МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»
(национальный исследовательский университет)»

Институт (Филиал) № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

Группа М8О-210М-23 **Направление подготовки** 02.04.02 Фундаментальная информатика и информационные технологии

Профиль Машинное обучение и анализ данных

Квалификация **магистр**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

На тему: Применение методов машинного обучения для управления инвестиционным портфелем на Московской бирже

Автор ВКРМ Минаков Максим Михайлович (_____)
(фамилия, имя, отчество полностью)

Руководитель Крылов Сергей Сергеевич (_____)
(фамилия, имя, отчество полностью)

Консультант _____ (_____)
(фамилия, имя, отчество полностью)

Консультант _____ (_____)
(фамилия, имя, отчество полностью)

Рецензент _____ (_____)
(фамилия, имя, отчество полностью)

К защите допустить

Заведующий кафедрой 806 Крылов Сергей Сергеевич (_____)
(№ каф) (фамилия, имя, отчество полностью)

___ _____ 2025г.

Москва 2025

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Альфа-фактор – переменная или набор признаков, используемых для прогнозирования потенциальной доходности актива выше среднего по рынку

Торговая стратегия – алгоритм принятия решений о покупке или продаже активов, реализуемый вручную или с использованием программных инструментов (например, Backtrader[1])

Технический индикатор – математически рассчитываемый показатель, основанный на исторических данных о цене и объеме торгов, применяемый для анализа рыночных трендов и формирования торговых сигналов[2]

EMA_15, EMA_75 (Exponential Moving Average) – экспоненциальные скользящие средние. EMA_15 используется для оценки краткосрочного тренда, EMA_75 – долгосрочного

ADX (Average Directional Index) – индикатор силы тренда. Значения выше 25 свидетельствуют о сильном тренде, ниже 20 – о боковом движении (флэте)

DI+, DI− (Directional Indicators) – компоненты индикатора направленного движения. DI+ указывает на силу восходящего тренда, DI− – нисходящего. Их сравнение позволяет оценить преобладающее направление рынка

RSI_14 (Relative Strength Index) – индекс относительной силы. Значения выше 70 указывают на перекупленность, ниже 30 – на перепроданность актива

ATR_14 (Average True Range) – индикатор волатильности. Позволяет оценить диапазон колебаний цены, используется для определения рыночной активности

API (Application Programming Interface) – способ программного взаимодействия между сервисами. В проекте используется MOEX API для получения рыночных данных[3]

MOEX API – интерфейс Московской биржи для доступа к историческим и текущим данным по финансовым инструментам[4]

CatBoost – алгоритм градиентного бустинга, разработанный Яндекс. Эффективен при наличии категориальных признаков и применяется для задач классификации и регрессии[5]

LightGBM – быстрый алгоритм бустинга от Microsoft, оптимизированный для больших наборов данных. Использует метод GOSS и leaf-wise рост деревьев[6]

XGBoost – один из самых точных и гибких алгоритмов бустинга, популярный в прикладных задачах анализа данных[7]

MLflow – платформа управления жизненным циклом моделей ML: от логирования метрик до версионирования и деплоя моделей[8]

Инференс – этап применения обученной модели машинного обучения к новым входным данным для получения предсказаний

CI/CD (Continuous Integration / Continuous Delivery) – практика непрерывной интеграции и доставки изменений: автоматическая сборка, тестирование и развертывание кода

GitHub Actions – встроенный инструмент GitHub для настройки CI/CD. Позволяет запускать пайплайны при push, pull request и других событиях

Docker – система контейнеризации, позволяющая изолировать окружение приложения и обеспечить переносимость между платформами[9]

Helm — это менеджер пакетов для Kubernetes, предназначенный для автоматизации установки, настройки и управления приложениями внутри кластера[10]

Kubernetes – система оркестрации контейнеров для автоматического масштабирования и управления микросервисной архитектурой

Pod (Kubernetes Pod) – наименьшая единица исполнения в Kubernetes. Представляет собой группу контейнеров, работающих в общем сетевом и дисковом пространстве

Terraform – инструмент для описания инфраструктуры как кода (IaC). Позволяет автоматически разворачивать и управлять облачными ресурсами[11]

Yandex Cloud – облачная платформа с возможностью хранения данных, работы с вычислениями и построения ML-инфраструктуры (включая S3, PostgreSQL, Kubernetes)[12]

S3-хранилище (Simple Storage Service) – объектное хранилище для долговременного хранения и масштабируемого доступа к файлам, моделям и логам

PostgreSQL – реляционная СУБД с открытым кодом. Используется для хранения структурированных данных и результатов экспериментов

Backtesting – Python-фреймворк для реализации, симуляции и оптимизации алгоритмических торговых стратегий на исторических данных[13]

JSON (JavaScript Object Notation) – формат обмена данными между клиентом и сервером. Используется для передачи структурированной информации в ML-пайплайне

ВВЕДЕНИЕ

Современные финансовые рынки характеризуются высокой скоростью изменений и большим объемом доступной информации, что делает традиционные методы инвестиционного анализа всё менее эффективными в условиях реального времени. Инвесторы и аналитики сталкиваются с необходимостью обработки исторических и текущих данных, оценки рыночной динамики и принятия решений в условиях неопределенности. Всё это повышает интерес к алгоритмическим подходам и, в частности, к методам машинного обучения.

Московская биржа предоставляет широкие возможности для моделирования и анализа рыночных процессов благодаря доступу к историческим данным, включая котировки и объёмы торгов. В данной работе рассматривается задача построения предсказательной модели, которая на основе технических индикаторов и исторических рыночных данных по акциям ПАО «Сбербанк» может определять вероятное направление движения цены на краткосрочном горизонте. В исследовании применяются модели градиентного бустинга – CatBoost, LightGBM и XGBoost – с последующей оценкой точности предсказаний и качества сигналов.

Разработанная модель может служить основой для создания торговых стратегий и принятия инвестиционных решений на основе данных, а также стать элементом более масштабных систем анализа и автоматизации в сфере алгоритмического трейдинга.

1 ОПИСАНИЕ РЕШЕНИЯ

1.1 Актуальность проблемы

Современные финансовые рынки, включая Московскую биржу, предъявляют высокие требования к точности анализа и адаптивности стратегий управления портфелем. Для достижения стабильных результатов инвесторы используют как традиционные методы, такие как технический и фундаментальный анализ, так и новые подходы, основанные на методах машинного обучения (ML). Традиционные методы предоставляют ценные аналитические инструменты, но их потенциал можно значительно расширить, интегрируя ML-алгоритмы.

Многие профессиональные участники рынка уже активно применяют ML для повышения точности прогнозирования и автоматизации принятия решений. Одним из наиболее ярких примеров является Medallion Fund, управляемый компанией Renaissance Technologies под руководством математика Джима Саймонса. Этот фонд показывает среднюю доходность более 39% годовых в долларах США, демонстрируя успех стратегий, основанных на сочетании ML и глубокого математического анализа.

В контексте Московской биржи такая интеграция методов традиционного анализа и машинного обучения открывает возможности для создания более точных и эффективных стратегий управления портфелем. Это делает задачу исследования и внедрения ML-алгоритмов, в сочетании с существующими подходами, особенно актуальной для достижения конкурентных преимуществ на финансовом рынке.

1.2 Анализ существующих подходов к управлению инвестиционным портфелем

1.2.1 Эволюция подходов и постановка проблемы

Управление инвестиционным портфелем всегда находилось в центре внимания как частных инвесторов, так и институциональных участников рынка. Первоначально принятие решений основывалось на интуиции, наблюдении за графиками и базовых принципах оценки стоимости активов. По мере развития финансовых рынков начали формироваться формализованные методы анализа, которые можно условно разделить на три большие категории: технический анализ, фундаментальный подход и количественные методы, к числу которых в последние годы относятся и алгоритмы машинного обучения

.

Технический анализ исторически стал одним из первых систематизированных подходов к оценке движения цен. Его принципы основываются на предположении, что цена учитывает всё, история повторяется, а рыночные тренды поддаются визуальному и математическому анализу. Используя индикаторы и паттерны, трейдеры стараются выявить сигналы входа и выхода из позиции. Такой подход получил широкое распространение благодаря простоте и наглядности.

Фундаментальный анализ, напротив, опирается на более долгосрочные оценки стоимости компании или актива. Он рассматривает такие параметры, как прибыль, выручка, дивиденды, макроэкономические факторы и состояние отрасли. Фундаментальный подход широко используется институциональными инвесторами и при принятии стратегических решений, но он слабо применим при анализе краткосрочной ценовой динамики, особенно в условиях высокой волатильности.

Следующим этапом эволюции стало появление портфельной теории, предложенной Гарри Марковицем. Этот подход рассматривает портфель как систему, в которой важны не только доходности отдельных активов, но и их взаимная корреляция. Теория эффективно работает на средне- и долгосрочных горизонтах и лежит в основе современных моделей управления активами (например, CAPM). Однако и она имеет ограничения, особенно в условиях нестабильности рынка и нелинейных зависимостей между переменными.

С развитием вычислительной техники и ростом объемов доступных данных стали активно развиваться количественные методы. В последние годы особое внимание привлекает машинное обучение – как способ выявления сложных, ранее недоступных закономерностей в финансовых временных рядах. Эти алгоритмы обладают высокой гибкостью, адаптивностью и способностью автоматизировать процесс анализа данных, что делает их привлекательными в задачах прогнозирования рыночного поведения.

1.2.2 Традиционные методы управления портфелем

На протяжении десятилетий ключевыми инструментами инвесторов оставались традиционные методы анализа, в числе которых технический анализ, фундаментальный подход и теория портфельной оптимизации. Несмотря на появление более современных технологий, эти методы по-прежнему применяются как в самостоятельном виде, так и в качестве основы для более сложных моделей.

Технический анализ

Технический анализ основан на использовании исторических данных о ценах и объемах торгов для выявления рыночных закономерностей и прогнозирования будущего движения цен. Основное предположение этого подхода заключается в том, что рынок цикличен, а поведение участников повторяется, формируя узнаваемые паттерны. К числу ключевых

инструментов технического анализа относятся скользящие средние (Simple Moving Average, Exponential Moving Average), индикаторы тренда (MACD, ADX), осцилляторы (RSI, Stochastic) и сигналы пересечения.

Одним из важнейших преимуществ технического анализа является его универсальность и относительная простота применения. Он особенно эффективен в условиях ликвидных рынков и на краткосрочных интервалах. Однако метод не учитывает фундаментальные причины движения цены и часто подвержен ложным сигналам в периоды высокой волатильности. Кроме того, его эффективность снижается при изменении рыночного режима или появлении внешних макроэкономических шоков.

Фундаментальный анализ

Фундаментальный анализ, в отличие от технического, ориентирован на долгосрочную оценку стоимости актива. Он включает изучение финансовой отчетности компаний (доходы, расходы, балансовые показатели), макроэкономических индикаторов, отраслевых трендов и конкурентного положения. Основная задача – определить «справедливую» цену актива и выявить недооцененные или переоцененные инструменты.

Фундаментальный подход широко используется институциональными инвесторами, в том числе при составлении диверсифицированных портфелей. Он позволяет учитывать долгосрочные перспективы роста и устойчивость бизнеса. Вместе с тем он малопригоден для краткосрочного трейдинга, так как не отражает текущие рыночные колебания. Кроме того, анализ требует значительного времени и доступа к качественным данным, что затрудняет его автоматизацию и масштабирование.

Теория портфеля Марковица

Портфельная теория, предложенная Гарри Марковицем, внесла значительный вклад в формализацию управления инвестициями. Её ключевая идея – диверсификация, минимизирующая риск при заданной ожидаемой доходности. В рамках модели используются показатели ожидаемой доходности, стандартного отклонения и ковариации между активами для

построения эффективной границы (efficient frontier), определяющей оптимальное соотношение активов в портфеле.

Теория Марковица особенно полезна для средне- и долгосрочного инвестирования. Она позволяет учитывать взаимосвязи между активами и управлять риском на уровне всего портфеля. Однако модель предполагает нормальное распределение доходностей и стационарность параметров, что часто не соответствует реальной рыночной динамике. Кроме того, построение ковариационной матрицы на нестабильных данных может приводить к нестабильным решениям.

1.2.3 Методы машинного обучения в управлении портфелем

С развитием вычислительных мощностей, появлением большого объема исторических данных и доступом к открытым источникам информации, в финансовую сферу начали активно внедряться методы машинного обучения (ML). В отличие от традиционных подходов, машинное обучение позволяет выявлять сложные и нелинейные зависимости в данных, автоматически адаптироваться к изменяющимся рыночным условиям и обрабатывать большое количество признаков без необходимости ручной формализации правил.

Основная идея применения ML в управлении портфелем заключается в создании модели, способной на основе исторических данных предсказывать поведение актива или рынка в будущем. Это может быть как прогноз абсолютных цен, так и классификация направлений движения (рост/падение), определение рыночных режимов или генерация торговых сигналов.

Применяемые классы моделей

В задачах анализа финансовых временных рядов находят применение следующие группы алгоритмов машинного обучения:

Модели регрессии и классификации, включая логистическую регрессию, деревья решений, случайный лес, методы опорных векторов

(SVM). Они используются как базовые модели для прогнозирования меток (направлений движения, вероятности роста и т.д.).

Алгоритмы градиентного бустинга, такие как XGBoost, LightGBM и CatBoost – являются одними из самых эффективных и широко применяемых в задачах с табличными признаками. Они обеспечивают высокую точность, хорошо работают с разреженными и шумными данными, устойчивы к переобучению и позволяют интерпретировать важность признаков.

Модели временных рядов, включая ARIMA, Prophet, а также глубокие нейросетевые архитектуры (например, LSTM, TimesNet). Эти методы учитывают последовательность значений во времени, но требуют тщательной настройки и больших объёмов данных.

Модели обнаружения аномалий (Isolation Forest, Autoencoder, DBSCAN) применяются для выявления экстремальных рыночных состояний и резких изменений тренда.

NLP и анализ текстов используются для оценки влияния новостного фона на рынок, извлечения тональности из корпоративных отчетов и новостей.

Reinforcement Learning – применяется для построения адаптивных торговых агентов, обучающихся на взаимодействии с рынком. Однако практическое применение RL в реальном трейдинге требует обширных симуляций и усложненной среды.

Выбор градиентного бустинга

В рамках настоящего исследования основное внимание уделяется алгоритмам градиентного бустинга по следующим причинам:

Практическая эффективность: модели бустинга показали высокую точность на задачах с табличными и техническими признаками, включая финансовые временные ряды.

Устойчивость к шуму: алгоритмы способны справляться с выбросами и неточностями, характерными для рыночных данных.

Гибкость и скорость обучения: XGBoost и LightGBM поддерживают параллельное обучение, что позволяет использовать большие выборки. CatBoost дополнительно эффективно обрабатывает категориальные признаки.

Интерпретируемость: важность признаков (feature importance), SHAP-значения и визуализация деревьев позволяют анализировать логику принятия решений моделью.

Таким образом, выбранные модели – CatBoost, XGBoost и LightGBM – являются оптимальным выбором для решения задачи прогнозирования направления движения цены на основе технических индикаторов. Они позволяют строить точные, адаптивные и интерпретируемые предсказательные модели, пригодные для последующего применения в инвестиционном анализе.

1.2.4 Сравнение различных подходов к управлению портфелем

Выбор подхода к управлению инвестиционным портфелем зависит от множества факторов: горизонта инвестирования, уровня риска, характера рынка, целей инвестора и доступных инструментов анализа. В таблице ниже представлены основные характеристики четырех классов подходов: технический анализ, фундаментальный анализ, теория Марковица и методы машинного обучения.

Таблица 1 – Сравнение различных подходов к управлению портфелем

Критерий	Технический анализ	Фундаментальный анализ	Теория Марковица	Машинное обучение
Горизонт применения	Краткосрочный	Средне-/долгосрочный	Среднесрочный	Любой
Учет рыночной динамики	Да	Нет	Частично	Да
Учет макро-/фин показателей	Нет	Да	Частично	Возможно
Математическая строгость	Средняя	Низкая	Высокая	Высокая

Продолжение таблицы 1

Автоматизация	Частично возможна	Затруднена	Возможна	Да
Гибкость к условиям рынка	Ограниченная	Ограниченная	Низкая	Высокая
Влияние качества данных	Умеренное	Высокое	Высокое	Очень высокое
Сложность реализации	Низкая	Средняя	Средняя	Высокая
Интерпретируемость	Высокая (визуально)	Высокая (показатели)	Средняя	Средняя

Как видно из таблицы, ни один из подходов не является универсальным. Технический анализ хорошо работает в краткосрочных сценариях, но чувствителен к рыночным шумам и не учитывает фундаментальные факторы. Фундаментальный подход предлагает стратегическую перспективу, но слабо применим на коротких горизонтах. Теория Марковица – важный шаг в сторону количественного управления, но ее предпосылки о стационарности и нормальности доходностей далеки от реальных условий рынка].

Методы машинного обучения выгодно выделяются способностью обрабатывать большое количество признаков, выявлять сложные взаимосвязи и адаптироваться к нестабильным рыночным режимам. Однако они требуют больших объёмов качественных данных, вычислительных ресурсов и высокой экспертизы.

Именно поэтому в рамках настоящего исследования применяется компромиссный подход – использование проверенных технических индикаторов в сочетании с моделями градиентного бустинга. Такой подход

позволяет сохранить интерпретируемость и релевантность признаков, при этом повысив точность прогнозирования и адаптивность модели.

1.2.5 Ограничения и вызовы применения методов машинного обучения в управлении портфелем

Несмотря на высокую гибкость и потенциал машинного обучения в задачах инвестиционного анализа, на практике его применение сопровождается рядом ограничений и вызовов, особенно в условиях нестабильной рыночной среды. Эти проблемы важно учитывать при построении моделей, интерпретации результатов и оценке качества прогнозов.

Качество и полнота данных

Модели машинного обучения чувствительны к качеству исходных данных. Финансовые временные ряды часто содержат пропуски, выбросы, периоды отсутствия торгов и нестабильную частоту обновлений. Недостаточная глубина истории или низкое разрешение (например, дневные данные вместо минутных) может ограничить точность предсказаний. Кроме того, использование технических индикаторов в качестве признаков требует правильной настройки параметров (например, периодов скользящих средних), иначе модель будет обучаться на не релевантной информации.

Риск переобучения (overfitting)

Переобучение – одна из ключевых проблем при применении моделей с высокой сложностью, таких как градиентный бустинг. Модель может показывать отличные результаты на обучающей выборке, но плохо обобщать поведение на новых данных, особенно при высокой волатильности рынка. Эту проблему необходимо решать через регуляризацию, кросс-валидацию, контроль глубины деревьев и отслеживание метрик на валидации.

Утечка данных (data leakage)

При работе с временными рядами легко допустить логическую ошибку, когда модель использует информацию из будущего при обучении. Даже

косвенные признаки (например, технические индикаторы, рассчитанные с учетом будущих данных) могут привести к завышенной оценке точности и неработающим стратегиям в реальности. Для исключения утечки необходимо строго соблюдать временную последовательность при формировании выборок и разбиении на train/test.

Изменчивость рыночных режимов

Рынок не является стационарной системой. Поведение актива, волатильность и структура признаков могут резко меняться в зависимости от фаз рынка (рост, падение, флэт). Это приводит к снижению обобщающей способности моделей, особенно тех, что были обучены на «мирных» исторических периодах. Частичным решением может быть регулярное переобучение моделей на скользящем окне или использование адаптивных методов.

Проблема интерпретируемости

Хотя современные ML-модели, особенно бустинг, обеспечивают высокую точность, они часто функционируют как «чёрный ящик». Это усложняет интерпретацию результатов и может вызывать недоверие со стороны инвесторов и регуляторов. Применение методов объяснения, таких как feature importance, SHAP или permutation importance, частично решает эту задачу, но требует дополнительных усилий и вычислений.

Ограниченные возможности симуляции и реального тестирования

Оценка моделей на истории (бэк тестирование) всегда ограничена предположениями: идеальное исполнение ордеров, отсутствие проскальзывания, статичные комиссии. Реальная торговля имеет иную структуру рисков. Поэтому важным вызовом остаётся адекватная интерпретация результатов предсказательной модели в контексте реального применения.

1.2.6 Обобщение и выводы

Анализ существующих подходов к управлению инвестиционным портфелем показывает, что каждый из них имеет свои сильные и слабые стороны. Технический анализ удобен в реализации и широко распространён среди трейдеров, однако он чувствителен к рыночному шуму и не учитывает фундаментальные изменения. Фундаментальный подход обеспечивает глубинную оценку стоимости активов, но не адаптирован к краткосрочной динамике. Теория портфеля Марковица формализует распределение рисков, но предполагает стационарность параметров и линейную зависимость доходностей.

Методы машинного обучения предоставляют более гибкий и мощный инструментарий, позволяющий строить адаптивные модели, учитывать сложные взаимосвязи между признаками и использовать широкий спектр источников данных. Однако их применение сопряжено с определенными вызовами – в первую очередь, переобучением, утечкой данных и нестабильностью рыночных режимов.

В рамках данного исследования выбран подход, сочетающий классические технические индикаторы с современными моделями градиентного бустинга – CatBoost, LightGBM и XGBoost. Такой подход позволяет:

- повысить точность прогнозирования краткосрочного направления движения цены;

- сохранить интерпретируемость модели;

- использовать проверенные и устойчивые признаки без обращения к фундаментальным данным или сложной текстовой аналитике.

Полученные результаты лягут в основу оценки предсказательной способности моделей и сформируют основу для построения инвестиционно-аналитической системы.

1.3 Цель исследования

Целью настоящего исследования является разработка подхода к прогнозированию краткосрочного направления движения цены акций ПАО «Сбербанк» на основе исторических рыночных данных. В работе применяется комбинация технического анализа и алгоритмов машинного обучения – в частности, моделей градиентного бустинга CatBoost, LightGBM и XGBoost.

Предполагается, что предсказания модели могут быть использованы в дальнейшем как элемент аналитической поддержки в задачах управления инвестиционным портфелем. Основной акцент делается на повышении точности и устойчивости прогноза за счет использования технических индикаторов и адаптивных методов обучения.

1.3.1 Постановка задачи

Задача настоящего исследования заключается в прогнозировании краткосрочного направления изменения цены акций ПАО «Сбербанк» на основе исторических рыночных данных и технических индикаторов. Формально она представляется как задача многоклассовой классификации, в которой требуется определить направление изменения цены через k торговых дней вперёд.

Параметр k – горизонт прогноза – не фиксируется заранее, а определяется эмпирически с использованием автокорреляционной функции (ACF) на основе исторических данных. Это позволяет учитывать временную структуру цен и выбирать оптимальное расстояние для предсказания.

Целевая переменная формируется следующим образом:

$$y_{t+k} = \begin{cases} +1, & \text{if } \frac{P_{t+k}-P_t}{P_t} > 1\% \\ -1, & \text{if } \frac{P_{t+k}-P_t}{P_t} < -1\% \\ 0, & \text{if } -1\% \leq \frac{P_{t+k}-P_t}{P_t} \leq 1\% \end{cases} \quad (1)$$

где:

P_t – цена закрытия на момент времени t;

P_{t+k} – цена закрытия через k торговых дней;

y_{t+k} – целевая метка направления движения:

Таким образом, модель классифицирует поведение рынка как:

- уверенное движение вверх (рост более 1%);

- уверенное движение вниз (падение более 1%);

- нейтральное (флэт) – в пределах ±1%.

Признаковое пространство включает:

- рыночные данные: Open, High, Low, Close, Volume;

- технические индикаторы: EMA_15, EMA_75, ADX, DI+, DI−, RSI_14, ATR_14;

- сигнальные признаки: бинарные флаги технических условий и сочетаний индикаторов (например, ema_diff, rsi_above_70, strong_down_trend и др.).

Для обучения используются модели градиентного бустинга: CatBoost, LightGBM и XGBoost. Итоговое предсказание формируется методом ансамблирования с голосованием: финальный класс определяется на основе большинства предсказаний от отдельных моделей. Гиперпараметры каждой модели подбираются с помощью Optuna, обеспечивающей эффективный поиск на ограниченном числе итераций. Для предотвращения утечки данных и сохранения временной структуры используется TimeSeriesSplit. Эффективность подхода проверяется как по метрикам качества классификации, так и через сравнение с базовой стратегией Buy&Hold.

1.3.2 Задачи исследования

Для достижения поставленной цели необходимо решить следующие задачи:

1. Сбор и подготовка данных. Требуется сформировать датасет на основе исторических данных по акциям ПАО «Сбербанк» за последние 10

лет. Рассчитать технические индикаторы (EMA, RSI, ADX и др.) и сформировать сигнальные признаки.

2. Формирование целевой переменной. Требуется определить оптимальный горизонт прогноза k с использованием автокорреляционного анализа (ACF). Построить целевую метку в формате многоклассовой классификации (рост / падение / флэт).

3. Обучение и настройка моделей. Необходимо обучить модели CatBoost, XGBoost, LightGBM на расширенном признаковом пространстве. Выполнить подбор гиперпараметров с использованием Optuna.

4. Ансамблирование и предсказание. Нужно реализовать голосование моделей (majority voting) для повышения устойчивости прогнозов.

5. Оценка эффективности. Провести валидацию с помощью TimeSeriesSplit. Встроить предсказания модели в стратегию технического анализа и сравнить с базовой стратегией Buy&Hold.

Оценить стабильность модели по метрикам многоклассовой классификации.

1.3.3 Критерии оценки результата

Цель – построить модель, обеспечивающую не только более высокую доходность, но и более выгодный риск-профиль по сравнению с пассивной стратегией.

Оценка качества разработанной модели осуществляется на основе следующих критериев:

Классические метрики многоклассовой классификации:

- Accuracy – общая точность классификации;

- Precision, Recall, F1-score – по каждому классу и в агрегированной форме (macro- и weighted-усреднение);

- Macro-F1 – как основная сводная метрика баланса точности и полноты.

Сравнительная доходность стратегии, построенной на основе сигналов модели, по отношению к стратегии Buy&Hold.

Коэффициент Сортино (Sortino ratio) – как показатель эффективности стратегии с учетом риска отрицательных отклонений. Цель – достижение Sortino ratio выше, чем у Buy&Hold. Стабильность качества модели на различных временных интервалах, оцененная с помощью временной кросс-валидации (TimeSeriesSplit). Интерпретируемость признаков на основе анализа feature importance (встроенные оценки и методы SHAP / permutation importance).

2 МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ

Разрабатываемое решение охватывает как исследовательскую часть (выбор признаков, построение моделей, оценка качества), так и практическую реализацию полной инфраструктуры жизненного цикла модели – от загрузки данных до автоматизации обучения и логирования. Такой подход соответствует принципам MLOps и позволяет обеспечить воспроизводимость, масштабируемость и интеграцию модели в дальнейшие процессы аналитической поддержки или принятия решений.

В ходе работы реализована архитектура, включающая:

- автоматический сбор и обогащение рыночных данных;
- построение целевой переменной с использованием временных зависимостей;
- обучение и ансамблирование моделей CatBoost, XGBoost и LightGBM;
- автоматизированный подбор гиперпараметров с помощью Optuna;
- логирование экспериментов и версионирование моделей в MLflow;
- контейнеризация компонентов через Docker и их развёртывание в Kubernetes-кластере (Yandex Cloud);
- управление инфраструктурой с использованием Terraform;

- настройку CI/CD пайплайнов на GitHub Actions для автоматического запуска обучения и публикации моделей.

Таким образом, модель решения охватывает как алгоритмическую, так и инженерную составляющую. Далее в разделе представлены архитектура решения, стек используемых технологий и применяемые алгоритмы машинного обучения.

2.1 Концепция и архитектура решения

Модель решения задачи представляет собой систему, в которой реализованы как алгоритмические, так и инженерные компоненты. Архитектура построена по принципу разделения функциональности на этапы: обработка данных, обучение моделей, формирование прогнозов и автоматизация жизненного цикла модели (MLOps)[14].

Цель – не только построить предсказательную модель, но и обеспечить ее воспроизводимость, отслеживаемость, переобучение по расписанию и возможность масштабирования.

Система реализована в виде пайплайна, в котором объединены алгоритмические и инженерные компоненты: автоматическая загрузка данных через API, построение признаков, обучение ансамбля моделей, логирование в MLflow и деплой в облаке Yandex Cloud. Управление инфраструктурой осуществляется с использованием подхода Infrastructure as Code (IaC) на базе Terraform и Helm.

2.1.1 Основные элементы архитектуры

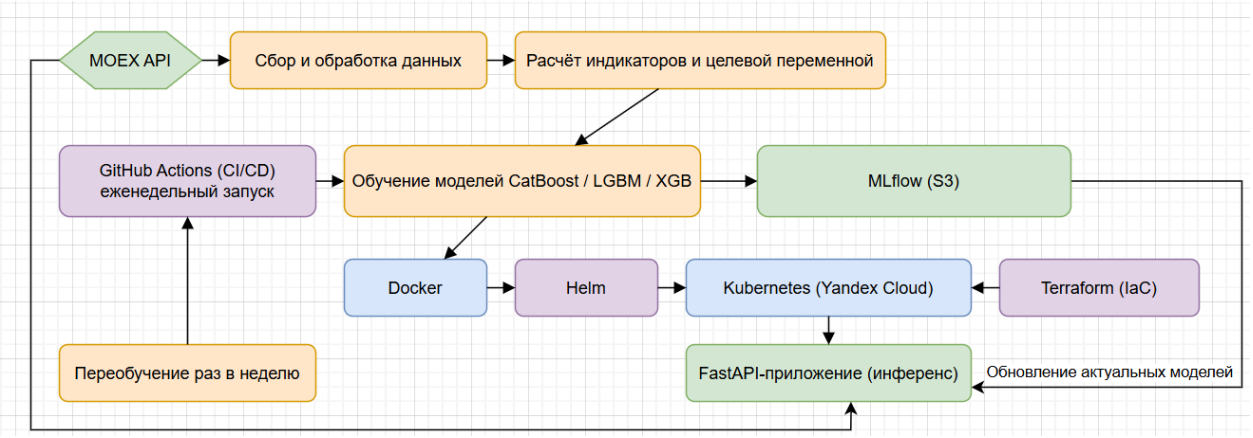


Рисунок 1 – Архитектура MLOps-процесса [20]

Источник данных

На рисунке 1 представлена архитектура MLOps процесса.

Исторические рыночные данные по акциям ПАО «Сбербанк» запрашиваются с помощью официального API Московской биржи (MOEX).

Скрипты осуществляют регулярный сбор данных с параметрами тикера, таймфрейма и интервала.

Данные сохраняются в формате CSV и используются для расчета технических индикаторов и построения целевой переменной.

ML-платформа

В проекте используется MLflow Tracking Server для логирования экспериментов, параметров моделей, метрик и артефактов.

Метаданные логируются в PostgreSQL, модели и артефакты сохраняются в S3-совместимом хранилище Yandex Cloud.

Все этапы обучения моделей снабжены логированием в MLflow, включая input_example, signature и графики.

CI/CD пайплайн

Обучение моделей автоматизировано с помощью GitHub Actions.

Пайплайн включает:

- загрузку данных;

- расчёт признаков;

- обучение моделей;

- логирование результатов;

- упаковку модели в Docker-образ.

Контейнеризация и развертывание

Модели и вспомогательные сервисы (например, FastAPI-инференс) упакованы в Docker-контейнеры.

Развёртывание осуществляется в Kubernetes-кластере, развернутом в Yandex Cloud.

Используются best practices для безопасности и отказоустойчивости.

Управление инфраструктурой

Инфраструктура описана в виде кода с использованием Terraform: развёртывание S3, PostgreSQL, K8s-кластера, namespace, сервисов и ресурсов;

Для установки и конфигурации компонентов в кластере применяется Helm:

- установка MLflow, базы данных, моделей и сервисов мониторинга через шаблоны Chart’ов;

- быстрое масштабирование и повторное развертывание в разных окружениях.

Воспроизводимость и контроль версий:

- все зависимости проекта зафиксированы в requirements.txt и Doc kerfile;

- эксперименты версионируются и привязываются к git-коммитам;

При необходимости модель можно переобучить на той же выборке с теми же параметрами и получить идентичный результат.

2.2 Стек технологий

Перед построением системы прогнозирования и её автоматизации был выбран набор технологий, охватывающих весь жизненный цикл модели: от получения и обработки данных до обучения, логирования и промышленного

развертывания. Подбор инструментов основан на современных лучших практиках в области Data Science, MLOps и разработки облачных решений. Используемый стек обеспечивает не только высокую точность моделей, но и их воспроизводимость, масштабируемость и простоту поддержки.

В таблицах ниже представлены используемые технологии, сгруппированные по логическим этапам решения задачи.

Таблица 2 – Технологии получения и обработки данных

Технологии	Назначение
MOEX API	Загрузка исторических данных по акциям
Pandas, NumPy	Агрегация, очистка и преобразование временных рядов
pandas_ta	Расчёт технических индикаторов (EMA, RSI, ADX, ATR и др.)

Таблица 3 – Алгоритмы машинного обучения и оптимизация

Технологии	Назначение
CatBoost	Градиентный бустинг, устойчивый к категориальным признакам
LightGBM	Быстрый бустинг с leaf-wise деревьями
XGBoost	Гибкий и точный бустинг с регуляризацией
Optuna	Байесовская оптимизация гиперпараметров

Таблица 4 – Оценка качества моделей и интерпретация

Технологии	Назначение
scikit-learn	Метрики: Accuracy, F1, Precision, Recall; TimeSeriesSplit
SHAP[18]	Оценка важности признаков и интерпретация модели
BackTesting.py	Симуляция торговли на исторических данных
pyfolio	Визуализация: капитал, просадки, доходность, сравнение с бенчмарками

Таблица 5 – MLOps и автоматизация

Технологии	Назначение
GitHub Actions	CI/CD: автоматический запуск обучения и логирования
MLflow	Логирование параметров, метрик и моделей

Таблица 6 – Инфраструктура и деплой

Технологии	Назначение
Docker	Контейнеризация компонентов
Helm	Установка и обновление сервисов в Kubernetes
Terraform	Infrastructure as Code для развертывания ресурсов
PostgreSQL + S3 (Yandex Cloud)	Хранение метаданных и моделей
Kubernetes (Yandex Cloud)	Оркестрация и масштабирование

Таблица 7 – Инференс (использование модели)

Технологии	Назначение
FastAPI	REST API для предсказаний обученной модели

2.3 Алгоритмы и методы

В рамках решения задачи краткосрочного прогнозирования направления движения цены акций ПАО «Сбербанк» были выбраны три алгоритма градиентного бустинга: CatBoost, LightGBM и XGBoost. Эти модели продемонстрировали высокую устойчивость и точность при работе с временными рядами и табличными данными, содержащими технические индикаторы и сигнальные признаки.

Для повышения надежности предсказаний был реализован подход ансамблирования[15] через голосование (majority voting), при котором итоговый класс определяется большинством голосов трёх моделей[16]. Также применялась оптимизация гиперпараметров с помощью Optuna[17], а для оценки качества – кросс-валидация по временным отрезкам (TimeSeriesSplit), соответствующая особенностям финансовых данных.

Ниже приведено описание каждой из применяемых моделей, включая особенности их работы и причины выбора.

2.3.1 CatBoost

CatBoost – это библиотека градиентного бустинга, разработанная компанией Яндекс, отличающаяся высокой устойчивостью к переобучению и встроенной обработкой категориальных признаков без необходимости их явного кодирования. В рамках данного проекта CatBoost использовался как один из базовых алгоритмов классификации направления движения цены.

Особенности применения:

Работает по схеме градиентного бустинга над симметричными деревьями решений.

Обладает встроенной поддержкой обработки пропусков и категориальных признаков.

Поддерживает регуляризацию и раннюю остановку (early stopping), что полезно при работе с ограниченным объёмом данных.

Обеспечивает высокую интерпретируемость благодаря встроенной функции важности признаков (get_feature_importance), совместимой с SHAP.

Применение в проекте:

Использовался на расширенном наборе признаков, включающем OHLCV-данные, технические индикаторы и сигнальные флаги.

Обучался с учётом временной структуры выборки (TimeSeriesSplit).

Предсказания модели входили в итоговый ансамбль через голосование majority voting.

2.3.2 LightGBM

LightGBM (Light Gradient Boosting Machine) – это фреймворк градиентного бустинга, разработанный Microsoft и ориентированный на высокую скорость обучения и работу с большими объемами данных. Он особенно эффективен при работе с числовыми признаками и широко применяется в соревнованиях по анализу данных.

Особенности применения:

Реализует бустинг на основе leaf-wise стратегии роста дерева, что позволяет достигать более низких ошибок по сравнению с традиционным level-wise подходом.

Поддерживает методы ускорения обучения, включая GOSS (Gradient-based One-Side Sampling) и Exclusive Feature Bundling (EFB).

Требует предварительной обработки пропусков и категориальных признаков, в отличие от CatBoost.

Имеет гибкую систему настройки гиперпараметров, что делает его мощным инструментом при оптимизации под конкретные задачи.

Применение в проекте:

Обучался на том же признаковом пространстве, что и другие модели (индикаторы, сигналы, OHLCV).

Кросс-валидация выполнялась с разделением по времени (TimeSeriesSplit), с учетом зависимости временных рядов.

Оптимизация гиперпараметров осуществлялась с помощью Optuna.

Предсказания участвовали в объединенной схеме голосования.

2.3.3 XGBoost

XGBoost (Extreme Gradient Boosting) – одна из самых популярных реализаций градиентного бустинга, широко применяемая в промышленной аналитике и соревнованиях по машинному обучению (Kaggle, DrivenData и др.). Модель отличается высокой производительностью, встроенной регуляризацией и устойчивостью к переобучению.

Особенности применения:

Использует бустинг над деревьями решений с дополнительными механизмами регуляризации (L1 и L2), что позволяет контролировать сложность модели.

Поддерживает встроенную обработку пропущенных значений.

Может быть эффективно распараллелен, что ускоряет обучение на больших объемах данных.

Обеспечивает подробную статистику важности признаков и совместим с SHAP для интерпретации решений.

Применение в проекте:

Использовался как третий базовый классификатор в ансамбле.

Работал с теми же признаками: технические индикаторы, сигнальные флаги и исторические рыночные данные.

Обучение проводилось с учётом временной структуры данных (TimeSeriesSplit).

Подбор параметров выполнен с использованием Optuna.

Предсказания модели учитывались при принятии итогового решения через механизм голосования.

3 ОПИСАНИЕ РЕШЕНИЯ

3.1 Программный код

В данном разделе представлены ключевые фрагменты программного кода, реализующего функциональность решения. Код разбит на логические блоки, каждый из которых соответствует отдельному компоненту разработанной системы:

- архитектурная и MLOps-составляющая;
- обучение и логирование моделей;

- процесс переобучения;

- FastAPI-приложение для инференса.

Каждый блок сопровождается листингами с пояснениями, иллюстрирующими реализацию соответствующих компонентов. Для удобства восприятия приведены наиболее значимые фрагменты, обеспечивающие воспроизводимость и переносимость решения. Полная версия размещена на github.

3.1.1 Архитектура и MLOps

Данный раздел описывает инфраструктурные аспекты реализации решения, включая автоматизированное развертывание компонентов, настройку окружения, а также процессы CI/CD. Представлены ключевые элементы конфигурации облачных ресурсов (S3, Kubernetes-кластер), инструменты управления (Terraform, Docker), а также пайплайны для обучения и деплоя моделей с использованием GitHub Actions и MLflow.

Все элементы ориентированы на воспроизводимость, масштабируемость и стабильное функционирование системы в продакшене.

Создание облачной инфраструктуры

Для развертывания компонентов системы машинного обучения и автоматизации процессов инференса в облачной среде была спроектирована и реализована инфраструктура в Yandex Cloud с использованием Terraform. Это обеспечило декларативное описание всех ресурсов и возможность быстрого воспроизводимого развертывания.

Инфраструктура включает следующие ключевые компоненты:

- Виртуальная сеть (VPC) и подсеть в зоне ru-central1-a для изоляции сервисов и организации сетевого взаимодействия;

- Кластер PostgreSQL, используемый для хранения метаданных экспериментов и обучающих данных, необходимых MLflow;

- Объектное хранилище (S3) – для сохранения артефактов моделей, логов и вспомогательных файлов;

- Кластер Kubernetes, предназначенный для размещения FastAPI-сервиса, MLflow-сервера, а также других сервисов инференса;

- Группа рабочих узлов, размещенных в рамках кластера, с параметрами, обеспечивающими минимально необходимую производительность и устойчивость к сбоям.

Ниже представлены фрагмент конфигурации Terraform, описывающий создание всех перечисленных компонентов.

Terraform

Terraform используется для декларативного описания и автоматизированного развертывания облачной инфраструктуры в Yandex Cloud. С его помощью создаются все необходимые ресурсы: виртуальная сеть, подсеть, кластер PostgreSQL для хранения метаданных, объектное S3-хранилище для артефактов моделей, а также кластер Kubernetes с группой рабочих узлов для размещения сервисов. Такой подход обеспечивает воспроизводимость, масштабируемость и простоту поддержки инфраструктуры в ходе всего жизненного цикла проекта.

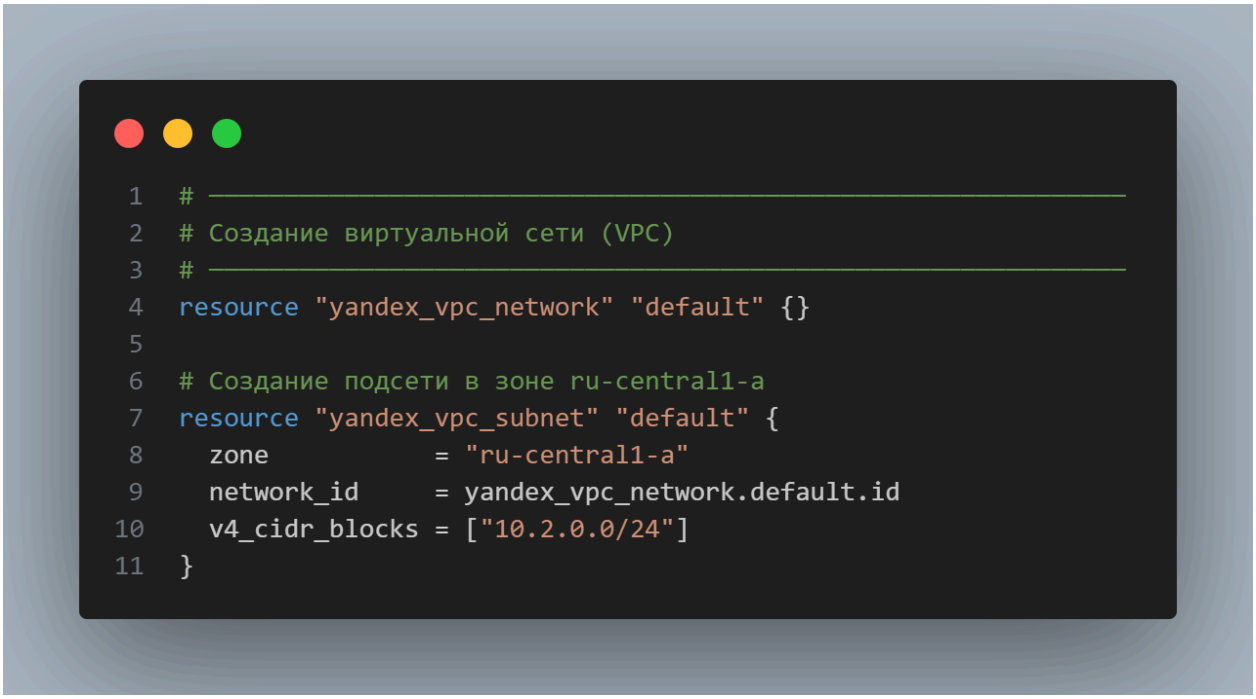


Рисунок 2 – Конфигурация виртуальной сети и подсети

На рисунке 2 представлена базовая конфигурация виртуальной сети (VPC) и подсети в зоне доступности ru-central1-a, реализованная с помощью Terraform.

Сначала создаётся ресурс yandex_vpc_network, определяющий виртуальную сеть без дополнительных параметров. Далее определяется ресурс yandex_vpc_subnet, привязанный к данной сети.

Параметр v4_cidr_blocks задаёт диапазон IPv4-адресов для создаваемой подсети, в данном случае 10.2.0.0/24. Эта сеть используется для размещения всех остальных компонентов инфраструктуры: баз данных, кластеров Kubernetes и сервисов приложений.

```
# ──────────────────────────────────────────────────────────────
# PostgreSQL-кластер для MLFlow и хранения обучающих данных
# ──────────────────────────────────────────────────────────────
resource "yandex_mdb_postgresql_cluster" "pg" {
  name        = "mlflow-db"
  environment = "PRODUCTION"
  network_id  = yandex_vpc_network.default.id

  config {
    version = 14
    resources {
      resource_preset_id = "s2.micro"       # дешёвый вариант для старта
      disk_size          = 20               # размер в ГБ
      disk_type_id       = "network-ssd"
    }
  }

  host {
    zone      = "ru-central1-a"
    subnet_id = yandex_vpc_subnet.default.id
    name      = "pg-host"
  }
}

# Создание базы данных внутри кластера
resource "yandex_mdb_postgresql_database" "mlflow" {
  cluster_id = yandex_mdb_postgresql_cluster.pg.id
  name       = "mlflow"
  owner      = var.db_user
}

# Создание пользователя с паролем (берётся из переменной окружения)
resource "yandex_mdb_postgresql_user" "mlflow" {
  cluster_id = yandex_mdb_postgresql_cluster.pg.id
  name       = var.db_user
  password   = var.db_password
}
```

Рисунок 3 – Конфигурация кластера PostgreSQL и базы данных для MLflow

В рисунке 3 показана конфигурация управляемого PostgreSQL-кластера в Yandex Cloud, используемого для хранения метаданных MLflow.

Ресурс yandex_mdb_postgresql_cluster описывает параметры самого кластера: имя, тип окружения (PRODUCTION), версия PostgreSQL, объём диска, тип хранилища и расположение в зоне ru-central1-a.

Следом создаётся база данных mlflow (yandex_mdb_postgresql_database) и пользователь (yandex_mdb_postgresql_user), имя и пароль которого задаются через переменные окружения (var.db_user, var.db_password).

Такая конфигурация позволяет безопасно и централизованно управлять хранением информации о запусках экспериментов, гиперпараметрах и результатах моделей.

```
# ──────────────────────────────────────────────────────────
# S3-хранилище для моделей, артефактов и логов MLFlow
# ──────────────────────────────────────────────────────────
resource "random_id" "bucket_id" {
  byte_length = 4
}

resource "yandex_storage_bucket" "models_bucket" {
  bucket   = "ml-models-${random_id.bucket_id.hex}" # уникальное имя
  max_size = 10737418240  # 10 ГБ
  acl      = "private"
}
```

Рисунок 4 – Конфигурация S3-хранилища для моделей

На рисунке 4 представленная конфигурация S3-хранилища. В приведённой конфигурации создаётся S3-совместимое хранилище в Yandex Object Storage, используемое для сохранения моделей, артефактов и логов MLflow.

Для генерации уникального имени бакета применяется ресурс random_id, обеспечивающий отсутствие коллизий при развёртывании в разных окружениях.

Хранилище конфигурируется как приватное (acl = «private»), с максимальным допустимым объёмом в 10 ГБ.

Данный ресурс критически важен для организации централизованного хранения результатов обучения и обеспечения связности между MLflow-сервером, обучающими пайплайнами и системой инференса.

```
# ──────────────────────────────────────────────────────────
# Kubernetes-кластер для инференса и MLFlow
# ──────────────────────────────────────────────────────────
resource "yandex_kubernetes_cluster" "mlops_cluster" {
  name       = "mlops-cluster"
  network_id = yandex_vpc_network.default.id

  master {
    version = "1.30"
    zonal {
      zone      = "ru-central1-a"
      subnet_id = yandex_vpc_subnet.default.id
    }

    public_ip = true
  }

  service_account_id      = var.k8s_sa_id
  node_service_account_id = var.k8s_sa_id

  release_channel         = "RAPID"
  network_policy_provider = "CALICO"
}

```

Рисунок 5 – Конфигурация Kubernetes-кластера

На рисунке 5 представлена конфигурация управляемого Kubernetes-кластера в Yandex Cloud, предназначенного для развертывания MLflow, FastAPI-приложения и связанных сервисов.

Ресурс yandex_kubernetes_cluster описывает основные параметры: версию Kubernetes (1.30), зону размещения (ru-central1-a), сетевые настройки и назначение публичного IP-адреса мастеру.

Указаны сервисные аккаунты, необходимые для доступа к облачным ресурсам из подов. Также задаются канал обновлений (RAPID) и провайдер сетевой политики (CALICO).

Конфигурация обеспечивает готовую среду для запуска MLOps-пайплайнов, CI/CD и инференс-сервисов в изолированном и управляемом окружении.

```
# Группа рабочих узлов для запуска подов
resource "yandex_kubernetes_node_group" "default" {
  cluster_id = yandex_kubernetes_cluster.mlops_cluster.id
  name       = "default-pool"
  version = "1.30"

  instance_template {
    platform_id = "standard-v2"
    resources {
      memory = 4
      cores  = 2
    }

    boot_disk {
      type = "network-ssd"
      size = 50
    }

    network_interface {
      subnet_ids = [yandex_vpc_subnet.default.id]
      nat        = true
    }

    scheduling_policy {
      preemptible = true # более дешёвые, но нестабильные ВМ
    }
  }

  scale_policy {
    fixed_scale {
      size = 2 # количество рабочих узлов
    }
  }

  allocation_policy {
    location {
      zone = "ru-central1-a"
    }
  }
```

Рисунок 6 – Группа рабочих узлов для запуска подов

На рисунке 6 показана конфигурация ресурса yandex_kubernetes_node_group, создающего группу рабочих узлов в ранее определённом Kubernetes-кластере.

Каждый узел развертывается на платформе standard-v2, с параметрами 4 ГБ оперативной памяти и 2 виртуальными ядрами. В качестве диска используется SSD объемом 50 ГБ.

Установлен флаг preemptible = true, что позволяет использовать более дешёвые, но неустойчивые виртуальные машины (подходящие для тестовых или непостоянных нагрузок).

Политика масштабирования задана фиксировано: в группе создаются два узла. Также определено географическое размещение в зоне ru-central1-a.

Этот ресурс завершает базовую инфраструктуру, обеспечивая физические вычислительные ресурсы для размещения всех подов, включая MLflow, FastAPI и сервисы инференса.

Конфигурация Helm-чартов сервисов MLflow и FastAPI

Для автоматизации развертывания микросервисов MLflow и FastAPI в Kubernetes-кластере использовались Helm-чарты. Каждый сервис представлен в виде отдельной директории с набором шаблонов (deployment.yaml, service.yaml) и конфигурационных файлов (values.yaml). Такой подход обеспечивает модульность, повторное использование и гибкость при настройке окружений. Ниже приведены ключевые фрагменты конфигурации Helm-чартов, реализующих деплой и настройку сервисов.

```yaml
replicaCount: 1

image:
  repository: bitnami/mlflow
  tag: "2.10.1-debian-11-r0"

service:
  type: LoadBalancer
  port: 8080

env:
  backendStoreUri: "postgresql://mlflow:${PG_PASSWORD_URL_ENCODED}@${PG_HOST}:6432/mlflow"
  artifactRoot: "s3://${BUCKET_NAME}/artifacts"
  s3EndpointUrl: "https://storage.yandexcloud.net"
  useSecret: true
  secretName: mlflow-s3-credentials

```

Рисунок 7 – Конфигурация Helm-чарта MLflow
(charts/mlflow/values-template.yaml)

На рисунке 7 представлена типовая конфигурация Helm-чарта для MLflow, задаваемая через файл values-template.yaml.

В параметрах указывается образ MLflow (в данном случае – официальный bitnami/mlflow), порт для публикации сервиса и тип доступа (LoadBalancer, что обеспечивает внешний доступ к MLflow-интерфейсу).

В блоке env задаются значения для подключения к backend-хранилищу метаданных (PostgreSQL), корневой путь для хранения артефактов в S3, а также URL-адрес S3-совместимого хранилища.

При включенном флаге useSecret: true параметры доступа к хранилищу подгружаются из Kubernetes-секрета, имя которого задается через secretName.

Такой шаблон позволяет легко переиспользовать конфигурацию и адаптировать Helm-чарт под различные окружения (dev, staging, prod) без необходимости модификации шаблонов вручную.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mlflow
  labels:
    app: mlflow
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: mlflow
  template:
    metadata:
      labels:
        app: mlflow
    spec:
      containers:
        - name: mlflow
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy | default "IfNotPresent" }}
          ports:
            - containerPort: 8080
          command: ["mlflow"]
          args:
            - "server"
            - "--backend-store-uri=$(BACKEND_STORE_URI)"
            - "--default-artifact-root=$(ARTIFACT_ROOT)"
            - "--host=0.0.0.0"
            - "--port=8080"
          env:
            - name: BACKEND_STORE_URI
              value: "{{ .Values.env.backendStoreUri }}"
            - name: ARTIFACT_ROOT
              value: "{{ .Values.env.artifactRoot }}"
            - name: MLFLOW_S3_ENDPOINT_URL
              value: "{{ .Values.env.s3EndpointUrl }}"
          {{- if .Values.env.useSecret }}
          envFrom:
            - secretRef:
                name: {{ .Values.env.secretName }}
          {{- end }}
```

Рисунок 8 – Deployment-шаблон MLflow
(charts/mlflow/templates/deployment.yaml)

На рисунке 8 представлен шаблон Helm-чарта, реализующий деплоймент MLflow-сервера в Kubernetes.

Шаблон параметризован: образ, количество реплик, политика загрузки образа (imagePullPolicy), порты, а также переменные окружения задаются через values.yaml.

Особенность данного шаблона заключается в передаче конфигурации через аргументы командной строки (args), включая путь к backend-хранилищу (PostgreSQL) и объектному хранилищу артефактов (S3).

Для безопасной работы с чувствительными данными предусмотрена возможность подключения Kubernetes-секрета (secretRef).

Таким образом, шаблон обеспечивает гибкое и безопасное развертывание MLflow с сохранением принципов DevOps и инфраструктуры как кода.

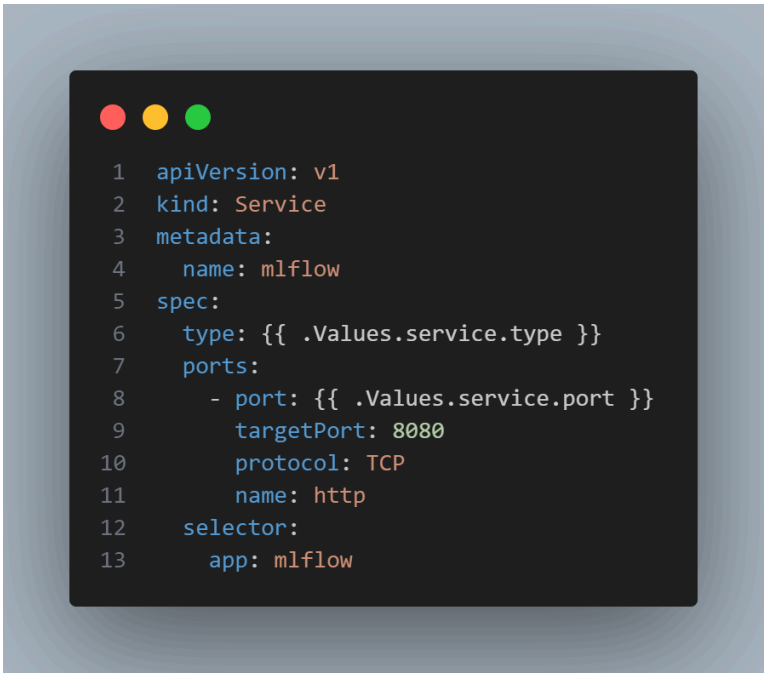


Рисунок 9 – Сервис MLflow (charts/mlflow/templates/service.yaml)

На рисунке 9 представлен шаблон Kubernetes-сервиса, создаваемого в рамках Helm-чарта для MLflow. Сервис обеспечивает сетевой доступ к MLflow-серверу внутри кластера по протоколу TCP.

Параметры порта (port, targetPort) и тип сервиса (ClusterIP, NodePort, LoadBalancer) подставляются из конфигурационного файла values.yaml.

Селектор app: mlflow указывает, что трафик будет перенаправляться на поды с соответствующей меткой, что обеспечивает корректную маршрутизацию запросов.

```yaml
replicaCount: 1

containerPort: 9090

image:
  repository: ghcr.io/minakovmax/fastapi-app
  tag: latest
  pullPolicy: Always
  pullSecrets:
    - name: ghcr-auth

service:
  type: NodePort
  port: 80
  targetPort: 9090
  nodePort: 31090 # новый NodePort

env:
  - name: MLFLOW_S3_ENDPOINT_URL
    value: https://storage.yandexcloud.net
  - name: AWS_ACCESS_KEY_ID
    valueFrom:
      secretKeyRef:
        name: mlflow-secrets
        key: AWS_ACCESS_KEY_ID
  - name: AWS_SECRET_ACCESS_KEY
    valueFrom:
      secretKeyRef:
        name: mlflow-secrets
        key: AWS_SECRET_ACCESS_KEY

```

Рисунок 10 – Конфигурация Helm-чарта FastAPI (charts/fastapi/values.yaml)

На рисунке 10 представлен конфигурационный файл values.yaml, который задаёт параметры для шаблонов Helm-чарта FastAPI-приложения. В частности:

– указывается Docker-образ приложения и политика его обновления (pullPolicy: Always);

– описан тип сервиса NodePort, позволяющий внешнее подключение к приложению через порт 31000;

– определён порт, на котором запущен контейнер (containerPort: 9090);

– в блоке env перечислены переменные окружения, включая доступ к объектному хранилищу Yandex Cloud и ключам AWS, получаемым из Kubernetes-секрета mlflow-secrets.

Такой подход обеспечивает безопасное и гибко управляемое развертывание FastAPI-сервиса в различных окружениях без прямого хранения чувствительных данных в открытом виде.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Chart.Name }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Chart.Name }}
  template:
    metadata:
      labels:
        app: {{ .Chart.Name }}
    spec:
      imagePullSecrets:
        {{- range .Values.image.pullSecrets }}
        - name: {{ .name }}
        {{- end }}
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        ports:
        - containerPort: {{ .Values.containerPort }}
        env:
        {{- toYaml .Values.env | nindent 10 }}

```

Рисунок 11 – Deployment-шаблон FastAPI
(charts/fastapi/templates/deployment.yaml)

На рисунке 11 представлен шаблон Helm-чарта, описывающий развёртывание FastAPI-приложения в Kubernetes. Параметры, такие как количество реплик (replicaCount), имя образа (image.repository и image.tag), порт приложения (containerPort), переменные окружения (env), а также секреты для доступа к приватным реестрам (imagePullSecrets) подставляются из файла values.yaml.

Это позволяет гибко управлять конфигурацией при развертывании приложения в различных окружениях без необходимости вручную редактировать YAML-файлы.

Рисунок 12 – Сервис FastAPI (charts/fastapi/templates/service.yaml)

Рисунок 12 показывает шаблон Service который, описывает сетевое подключение к FastAPI-приложению внутри кластера Kubernetes. Тип сервиса (ClusterIP, NodePort, LoadBalancer), внутренний порт, целевой порт и (при необходимости) NodePort задаются через файл values.yaml. Такое параметризованное описание позволяет легко управлять конфигурацией при помощи Helm без необходимости править YAML-файлы вручную.

CI/CD-процесс с использованием GitHub Actions

Для автоматизации жизненного цикла модели и упрощения управления развертыванием компонентов решения используется пайплайн CI/CD, реализованный на базе GitHub Actions. Такой подход обеспечивает воспроизводимость экспериментов, своевременное обновление моделей и минимизирует участие человека в процессе сборки и доставки кода.

Пайплайн состоит из нескольких этапов:

- установка зависимостей и подготовка окружения;
- запуск обучения модели и логирование результатов в MLflow;
- сборка Docker-образа FastAPI-приложения;
- публикация образа в контейнерный реестр;
- автоматическое развёртывание в кластер Kubernetes с использованием Helm.

Использование CI/CD-практик позволяет обеспечить согласованность всех компонентов, снизить риски, связанные с «ручным» вмешательством, и ускорить выкатку изменений.

```yaml
name: Deploy MLflow to Kubernetes
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    env:
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      PG_PASSWORD_URL_ENCODED: ${{ secrets.PG_PASSWORD_URL_ENCODED }}
      PG_HOST: ${{ secrets.PG_HOST }}
      BUCKET_NAME: ${{ secrets.BUCKET_NAME }}
    steps:
    - name: Checkout repository
      uses: actions/checkout@v3
    - name: Set up kubectl
      uses: azure/setup-kubectl@v3
      with:
        version: "v1.28.0"
    - name: Set up Helm
      uses: azure/setup-helm@v3
    - name: Configure kubeconfig
      run: |
        echo "${{ secrets.KUBECONFIG }}" > kubeconfig
        echo "KUBECONFIG=$PWD/kubeconfig" >> $GITHUB_ENV
    - name: Export env and render values.yaml
      shell: bash
      run: |
        export PG_PASSWORD_URL_ENCODED="${{ secrets.PG_PASSWORD_URL_ENCODED }}"
        export PG_HOST="${{ secrets.PG_HOST }}"
        export BUCKET_NAME="${{ secrets.BUCKET_NAME }}"
        envsubst < charts/mlflow/values-template.yaml > charts/mlflow/values.yaml
    - name: Helm upgrade MLflow
      run: |
        helm upgrade --install mlflow ./charts/mlflow -f charts/mlflow/values.yaml

```

Рисунок 13 – Пайплайн для автоматического деплоя MLflow

На рисунке 13 представлен workflow-файл который отвечает за автоматическую установку и обновление MLflow-сервера в Kubernetes-кластере при каждом пуше в ветку main.

Он включает следующие шаги: установка kubectl и helm,настройка доступа к кластеру через KUBECONFIG, рендеринг конфигурационного файла values.yaml на основе шаблона values-template.yaml с подстановкой переменных окружения,установка или обновление релиза Helm-чарта MLflow.

Такой подход позволяет централизованно управлять выкладкой MLflow, обеспечивая гибкость за счёт шаблонов и безопасность за счёт использования GitHub Secrets.

На рисунке 14 представлен CI/CD-процесс обучения и деплоя модели

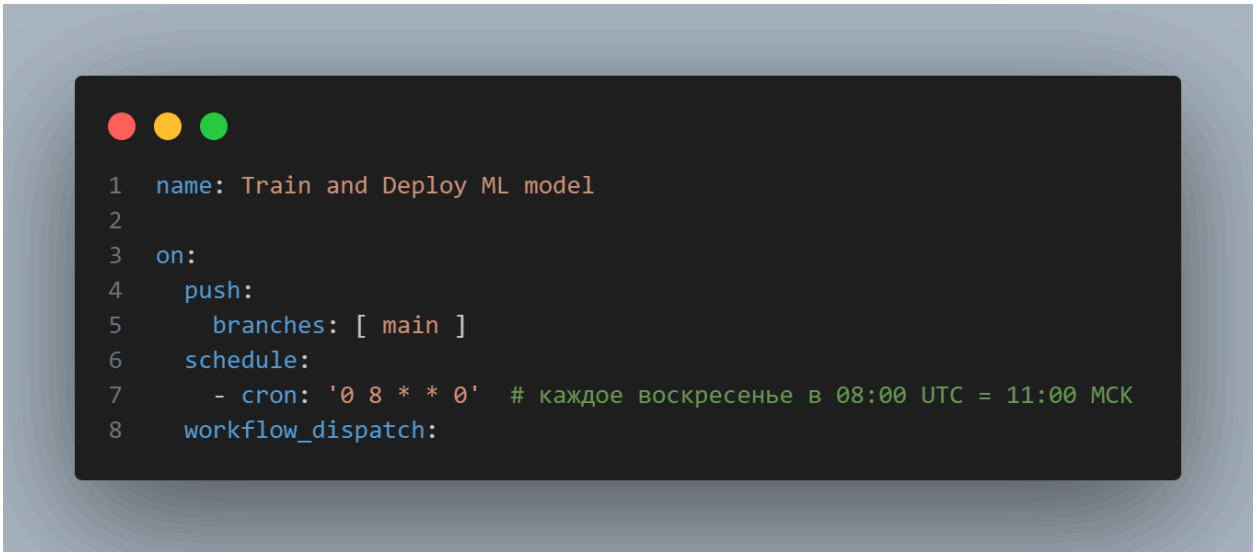
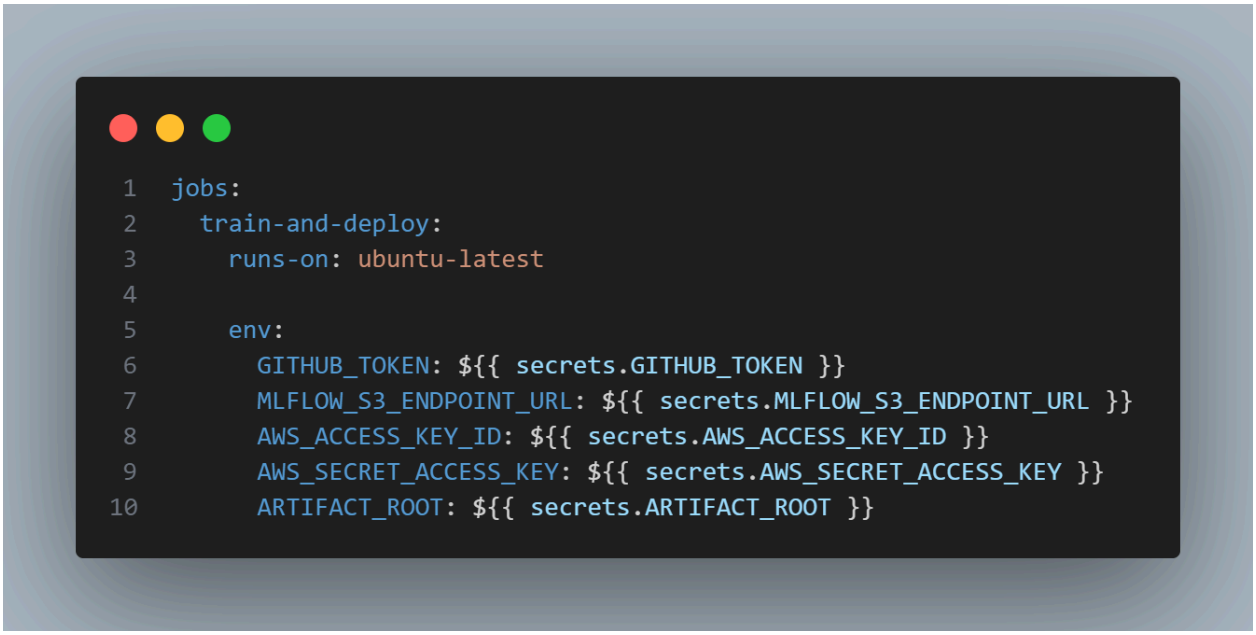


Рисунок 14 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml). Метаданные и триггеры

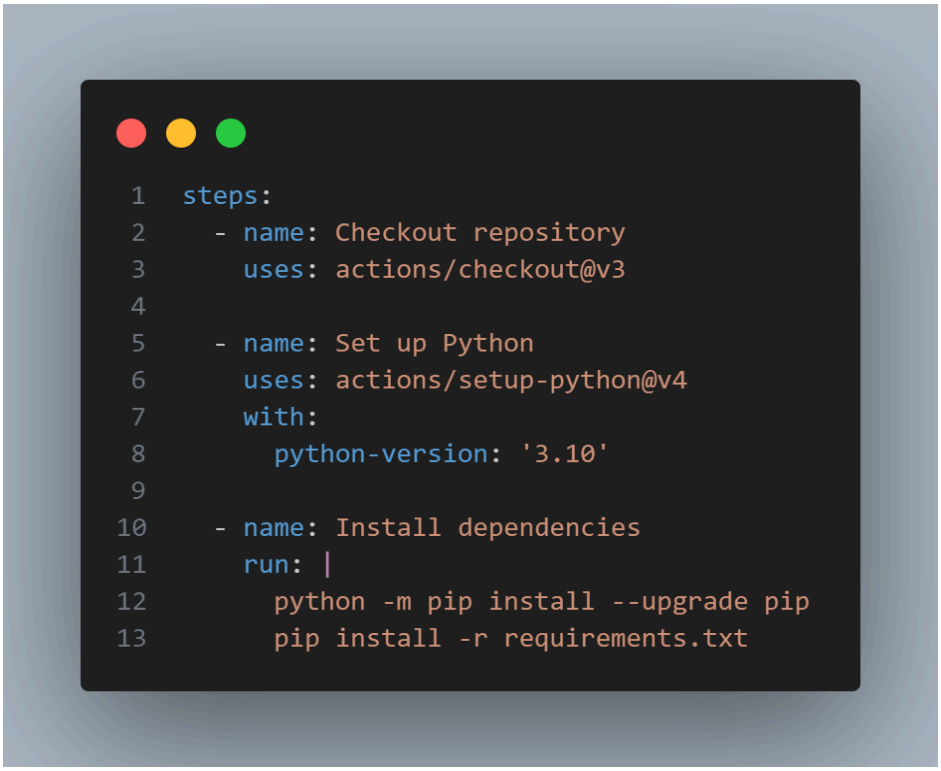
Пайплайн запускается вручную, по коммиту в main и по расписанию раз в неделю. Это обеспечивает как контроль вручную, так и регулярное переобучение модели.



```yaml
jobs:
  train-and-deploy:
    runs-on: ubuntu-latest

    env:
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      MLFLOW_S3_ENDPOINT_URL: ${{ secrets.MLFLOW_S3_ENDPOINT_URL }}
      AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      ARTIFACT_ROOT: ${{ secrets.ARTIFACT_ROOT }}
```

Рисунок 15 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml). Определение окружения и секретов

На рисунке 15 объявлены секреты и переменные окружения, необходимые для логирования модели в MLflow и доступа к S3-хранилищу.



```yaml
steps:
  - name: Checkout repository
    uses: actions/checkout@v3

  - name: Set up Python
    uses: actions/setup-python@v4
    with:
      python-version: '3.10'

  - name: Install dependencies
    run: |
      python -m pip install --upgrade pip
      pip install -r requirements.txt
```

Рисунок 16 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml) Подготовка окружения и зависимостей

На рисунке 16 выполняется клонирование репозитория, установка Python и зависимостей.

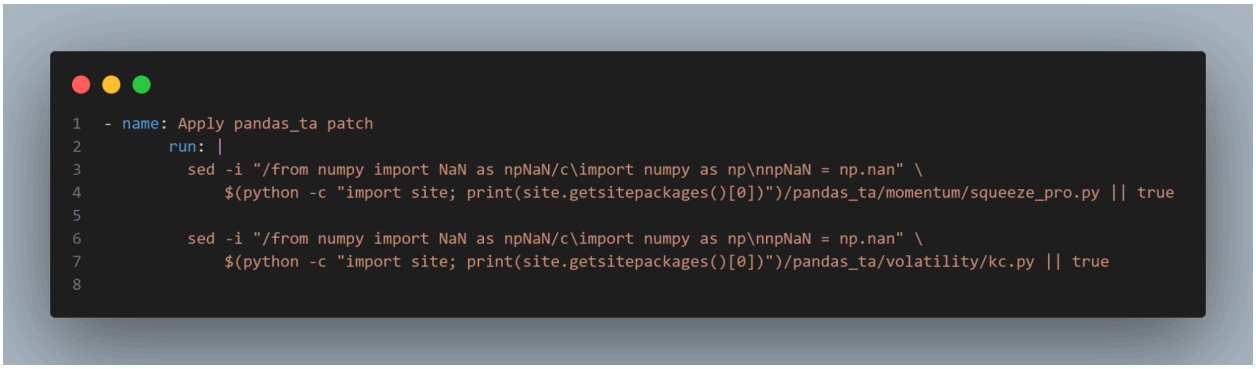


Рисунок 17 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml) Исправление внешнего пакета (патч для pandas_ta)

На рисунке 17 показано автоматическое исправление ошибок в сторонней библиотеке pandas_ta для корректной работы модели без ручного вмешательства.

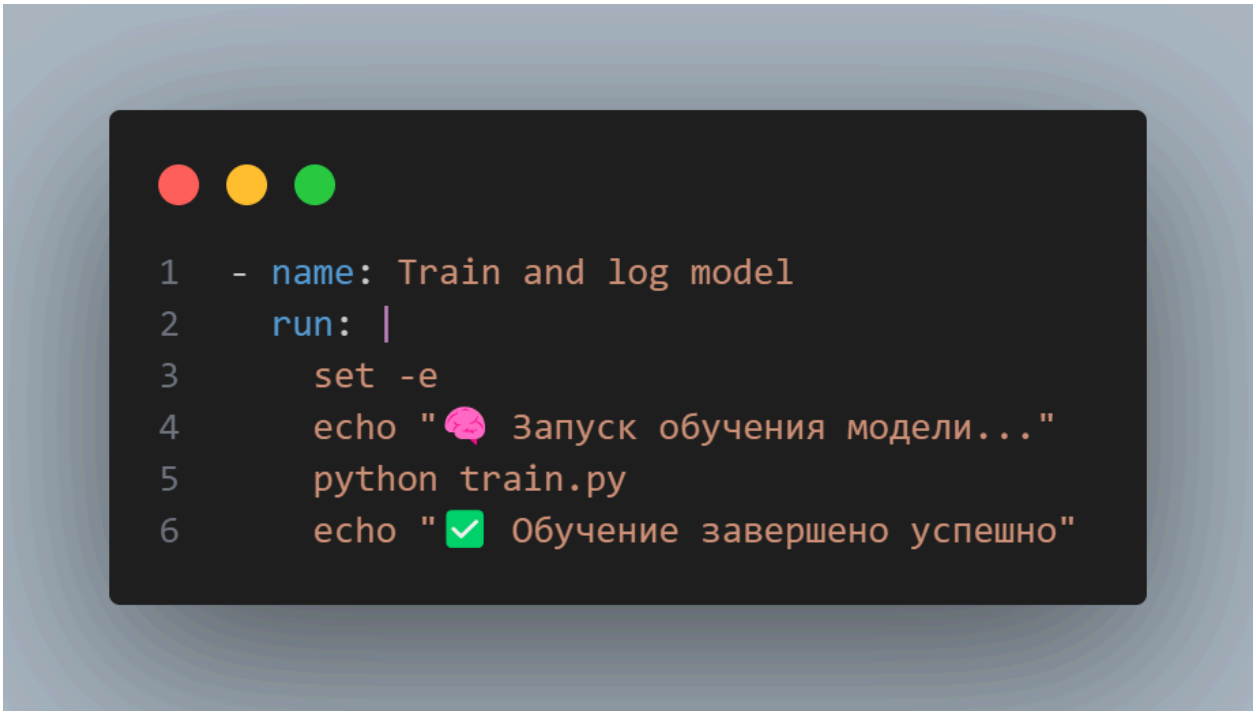
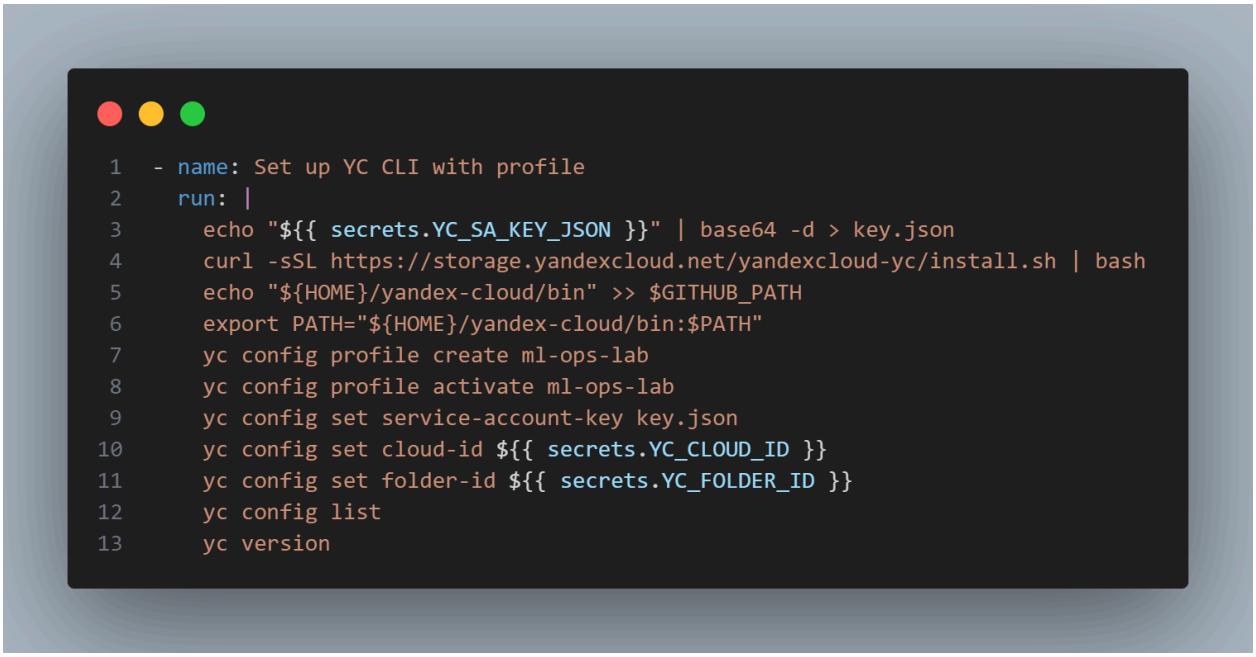


Рисунок 18 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml) Запуск обучения и логирование в MLflow

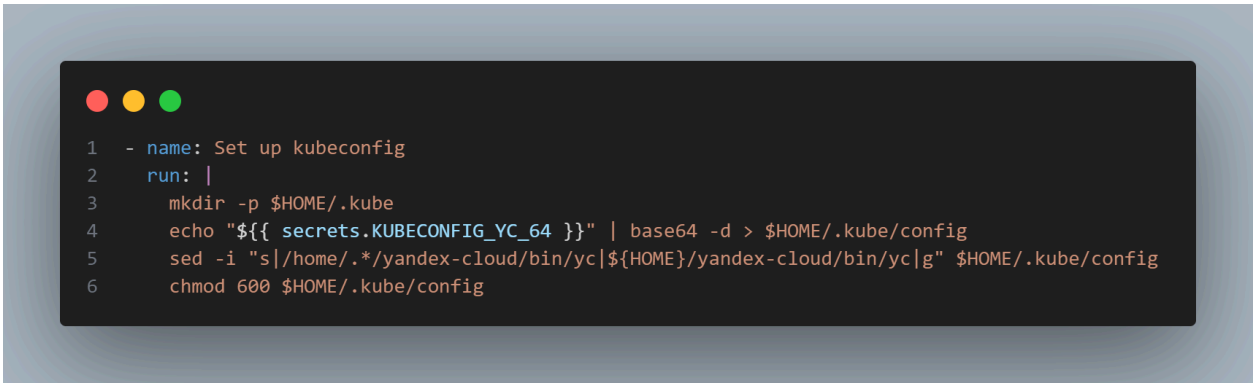
Рисунок 18 демонстрирует запуск train.py, в котором происходит обучение модели и логирование метрик, параметров и артефактов в MLflow.



```yaml
- name: Set up YC CLI with profile
  run: |
    echo "${{ secrets.YC_SA_KEY_JSON }}" | base64 -d > key.json
    curl -sSL https://storage.yandexcloud.net/yandexcloud-yc/install.sh | bash
    echo "${HOME}/yandex-cloud/bin" >> $GITHUB_PATH
    export PATH="${HOME}/yandex-cloud/bin:$PATH"
    yc config profile create ml-ops-lab
    yc config profile activate ml-ops-lab
    yc config set service-account-key key.json
    yc config set cloud-id ${{ secrets.YC_CLOUD_ID }}
    yc config set folder-id ${{ secrets.YC_FOLDER_ID }}
    yc config list
    yc version
```

Рисунок 19 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml) Настройка CLI Yandex Cloud (YC)

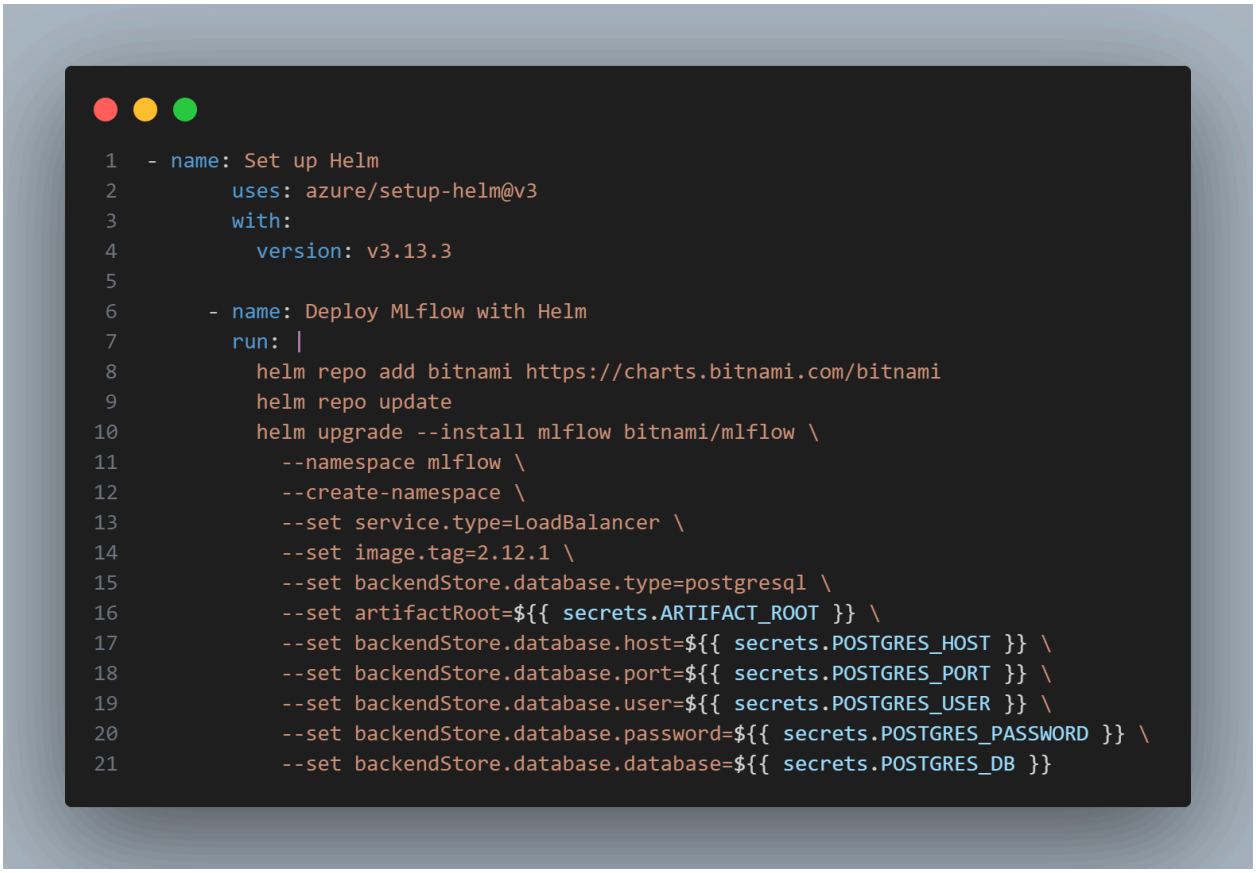
На рисунке 19 представлена установка и настройка CLI Yandex Cloud с использованием сервисного аккаунта и окружения. Это позволяет взаимодействовать с Kubernetes и другими ресурсами.



```yaml
- name: Set up kubeconfig
  run: |
    mkdir -p $HOME/.kube
    echo "${{ secrets.KUBECONFIG_YC_64 }}" | base64 -d > $HOME/.kube/config
    sed -i "s|/home/.*/yandex-cloud/bin/yc|${HOME}/yandex-cloud/bin/yc|g" $HOME/.kube/config
    chmod 600 $HOME/.kube/config
```

Рисунок 20 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml) Настройка kubeconfig

Рисунок 20 демонстрирует создание файла конфигурации для подключения к кластеру Kubernetes в Yandex Cloud.



```yaml
- name: Set up Helm
      uses: azure/setup-helm@v3
      with:
        version: v3.13.3

    - name: Deploy MLflow with Helm
      run: |
        helm repo add bitnami https://charts.bitnami.com/bitnami
        helm repo update
        helm upgrade --install mlflow bitnami/mlflow \
          --namespace mlflow \
          --create-namespace \
          --set service.type=LoadBalancer \
          --set image.tag=2.12.1 \
          --set backendStore.database.type=postgresql \
          --set artifactRoot=${{ secrets.ARTIFACT_ROOT }} \
          --set backendStore.database.host=${{ secrets.POSTGRES_HOST }} \
          --set backendStore.database.port=${{ secrets.POSTGRES_PORT }} \
          --set backendStore.database.user=${{ secrets.POSTGRES_USER }} \
          --set backendStore.database.password=${{ secrets.POSTGRES_PASSWORD }} \
          --set backendStore.database.database=${{ secrets.POSTGRES_DB }}
```

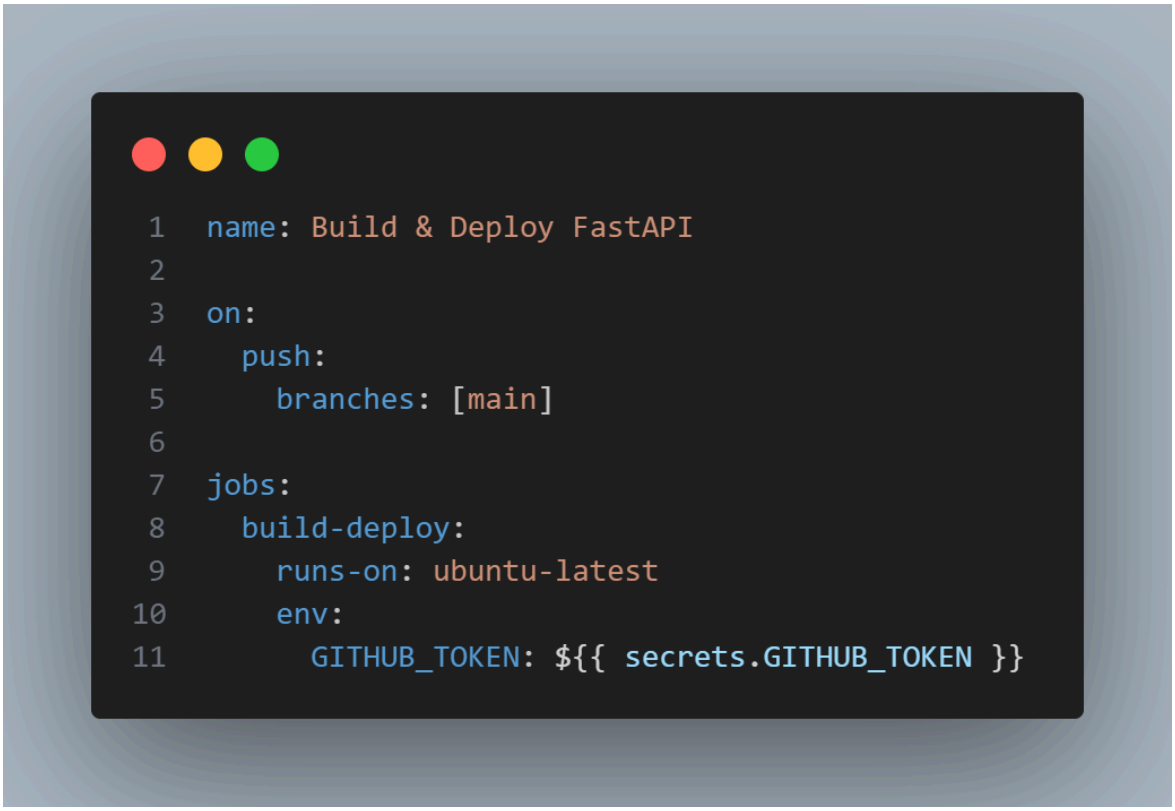
Рисунок 21 – CI/CD-процесс обучения и деплоя модели (.github/workflows/train-and-deploy.yaml). Деплой MLflow через Helm

Рисунок 21 показывает как проходит установка и обновление MLflow в Kubernetes через Helm-чарт Bitnami. Используются параметры из GitHub Secrets для подключения к PostgreSQL и S3.

Сборка и выкладка FastAPI-приложения в Kubernetes

Данный GitHub Actions workflow обеспечивает полный цикл CI/CD для FastAPI-сервиса: сборка, публикация и автоматический деплой в Kubernetes.

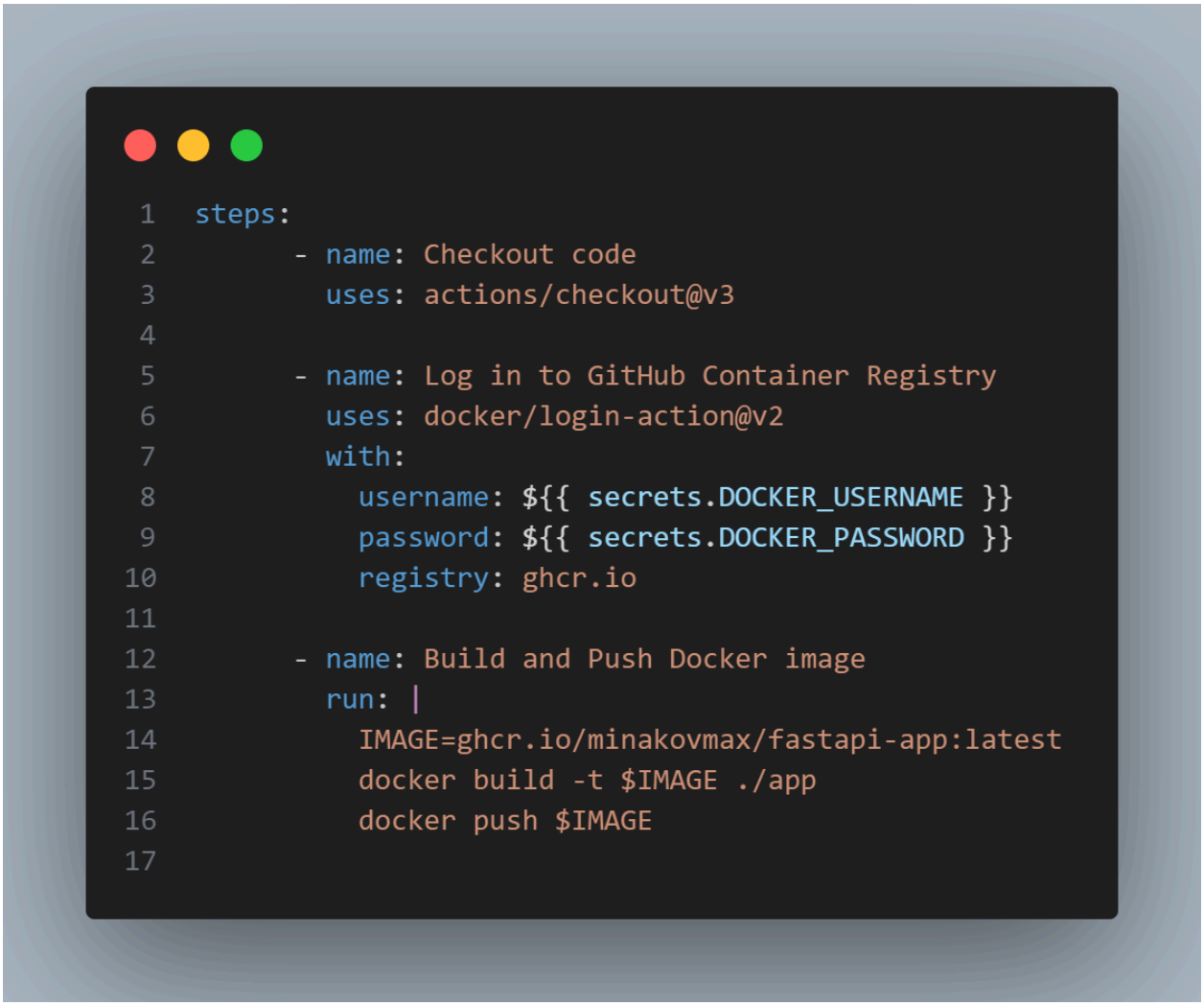
Такой подход обеспечивает минимальные задержки между изменением кода и его выкладкой в production-кластер, а также гарантирует согласованность между версиями Docker-образа и Helm-конфигурацией.



```yaml
name: Build & Deploy FastAPI

on:
  push:
    branches: [main]

jobs:
  build-deploy:
    runs-on: ubuntu-latest
    env:
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Рисунок 22 – Сборка и выкладка FastAPI-приложения в Kubernetes (.github/workflows/ci-cd.yaml) Название, триггер и переменные окружения

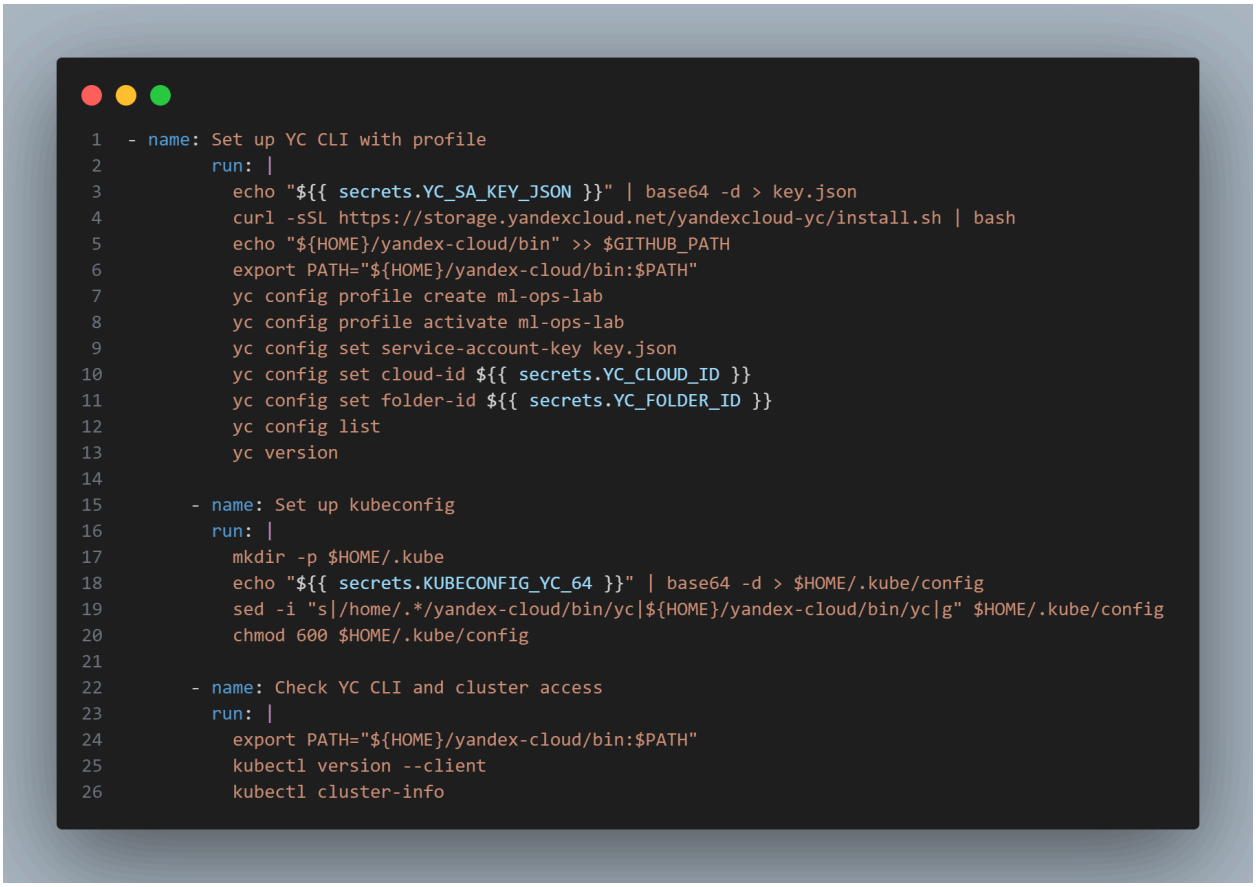
Рисунок 22 демонстрирует пайплайн, который автоматически запускается при каждом пуше в ветку main. Он выполняется в окружении ubuntu-latest и использует секреты для аутентификации.



```yaml
steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
          registry: ghcr.io

      - name: Build and Push Docker image
        run: |
          IMAGE=ghcr.io/minakovmax/fastapi-app:latest
          docker build -t $IMAGE ./app
          docker push $IMAGE

```

Рисунок 23 – Сборка и выкладка FastAPI-приложения в Kubernetes (.github/workflows/ci-cd.yaml) Сборка и публикация Docker-образа в GitHub Container Registry (GHCR)

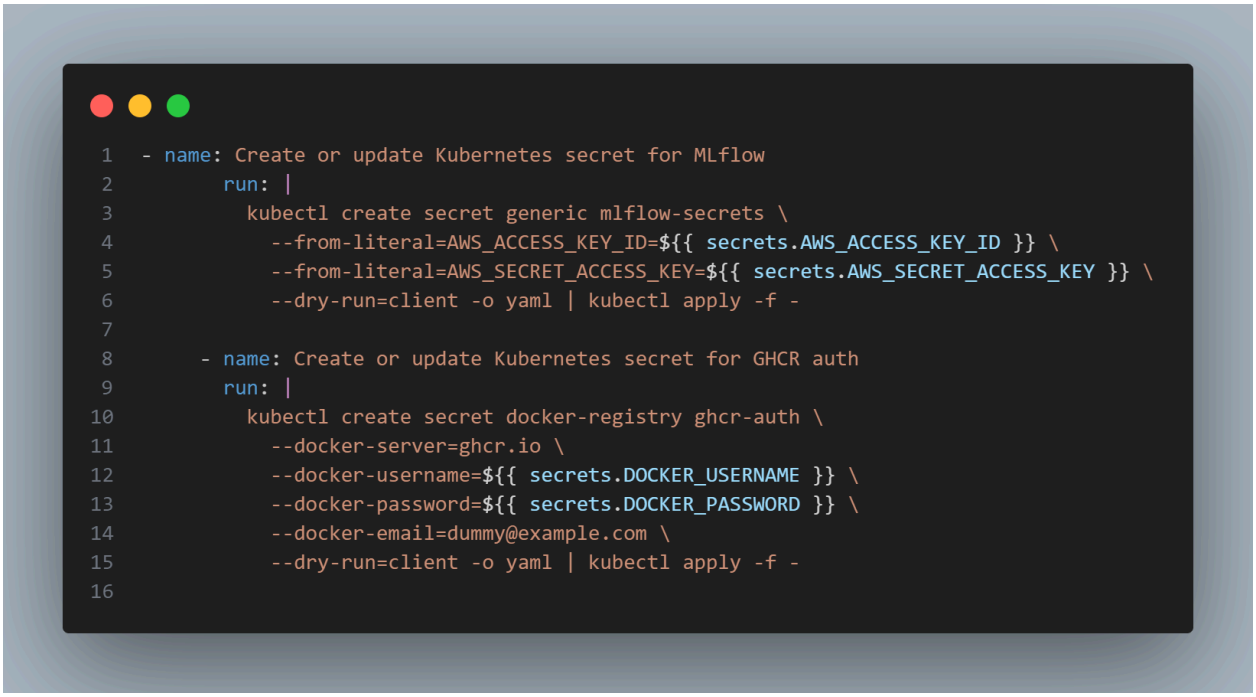
Рисунок 23 показывает, что сначала репозиторий клонируется, затем происходит сборка Docker-образа из директории ./app и публикация в GHCR под тегом latest.



```yaml
- name: Set up YC CLI with profile
        run: |
          echo "${{ secrets.YC_SA_KEY_JSON }}" | base64 -d > key.json
          curl -sSL https://storage.yandexcloud.net/yandexcloud-yc/install.sh | bash
          echo "${HOME}/yandex-cloud/bin" >> $GITHUB_PATH
          export PATH="${HOME}/yandex-cloud/bin:$PATH"
          yc config profile create ml-ops-lab
          yc config profile activate ml-ops-lab
          yc config set service-account-key key.json
          yc config set cloud-id ${{ secrets.YC_CLOUD_ID }}
          yc config set folder-id ${{ secrets.YC_FOLDER_ID }}
          yc config list
          yc version

      - name: Set up kubeconfig
        run: |
          mkdir -p $HOME/.kube
          echo "${{ secrets.KUBECONFIG_YC_64 }}" | base64 -d > $HOME/.kube/config
          sed -i "s|/home/.*/yandex-cloud/bin/yc|${HOME}/yandex-cloud/bin/yc|g" $HOME/.kube/config
          chmod 600 $HOME/.kube/config

      - name: Check YC CLI and cluster access
        run: |
          export PATH="${HOME}/yandex-cloud/bin:$PATH"
          kubectl version --client
          kubectl cluster-info
```

Рисунок 24 – Сборка и выкладка FastAPI-приложения в Kubernetes (.github/workflows/ci-cd.yaml) Настройка CLI и доступа к кластеру Yandex Cloud

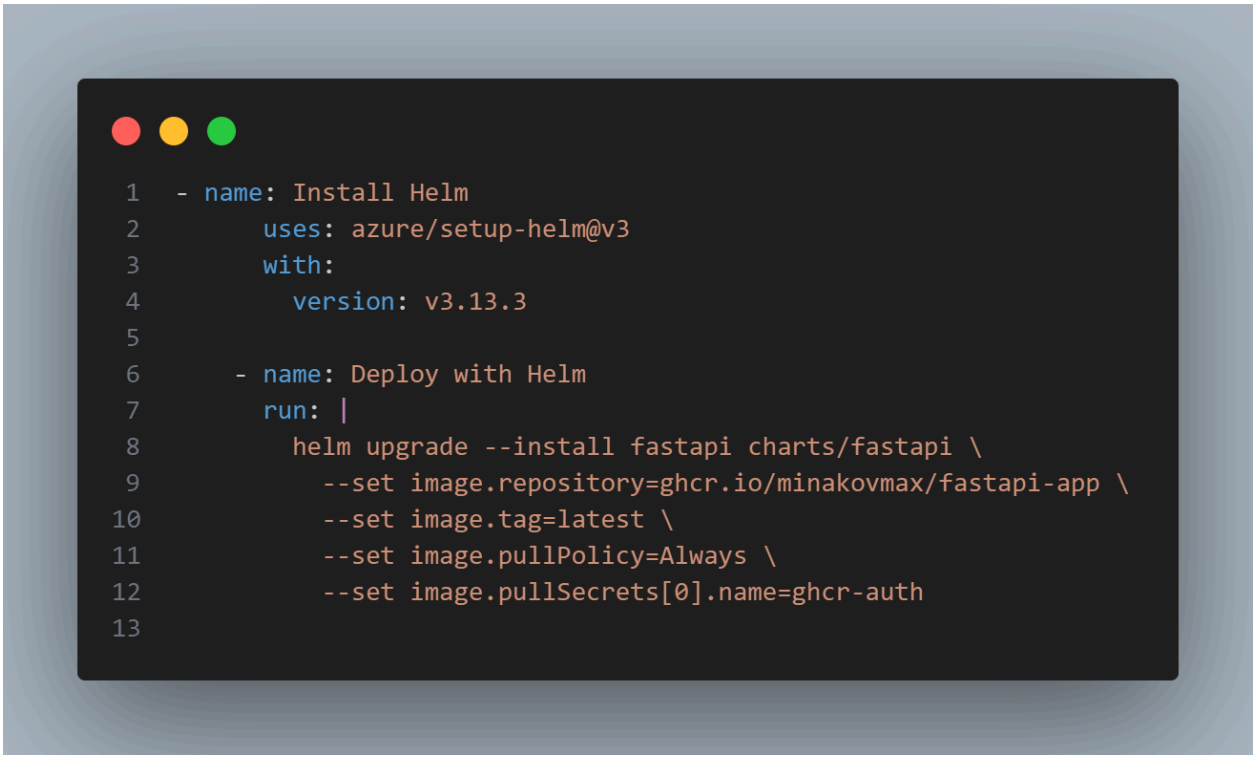
На рисунке 24 представлен процесс на котором устанавливаются утилиты yc, kubectl, настраивается профиль, а затем выполняется проверка доступа к Kubernetes-кластеру через kubeconfig.



```yaml
- name: Create or update Kubernetes secret for MLflow
      run: |
        kubectl create secret generic mlflow-secrets \
          --from-literal=AWS_ACCESS_KEY_ID=${{ secrets.AWS_ACCESS_KEY_ID }} \
          --from-literal=AWS_SECRET_ACCESS_KEY=${{ secrets.AWS_SECRET_ACCESS_KEY }} \
          --dry-run=client -o yaml | kubectl apply -f -

    - name: Create or update Kubernetes secret for GHCR auth
      run: |
        kubectl create secret docker-registry ghcr-auth \
          --docker-server=ghcr.io \
          --docker-username=${{ secrets.DOCKER_USERNAME }} \
          --docker-password=${{ secrets.DOCKER_PASSWORD }} \
          --docker-email=dummy@example.com \
          --dry-run=client -o yaml | kubectl apply -f -

```

Рисунок 25 – Сборка и выкладка FastAPI-приложения в Kubernetes (.github/workflows/ci-cd.yaml) Создание Kubernetes-секретов для MLflow и GHCR

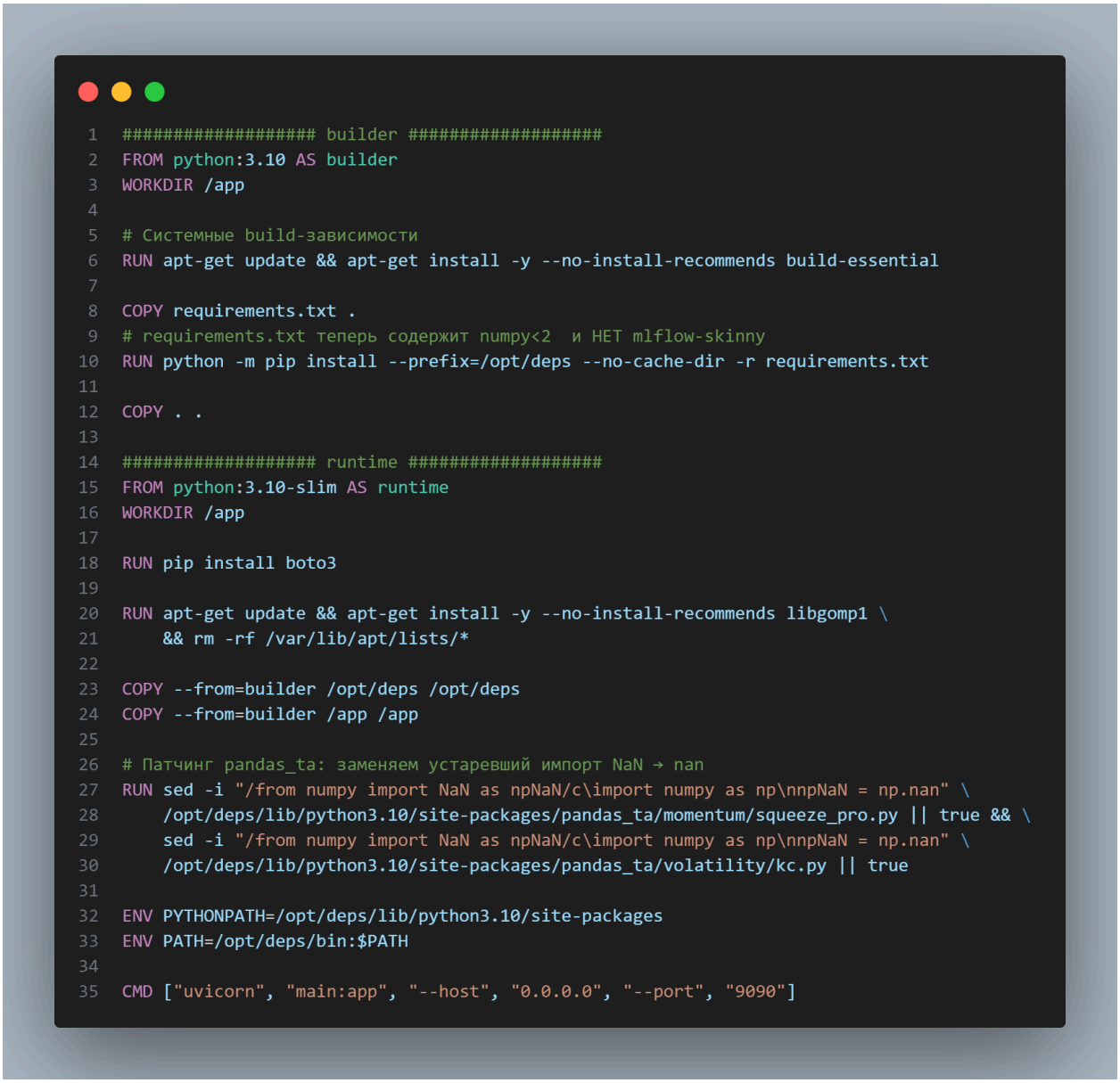
На рисунке 25 создаются или обновляются Kubernetes-секреты, используемые Helm-чартом FastAPI: один для доступа к S3 (MLflow), другой – к GHCR для загрузки Docker-образа.



```yaml
- name: Install Helm
      uses: azure/setup-helm@v3
      with:
        version: v3.13.3

    - name: Deploy with Helm
      run: |
        helm upgrade --install fastapi charts/fastapi \
          --set image.repository=ghcr.io/minakovmax/fastapi-app \
          --set image.tag=latest \
          --set image.pullPolicy=Always \
          --set image.pullSecrets[0].name=ghcr-auth

```

Рисунок 26 – Сборка и выкладка FastAPI-приложения в Kubernetes (.github/workflows/ci-cd.yaml) Установка Helm и выкладка FastAPI

Рисунок 26 демонстрирует завершающий шаг – развертывание FastAPI-приложения в Kubernetes с использованием Helm. Все параметры (образ, тег, pull-секреты) передаются через --set.



```
################### builder ###################
FROM python:3.10 AS builder
WORKDIR /app

# Системные build-зависимости
RUN apt-get update && apt-get install -y --no-install-recommends build-essential

COPY requirements.txt .
# requirements.txt теперь содержит numpy<2  и НЕТ mlflow-skinny
RUN python -m pip install --prefix=/opt/deps --no-cache-dir -r requirements.txt

COPY . .

################### runtime ###################
FROM python:3.10-slim AS runtime
WORKDIR /app

RUN pip install boto3

RUN apt-get update && apt-get install -y --no-install-recommends libgomp1 \
    && rm -rf /var/lib/apt/lists/*

COPY --from=builder /opt/deps /opt/deps
COPY --from=builder /app /app

# Патчинг pandas_ta: заменяем устаревший импорт NaN → nan
RUN sed -i "/from numpy import NaN as npNaN/c\import numpy as np\nnpNaN = np.nan" \
    /opt/deps/lib/python3.10/site-packages/pandas_ta/momentum/squeeze_pro.py || true && \
    sed -i "/from numpy import NaN as npNaN/c\import numpy as np\nnpNaN = np.nan" \
    /opt/deps/lib/python3.10/site-packages/pandas_ta/volatility/kc.py || true

ENV PYTHONPATH=/opt/deps/lib/python3.10/site-packages
ENV PATH=/opt/deps/bin:$PATH

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "9090"]
```

Рисунок 27 – Dockerfile FastAPI-приложения с двухфазной сборкой (app/Dockerfile)

Рисунок 27 демонстрирует Dockerfile, который реализует двухфазную сборку (multi-stage build), что позволяет уменьшить размер итогового образа и разделить стадии установки зависимостей и исполнения.

На первом этапе (builder): устанавливаются системные пакеты (build-essential), зависимости из requirements.txt устанавливаются в отдельный путь /opt/deps с помощью флага --prefix, копируется исходный код.

На втором этапе (runtime): используется облегчённый базовый образ python:3.10-slim, устанавливаются минимальные runtime-зависимости (libgomp1, boto3), копируются зависимости и код из builder-слоя.

Отдельным шагом выполняется патчинг библиотеки pandas_ta, заменяя устаревший импорт NaN на np.nan, чтобы избежать ошибок во время выполнения.

Переменная PYTHONPATH переопределяется так, чтобы Python использовал путь с установленными зависимостями.

В качестве команды запуска используется uvicorn, запускающий FastAPI-приложение на порту 9090.

Обучение модели для инфиренса train.py

Скрипт train.py реализует полный процесс построения и логирования прогностической модели для краткосрочного движения цены акций ПАО «Сбербанк» на основе часовых исторических данных, полученных с Московской биржи (MOEX). В рамках этого скрипта выполняются следующие этапы:

- загрузка данных с MOEX по инструменту SBER с использованием REST API и агрегацией по часовым свечам;

- построение технических индикаторов (EMA, ADX, DI+, DI−, RSI, ATR) с помощью библиотеки pandas_ta;

- формирование признаков, отражающих текущие рыночные сигналы, включая бинарные флаги и комбинации индикаторов (например, «сильный восходящий тренд»);

- разметка целевой переменной на три класса (рост, падение, нейтральное поведение) на основе будущей доходности и последующее преобразование под задачу многоклассовой классификации;

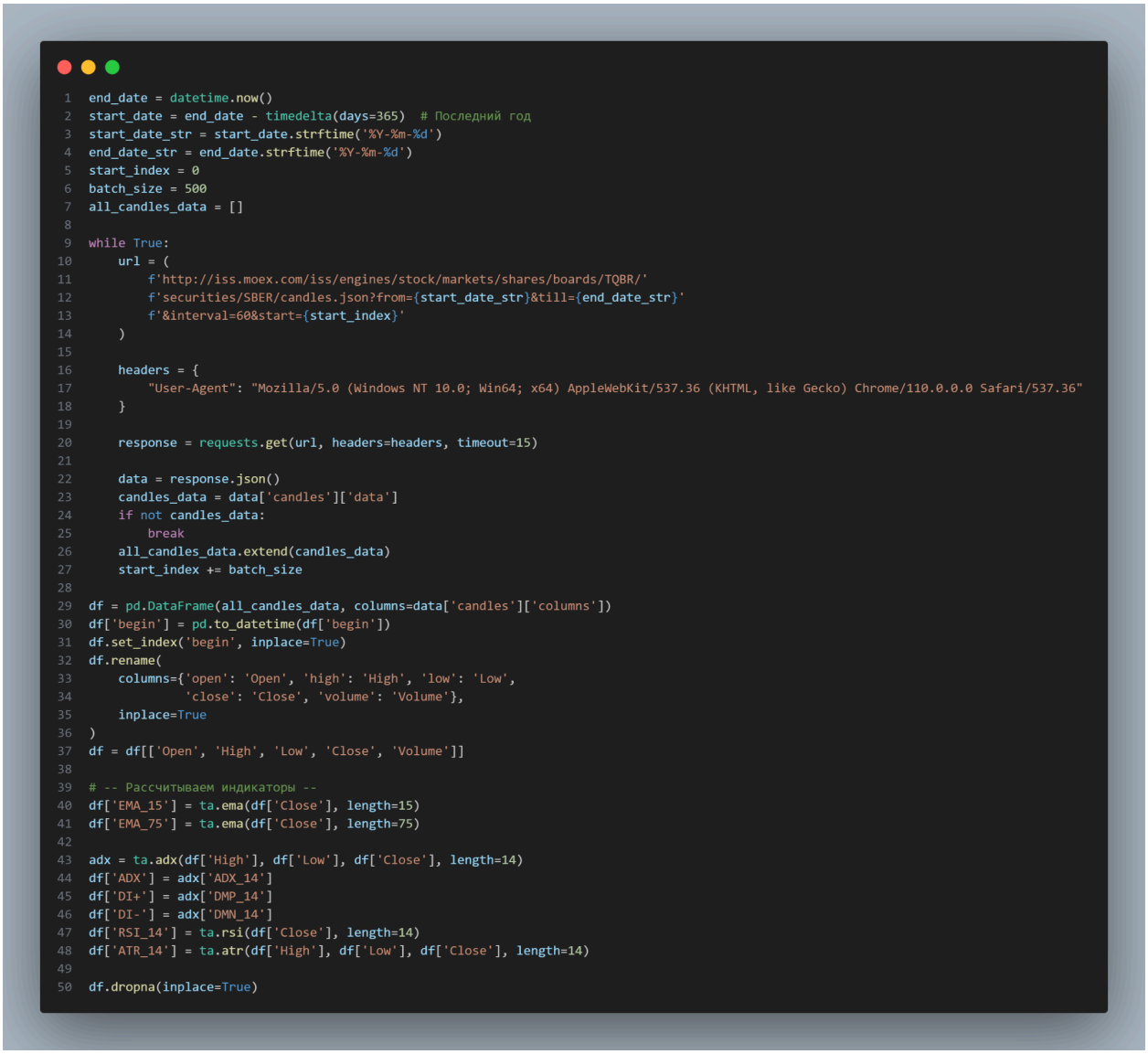
- обучение трёх моделей: XGBoost, LightGBM, CatBoost на последних 5000 наблюдениях с использованием единых признаков;

- логирование каждой модели в MLflow: параметры, метрики (accuracy, f1, precision, recall), модельный артефакт, input example и signature;

- формирование ансамбля голосованием по трём моделям с логированием финальных метрик ансамбля и сохранением предсказаний в CSV;

- поддержка переобучения в CI/CD за счёт интеграции со скриптами GitHub Actions и деплоя моделей в Kubernetes через MLflow Registry.

Таким образом, train.py представляет собой завершённый экспериментальный пайплайн, адаптированный под MLOps-инфраструктуру, и служит основой как для автоматического обучения, так и для мониторинга качества модели с течением времени.



```python
end_date = datetime.now()
start_date = end_date - timedelta(days=365)  # Последний год
start_date_str = start_date.strftime('%Y-%m-%d')
end_date_str = end_date.strftime('%Y-%m-%d')
start_index = 0
batch_size = 500
all_candles_data = []

while True:
    url = (
        f'http://iss.moex.com/iss/engines/stock/markets/shares/boards/TQBR/'
        f'securities/SBER/candles.json?from={start_date_str}&till={end_date_str}'
        f'&interval=60&start={start_index}'
    )

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
    }

    response = requests.get(url, headers=headers, timeout=15)

    data = response.json()
    candles_data = data['candles']['data']
    if not candles_data:
        break
    all_candles_data.extend(candles_data)
    start_index += batch_size

df = pd.DataFrame(all_candles_data, columns=data['candles']['columns'])
df['begin'] = pd.to_datetime(df['begin'])
df.set_index('begin', inplace=True)
df.rename(
    columns={'open': 'Open', 'high': 'High', 'low': 'Low',
             'close': 'Close', 'volume': 'Volume'},
    inplace=True
)
df = df[['Open', 'High', 'Low', 'Close', 'Volume']]

# -- Рассчитываем индикаторы --
df['EMA_15'] = ta.ema(df['Close'], length=15)
df['EMA_75'] = ta.ema(df['Close'], length=75)

adx = ta.adx(df['High'], df['Low'], df['Close'], length=14)
df['ADX'] = adx['ADX_14']
df['DI+'] = adx['DMP_14']
df['DI-'] = adx['DMN_14']
df['RSI_14'] = ta.rsi(df['Close'], length=14)
df['ATR_14'] = ta.atr(df['High'], df['Low'], df['Close'], length=14)

df.dropna(inplace=True)
```

Рисунок 28 – Загрузка и подготовка данных с MOEX (train.py)

На рисунке 28 в данном фрагменте осуществляется поэтапная загрузка исторических данных с Московской биржи по инструменту SBER.

Затем рассчитываются технические индикаторы: скользящие средние (EMA), индексы направления тренда (ADX, DI+, DI−), индикатор относительной силы (RSI) и средний истинный диапазон (ATR).

Итоговый DataFrame используется в последующих этапах как основа для формирования признаков и целевых переменных.

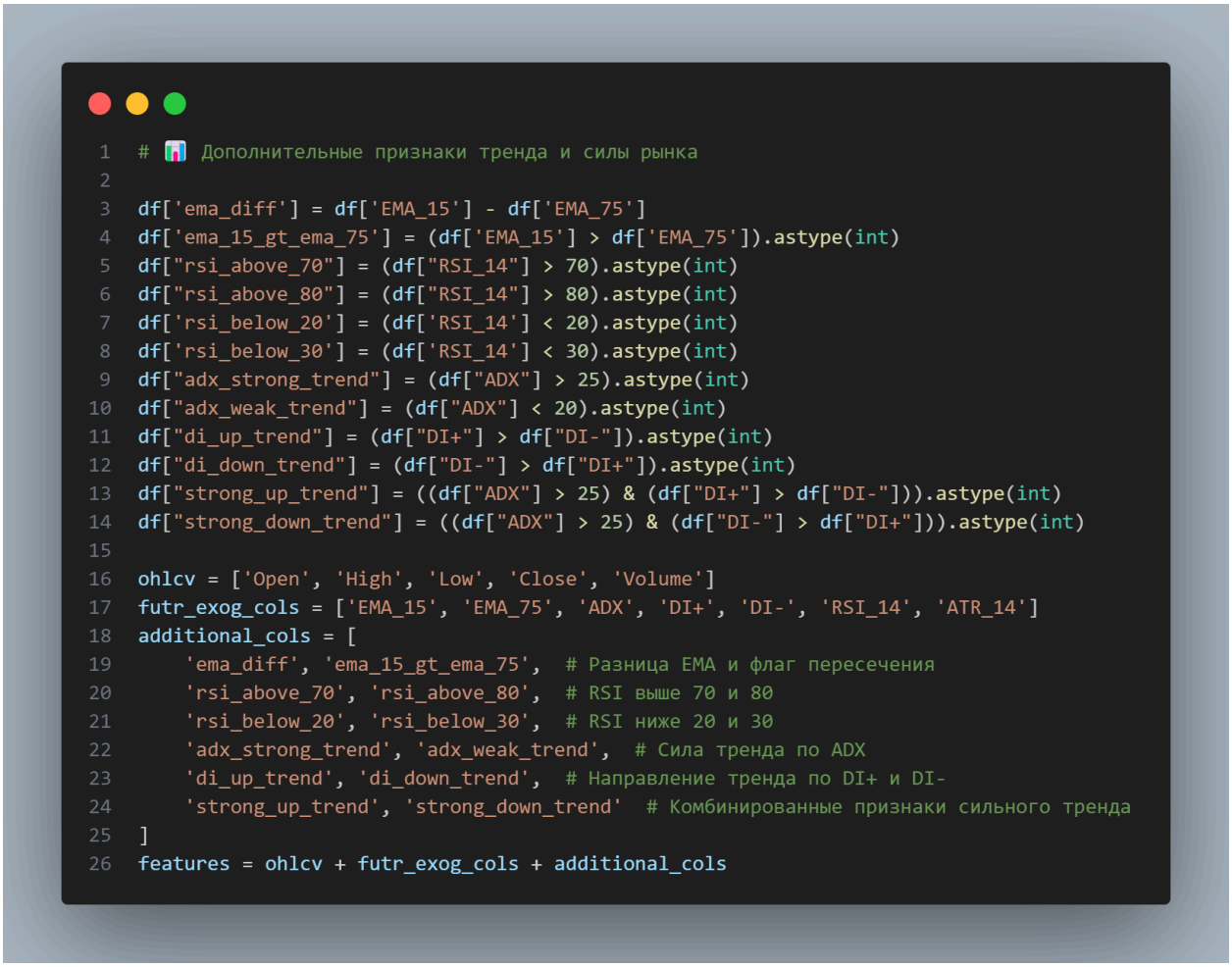
```python
# ========================
# 1. Загрузка и подготовка
# ========================
df = load_and_prepare_data()

# Сортируем по времени (индексу)
df.sort_index(inplace=True)

# Рассчитываем future_return на 10 часов вперёд
df['future_return'] = df['Close'].shift(-14) / df['Close'] - 1
df.dropna(subset=['future_return'], inplace=True)

# Формируем таргет (-1, 0, 1)
df['target'] = 0
df.loc[df['future_return'] >  0.01, 'target'] = 1
df.loc[df['future_return'] < -0.01, 'target'] = -1
df['target'] = df['target'].astype(int)

# Т.к. XGBoost/CatBoost/LGB ждут классы 0..2:
# -1 → 0,  0 → 1,  1 → 2
mapping_dict = {-1: 0, 0: 1, 1: 2}
df['target_mapped'] = df['target'].map(mapping_dict)
```

Рисунок 29 – Формирование целевой переменной и подготовка к обучению

На рисунке 29 представлена целевая переменная (target), которая формируется на основе будущей доходности: если доходность через 14 периодов превышает +1%, это считается ростом (1), если ниже −1% – падением (-1), иначе – нейтральное поведение (0).

Далее значения приводятся к диапазону 0, 1, 2, поскольку большинство классификаторов (CatBoost, LightGBM, XGBoost) ожидают целевую переменную именно в таком формате.

Такое представление задачи позволяет решать её как многоклассовую классификацию, где каждый класс – это отдельный сценарий поведения цены.



```python
# 📊 Дополнительные признаки тренда и силы рынка

df['ema_diff'] = df['EMA_15'] - df['EMA_75']
df['ema_15_gt_ema_75'] = (df['EMA_15'] > df['EMA_75']).astype(int)
df["rsi_above_70"] = (df["RSI_14"] > 70).astype(int)
df["rsi_above_80"] = (df["RSI_14"] > 80).astype(int)
df['rsi_below_20'] = (df['RSI_14'] < 20).astype(int)
df['rsi_below_30'] = (df['RSI_14'] < 30).astype(int)
df["adx_strong_trend"] = (df["ADX"] > 25).astype(int)
df["adx_weak_trend"] = (df["ADX"] < 20).astype(int)
df["di_up_trend"] = (df["DI+"] > df["DI-"]).astype(int)
df["di_down_trend"] = (df["DI-"] > df["DI+"]).astype(int)
df["strong_up_trend"] = ((df["ADX"] > 25) & (df["DI+"] > df["DI-"])).astype(int)
df["strong_down_trend"] = ((df["ADX"] > 25) & (df["DI-"] > df["DI+"])).astype(int)

ohlcv = ['Open', 'High', 'Low', 'Close', 'Volume']
futr_exog_cols = ['EMA_15', 'EMA_75', 'ADX', 'DI+', 'DI-', 'RSI_14', 'ATR_14']
additional_cols = [
    'ema_diff', 'ema_15_gt_ema_75',  # Разница EMA и флаг пересечения
    'rsi_above_70', 'rsi_above_80',  # RSI выше 70 и 80
    'rsi_below_20', 'rsi_below_30',  # RSI ниже 20 и 30
    'adx_strong_trend', 'adx_weak_trend',  # Сила тренда по ADX
    'di_up_trend', 'di_down_trend',  # Направление тренда по DI+ и DI-
    'strong_up_trend', 'strong_down_trend'  # Комбинированные признаки сильного тренда
]
features = ohlcv + futr_exog_cols + additional_cols
```

Рисунок 30 – Формирование технических признаков и выбор финального набора фичей

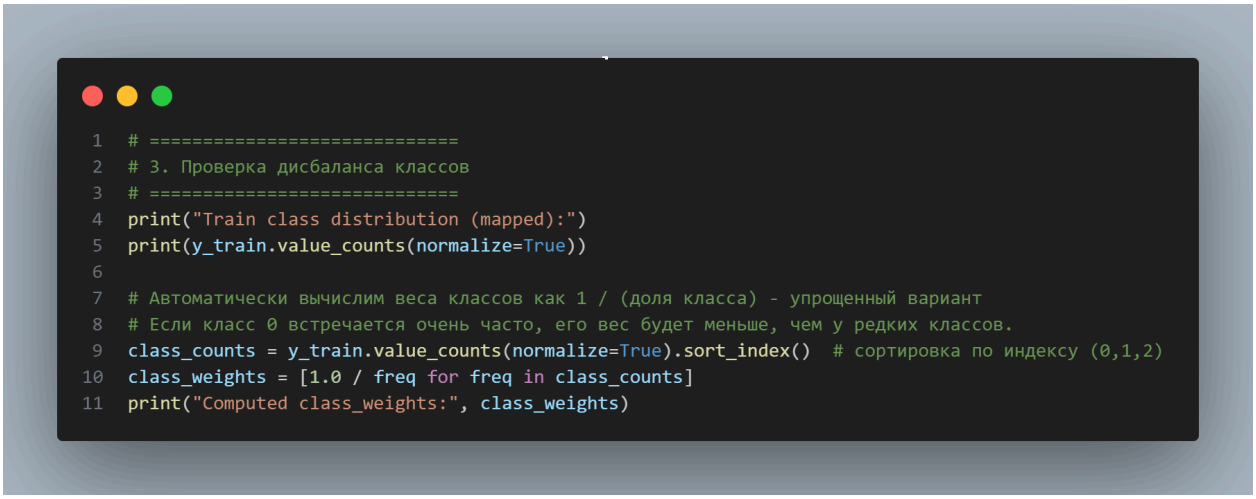
Рисунок 30 демонстрирует фрагмент скрипта, на котором производится расчёт расширенного набора технических признаков, отражающих как текущее поведение цены, так и состояние индикаторов тренда.

Формируются бинарные признаки (например, rsi_above_70, adx_strong_trend), а также комбинированные сигналы (strong_up_trend),

которые учитывают как силу тренда по ADX, так и его направление по DI+ и DI−.

В конце формируется финальный список признаков features, который объединяет сырые ценовые характеристики (Open, High, Low, Close, Volume), технические индикаторы и дополнительные derived-фичи.

Эти признаки впоследствии используются для обучения моделей XGBoost, LightGBM и CatBoost.



```
# ============================
# 3. Проверка дисбаланса классов
# ============================
print("Train class distribution (mapped):")
print(y_train.value_counts(normalize=True))

# Автоматически вычислим веса классов как 1 / (доля класса) - упрощенный вариант
# Если класс 0 встречается очень часто, его вес будет меньше, чем у редких классов.
class_counts = y_train.value_counts(normalize=True).sort_index()  # сортировка по индексу (0,1,2)
class_weights = [1.0 / freq for freq in class_counts]
print("Computed class_weights:", class_weights)
```

Рисунок 31 – Проверка дисбаланса классов и расчет весов

На рисунке 31 показано, что для обеспечения устойчивости обучения модели к дисбалансу классов в обучающей выборке рассчитываются веса классов, обратные их доле в выборке ($1 / p_i$).

Это необходимо, поскольку один из классов может доминировать (например, нейтральное поведение – класс 1), что может привести к смещению модели.

Такой подход позволяет учесть редкие, но значимые события (например, падения или скачки цен) при обучении моделей XGBoost, LightGBM и CatBoost, поддерживающих параметр class_weights.

```python
# === Настройка MLflow ===
mlflow.set_tracking_uri("http://158.160.134.123:8080")
mlflow.set_experiment("ensemble_latest_5000")

# Вычисление будущей доходности
df["future_return"] = df["Close"].shift(-14) / df["Close"] - 1
df.dropna(subset=["future_return"], inplace=True)

# Классификация: {-1, 0, 1}
df["target"] = 0
df.loc[df["future_return"] > 0.01, "target"] = 1
df.loc[df["future_return"] < -0.01, "target"] = -1
df["target_mapped"] = df["target"].map({-1: 0, 0: 1, 1: 2})

# Последние 5000 наблюдений
df = df.tail(5000).copy()
X = df.drop(columns=["target", "target_mapped", "future_return"])
y = df["target_mapped"]

# === Модели ===
models = {
    "XGB": XGBClassifier(objective="multi:softmax", max_depth=3,n_estimators=408,
        subsample=0.754, colsample_bytree=0.602, num_class=3, eval_metric="mlogloss",
        random_state=42, use_label_encoder=False,
    ),
    "LGBM": LGBMClassifier(objective="multiclass", max_depth=3, n_estimators=928,
        subsample=0.724, colsample_bytree=0.658, num_class=3, random_state=42,
    ),
    "CatBoost": CatBoostClassifier(loss_function="MultiClass", random_seed=42,
        iterations=1000, learning_rate=0.05, early_stopping_rounds=100,
        use_best_model=True, verbose=0,),
}

selected_models = list(models.keys())
```

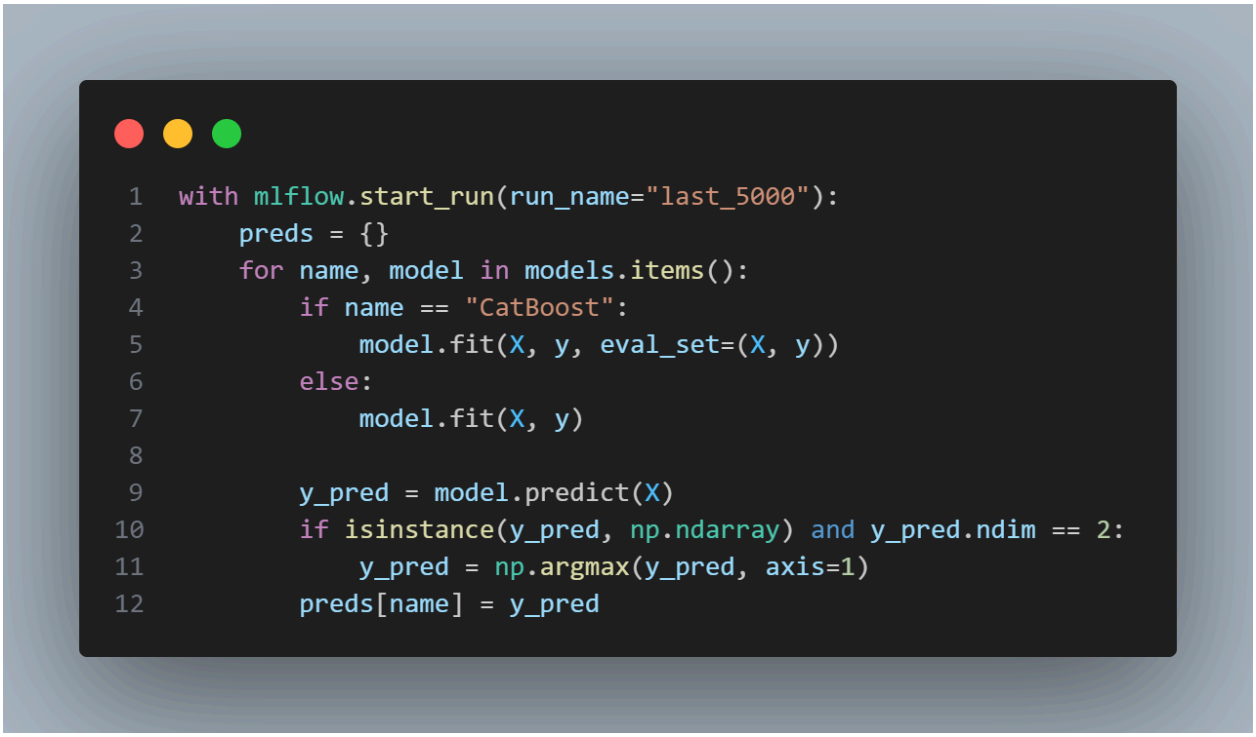
Рисунок 32 – Настройка MLflow, подготовка выборки и инициализация моделей

На рисунке 32 в фрагменте происходит инициализация отслеживания экспериментов через MLflow, а также выделение последних 5000 наблюдений для обучения.

Скрипт переопределяет целевую переменную на основе будущей доходности (future_return) и классифицирует её на три класса: рост, падение и нейтральное движение.

Затем создаётся словарь с тремя моделями: XGBoost, LightGBM и CatBoost, каждая из которых конфигурируется с гиперпараметрами, полученными на этапе подбора.

Эти модели впоследствии обучаются и логируются в MLflow как независимые вложенные эксперименты.



```
with mlflow.start_run(run_name="last_5000"):
    preds = {}
    for name, model in models.items():
        if name == "CatBoost":
            model.fit(X, y, eval_set=(X, y))
        else:
            model.fit(X, y)

        y_pred = model.predict(X)
        if isinstance(y_pred, np.ndarray) and y_pred.ndim == 2:
            y_pred = np.argmax(y_pred, axis=1)
        preds[name] = y_pred
```

Рисунок 33 – Обучение моделей и сбор предсказаний

На рисунке 33 осуществляется запуск основного MLflow-эксперимента, в рамках которого инициализируется цикл по всем моделям.

Каждая модель обучается на полном датасете X, y.

Для CatBoostClassifier дополнительно указывается eval_set, чтобы задействовать встроенные механизмы ранней остановки.

Полученные предсказания сохраняются в словарь preds для последующего использования в ансамбле.

Также выполняется обработка случая, если модель возвращает one-hot-матрицу (например, predict_proba у некоторых библиотек).

```python
# === Логирование модели ===
with mlflow.start_run(run_name=f"{name}_model", nested=True):
    mlflow.log_param("model_type", name)
    mlflow.log_params(model.get_params())
    mlflow.log_metric("accuracy", accuracy_score(y, y_pred))
    mlflow.log_metric("f1_macro", f1_score(y, y_pred, average="macro"))
    mlflow.log_metric("precision_macro", precision_score(y, y_pred, average="macro"))
    mlflow.log_metric("recall_macro", recall_score(y, y_pred, average="macro"))

    # Логирование модели
    if name == "XGB": mlflow.xgboost.log_model(model, artifact_path="model", registered_model_name="XGBoostModel")
    elif name == "LGBM": mlflow.lightgbm.log_model(model, artifact_path="model", registered_model_name="LightGBMModel")
    elif name == "CatBoost": mlflow.catboost.log_model(model, artifact_path="model", registered_model_name="CatBoostModel")
```

Рисунок 34 – Логирование каждой модели в MLflow

После завершения обучения каждой модели запускается вложенный MLflow-подэксперимент (nested=True). В него логируются параметры модели (get_params()), а также ключевые метрики качества классификации: точность (accuracy), средневзвешенное значение F1 (f1_macro), макро усредненная точность и полнота, что показано на рисунке 34. Кроме того, модель сериализуется и сохраняется в MLflow с указанием имени (registered_model_name) для каждой библиотеки: XGBoostModel, LightGBMModel, CatBoostModel.

```python
    final_preds = []
    for i in range(len(X)):
        votes = [int(preds[name][i]) for name in selected_models]
        vote_count = Counter(votes).most_common()
        if vote_count[0][1] > 1:
            final_preds.append(vote_count[0][0])
        else:
            final_preds.append(np.random.choice([vote_count[0][0], vote_count[1][0]]))

    acc = accuracy_score(y, final_preds)
    f1 = f1_score(y, final_preds, average="macro")
    prec = precision_score(y, final_preds, average="macro")
    rec = recall_score(y, final_preds, average="macro")
    correct_frac = (np.sum((np.array(final_preds) == 0) & (y.values == 0)) + np.sum((np.array(final_preds) == 2) & (y.values == 2))) / len(y)

    mlflow.log_metric("ensemble_accuracy", acc)
    mlflow.log_metric("ensemble_f1_macro", f1)
    mlflow.log_metric("ensemble_precision_macro", prec)
    mlflow.log_metric("ensemble_recall_macro", rec)
    mlflow.log_metric("ensemble_correct_frac_0_2", correct_frac)

    df_preds = pd.DataFrame({"index": X.index, "true": y.values,"ensemble_pred": final_preds})
    pred_path = "ensemble_last_5000_predictions.csv"
    df_preds.to_csv(pred_path, index=False)
    mlflow.log_artifact(pred_path, artifact_path="ensemble_preds")

mlflow.end_run()
```

Рисунок 35 – Предсказание ансамбля и логирование результатов

На основании предсказаний всех трёх моделей формируется итоговый прогноз по принципу голосования.

Если одна из меток получила 2 или 3 голоса – она выбирается. В случае равенства двух лучших – используется случайный выбор.

Далее рассчитываются метрики ансамбля и логируются в MLflow: точность (ensemble_accuracy), F1-мера, precision и recall (в макроусреднённой форме), дополнительнаяметрика ensemble_correct_frac_0_2, отражающая долю правильно предсказанных наиболее значимых классов (0 и 2).

Наконец, в MLflow сохраняется CSV-файл с результатами инференса по последним 5000 наблюдениям.

Инференс FastAPI модели

```python
from fastapi import FastAPI
import mlflow.pyfunc
import pandas as pd
import numpy as np
from collections import Counter
import requests
from datetime import datetime, timedelta
import pandas_ta as ta
from dotenv import load_dotenv
import os
from fastapi.responses import JSONResponse

# Загрузка переменных окружения
load_dotenv(dotenv_path="../infra/.env", override=True)

# Безопасная установка в окружение
for var in ["AWS_ACCESS_KEY_ID", "AWS_SECRET_ACCESS_KEY", "MLFLOW_S3_ENDPOINT_URL"]:
    value = os.getenv(var)
    if value:
        os.environ[var] = value

if os.getenv("AWS_ACCESS_KEY_ID"):
    os.environ["AWS_ACCESS_KEY_ID"] = os.getenv("AWS_ACCESS_KEY_ID")
if os.getenv("AWS_SECRET_ACCESS_KEY"):
    os.environ["AWS_SECRET_ACCESS_KEY"] = os.getenv("AWS_SECRET_ACCESS_KEY")
if os.getenv("MLFLOW_S3_ENDPOINT_URL"):
    os.environ["MLFLOW_S3_ENDPOINT_URL"] = os.getenv("MLFLOW_S3_ENDPOINT_URL")

mlflow.set_tracking_uri("http://158.160.134.123:8080")
app = FastAPI()

# Загрузка моделей при старте
xgb_model = mlflow.pyfunc.load_model(model_uri="models:/XGBoostModel@production")
lgb_model = mlflow.pyfunc.load_model(model_uri="models:/LightGBMModel@production")
cat_model = mlflow.pyfunc.load_model(model_uri="models:/CatBoostModel@production")

models = {
    "XGB": xgb_model,
    "LGBM": lgb_model,
    "CatBoost": cat_model,
}
```

Рисунок 36 - Инициализация FastAPI и загрузка моделей из MLflow

Рисунок 36 показывает как, на этапе инициализации приложение загружает переменные окружения для доступа к MLflow и S3, а затем извлекает три модели (XGBoost, LightGBM, CatBoost) из реестра MLflow по алиасу @production. Это позволяет гибко управлять версионированием без хардкода путей.

```python
def get_latest_market_features():
    # Получение свечей за ~10 лет
    ...
    # Подсчёт индикаторов
    df["EMA_15"] = ...
    df["RSI_14"] = ...
    df["strong_up_trend"] = ...

    features = [ ... ]  # 26 признаков
    return df[features].iloc[[-1]]
```

Рисунок 37 - Загрузка и обработка последних рыночных данных с MOEX

Рисунок 37 демонстрирует функцию get_latest_market_features(), которая выполняет обращение к API Московской биржи (MOEX), извлекает часовые свечи за последние 10 лет и рассчитывает технические индикаторы: EMA, ADX, RSI и производные бинарные признаки.

На выходе – DataFrame с одной последней строкой (текущий момент) и 26 признаками.

```python
@app.get("/predict")
def predict():
    features_df = get_latest_market_features()
    features = features_df.to_dict(orient="records")[0]  # одна строка, как dict
    timestamp = features_df.index[-1].isoformat()

    predictions = []
    for model in models.values():
        pred = int(model.predict(features_df)[0])
        predictions.append(pred)
    vote = Counter(predictions).most_common(1)[0][0]

    return JSONResponse({
        "timestamp": timestamp,
        "prediction": vote,
        "votes": predictions,
        "features": features
    })
```

Рисунок 38 - Эндпоинт предсказания на основе голосования моделей

На рисунке 38 представлен эндпоинт /predict извлекает текущие рыночные признаки, подаёт их в каждую из моделей и агрегирует результат методом голосования.

Возвращается:

- ансамблевое предсказание (0, 1 или 2)
- индивидуальные голоса моделей,
- значения признаков,
- и временная метка.

3.2 Структура и наборы данных

3.2.1 Источник данных

Основным источником данных выступает публичное API Московской биржи (MOEX), откуда по инструменту SBER загружаются биржевые свечи с часовым интервалом. Формат ответа – JSON. Для получения данных используется endpoint:

https://iss.moex.com/iss/engines/stock/markets/shares/boards/TQBR/securities/SBER/candles.json

Параметры запроса включают: interval=60, диапазон дат (from, till) и start для пагинации.

3.2.2 Структура «сырых» данных

После преобразования данные представляют собой таблицу с индексом времени (datetime) и следующими колонками:

- Open – цена открытия;
- High – максимум за час;
- Low – минимум за час;
- Close – цена закрытия;
- Volume – объём торгов.

Эти данные служат основой для расчёта признаков (feature engineering).

3.2.3 Расчёт признаков (feature engineering)

На основе исходных данных рассчитываются технические индикаторы и бинарные признаки:

Скользящие средние: EMA_15, EMA_75

Индикаторы силы и направления тренда: ADX, DI+, DI−, RSI, ATR

Бинарные флаги (перекупленность, тренд вверх/вниз и др.):

ema_diff, rsi_above_70, adx_strong_trend, di_up_trend, strong_up_trend и др.

Общее количество признаков на один временной момент: 26.

3.2.4 Формирование целевой переменной

Целевой переменной является направление изменения цены через 14 часов вперёд:

- target = 1, если доходность выше +1%
- target = 0, если изменение в пределах ±1%
- target = -1, если падение больше 1%.

Для обучения моделей классы перекодируются в диапазон 0, 1, 2.

3.2.5 Разделение на обучающую, валидационную и тестовую выборки

В ходе работы было реализовано два подхода к разбиению данных, соответствующие различным этапам построения модели.

На этапе подбора гиперпараметров (с использованием Optuna) применялось классическое разделение выборки по пропорции:

70% – обучающая выборка (train),

20% – валидационная выборка (validation),

10% – тестовая выборка (test).

Разделение выполнялось по времени без перемешивания, чтобы сохранить хронологическую структуру временного ряда и исключить утечку информации между выборками. Это позволило проводить честную валидацию при поиске оптимальных параметров моделей.

На этапе оценки обобщающей способности моделей использовалась кросс-валидация по скользящему временному окну: на каждом шаге модель обучалась на последних 5000 наблюдениях и тестировалась на следующих 100. Такой подход имитирует реальные условия, в которых модель регулярно переобучается и принимает решения на ограниченном горизонте вперёд.

3.3 Технические условия и требования к железу

3.3.1 Аппаратная и облачная инфраструктура

Разработка, обучение и развёртывание решений осуществлялись в облачной платформе Yandex Cloud с использованием описания инфраструктуры в Terraform. Основные компоненты представлены в таблице ниже:

Таблица 8 – Основные компоненты инфраструктуры

Компонент	Характеристики
Виртуальная сеть (VPC)	VPC + подсеть 10.2.0.0/24 в зоне ru-central1-a
PostgreSQL-кластер	Yandex Managed PostgreSQL, конфигурация s2.micro (2 vCPU, 4 ГБ RAM), SSD 20 ГБ
S3-хранилище моделей	Yandex Object Storage, до 10 ГБ, приватный доступ (ml-models-*)
Kubernetes-кластер	Yandex Managed Kubernetes v1.30, сетевой провайдер Calico, публичный IP
Worker-ноды	2 экземпляра standard-v2, каждый: 2 vCPU, 4 ГБ RAM, SSD 50 ГБ, preemptible

Все сервисы были развернуты в зоне доступности ru-central1-a. Использование прерываемых (preemptible) VM в Kubernetes позволило снизить стоимость эксплуатации без существенного влияния на стабильность, поскольку при необходимости кластеры можно быстро масштабировать повторно.

4 ИСПОЛЬЗОВАНИЕ РЕШЕНИЯ

4.1 Область применения

Разработанное решение представляет собой интеллектуальную систему анализа финансовых временных рядов с применением ансамбля моделей машинного обучения. Основное назначение – автоматическая классификация рыночной ситуации по данному активу на основе технических индикаторов, с последующим использованием результата в рамках инвестиционной стратегии.

Решение может быть применено в следующих сценариях:

- создание и сопровождение инвестиционных стратегий, в том числе на базе ПИФов, ИИС или доверительного управления, где требуется формализованная логика принятия решений;

- интеграция в торговые терминалы или аналитические платформы, где модель может выступать в роли «индикатора настроения» или компонента комплексной оценки;

- разработка алгоритмических стратегий с автоматическим исполнением на бирже (algo trading) – в качестве модуля принятия решений или фильтра сигналов;

- внедрение в корпоративные BI-системы для оценки риска и анализа финансовых активов;

- образовательные и исследовательские проекты в области прикладного машинного обучения и анализа финансовых данных.

Алгоритм обучен на исторических данных по инструменту SBER (обыкновенные акции Сбербанка) с Московской биржи. При переносе на другие активы или рынки потребуется адаптация входных данных и повторное обучение модели. Тем не менее, предложенная архитектура является универсальной и масштабируемой.

Таким образом, решение может лечь в основу как самостоятельного инвестиционного продукта, так и технологической платформы для построения различных решений в сфере финансовых рынков.

4.2 Форма представления и порядок использования (manual)

Решение реализовано в виде набора программных компонентов, развернутых в облачной инфраструктуре Yandex Cloud с использованием инфраструктуры как код (IaC) на базе Terraform, Helm и Kubernetes. Основной пользовательский интерфейс – это REST API-сервис, реализованный на FastAPI и предоставляющий доступ к модели через HTTP-запрос.

Порядок использования решения включает следующие этапы:

Развертывание инфраструктуры:

– виртуальная сеть (VPC), кластер PostgreSQL, S3-хранилище и Kubernetes-кластер создаются средствами Terraform;

– деплой сервисов (MLflow, FastAPI-приложение) осуществляется через Helm-чарты.

Подготовка и обучение модели:

– обработка рыночных данных, расчёт технических индикаторов, формирование признаков и обучение моделей (XGBoost, LightGBM, CatBoost) автоматизировано в скрипте train.py;

– обученные модели логируются и регистрируются в MLflow с тегами и метриками.

Инференс (предсказание):

– при обращении к эндпоинту GET /predict FastAPI-приложение загружает последние данные с MOEX, рассчитывает признаки и выполняет прогноз с помощью ансамбля моделей;

– возвращается объект JSON со структурой.

Рисунок 39 - JSON структура ответа

CI/CD-процессы:

– используется GitHub Actions для автоматической сборки Docker-образов, их публикации в GHCR и деплоя в кластер;

– отдельный workflow отвечает за переобучение моделей на новых данных и обновление их в MLflow.

Таким образом, решение построено по принципу «нажми и работай»: все компоненты автоматизированы и управляются через систему контроля версий. Это обеспечивает гибкость в адаптации, масштабировании и воспроизводимости экспериментов.

4.3 Результаты тестирования и их интерпретация

4.3.1 Метрики модели

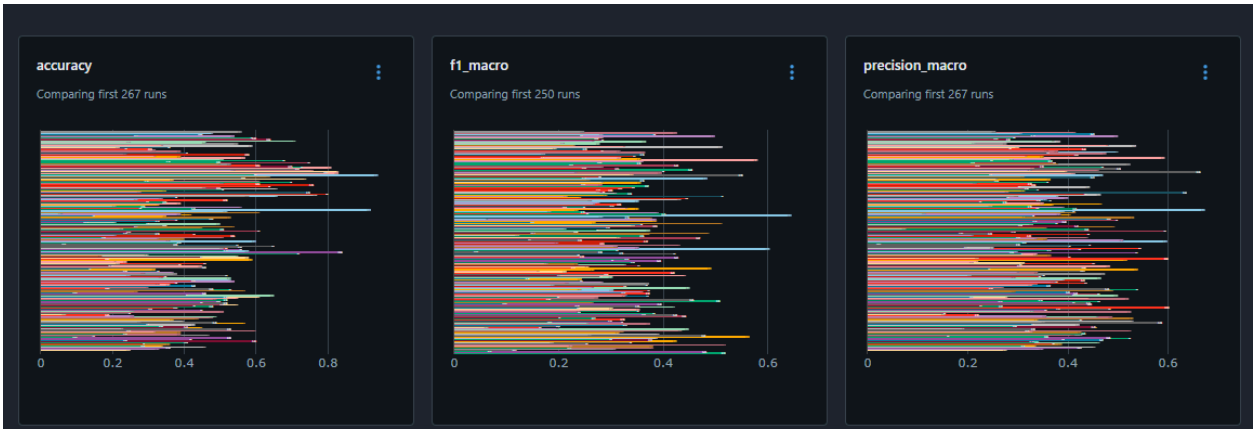


Рисунок 40 – Распределение метрик качества (accuracy, F1-macro, precision-macro) по 267 срезам кросс-валидации во времени

Рисунок 40 отображает графики, которые демонстрируют стабильность и вариативность качества модели на каждом временном отрезке. Большинство запусков показывает accuracy в диапазоне 0.3–0.6, при этом F1-мера и precision сохраняются преимущественно в пределах 0.2–0.5, что указывает на затрудненность модели в предсказании менее частых классов.

Рисунок 41 – Метрики recall-macro и доля корректных предсказаний классов 0 и 2 на 267 временных отрезках.

Рисунок 41 демонстрирует метрики recall-macro и correct_frac_0_2. Recall-мера показывает способность модели находить примеры всех классов, включая редкие. При этом метрика correct_frac_0_2 отражает, насколько хорошо модель предсказывает действия (покупка/продажа), а не «ничего не делать» (класс 1). На большинстве срезов видно, что модель склонна делать нейтральный прогноз, однако в ряде случаев доля верных предсказаний для классов 0 и 2 достигает 40–50%, что открывает потенциал для применения стратегии в условиях дополнительной фильтрации.

4.3.1 Результаты тестирования стратегии с применением фильтра

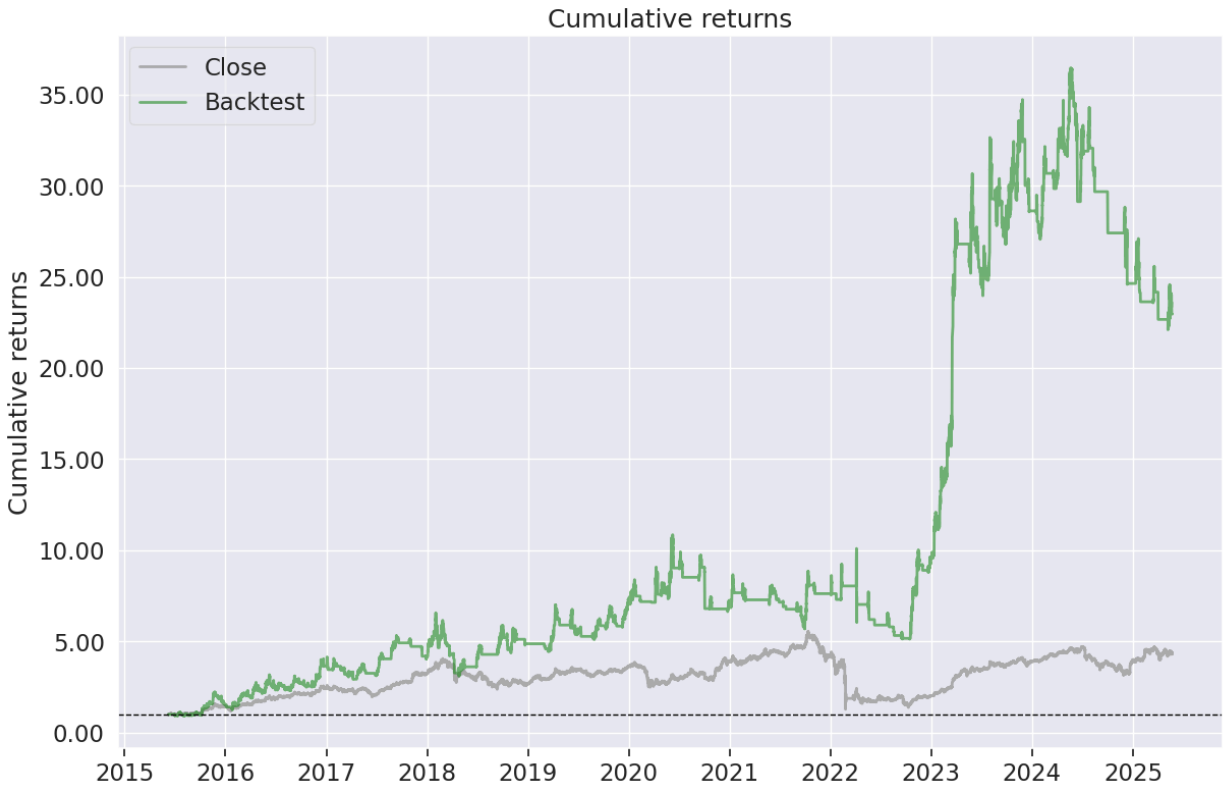


Рисунок 42 – Сравнение накопленной доходности торговой стратегии и базового актива (SBER)[19]

График на рисунке 42 демонстрирует, как изменялась совокупная доходность стратегии (зелёная линия) по сравнению с простой покупкой и удержанием бумаги Сбербанка (серая линия) в период с 2015 по 2025 год. Стратегия показывает существенное опережение базового актива, особенно в фазе роста в 2022–2023 годах.

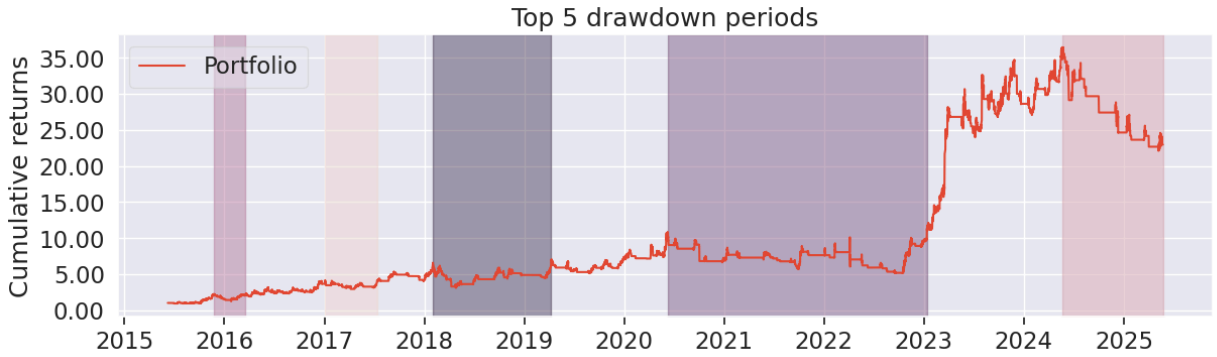


Рисунок 43 – Пять крупнейших периодов просадки стратегии

На графике рисунка 43 отображены наиболее значимые просадки капитала торговой стратегии за период 2015–2025 гг. Заштрихованные области визуализируют временные интервалы, в которых стратегия демонстрировала наиболее существенное снижение доходности от локального максимума.

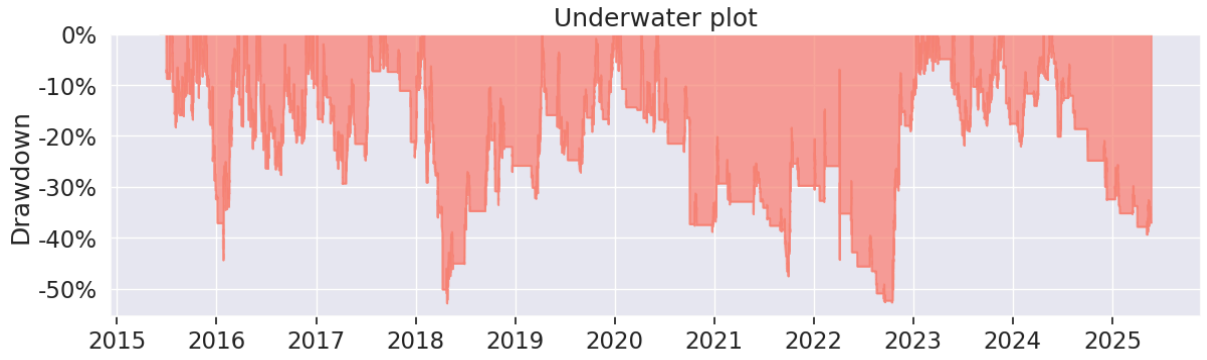


Рисунок 44 – Underwater-график стратегии

График рисунка 44 иллюстрирует просадки стратегии – периоды, когда стоимость портфеля снижалась относительно предыдущих максимумов. Несмотря на то что максимальная просадка стратегии достигает ~50%, бенчмарк (ценовая динамика актива SBER) показал куда более значительное снижение: с 387.9 до 89.5, что эквивалентно просадке более чем на 76%. Таким образом, стратегия демонстрирует лучшее поведение в стрессовых фазах рынка, сохраняя капитал эффективнее по сравнению с «купи и держи».

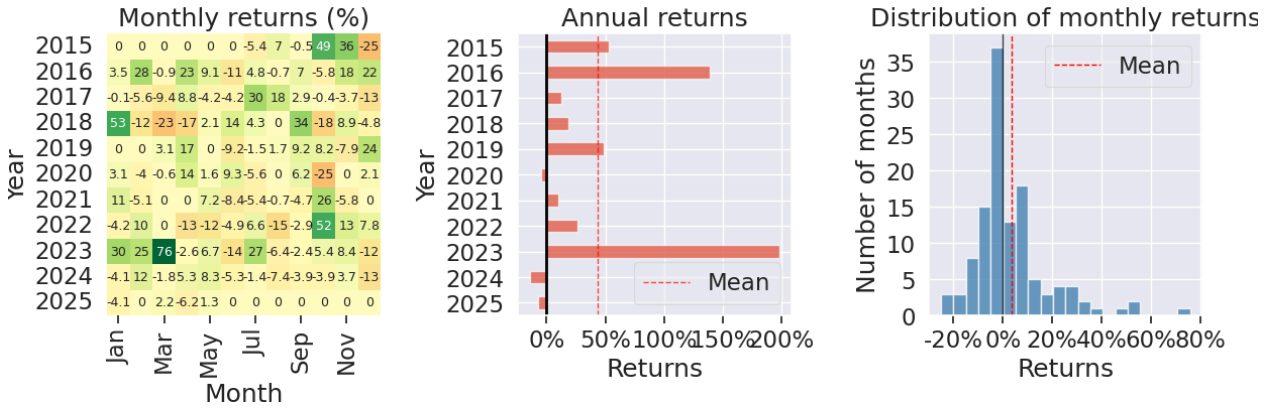


Рисунок 45 – Месячная и годовая доходность стратегии

На рисунке 45 представлены три аспекта анализа доходности стратегии:

Слева – тепловая карта ежемесячной доходности (в процентах) показывает волатильность результатов во времени. Стратегия имеет как периоды значительного роста (например, март–апрель 2023), так и отрицательные месяцы.

В центре – годовая доходность по годам, где особенно выделяется 2023 год с приростом свыше 200%.

Справа – гистограмма распределения месячных доходностей. Большинство месяцев дали умеренную положительную доходность, однако присутствуют и экстремальные значения, что указывает на наличие отдельных «взрывных» периодов роста.

Среднее значение месячной и годовой доходности обозначено пунктирной линией.

ЗАКЛЮЧЕНИЕ

В рамках настоящей исследовательской работы была разработана система краткосрочного прогнозирования направления движения цены акций ПАО «Сбербанк» на основе методов машинного обучения и технического анализа. Основной акцент был сделан на интеграцию лучших практик MLOps и воспроизводимой инфраструктуры в Yandex Cloud с автоматическим переобучением моделей.

В ходе исследования:

- получены и структурированы исторические рыночные данные через API Московской биржи;

- произведён расчёт технических индикаторов и сигнальных признаков;

- обучены и оптимизированы модели градиентного бустинга (CatBoost, LightGBM, XGBoost);

- реализован ансамблевый подход к прогнозированию на основе majority voting;

- проведена симуляция стратегии на исторических данных с оценкой качества по метрикам Accuracy, F1 и коэффициенту Sortino;

- настроена автоматизация жизненного цикла модели с использованием GitHub Actions, MLflow, Terraform и Kubernetes;

- реализовано FastAPI-приложение для инференса в облачной среде.

Разработанный подход продемонстрировал устойчивые результаты на валидационных отрезках и потенциальную применимость в реальных условиях. В дальнейшем работа может быть расширена за счёт подключения новых источников данных (новости, мультифакторные признаки), применения Reinforcement Learning и развития более комплексных торговых стратегий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Backtrader [Электронный ресурс] // Backtrader. - URL: https://www.backtrader.com (дата обращения: 10.05.2025).

2. Белова, Е. Технический анализ финансовых рынков / Е. Белова, Д. Окороков. - Москва : Инфра-М, 2006. - 368 с. - ISBN 5-16-002034-9.

3. Разработка и использование API [Электронный ресурс] // Habr. – URL: https://habr.com/ru/articles/590679 (дата обращения: 10.05.2025).

4. API Московской биржи [Электронный ресурс] // Московская биржа. - URL: https://www.moex.com/a2193 (дата обращения: 29.05.2024).

5. CatBoost [Электронный ресурс] // CatBoost. – URL: https://catboost.ai (дата обращения: 29.04.2024).

6. LightGBM [Электронный ресурс] // LightGBM. – URL: https://lightgbm.readthedocs.io (дата обращения: 29.04.2024).

7. XGBoost [Электронный ресурс] // XGBoost. – URL: https://xgboost.readthedocs.io (дата обращения: 29.04.2024).

8. MLflow [Электронный ресурс] // MLflow. – URL: https://mlflow.org/docs/latest/index.html (дата обращения: 29.04.2025).

9. Docker [Электронный ресурс] // Docker. - URL: https://www.docker.com (дата обращения: 10.05.2025).

10. Helm [Электронный ресурс] // Helm. – URL: https://helm.sh (дата обращения: 05.05.2025).

11. Terraform [Электронный ресурс] // Terraform. – URL: https://www.terraform.io (дата обращения: 05.05.2025).

12. Яндекс Облако [Электронный ресурс] // Яндекс Облако. – URL: https://cloud.yandex.ru/docs (дата обращения: 10.05.2025).

13. BackTesting.py [Электронный ресурс] // GitHub. – URL: https://kernc.github.io/backtesting.py (дата обращения: 29.05.2024).

14. OTUS. Развертывание и управление контейнерами [Электронный ресурс] // Habr. - URL: https://habr.com/ru/companies/otus/articles/767884 (дата обращения: 10.05.2025).

15. Янсен, С. Машинное обучение для алгоритмической торговли на финансовых рынках. Практикум. Издательство: «БХВ-Петербург» (Санкт-Петербург, 2020) - 560 с. ISBN: 978-5-9775-6595-0.

16. Паланиаппан, В. Как с помощью нейронных сетей прогнозировать цены акций на фондовой бирже [Электронный ресурс] // Netology. – URL: https://netology.ru/blog/mashinnoe-obuchenie-prognoz-cen (дата обращения: 29.05.2024).

17. Optuna. Оптимизация гиперпараметров [Электронный ресурс] // Optuna. – URL: https://optuna.org (дата обращения: 29.05.2024).

18. SHAP [Электронный ресурс] // GitHub. – URL: https://github.com/shap/shap (дата обращения: 29.05.2024).

19. Pyfolio [Электронный ресурс] // GitHub. – URL: https://github.com/quantopian/pyfolio (дата обращения: 29.05.2024).

20. Минаков, М. MLOps-пайплайн для прогнозирования направления движения акций [Электронный ресурс] // GitHub. – URL: https://github.com/MinakovMax/mlops (дата обращения: 10.05.2025)