Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский ядерный университет «МИФИ»

УДК 539.12.01

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

РЕКОНСТРУКЦИЯ ТРЕКОВ ЧАСТИЦ В МАГНИТНОМ ПОЛЕ С ПОМОЩЬЮ ГРАФОВОЙ НЕЙРОННОЙ СЕТИ (GNN) ДЛЯ ЭКСПЕРИМЕНТА BM@N

Научный руководитель
к.т.н _____ К. В. Герценбергер
Студент _____ Д. Н. Джавадов

Москва 2026

Содержание

Перечень сокращений и обозначений

В настоящем отчете о НИР применяют следующие сокращения и обозначения:

GAT – Graph Attention Network (графовая сеть внимания)

GNN – Graph Neural Network (графовая нейронная сеть);

CUDA – Compute Unified Device Architecture (параллельная вычислительная платформа NVIDIA);

Numba – JIT-компилятор для ускорения Python-кода;

Optuna – фреймворк для автоматической оптимизации гиперпараметров;

ML – Machine Learning (машинное обучение)

DL – Deep Learning (глубокое обучение)

Введение

Эксперимент BM@N (Baryonic Matter at Nuclotron) в рамках мегасайенс комплекса NICA (Nuclotron-based Ion Collider fAcility) в ОИЯИ (Объединенный институт ядерных исследований) направлен на исследование свойств ядерной материи при экстремальных плотностях, создаваемых в столкновениях пучков тяжелых ионов с фиксированной мишенью [1]. Получение физических результатов напрямую зависит от точности и эффективности обработки сырых данных, регистрируемых сложной системой детекторов.

Одним из важных и нетривиальных этапов при обработке полученных данных эксперимента является реконструкция треков — восстановление траектории заряженных частиц по координатным сигналам (хитам), оставленным в различных слоях детекторной системы. Различные эксперименты используют свои индивидуальные подходы к решению этой задачи: к примеру, на эксперименте BM@N используются методы, основанные на геометрических и комбинаторных принципах, а также разрабатывается метод реконструкции с помощью применения машинного (ML) и глубокого обучения (DL) [2, 3, 4], о чем пойдет речь в данной работе.

Методы, основанные на разработке нейронных сетей, являются перспективным направлением, позволяющим как сравнивать эффективность различных подходов, так и значительно ускорить обработку данных, вплоть до использования в онлайн-системе эксперимента. Этот подход позволяет не явно программировать правила связывания хитов, а обучить модель распознавать сложные паттерны и закономерности. В данной работе речь пойдет о графовой нейронной сети (GNN). Она позволяет напрямую представить набор хитов как набор вершин графа, а задачу поиска треков — как задачу бинарной классификации ребер (связей между вершинами-хитами).

О прошлом семестре. По итогам осеннего семестра (2025) был разработан дальнейший план разработки графовой нейросети для задачи реконструкции треков заряженных частиц в магнитном поле. Основные задачи, решаемые в ходе проекта:

- генерация модельных событий столкновения частиц с фиксированной мишенью с использованием программного комплекса BmnRoot с последующим анализом датасета;
- учет физических параметров, связанных с кинематикой частиц в магнитном поле;
- построение графа с физическими ограничениями;
- реализация классификатора ребер с механизмом внимания (GAT) [5];
- анализ метрик для задачи бинарной классификации с редкими классами;
- использование пакета Optuna для подбора гиперпараметров модели и модуля построения графов;
- оптимизация разработанного решения (CUDA, Numba);
- анализ метрик на уровне ребер и треков.

В весеннем семестре были поставлены следующие задачи:

- разработать и реализовать конвейер обработки данных и инференса;
- реализовать физически мотивированное построение графов;

- вычислить базовые метрики качества на уровне ребер графа;
- провести локальную оптимизацию разработанного решения;
- с использованием пакета Optuna выполнить подбор гиперпараметров модели;
- в рамках отдельного эксперимента сравнить две функции потерь, зафиксировав одинаковые условия валидации и число итераций Optuna для каждой из них;
- по результатам сравнения выбрать лучшую функцию потерь, обучить финальную модель и оценить ее качество на уровне треков.

Цель научно-исследовательской работы — разработка и экспериментальная оценка графовой нейронной сети с механизмом внимания (GATv2) для реконструкции треков заряженных частиц в магнитном поле детектора BM@N в условиях большого количества ложных треков.

1 Постановка задачи

1.1 Геометрия детекторной установки

Экспериментальная установка BM@N представляет собой спектрометр, регистрирующий частицы, летящие вперед (под малыми углами к оси пучка), с фиксированной мишенью, работающий на пучках тяжелых ионов ускорительного комплекса NICA [1]. Спектрометр перекрывает область псевдобыстрот $1.6 \leq \eta \leq 4.4$ и включает в себя систему детекторов, расположенных как внутри, так и за пределами анализирующего магнита SP-41 (рис.1).

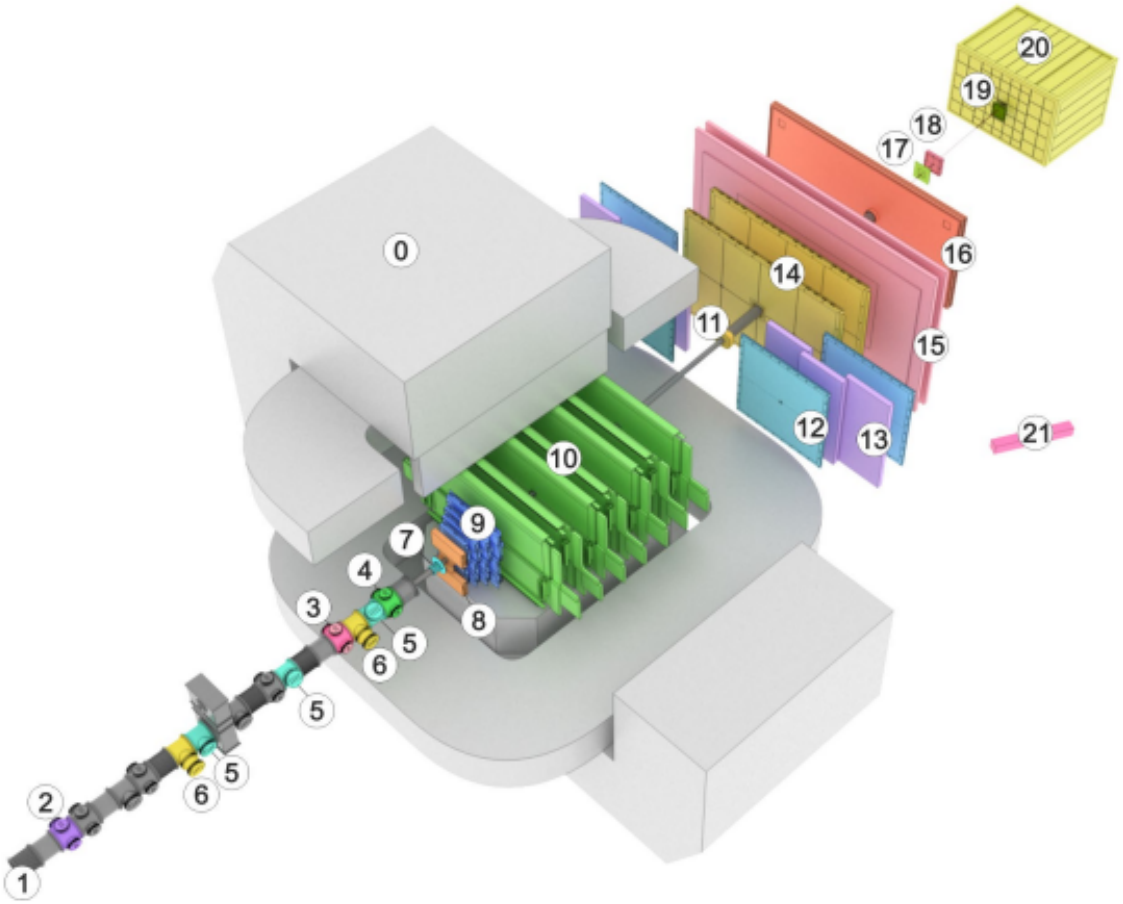

Рис. 1: Схема экспериментальной установки BM@N: (0) анализирующий магнит SP-41; (1) вакуумная труба пучка; (2) сцинтилляционный счетчик BC1; (3) сцинтилляционный счетчик VC; (4) сцинтилляционный счетчик BC2; (5) кремниевый трекер пучка SiBT; (6) кремниевый профилометр SiProf; (7) баррельный детектор BD; (8) вакуумный стартовый детектор VSP; (9) кремниевый детектор FSD; (10) GEM-детекторы; (11) передний детектор FD; (12) четыре катодные камеры CSC (1×1 м^2); (13) времяпролетная система TOF400; (14) две катодные камеры CSC ($2 \times 1{,}5$ м^2); (15) времяпролетная система TOF700; (16) сцинтилляционная стена ScWall; (17) малые GEM-детекторы; (18) профилометр пучка; (19) кварцевый детектор FQH; (20) форвардный адронный калориметр FHCal; (21) детектор HGN.

Магнитное поле с индукцией B (конкретное значение определяется условиями экспе-

римента) создается дипольным магнитом с полюсным зазором 1.07 м и размерами полюсов 1.4×2.5 м^2 по осям X и Z соответственно [1, раздел 2.4]. Ось пучка проходит через центр магнита со смещением по вертикали на 40 мм в сторону нижнего полюса. Мишень устанавливается на входе в магнитное поле, на переднем крае полюса, что определяет начало отсчета продольной координаты $z = 0$.

Детекторная система, участвующая в реконструкции треков, включает следующие подсистемы (рис.1):

- **Центральная трековая система (CTS)** внутри магнита:
  - **Кремниевый детектор FSD** (Forward Silicon Detector) — *4 координатные плоскости на расстояниях* $z \approx 10$–30 *см от мишени*, непосредственно за ней. Это *первые детекторы*, в которых первичная заряженная частица оставляет хиты сразу после рождения, поэтому качество реконструкции начального участка трека критически зависит от пространственного разрешения FSD. Каждая плоскость разделена на верхнюю и нижнюю половины с мертвой зоной 57×57 мм^2 для вакуумной трубы пучка. Детектор выполнен на основе двухсторонних кремниевых стрип-детекторов (DSSD) с шагом полосок 95–103 мкм [1, раздел 5.1], что обеспечивает высокое пространственное разрешение (порядка десятков микрон) и позволяет точно измерить кривизну траектории на начальном участке.
  - **GEM-детекторы** (Gas Electron Multiplier) — 7 трековых плоскостей (14 детекторов) в интервале $z \approx 40$–110 см. Каждая плоскость состоит из верхнего и нижнего детекторов с активной площадью 163×45 см^2 и 163×39 см^2 соответственно.
- **Внешний трекер (Outer Tracker)** за магнитом ($z \gtrsim 300$ см):
  - **Катодные камеры (CSC)** — малые (1.1×1.1 м^2) и большие (2.2×1.5 м^2) с шагом катодных полосок 2.5 мм.
- **Системы идентификации частиц (TOF)**: TOF400 ($z \approx 4$ м) и TOF700 ($z \approx 7$ м).

Каждый хит, регистрируемый в этих системах, характеризуется не только пространственными координатами (x, y, z), но и дискретными индексами *station* (номер станции) и *module* (номер модуля), однозначно определяющими его положение в иерархической структуре детектора. Номера станций возрастают вдоль направления пучка ($+z$), что накладывает фундаментальное топологическое ограничение при построении ребер-кандидатов: связь возможна только между хитами с $station_j > station_i$ (движение только вперед), а максимальный пропуск станций ограничен величиной $\Delta station \leq 2$ (см. раздел 2.3). Это ограничение, совместно с геометрическим аксептансом детекторов по полярному углу θ, позволяет радикально сократить число ложных комбинаций еще до этапа физической проверки кривизны траектории в магнитном поле.

1.2 Физическая задача

Заряженная частица, движущаяся в магнитном поле B, совершает спиральную траекторию. Радиус кривизны определяется

$$R = \frac{p_T}{0{,}3 \cdot B \cdot |q|}, \tag{1}$$

где p_T — поперечный импульс частицы [ГэВ/с], B — магнитное поле [Тл], q — заряд частицы в единицах элементарного заряда.

При прохождение через детекторную систему частица оставляет набор сигналов – **хитов**. Задача реконструкции состоит в том, чтобы по множеству хитов восстановить исходные траектории частиц.

1.3 Формализация как задача на графе

Набор хитов одного события представляется как граф $G = (V, E)$, где:

- V — множество вершин, каждая вершина соответствует одному хиту;
- E — множество ребер-кандидатов между хитами, которые физически могут принадлежать одному треку.

Каждому ребру $(i, j) \in E$ присваивается метка:

$$y_{ij} = \begin{cases} 1, & \text{если хиты } i \text{ и } j \text{ принадлежат одному треку,} \\ 0, & \text{иначе.} \end{cases} \tag{2}$$

Формально, пусть $Y \in \{0, 1\}^{N \times N}$ — истинная матрица смежности, построенная на основе $trackID$ (меток истинных треков). Требуется найти такую бинарную матрицу предсказанных ребер $\hat{Y} \in \{0, 1\}^{N \times N}$, которая минимизирует расхождение с Y:

$$\hat{Y} = \arg\min_{Y' \in \mathcal{Y}} \mathcal{L}(Y', Y), \tag{3}$$

где $\mathcal{L}$ — функция потерь (например, бинарная кросс-энтропия(BCE) или фокальная (FL)). При этом накладывается дополнительное физическое ограничение: каждая связная компонента в графе $G' = (V, \hat{E})$, соответствующем матрице $\hat{Y}$, должна образовывать **физически допустимый трек** — то есть существует заряженная частица с некоторыми параметрами (p_T, q), траектория которой в магнитном поле B проходит через все хиты данной компоненты, а радиус кривизны $R = p_T/(0.3 \cdot B \cdot |q|)$ удовлетворяет геометрическому соотношению хорды и дуги для каждой последовательной пары хитов.

Таким образом, задача реконструкции сводится к **бинарной классификации ребер графа**. После отбора ребер с высокой вероятностью из оставшихся связей строятся треки.

1.4 Метрики качества

Оценка качества модели проводится на двух уровнях: на уровне отдельных ребер графа (классификация) и на уровне восстановленных треков (итоговый результат реконструкции). Поскольку задача характеризуется сильным *дисбалансом классов* — доля истинных ребер значительно меньше доли ложных, — для сравнения в работе используются две функции потерь: стандартная бинарная кросс-энтропия (BCE) и *Focal Loss*, позволяющая фокусироваться на трудно классифицируемых примерах.

1.4.1 Метрики на уровне ребер

Метрики данной группы вычисляются на основе матрицы ошибок бинарной классификации ребер, где TP (true positive) — число верно классифицированных истинных ребер, TN (true negative) — верно классифицированных ложных ребер, FP (false positive) — ложных ребер, ошибочно отнесенных к истинным, FN (false negative) — истинных ребер, ошибочно отнесенных к ложным.

Accuracy — доля верных предсказаний среди всех ребер:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{4}$$

При сильном дисбалансе классов данная метрика малоинформативна: классификатор, всегда предсказывающий «ложное ребро», уже достигает высокой accuracy за счет преобладающего класса.

Precision (точность) — доля истинных ребер среди всех ребер, отнесенных моделью к истинным:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{5}$$

Низкое значение precision означает, что среди отобранных ребер велика доля ложных, что увеличивает зашумленность графа перед этапом сборки треков.

Recall (полнота) — доля истинных ребер, успешно обнаруженных моделью, от общего числа истинных ребер:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{6}$$

Низкое значение recall означает потерю реальных связей между хитами одного трека, что напрямую снижает эффективность последующей реконструкции треков.

F1-score — гармоническое среднее precision и recall, дающее сбалансированную оценку при дисбалансе классов:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{7}$$

AUC-ROC — площадь под ROC-кривой, которая строится в координатах доли истинно положительных (TPR) и доли ложно положительных (FPR) предсказаний при варьировании порога классификации $t \in [0, 1]$:

$$\text{TPR}(t) = \frac{TP(t)}{TP(t) + FN(t)}, \qquad \text{FPR}(t) = \frac{FP(t)}{FP(t) + TN(t)}, \tag{8}$$

$$\text{AUC-ROC} = \int_0^1 \text{TPR}\big(\text{FPR}^{-1}(x)\big)\, dx. \tag{9}$$

Метрика AUC-ROC не зависит от выбора порога классификации и отражает способность модели в целом ранжировать истинные ребра выше ложных; значение 0,5 соответствует случайному классификатору, 1,0 — идеальному.

1.4.2 Метрики на уровне треков

Метрики на уровне треков — *purity* (чистота) и *efficiency* (эффективность) — характеризуют качество итоговой реконструкции и рассматриваются как цель последующих этапов работы, направленных на улучшение качества графовой нейронной сети. Достижение высоких значений данных метрик не являлось приоритетной задачей текущего этапа исследования.

Метрики вычисляются на основе сопоставления предсказанных треков T_k^{pred} ($k = 1, \ldots, N_{pred}$) с истинными треками T_m^{true} ($m = 1, \ldots, N_{true}$), где N_{pred} и N_{true} — количество предсказанных и истинных треков соответственно, а $|\cdot|$ обозначает число хитов в треке.

Track Purity. Для каждого предсказанного трека вычисляется доля хитов, совпадающих с наиболее перекрывающимся истинным треком:

$$\text{purity}_k = \frac{\max\limits_m \left| T_k^{pred} \cap T_m^{true} \right|}{\left| T_k^{pred} \right|}. \tag{10}$$

Итоговое значение метрики усредняется по всем предсказанным трекам:

$$\text{Track Purity} = \frac{1}{N_{pred}} \sum_{k=1}^{N_{pred}} \text{purity}_k. \tag{11}$$

Track Efficiency. Истинный трек считается успешно восстановленным, если хотя бы один предсказанный трек перекрывает его более чем на 50%:

$$\text{found}(T_m^{true}) = \begin{cases} 1, & \exists\, k : \ \dfrac{\left| T_k^{pred} \cap T_m^{true} \right|}{|T_m^{true}|} > 0.5, \\ 0, & \text{иначе.} \end{cases} \tag{12}$$

Эффективность реконструкции определяется как доля успешно восстановленных истинных треков:

$$\text{Track Efficiency} = \frac{1}{N_{true}} \sum_{m=1}^{N_{true}} \text{found}(T_m^{true}). \tag{13}$$

2 Данные и предобработка

2.1 Датасет

Для обучения и тестирования использовался смоделированный датасет, сгенерированный с помощью программного комплекса BmnRoot. В процессе моделирования задавались следующие параметры: тип генератора физических событий – DCMSMM, мишень CsI и пучок ионов Xe с энергией - $E = 3.0$ ГэВ/нуклон, а также количество сгенерированных событий (10000). Каждый хит описывается набором параметров представленных в таблице 1.

Таблица 1: Основные параметры хитов в датасете

Параметр	**Тип**	**Описание**
`eventID`	int	Номер события
`x, y, z`	float	Координаты хита частицы в детекторе [см]
`px, py, pz`	float	Импульс частицы в данном хите [ГэВ/с]
`station`	int	Номер станции детектора
`module`	int	Номер модуля внутри станции
`trackID`	int	ID трека (метка для обучения)
`particle_charge`	int	Заряд частицы
`primary`	int	Флаг первичной частицы (1/0)

Магнитное поле детектора составляет $B = 0.7$ Тл.

Различие режимов обучения и инференса по набору параметров. В режиме обучения используется полный набор столбцов датасета, включая импульс (p_x, p_y, p_z), заряд `particle_charge` и метку трека `trackID`. В режиме инференса эти параметры считаются недоступными: используются только геометрические и детекторные признаки — (x, y, z), `station`, `module`. Это отражает реальное ограничение эксперимента: импульс частицы измеряется лишь после полной реконструкции трека и не может быть известен заранее на этапе построения графа.

2.2 Система координат и признаки

Для описания хитов используется переход от декартовых к **полярным координатам** (r, φ, θ), где $r = \sqrt{x^2 + y^2}$ — расстояние до оси Z, $\varphi = \arctan(y/x)$ — азимутальный угол, $\theta = \arctan(z/r)$ — полярный угол. Полярные координаты более естественны для описания цилиндрически симметричного детектора и физики движения в магнитном поле.

Каждый узел графа (хит) описывается вектором из 9 признаков. Выбор признаков обусловлен их физической значимостью:

- полярные координаты (r, φ) и продольная координата z полностью определяют геометрическое положение хита;
- номера станции и модуля кодируют топологию детектора;
- $\log_{10}(p_T)$ – логарифм поперечного импульса; данный признак хорошо разделяет мягкие ($p_T < 0.5$ ГэВ) и жесткие треки;
- псевдобыстрота $\eta = -\ln(\tan(\theta/2))$ инвариантна относительно преобразований Лоренца вдоль оси пучка; обрезание η в пределы $[-0.999, 0.999]$ выполнено для избежания численной нестабильности $\operatorname{arctanh}(1) \to \infty$;
- заряд частицы и флаг первичной частицы предоставляют важную контекстную информацию.

2.3 Построение графа ребер-кандидатов

Построение графа выполняется модулем `EdgeBuilder`. Для каждой пары хитов (i, j) применяется двухэтапная фильтрация.

Быстрая предварительная фильтрация проверяет:

- движение только вперед по детектору $\text{station}_j > \text{station}_i$;
- максимальный разрыв по станциям: $\Delta\text{station} \leq 2$;
- расстояние: $d_{min} \leq d \leq d_{max}$ см, где d - полное евклидово расстояние между хитами по X, Y, Z, d_{min}/d_{max} - задаваемые параметры.

Физическая проверка в режиме обучения (при наличии импульсов) использует совместимость кривизны траектории — булеву величину, показывающую, может ли хит j физически лежать на продолжении трека после хита i с заданными p_T и зарядом q. Ожидаемое изменение азимутального угла вычисляется через геометрию хорды:

$$\Delta\varphi_{\text{exp}} = 2 \arcsin\left(\frac{dr}{2R}\right), \tag{14}$$

и сравнивается с фактическим изменением азимутального угла $\Delta\varphi_{\text{act}}$:

$$|\Delta\varphi_{\text{act}} - \Delta\varphi_{\text{exp}}| < \tau, \tag{15}$$

где τ — подобранный параметр допуска. При отсутствии импульсов (режим инференса) заряд и p_T оцениваются из геометрии хитов.

Каждое ребро описывается вектором из **18 признака**: полярные координаты обоих хитов, геометрические параметры пары $(dr, dz, d, \Delta\varphi, \Delta\theta)$, характеристики кривизны, физические параметры и номер станции.

Для каждого хита отбираются **топ-10 кандидатов** по score совместимости — взвешенной комбинации кривизны, направления импульса, расстояния и разрыва станций.

Режим обучения с dropout импульсов. Чтобы модель оставалась работоспособной при отсутствии импульсов (как на инференсе), в 30% случаев во время обучения импульсы искусственно скрываются — в этих случаях ребра строятся только по геометрии. Использование `momentum_dropout` преследует две цели: (1) регуляризация — модель не может полностью полагаться на «легкий» признак (p_x, p_y); (2) робастность — обучение предсказывает геометрические признаки $(p_T^{\text{est}}, q^{\text{est}})$ из голых координат, что критически важно для инференса на реальных данных, где импульс неизвестен.

3 Архитектура модели

3.1 Общая структура конвейера

Полный конвейер обработки данных выглядит следующим образом:

`CSV` → `dataset.py` → `edge_builder.py` → `gat_model.py` → `trainer.py` → `predictor.py` → `track_builder.py`

`dataset.py` Загружает CSV-файл с хитами детектора, группирует их по событиям (`eventID`) и строит граф для каждого события.

`edge_builder.py` Для каждой пары хитов применяет двухэтапную фильтрацию. Для каждого хита отбираются топ-K кандидатов по оценки (score) совместимости.

`gat_model.py` Реализует классификатор ребер на основе Graph Attention Network v2 (GATv2) [5]. Модель кодирует признаки узлов и ребер линейными проекциями, пропускает их через стек GAT-слоев с механизмом внимания, после чего MLP-классификатор присваивает каждому ребру вероятность принадлежности к истинному треку.

`trainer.py` Реализует цикл обучения: на каждом батче вычисляется функция потерь, веса обновляются оптимизатором Adam с косинусным расписанием скорости обучения. После каждой эпохи выполняется валидация, сохраняется модель с минимальным значениеи потерь на валидационной выборке (`val_loss`). Ранняя остановка (early stopping) останавливает обучение при отсутствии улучшений.

`predictor.py` Запускает обученную модель на тестовых данных в режиме `eval()` без градиентов, применяет сигмоиду к логитам и фильтрует ребра по пороговому значению вероятности.

`track_builder.py` Строит треки из отобранных ребер двумя алгоритмами: жадным (greedy) — последовательное продление трека от ребра с наибольшей оценкой (score), и через поиск связных компонент графа обходом в глубину (DFS).

3.2 Graph Attention Network (GATv2)

В качестве основной модели используется **Graph Attention Network v2** (GATv2Conv) [5]. В отличие от классического GAT, GATv2 вычисляет динамические коэффициенты внимания, зависящие от обоих концов ребра одновременно, что устраняет проблему статического внимания.

Коэффициент внимания для ребра (i, j):

$$\alpha_{ij} = \frac{\exp\big(a^\top \mathrm{LeakyReLU}(W[h_i \| h_j \| e_{ij}])\big)}{\sum_{k \in \mathcal{N}(i)} \exp\big(a^\top \mathrm{LeakyReLU}(W[h_i \| h_k \| e_{ik}])\big)}, \tag{16}$$

где h_i — эмбеддинг вершины, e_{ij} — признаки ребра, W и a — обучаемые параметры.

Коэффициент внимания α_{ij} определяет вклад соседнего хита j при агрегации информации в вершине i. Чем выше α_{ij}, тем сильнее признаки хита j влияют на обновленный эмбеддинг хита i.

В задаче трекинга это имеет прямой физический смысл: каждый хит соединен с несколькими кандидатами на продолжение трека, среди которых большинство являются ложными ребрами. Механизм внимания позволяет модели самостоятельно научиться подавлять ложные связи и усиливать физически совместимые — те, где два хита действительно принадлежат одной траектории.

Ключевое отличие GATv2 от классического GAT состоит в том, что аргумент функции LeakyReLU зависит от конкатенации эмбеддингов *обоих* концов ребра $[h_i \| h_j \| e_{ij}]$, тогда как в GAT v1 веса вычислялись независимо для каждой вершины. Это устраняет проблему статического внимания — ситуацию, когда ранжирование соседей не зависит от запрашивающей вершины i.

3.3 Детали архитектуры

Модель состоит из следующих блоков:

1. **Входные проекции**: линейные слои отображают признаки узлов (9-мерные) и ребер (18-мерные) в пространство размерности `hidden_dim`.

2. **Стек GAT-слоев**: L слоев GATv2Conv с H головами внимания (attention head). После каждого слоя применяются BatchNorm, ELU-активация и Dropout (кроме последнего слоя).

3. **MLP-классификатор ребер**: для каждого ребра формируется представление как конкатенация эмбеддингов исходной вершины, конечной вершины и признаков ребра:

$$\mathbf{e}_{ij} = [\mathbf{h}_i \| \mathbf{h}_j \| \mathbf{f}_{ij}], \tag{17}$$

   которое затем пропускается через многослойный перцептрон с выходом в виде одного логита.

Такая архитектура, известная как *edge-wise MLP после агрегации узлов*, является стандартной для задач классификации ребер. Она позволяет классификатору видеть полную информацию о паре вершин в изолированном виде, не полагаясь только на их обновленные окрестностями эмбеддинги.

Конфигурация модели с фиксированными параметрами (по итогам предварительного подбора) представлены в таблице 2

Таблица 2: Архитектурные гиперпараметры модели

Параметр	**Значение**
`hidden_dim`	256
`num_layers`	4
`num_heads`	8

4 Обучение

4.1 Функции потерь

Дисбаланс классов (истинных ребер значительно меньше ложных ребер) требует специального подхода к функции потерь. В данной работе исследовались два варианта. Из статистики датасета следует, что количество истинных ребер против количества ложных составляет примерно $1:6$.

Бинарная кросс-энтропия с взвешиванием классов (Binary Cross-Entropy with pos_weight):

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\big[w_+ \cdot y_i \log \hat{p}_i + (1-y_i)\log(1-\hat{p}_i)\big], \tag{18}$$

где w_+ — вес для положительного класса. Параметр w_+ искусственно увеличивает штраф за пропуск истинного ребра, компенсируя дисбаланс. При $w_+ = 1$ функция стандартная; при $w_+ > 1$ модель «дороже платит» за пропуск истинного ребра (false negative).

Фокальная функция потерь (Focal Loss) — снижает вклад «легких» примеров, фокусируя обучение на сложных:

$$\mathcal{L}_{\text{FL}} = -\frac{\alpha}{N}\sum_{i=1}^{N}(1-\hat{p}_i)^{\gamma} \cdot y_i \log \hat{p}_i, \tag{19}$$

где α и γ — параметры фокусировки. Множитель $(1-\hat{p}_i)^{\gamma}$ автоматически снижает вклад примеров, которые модель уже уверенно классифицирует (большое $\hat{p}_i$ для истинных ребер), и усиливает внимание к трудным случаям — граничным ребрам, где модель сомневается. Это особенно актуально в задаче трекинга, где большинство ложных ребер легко отсеиваются геометрическими ограничениями, а оставшиеся сложные случаи и определяют качество реконструкции.

4.2 Оптимизатор и расписание скорости обучения (learning rate)

Используется оптимизатор **Adam** с L2-регуляризацией (weight decay). Adam выбран ввиду его робастности к выбору скорости обучения и способности эффективно работать с зашумленными градиентами, характерными для разреженных графов. Расписание скорости обучения (learning rate) задается **CosineAnnealingLR** — скорость обучения плавно убывает от начального значения $\eta_{\max}$ до минимального $\eta_{\min}$ по форме косинуса на протяжении всего обучения:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\frac{\pi t}{T_{\max}}\right), \tag{20}$$

где t — номер текущей эпохи, $T_{\max}$ — общее число эпох.

Такой подход позволяет модели в начале обучения делать крупные шаги и быстро находить область минимума, а ближе к концу — мелкие шаги и точнее в него попадать. В данной работе $\eta_{\max} = 7 \cdot 10^{-4}$, $\eta_{\min} = 10^{-6}$.

Дополнительно применяется **отсечение градиентов** (gradient clipping) с максимальной нормой `max_norm` = 1,0. Перед каждым шагом оптимизатора норма градиентов проверяется, и, если она превышает порог, все градиенты масштабируются так, чтобы их

суммарная норма равнялась 1,0:

$$\mathbf{g} \leftarrow \mathbf{g} \cdot \frac{1{,}0}{\max(1{,}0,\ \|\mathbf{g}\|)}. \tag{21}$$

Это предотвращает взрыв градиентов — ситуацию, когда на отдельных батчах градиенты резко возрастают и обновление весов выбивает модель из области минимума. В графовых сетях это особенно актуально из-за неоднородности графов: батч может содержать как простые события с малым числом ребер, так и сложные — с сотнями кандидатов.

4.3 Подбор гиперпараметров (Optuna)

Для систематического поиска гиперпараметров используется фреймворк **Optuna** с алгоритмом TPE (Tree-structured Parzen Estimator). Стратегия поиска — двухэтапная:

1. **Быстрый поиск** на подвыборке (первые 2000 событий): 85 испытаний, по 15 эпох каждое. Цель — найти перспективную область гиперпараметров.
2. **Точный поиск** на полном датасете: из лучших параметров быстрого поиска.

Диапазоны поиска для текущих экспериментов представлены в таблице 3.

Таблица 3: Пространство поиска гиперпараметров

Параметр	**Диапазон**	**Тип**	**Обоснование**
`dropout`	[0,1; 0,3]	float (log)	Защита от переобучения без потери выразительности модели.
`learning_rate`	$[2{\cdot}10^{-4};\ 7{\cdot}10^{-4}]$	float (log)	Баланс между скоростью и стабильностью сходимости Adam.
`weight_decay`	$[10^{-7};\ 10^{-5}]$	float (log)	Легкая регуляризация, не подавляющая слабые физические признаки.
`pos_weight`	[3,5; 5,0]	float	Компенсация дисбаланса классов ($\approx$ 1:6).
`grad_accum`	$\{2, 4\}$	int	Увеличение эффективного batch size при ограниченной памяти GPU.

Для ускорения поиска применяется **MedianPruner** — испытания, показывающие результат хуже медианы предыдущих на промежуточных эпохах, прерываются досрочно.

4.4 Ранняя остановка (early stopping) и сохранение модели

Обучение прекращается досрочно если потери на валидационной части не улучшается в течение 15 эпох. Лучшая модель (по минимальному `val_loss`) автоматически сохраняется в файл `best_model.pt` вместе с конфигурацией.

5 Результаты

5.1 Предварительные результаты

В таблице 4 представлены промежуточные результаты обучения модели с грубо оцененными значениями гиперпараметров на начальном этапе реализации, целью которого была проверка работоспособности кода.

Таблица 4: Метрики классификации ребер

Метрика	**Значение**
Train Loss	0.1729
Val Loss	0.1704
Accuracy	0.9400
Precision	0.7432
Recall	0.8520
F1-score	0.7939
AUC-ROC	0.9740

По промежуточным данным были сделаны следующие гипотезы:

1. **Неоптимальное построение графов**: стоит добавить и изменить существующие физические ограничения для более корректного связывания хитов.
2. **Недостаток параметров edge_features**: стоит добавить физические величины, увеличивающие вероятность связывания хитов одного трека.
3. **Недостаточно точная настройка гиперпараметров**: модель не достигает своего реального потенциала, требуется более точная подборка гиперпараметров.
4. **Неподходящая под задачу функция потерь**: стоит оценить влияние другой функции потерь, специализированной на редкие классы.

5.2 Улучшение качества модели

Для проверки выдвинутых гипотез был проведен ряд экспериментов, направленных на повышение качества классификации ребер.

Модификация построения графов и расширение признакового пространства. На вход графовой нейронной сети подается граф, поэтому корректность его построения является ключевым фактором, влияющим на качество классификации. На данном этапе проводилась оптимизация параметров построения графа, а также расширение признакового пространства для повышения дискриминационной способности модели.

В исходном варианте в качестве признаков ребер не использовалась псевдобыстрота η, которая характеризует продольную составляющую трека. Ее добавление в набор признаков позволило значительно улучшить качество классификации: точность (Precision) увеличилась с 0.7432 до 0.7940, полнота (Recall) — с 0.8520 до 0.9214, а F1-мера — с 0.7939 до 0.8530.

Настройка гиперпараметров модели Было проведено сравнение двух функций потерь: бинарной кросс-энтропии (BCE) с взвешиванием с фокальной функцией потерь

(Focal Loss), под каждую были найдены индивидуальные гиперпараметры, представленные в таблице 5.

Таблица 5: Сравнение лучших гиперпараметров и результатов обучения

Параметр	**BCE**	**Focal Loss**	**Что означает**
val_loss	0,02566	0,00028	Значение функции потерь на валидационных данных
hidden_dim	256	256	Размерность эмбеддинга узла/ребра внутри сети
num_layers	3	3	Число последовательных GAT-слоев
num_heads	8	8	Число голов внимания в каждом GAT-слое
dropout	0,23952	0,11281	Вероятность отключения нейрона при обучении (регуляризация)
learning_rate	0,00074	0,00023	Начальная скорость обучения оптимизатора Adam
weight_decay	$5{,}6062 \cdot 10^{-6}$	$1{,}8914 \cdot 10^{-6}$	Коэффициент L2-регуляризации весов модели
lr_min	$2{,}8824 \cdot 10^{-6}$	$3{,}4930 \cdot 10^{-7}$	Минимальная скорость обучения в конце расписания CosineAnnealingLR.
momentum_dropout	0,40303	0,38363	Доля случаев, когда импульс искусственно скрывается на обучении
grad_accum	1	1	Число шагов накопления градиентов перед обновлением весов
pos_weight	3,43023	–	Вес положительного класса в BCE (компенсация дисбаланса)
focal_alpha	–	0,10832	Весовой коэффициент положительного класса в Focal Loss
focal_gamma	–	2,54615	Параметр фокусировки на сложных примерах в Focal Loss

Анализ полученных результатов. Потери на валидационной части у фокальной функции потерь (Focal Loss) оказались на порядок меньше, чем у BCE. Это свидетельствует о том, что Focal Loss лучше справляется с проблемой дисбаланса классов.

Результаты с использованием фокальной функции потерь (Focal Loss). Обучение модели с подобранными гиперпараметрами и фокальной функцией потерь (Focal Loss) показало значительное улучшение качества классификации по сравнению с промежуточными результатами. На обучении были достигнуты следующие значения метрик:

- точность (Accuracy) составила 0.9963;
- точность (Precision) — 0.9885;
- полнота (Recall) — 0.9978;
- F1-мера — 0.9871;

- площадь под ROC-кривой (AUC-ROC) — 0.9998.

Метрики на уровне треков (track-level): чистота треков (Track Purity) достигла 0.9975, а эффективность восстановления треков (Track Efficiency) составила 0.2672.

Анализ расхождения метрик на уровне ребер и треков. Высокое качество классификации на уровне ребер является необходимым, но недостаточным условием для эффективного восстановления треков. Причины низкого значения Track Efficiency требуют дальнейшего анализа. В ближайшее время планируется провести более детальное исследование влияния различных факторов на восстановление треков, на основе которого будут разработаны и реализованы необходимые доработки методов постобработки, позволяющие повысить Track Efficiency.

6 Оптимизация производительности

6.1 Анализ узких мест

Основное время при первом запуске уходит на **построение графов**: для каждого события выполняется попарная проверка всех хитов во вложенных циклах Python (через `iterrows`), что при больших событиях является существенным узким местом по производительности.

6.2 Реализованные оптимизации

6.2.1 Двухэтапная фильтрация ребер

Введена быстрая предварительная фильтрация, которая отсеивает заведомо невозможные пары до дорогостоящей физической проверки. Проверки выполняются в порядке возрастания стоимости: сначала сравнение номеров станций (целочисленное), затем расстояние (одно умножение) и только затем вычисление полярных координат и кривизны.

6.2.2 Кэширование полярных координат

Каждый хит преобразуется один раз и сохраняется в словаре. При построении ребер используются готовые значения без повторных вычислений.

6.2.3 Оптимизация пакетной загрузки данных DataLoader

Для ускорения подачи данных в модель настроены параметры DataLoader: `pin_memory=True` (закрепленная память) для ускорения передачи на графический процессор (GPU), количество параллельных процессов которые загружают данные пока модель обучается `num_workers=4`.

6.2.4 Накопление градиентов (Gradient accumulation)

Введена поддержка накопления градиентов (`gradient_accumulation_steps`), что позволяет эффективно использовать большие батчи при ограниченной видеопамяти.

6.3 Дальнейшие направления оптимизации

- **C++/CuPy гибрид**: переписывание некоторых модулей на C++ с экспортом через Python bindings (pybind11) и их вызов из основного проекта; для задействования GPU-ускорения CUDA используется как напрямую (написание пользовательских ядер), так и через высокоуровневые операции CuPy, интегрированные в C++-модули.
- **Оффлайн-препроцессинг**: однократное построение всех графов с сохранением в файлы `.pt`, что исключает повторные вычисления при каждом запуске обучения.
- **Радиусный поиск соседей**: замена полного перебора пар $O(N^2)$ на KD-tree или ball tree для поиска кандидатов в окрестности.
- **Numba/NumPy**: перевод горячих функций с Python-циклов на векторизованные операции.

Заключение

В ходе данной научно-исследовательской работы была продолжена разработка системы реконструкции треков заряженных частиц на основе графовой нейронной сети с механизмом внимания (GATv2) [5] для эксперимента BM@N.

Основные результаты:

- реализован полный конвейер обработки от сырых данных до построения треков, включая физически мотивированное построение графа с проверкой совместимости кривизны;
- проведен систематический подбор гиперпараметров с помощью Optuna с использованием двухэтапной стратегии (быстрый поиск + точный поиск);
- исследованы две функции потерь (бинарная кросс-энтропия (BCE) с взвешиванием и фокальная функция потерь (Focal Loss)) для борьбы с дисбалансом классов;
- выполнен ряд оптимизаций производительности конвейера.

В ходе работы были получены высокие значения метрик на уровне ребер (edge-level), однако эффективность восстановления треков (Track Efficiency) остается низкой. Это указывает на необходимость дополнительного анализа причин, по которым целостные треки не восстанавливаются, несмотря на качественную классификацию отдельных связей между хитами.

Дальнейшее развитие работы предполагает два основных направления:

- улучшение качества реконструкции на уровне треков (track-level) за счет проведения анализа причин низкой Track Efficiency, доработки архитектуры модели, расширения набора признаков и увеличения объема данных для обучения;
- оптимизация производительности разработанного решения с использованием C++/CuPy гибридного подхода, оффлайн-препроцессинга графов, пространственных структур для ускорения поиска соседей и векторизации вычислений с помощью Numba.

Список литературы

[1] S. Afanasiev et al. The BM@N spectrometer at the NICA accelerator complex // Nucl. Instr. and Meth. A. — 2024. — Vol. 1065. — P. 169532.

[2] Воронцов К. В. и др. Машинное обучение [Электронный ресурс] // Хендбук Яндекс. — URL: https://education.yandex.ru/handbook/ml/article/about (дата обращения: 10.06.2026).

[3] Китов В. В. Машинное обучение. Глубокое обучение [Электронный ресурс]. — URL: https://deepmachinelearning.ru/docs/Machine-learning/book-title (дата обращения: 10.06.2026).

[4] Машинное обучение в физике [Электронный ресурс] : курс лекций // Teach-in. — 2022. — URL: https://teach-in.ru/course/machine-learning-in-physics (дата обращения: 10.06.2026).

[5] PyTorch Geometric. GATConv [Электронный ресурс] // PyTorch Geometric Documentation. — URL: `https://pytorch-geometric.readthedocs.io/en/2.7.0/generated/torch_geometric.nn.conv.GATConv.html` (дата обращения: 10.06.2026).

[6] Scikit-learn [Электронный ресурс] // Scikit-learn Documentation. — URL: https://scikit-learn.org/stable/ (дата обращения: 10.06.2026).

[7] Girafe-ai. ML Course [Электронный ресурс] : открытый курс по машинному обучению // GitHub. — URL: https://github.com/girafe-ai/ml-course/ (дата обращения: 10.06.2026).