm{s_{NN}}}$ = 5.02 TeV. *Phys. Rev. Lett.*, 116(22):222302, 2016.

[219] Jaroslav Adam et al. Anisotropic flow of charged particles in Pb-Pb collisions at $\sqrt{\mathrm{s_{NN}}} = 5.02$ TeV. *Phys. Rev. Lett.*, 116(13):132302, 2016.

[220] Bjoern Schenke, Chun Shen, and Prithwish Tribedy. Running the gamut of high energy nuclear collisions. *Phys. Rev. C*, 102(4):044905, 2020.

[221] Chun Shen. private communication. 2020.

[222] Bjoern Schenke, Sangyong Jeon, and Charles Gale. (3+1)D hydrodynamic simulation of relativistic heavy-ion collisions. *Phys. Rev. C*, 82:014903, 2010.

[223] Bjorn Schenke, Sangyong Jeon, and Charles Gale. Elliptic and triangular flow in event-by-event (3+1)D viscous hydrodynamics. *Phys. Rev. Lett.*, 106:042301, 2011.

[224] Bjorn Schenke, Sangyong Jeon, and Charles Gale. Higher flow harmonics from (3+1)D event-by-event viscous hydrodynamics. *Phys. Rev. C*, 85:024901, 2012.

[225] J. Scott Moreland and Ron A. Soltz. Hydrodynamic simulations of relativistic heavy-ion collisions with different lattice quantum chromodynamics calculations of the equation of state. *Phys. Rev. C*, 93(4):044913, 2016.

[226] Ante Bilandzic, Raimond Snellings, and Sergei Voloshin. Flow analysis with cumulants: Direct calculations. *Phys. Rev. C*, 83:044913, 2011.

[227] Jean-François Paquet, Chun Shen, Gabriel S. Denicol, Matthew Luzum, Björn Schenke, Sangyong Jeon, and Charles Gale. Production of photons in relativistic heavy-ion collisions. *Phys. Rev. C*, 93(4):044906, 2016.

[228] Sergei A. Voloshin, Arthur M. Poskanzer, Aihong Tang, and Gang Wang. Elliptic flow in the Gaussian model of eccentricity fluctuations. *Phys. Lett. B*, 659:537–541, 2008.

[229] Jiangyong Jia and Peng Huo. A method for studying the rapidity fluctuation and decorrelation of harmonic flow in heavy-ion collisions. *Phys. Rev. C*, 90(3):034905, 2014.

[230] Vardan Khachatryan et al. Evidence for transverse momentum and pseudorapidity dependent event plane fluctuations in PbPb and pPb collisions. *Phys. Rev. C*, 92(3):034911, 2015.

[231] Albert M Sirunyan et al. Measurement of prompt D^0 and $\overline{\mathrm{D}}^0$ meson azimuthal anisotropy and search for strong electric fields in PbPb collisions at $\sqrt{s_{\mathrm{NN}}} = 5.02$ TeV. *Phys. Lett. B*, 816:136253, 2021.

[232] Shreyasi Acharya et al. Prompt D^0, D^+, and D^{*+} production in Pb–Pb collisions at $\sqrt{s_{\mathrm{NN}}}$ = 5.02 TeV. *JHEP*, 01:174, 2022.

[233] Shreyasi Acharya et al. Transverse-momentum and event-shape dependence of D-meson flow harmonics in Pb–Pb collisions at $\sqrt{s_{NN}}$ = 5.02 TeV. *Phys. Lett. B*, 813:136054, 2021.

Biography of the author

Dušan Žigić was born on October 19, 1991. He graduated from Mitrovačka Gimnazija in 2010 and pursued higher education at the Faculty of Physics, University of Belgrade in 2012. He earned a bachelor's degree in applied and computational physics in 2017 with a GPA of 9.75/10.0.

Building on this foundation, he completed his master's studies in theoretical and experimental physics in 2018 with a GPA of 9.67/10.0. His master's thesis was recognized with the prestigious Prof. Dr. Ljubomir Ćirković Award for the best master's thesis.

In 2018, he began PhD studies in the field of heavy-ion collisions, working under the guidance of Dr. Magdalena Djordjević and Dr. Igor Salom. During his PhD, he delivered 14 talks, including one invited talk, at international conferences, presented 5 postersa and he also participated in 6 specialized schools. He authored 11 publications in heavy-ion physics and computational biology. Additionally, he served as a journal referee for Physical Review C.

Journal articles:

A. **Theoretical high energy physics**

1 D. Zigic, I. Salom, J. Auvinen, M. Djordjevic and M. Djordjevic, *DREENA-C framework: joint R_{AA} and v_2 predictions and implications to QGP tomography*, J. Phys. G **46**, no. 8, 085101 (2019).

2 M. Djordjevic, D. Zigic, M. Djordjevic and J. Auvinen, *How to test path-length dependence in energy loss mechanisms: analysis leading to a new observable*, Phys. Rev. C **99**, no. 6, 061902(R) (2019).

3 D. Zigic, I. Salom, J. Auvinen, M. Djordjevic and M. Djordjevic, *DREENA-B framework: first predictions of R_{AA} and v_2 within dynamical energy loss formalism in evolving QCD medium*, Phys. Lett. B **791**, 236 (2019).

4 D. Zigic, B. Ilic, M. Djordjevic and M. Djordjevic, *Exploring the initial stages in heavy-ion collisions with high-$p_\perp$ R_{AA} and v_2 theory and data*, Phys. Rev. C **101**, no. 6, 064909 (2020).

5 D. Zigic, I. Salom, J. Auvinen, P. Huovinen and M. Djordjevic, *DREENA-A framework as a QGP tomography tool*, Front. Phys. **10**:957019 (2022).

6 D. Zigic, J. Auvinen, I. Salom, M. Djordjevic and P. Huovinen, *Importance of higher harmonics and v_4 puzzle in quark-gluon plasma tomography*, Phys. Rev. C **106**, no.4, 044909 (2022).

7 B. Karmakar, D. Zigic, I. Salom, J. Auvinen, P. Huovinen, M. Djordjevic and M. Djordjevic, *Constraining η/s through high-$p\perp$ theory and data*, Phys. Rev. C **108**, no.4, 044907 (2023)

8 B. Karmakar, D. Zigic, M. Djordjevic, P. Huovinen, M. Djordjevic and J. Auvinen, *Probing the shape of the quark-gluon plasma droplet via event-by-event quark-gluon plasma tomography*, Phys. Rev. C **110**, no.4, 044906 (2024)

B. **Computational systems biology**

9 M. Djordjevic, A. Rodic, I. Salom, D. Zigic, O. Milicevic, B. Ilic, M. Djordjevic, *A systems biology approach to COVID-19 progression in population*, Advances in Protein Chemistry and Structural Biology, **127**, 291 (2021).

10 I. Salom, A. Rodic, O. Milicevic, D. Zigic, M. Djordjevic, M. Djordjevic, *Effects of demographic and weather parameters on COVID-19 basic reproduction number*, Frontiers in Ecology and Evolution **8**, 524, (2021).

11 O. Milicevic, I. Salom, M. Tumbas, A. Rodic, S. Markovic, D. Zigic, M. Djordjevic, M. Djordjevic, *PM2.5 as a major predictor of COVID-19 basic reproduction number in the USA*, Environmental Research, **201**, 111526 (2021).

Изјава о ауторству

Име и презиме аутора – Душан Жигић

Број индекса – 8014/2018

Изјављујем

да је докторска дисертација под насловом

Development of the DREENA model for quark-gluon plasma tomography

(Развој ДРЕЕНА модела за томографију кварк-глуонске плазме)

- резултат сопственог истраживачког рада;
- да дисертација у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршила ауторска права и користила интелектуалну својину других лица.

У Београду, 2024 Потпис аутора

_____

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора – Душан Жигић

Број индекса – 8014/2018

Студијски програм – Физика високих енергија и нуклеарна физика

Наслов рада – **Development of the DREENA model for quark-gluon plasma tomography**

(Развој ДРЕЕНА модела за томографију кварк-глуонске плазме)

Ментори – др Магдалена Ђорђевић и др Игор Салом

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао ради похрањивања у Дигиталном репозиторијуму Универзитета у Београду.

Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

У Београду, 2024 Потпис аутора

_____

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Development of the DREENA model for quark-gluon plasma tomography

(Развој ДРЕЕНА модела за томографију кварк-глуонске плазме)

која је моје ауторско дело.

Дисертацију са свим прилозима предао сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио.

1. Ауторство (CC BY)
2. Ауторство – некомерцијално (CC BY-NC)
3. Ауторство – некомерцијално – без прерада (CC BY-NC-ND)
4. Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)
5. Ауторство – без прерада (CC BY-ND)
6. Ауторство – делити под истим условима (CC BY-SA)

(Молимо да заокружите само једну од шест понуђених лиценци.
Кратак опис лиценци је саставни део ове изјаве).

У Београду, 2024 Потпис аутора

_____

1. Ауторство. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. Ауторство – некомерцијално. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. Ауторство – некомерцијално – без прерада. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. Ауторство – некомерцијално – делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. Ауторство – без прерада. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. Ауторство – делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.