О задаче построение независимых архитектур обработки данных в интеллектуальных транспортных системах

М.Г. Городничев, Д.В. Титов, А.Д. Липатова

Московский технический университет связи и информатики, Москва

Аннотация: В статье рассматриваются современные подходы к проектированию и реализации архитектур обработки данных в интеллектуальных транспортных системах (ИТС) с фокусом на обеспечение технологического суверенитета. Особое внимание уделяется интеграции практик машинного обучения для автоматизации полного жизненного цикла моделей машинного обучения: от подготовки и потоковой обработки данных до мониторинга и обновления моделей в реальном времени. Анализируются архитектурные решения с использованием распределённых вычислительных платформ, таких как Hadoop и Apache Spark, баз данных в оперативной памяти на Apache Ignite, а также брокеров обмена сообщениями Kafka для обеспечения надёжной передачи событий. Подчёркивается важность гибкости и масштабируемости инфраструктуры, поддержки параллельной работы множества моделей и надёжного управления доступом, включая вопросы обеспечения безопасности, и использование протоколов безопасности транспортного уровня. Приводятся рекомендации по организации системы логирования и мониторинга для оперативного реагирования на изменения и инциденты. Представленные решения ориентированы на обеспечение высокой отказоустойчивости, безопасности и соответствие требованиям промышленной эксплуатации, что позволяет поддерживать эффективную обработку больших объёмов транспортных данных и адаптацию систем ИТС к условиям импортонезависимости.

Ключевые слова: обработка данных, интеллектуальные транспортные системы, ,распределенная обработка, масштабируемость, отказоустойчивость.

Введение

В последние годы интеллектуальные транспортные системы (ИТС) становятся неотъемлемой частью современной городской инфраструктуры, помогая решать задачи управления транспортными потоками, повышения безопасности дорожного движения и оптимизации работы транспорта. Эффективность подобных систем во многом определяется способностью обрабатывать и анализировать большие объемы данных в реальном времени, что требует внедрения современных методов машинного обучения и продвинутых архитектур обработки данных. Однако для того, чтобы такие методы могли полноценно функционировать и приносить измеримую пользу, необходима надежная и гибкая инфраструктура, позволяющая

автоматизировать процессы развертывания, мониторинга и обновления моделей — так называемые практики машинного обучения (Machine Learning Operations – MLOps) [1].

Внедрение принципов MLOps в ИТС не ограничивается простым развертыванием моделей машинного обучения. Оно охватывает полный жизненный цикл обработки данных: от сбора и очистки до потокового внедрения моделей и анализа результатов. Особое значение здесь приобретает построение эффективных архитектур, позволяющих системам не только быстро и безопасно внедрять изменения, но и работать с высоконагруженными, распределенными потоками данных, характерными для транспортных приложений. В контексте ИТС MLOps обеспечивает сквозную интеграцию инструментов для автоматизации процессов обучения, валидации и эксплуатации моделей с высокими требованиями к надежности и производительности [2].

На этапе реализации системы критически важно детализировать концептуальную архитектуру и определить, как именно интегрируются компоненты потоковой обработки данных, механизмы доставки и обновления моделей. Учитывая особенности задач ИТС [3] (например, обработку данных с дорожных датчиков, видеоаналитику с камер, прогнозирование трафика), необходимо особое внимание уделять как техническим аспектам выбранных решений (устойчивость, масштабируемость, скорость отклика), так и сквозной автоматизации — от бизнес-логики до операционного мониторинга и управления. Ключевыми факторами, определяющими эффективность влияния искусственного интеллекта на оперативность и качество управленческих решений, выступают: качество и объём данных, вычислительные ресурсы, сложность моделей, уровень интеграции в бизнес-процессы и интерпретируемость используемых алгоритмов [4].

В этой статье акцентируется внимание на особенностях построения архитектурных решений для MLOps в интеллектуальных транспортных системах: детальном анализе ключевых этапов, специфику потоковой обработки и эксплуатацию моделей машинного обучения в реальном времени [5]. Мы рассмотрим лучшие практики проектирования, интеграции и сопровождения компонентов, обеспечивающих эффективное и надежное функционирование ИТС в условиях постоянно растущих объемов и скорости данных, поступающих из транспортной среды.

Исполнение модели с использованием систем управления базами данных

Проектирование и внедрение MLOps-архитектуры для интеллектуальных транспортных систем требует четкой формализации компонентов, обеспечивающих надежную и масштабируемую работу с потоковыми и историческими данными. Одним из ключевых модулей в подобных системах является компонент controller-database, задачей которого является организация корректного взаимодействия операционных сервисов с системами управления базами данных (СУБД) родительской платформы. Приведенные ниже функциональные требования отражают специфику применения данного модуля в рамках ИТС [6].

1. **Динамическая интеграция в систему.** Компонент должен иметь опциональный характер, то есть его подключение к системе, с использованием алгоритмов машинного обучения (Machine Learning Application – ML-Application), осуществляется посредством задания соответствующих параметров в конфигурационном файле системы. Такое требование обеспечивает гибкость архитектуры и возможность адаптации к различным сценариям эксплуатации в рамках ИТС.

2. **Поддержка многомодельности на уровне проекта.** Компонент обязан функционировать с каждой интегрированной моделью машинного обучения в строго синхронном режиме [7]. Это позволяет реализовать

одновременное и независимое обслуживание множества моделей в пределах единого транспортного проекта, что отражает требования ИТС к мультифункциональности и совместной работе нескольких аналитических модулей.

3. **Изолированная работа сервисов с родительской СУБД.** В рамках одного ML-Application допускается развертывание нескольких экземпляров компонентов как микросервисов. При этом каждый такой сервис должен обращаться исключительно к одной базе данных родительской системы, что обеспечивает изоляцию данных и защищает целостность информации при масштабировании приложений в условиях распределенной ИТС.

4. **Безопасное подключение к СУБД.** Для установления пул соединений (Connection Pooling) с базой данных компонент обязан использовать учетные данные (логин и пароль), хранящиеся в менеджере системы – Secrets. Такое решение соответствует современным требованиям к безопасности обработки и хранения чувствительных данных в транспортных системах.

5. **Встроенная поддержка драйверов для подключения.** В состав компонента должен входить актуальный драйвер для установления соединения с целевой СУБД родительской системы [8]. Это гарантирует технологическую совместимость с выбранным типом базы данных и упрощает процедуру развертывания и интеграции новых модулей интеллектуальных транспортных систем.

6. **Гибкая настройка взаимодействия с базой данных.** Модуль должен содержать файл конфигурации (с расширение. map) с описанием структуры и списка разрешённых запросов к СУБД. Указанный файл предоставляется пользователем при развертывании компонента, что позволяет оперативно адаптировать функциональность сервиса под конкретные бизнес-процессы в транспортной инфраструктуре.

7. **Ограничение на операции с данными.** Компонент должен инициировать выполнение исключительно операций языка манипулирования данными (Data Manipulation Language – DML), что исключает изменения структуры базы данных на этапе работы сервиса и минимизирует риски возникновения критических сбоев.

8. **Маршрутизация технических и бизнес-логов.** Для интеграции с системами мониторинга и аналитики, все бизнес- и технические записи событий, а также метрики, генерируемые компонентом, должны маршрутизироваться в отдельный сервис controller-logger [9]. Такое требование улучшает трассируемость событий и поддерживает соответствие стандартам операционного контроля в ИТС.

Данные требования определяют основу для построения отказоустойчивой и масштабируемой MLOps-архитектуры, предназначенной для эксплуатации в интенсивных условиях цифровой транспортной среды.

Нефункциональные требования к компоненту controller-database в архитектуре MLOps для интеллектуальных транспортных систем

Разработка и эксплуатация инфраструктуры обработки данных для интеллектуальных транспортных систем предъявляет высокие ожидания не только к функциональным, но и к нефункциональным характеристикам компонентов [10]. Ниже сформулированы ключевые нефункциональные требования в составе MLOps-архитектуры, нацеленной на обеспечение надежности, безопасности, интероперабельности и сопровождаемости решений в области ИТС.

1. **Сетевые требования и доступность.** Каждый экземпляр модели машинного обучения должен иметь возможность взаимодействовать с компонентом посредством сети, что обеспечивается открытием сервисного доступа по типу ClusterIP в Kubernetes [11]. Данная архитектурная особенность гарантирует изоляцию внутреннего трафика к компоненту

внутри кластера, повышая устойчивость и предотвращая несанкционированный доступ из внешних источников.

2. **Безопасное хранение учетных данных.** Все данные для установления пул соединения между компонентом и базой данных родительской автоматизированной системы (АС) хранятся в защищенных объектах Secrets Kubernetes. Это соответствует современным стандартам информационной безопасности и снижает риски компрометации учетных данных при эксплуатации ИТС.

3. **Административные меры подготовки среды.** Администратор среды обязан до развертывания сервисов вручную (или автоматически по утверждённым сценариям) разместить все необходимые данные аутентификации (логин, пароль, а также объекты типа keystore/truststore и пароли к ним) в Secrets-контуре Kubernetes [12]. Такое требование обеспечивает дисциплину и прозрачность операций при подготовке к промышленному внедрению ИТС-модулей.

4. **Технологическая независимость и универсальность.** Архитектура компонента принципиально независима от применяемого языка программирования, типа и количества обслуживаемых моделей машинного обучения. Взаимодействие между моделью и компонентом стандартизируется через сетевые интерфейсы REST или gRPC, что гарантирует масштабируемость и облегчает межплатформенную интеграцию в гетерогенных ИТС-средах.

5. **Синхронность и поддержка многомодельных сценариев.** Для минимизации задержек и обеспечения высокой согласованности обработки потоков данных во всех моделях, компонент обслуживает каждый вызов в строгом синхронном режиме. Это критично для задач ИТС, где результаты обработки должны быть доступны с минимальной временной дисперсией.

6. **Повышенные требования к безопасности.** В случаях, когда бизнес-процессы или нормативные акты требуют внедрения дополнительных мер защиты (многофакторная аутентификация, сквозное шифрование сетевого трафика и пр.), сервис компонента должен поставляться с преднастроенной поддержкой безопасности транспортного уровня (Transport Layer Security – TLS) [13, 14] в соответствии с корпоративными стандартами. Это обеспечивает защиту критически важных данных ИТС от угроз перехвата или несанкционированного доступа.

7. **Соответствие подходам промышленного DevOps.** В рамках согласованных DevOps-процессов запрещается развертывать в промышленной среде артефакты, отличные от финализированной ML-модели. Это исключает внедрение дополнительных программных зависимостей, повышая воспроизводимость, прозрачность и управляемость изменений в ИТС среде.

На рис.1 представлена архитектурная схема системы исполнения моделей машинного обучения с интеграцией с реляционной СУБД и компонентами мониторинга, реализованная в контейнерной среде Kubernetes. Описанная архитектура разрабатывалась с учётом специфики интеллектуальных транспортных систем, где критически важны надежная потоковая обработка данных, высокий уровень автоматизации и сквозной мониторинг ключевых процессов [15].

Основными сущностями здесь являются: инициатор и получатель. В качестве инициатора выступают запросы, которые могут инициироваться как внешней системой (например, транспортным аналитическим сервисом), так и внутренними компонентами ML-Application. В типичном сценарии данные исходят либо из транспортных сенсоров и агрегируются в СУБД, либо обработанные модели формируют обратный сигнал для других транспортных подсистем.

Основными архитектурными компонентами являются маршрутизатор, POD controller proxy, POD Model, POD controller database, POD controller logger [16]. Маршрутизатор служит входной точкой для всех запросов, поступающих как от сторонних систем, так и от внутренних сервисов. Прокси-маршрутизатор необходим для выполнения функции дополнительного уровня абстракции, предоставляя унифицированный интерфейс для маршрутизации вызовов к ML-моделям. В частности, при работе с несколькими моделями различной архитектуры данный слой снимает ограничения на совместимость и повышает устойчивость системы.

Модель POD представляет из себя контейнер реализованной моделью машинного обучения принимает обработанные запросы и производит вычисления, опираясь на потоковые или исторические транспортные данные. Модель POD является специализированным компонентом для синхронного получения данных из СУБД, с использованием заранее сконфигурированных соединений с базой данных Java (Java Database Connectivity – JDBC-соединения) и защищенных учетных данных, размещённых в защищённом хранилище конфиденциальной информации (Secrets) внутри кластера Kubernetes. В разрезе задачи ИТС подобный модуль позволяет оперативно извлекать необходимые данные (например, дорожную обстановку, параметры трафика) для моделей, что реализуется через REST/gRPC-интерфейсы и прямой доступ к СУБД.

Все технические и бизнес-события, а также метрики, генерируемые компонентами ML-Application, маршрутизируются в специализированный сервис логирования. Для передачи используется, в зависимости от типа данных, протокол WebSocket или очередь сообщений Kafka, что обеспечивает масштабируемую интеграцию с промышленными системами мониторинга. Такой подход критичен для ИТС, где требуется оперативная диагностика и прозрачное отслеживание состояния сервисов. В данной

архитектуре предлагается использовать рассредоточенные учетные данные для обеспечения защищённого соединения с внешними ресурсами (СУБД и иными сервисами), хранятся в виде объектов Secrets Kubernetes, реализуя стандарты безопасности промышленной эксплуатации.

Протоколы REST и gRPC используются в качестве основных транспортных механизмов обмена данными между компонентами приложения, что позволяет обеспечить гибкость, технологическую независимость и адаптацию к высоким нагрузкам, характерным для ИТС. Для мониторинга применяется публикуемая через Kafka или WebSocket телеметрия, с последующей интеграцией в платформы визуализации Grafana.

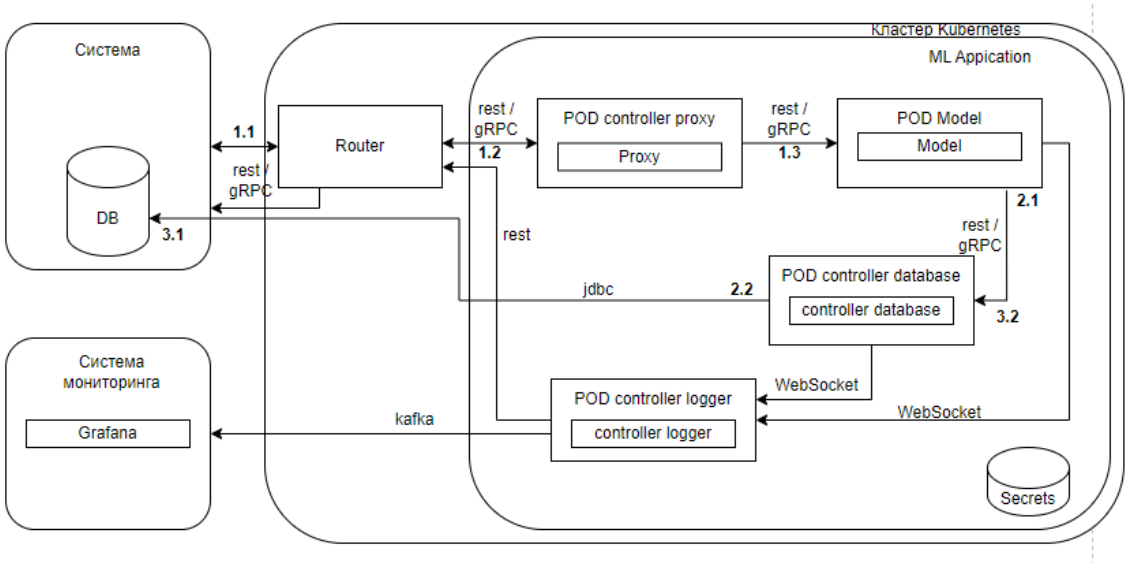

Рис. 1. – Архитектура системы исполнения модели с использованием СУБД

Исполнение модели с использованием графических процессоров

Разработка архитектуры современных ИТС неизбежно сталкивается с необходимостью выполнения ресурсоемких вычислений, связанных с обработкой и анализом больших объемов транспортных данных в реальном времени. Одним из ключевых факторов повышения эффективности работы аналитических моделей машинного обучения в подобных задачах становится

использование специализированных вычислительных ускорителей, прежде всего графических процессоров (Graphics Processing Unit - GPU) [17].

Включение GPU в вычислительную инфраструктуру открывает новые возможности как для ускоренного обучения, так и для быстрого исполнения сложных моделей, применяемых, например, для видеоаналитики трафика, распознавания событий на транспортных объектах и прогнозирования потоков на городской дорожной сети [18]. Для обеспечения высокой производительности система развертывается в облачной среде с поддержкой автоматизированного управления вычислительными ресурсами на уровне Kubernetes, что позволяет динамически определять и подключать подходящие узлы с установленными GPU на основании спецификации в конфигурационных файлах.

Для мониторинга состояния инфраструктуры, а также сбора технических и бизнес-метрик в режиме реального времени в системе выделен специализированный микросервис controller-logger. Этот компонент агрегирует событийную информацию и журналы событий, интегрируется с инструментами визуализации и предоставляет возможность оперативной диагностики сбоев или аномалий.

Представленная на рис.2 архитектура, иллюстрирует систему исполнения моделей посредством GPU, ориентированную на использование в рамках ИТС. Архитектура развертывается в контейнеризированной среде Kubernetes, что обеспечивает гибкость, масштабируемость и стандартизированные механизмы управления вычислительными ресурсами, критически важные для задач транспортной отрасли [19].

В качестве инициатора обработки могут выступать различные внешние или внутренние транспортные системы, которые через типовые протоколы REST или gRPC направляют запросы на выполнение моделей, реализующих

анализ данных или интеллектуальное принятие решений в транспортной среде.

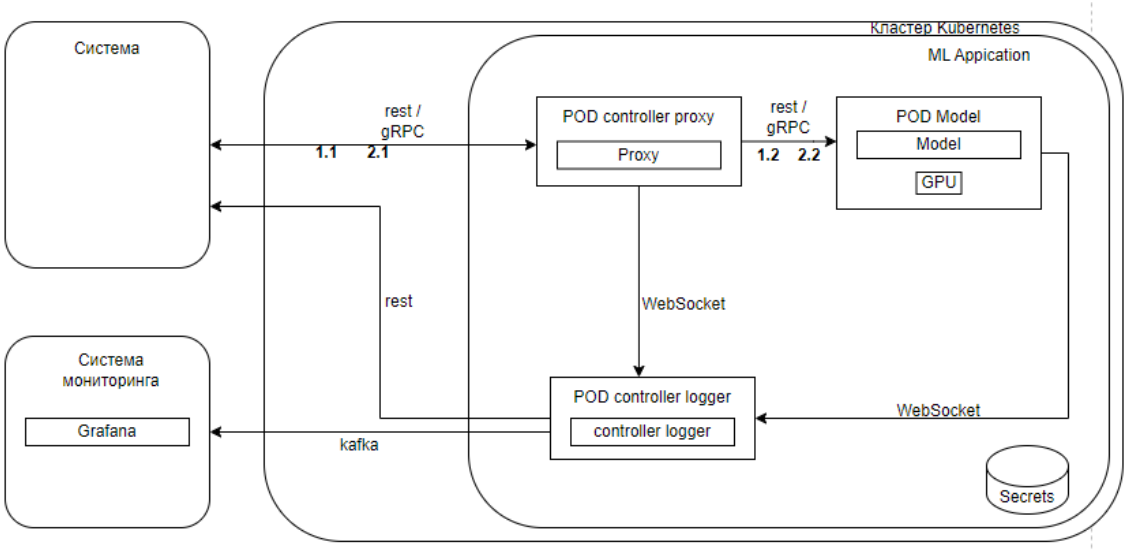

Рис. 2. – Архитектура системы исполнения моделей с использованием GPU

Первичной точкой входа этих запросов выступает микросервис POD controller proxy, который реализует функции маршрутизации и управления потоками данных, обеспечивая балансировку нагрузки и подготовку трафика к дальнейшей обработке. В непосредственной связке с этим компонентом функционирует модель POD — контейнер, содержащий модель машинного обучения с доступом к аппаратным ресурсам GPU.

Потоки данных между прокси-слоем и моделью организованы с максимальной производительностью за счет поддержки как REST, так и высокоэффективных gRPC-вызовов, что соответствует необходимости минимизации задержек в ИТС [20]. Сервис Secrets Kubernetes централизует хранение и управление аутентификационными данными, необходимыми для защищённых взаимодействий между компонентами, что важно с точки зрения информационной безопасности. Обеспечение такой инфраструктуры логирования, мониторинга и разграничения доступа позволяет оперативно

реагировать на аномалии и инциденты, а также поддерживать высокие стандарты устойчивости при промышленной эксплуатации ИТС.

Исполнение модели с использованием Hadoop

Современные ИТС требует применения гибких, масштабируемых и отказоустойчивых платформ. Одной из таких платформ является Hadoop, предоставляющий инструменты для хранения и параллельной обработки значительных массивов транспортных данных. В контексте повышения аналитических и вычислительных возможностей ИТС интеграция Hadoop-экосистемы с кластерами Spark [21] становится стратегическим преимуществом, позволяя использовать распределённую обработку данных как по инициативе пользователя, так и для автоматизации сложных бизнес-процессов анализа трафика, прогнозирования событий и поддержки принятия решений в квазиреальном режиме.

Кластер Spark, развернутый в Standalone Mode, может быть подключён к архитектуре как опциональный вычислительный компонент путём активации соответствующих параметров в конфигурации. Приложения, работающие с Hadoop через Spark, организуют раздельные процессы управления и исполнения: компонент spark-driver инициирует создание SparkContext и устанавливает соединение с Spark Master, который, в свою очередь, распределяет исполняемые задачи между множеством процессов spark-executor, развернутых на отдельных узлах кластера. Весь цикл передачи задач, сбора результатов и управления состоянием рабочих процессов осуществляется посредством RPC-протокола, что способствует устойчивости системы по отношению к нагрузкам и сбоям отдельных компонентов.

Особое значение имеет система мониторинга и логирования. Spark natively предоставляет интерфейс прикладного программирования (Application Programming Interface – API) для передачи телеметрии сервисов, а также поддерживает богатую систему метрик на основе Dropwizard Metrics,

что позволяет интегрировать данные с внешними платформами мониторинга, такими как Prometheus [22]. Метрики детализируются по уровням: master, worker, executor и driver, что делает контроль качества исполнения задач максимально прозрачным, а выявление аномалий — оперативным.

Функции управления развертыванием сервисов архитектуры реализуются на основе DeploymentConfigs, что обеспечивает как повторяемость и управляемость жизненного цикла компонентов, так и возможность оперативной адаптации системы под новые транспортные задачи без ущерба для устойчивости и безопасности обработки данных. Такой подход позволяет строить единое пространство обработки и анализа транспортных данных, обеспечивая высокую эффективность и отказоустойчивость решений в области ИТС.

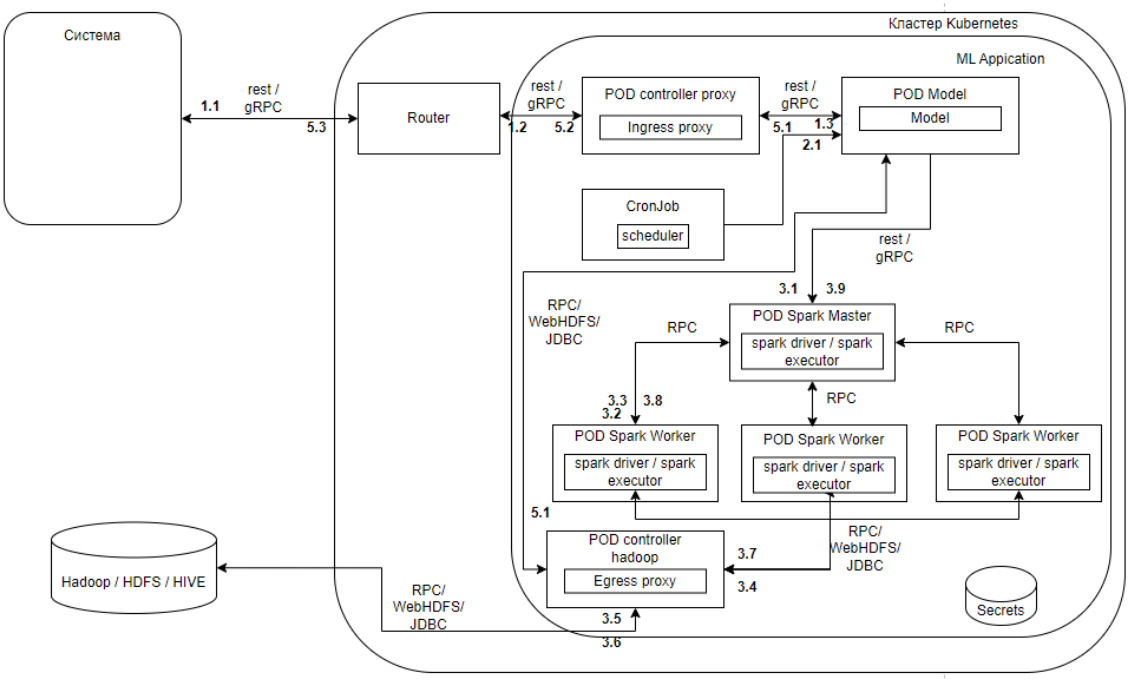

Рис. 3. – Архитектура системы исполнения модели на Hadoop

На рис.3 представлена архитектура системы исполнения моделей в рамках ИТС с использованием технологий Hadoop. Маршрутизатор распределяет входящие запросы в Kubernetes-кластер, внутри которого

функционирует ML-Application, состоящее из нескольких ключевых модулей и прокси-слоёв.

Для выполнения вычислительно сложных операций система использует компоненты Spark, развернутые в виде POD-ов. В частности, планировщик (CronJob scheduler) осуществляет планирование периодических и фоновых задач, направленных в POD мастера (spark master), где размещён драйвер (spark driver) и исполнитель (spark executor). Мастер посредством протокола удаленного вызова процедур (Remote Procedure Call – RPC) взаимодействует с множеством работников (spark worker), каждый из которых также содержит драйверы и исполнители для параллельной обработки данных.

Для интеграции с большими массивами данных система использует Hadoop-контроллеры, предоставляющие доступ к файловой системе Hadoop, Hadoop Distributed File System (HDFS) или базе данных Hive через RPC, WebHDFS либо JDBC-протоколы. Эти контроллеры (например, Egress proxy) обеспечивают безопасный доступ к данным, размещая запросы из Spark Worker-ов к хранилищу, и наоборот — предоставляя сервисам возможность выгружать обработанные данные обратно в Hadoop [23].

Общая архитектура предполагает тесную интеграцию между отдельными слоями — сеть взаимодействий реализована по протоколам REST, gRPC, RPC и JDBC, что обеспечивает масштабируемость, надёжность и возможность гибкого управления вычислениями при выполнении моделей машинного обучения для задач ИТС.

Исполнение модели с использованием объектного хранилища

В современных высоконагруженных вычислительных системах и платформах машинного обучения особое значение приобретает эффективная и масштабируемая обработка больших объемов данных, хранящихся во внешних файловых хранилищах, таких как объектные хранилища, например Simple Storage Service (S3). При разработке архитектурных решений для

подобных систем целесообразно внедрять специализированные прокси-сервисы, обеспечивающие прозрачное и контролируемое взаимодействие между пользовательской моделью и хранилищем данных. Одним из таких сервисов выступает компонент controller-storage, реализуемый как опциональный инфраструктурный сервис и активируемый по мере необходимости через соответствующие параметры конфигурации. Данный сервис предназначен исключительно для посредничества в потоках данных между моделью и объектным хранилищем. Сервис принимает ссылки на необходимые файлы в S3 bucket [24], предоставленные системой по запросу к модели, выбирает целевой ресурс и, используя протокол S3, обеспечивает взаимодействие по сценарию «только чтение».

Для корректной интеграции и стандартизации обмена данными компонент использует библиотеку Minio Java Client SDK, что позволяет абстрагироваться от низкоуровневых особенностей протокола S3 и поддерживать передачу данных в широком диапазоне форматов, диктуемых спецификациями соответствующего интерфейса. Архитектурно предусматривается единичный экземпляр данного компонента на ML-Application, что упрощает управление конфигурациями, централизует обработку запросов и способствует повышению контролируемости доступов к объектному хранилищу. В интересах безопасности взаимодействие компонента с S3 происходит с использованием клиентских учетных записей, каждая из которых обладает уникальными учетными данными (ключом доступа, секретным ключом, идентификатор пользователя и иные атрибуты доступа).

Сетевое взаимодействие и доступ к компоненту детально регулируются с помощью настроек iptables на стороне кластера, что ограничивает взаимодействие необходимыми компонентами (например, только ClusterIP модели) [25]. В качестве объекта развертывания сервисы, реализующие

данные функции, оформляются с использованием DeploymentConfigs, что позволяет гибко управлять масштабируемостью и обновлением компонентов в Kubernetes-окружении. Такой подход формирует единый, управляемый и защищённый контур доступа к S3 для задач интеллектуального анализа данных и построения машинного обучения.

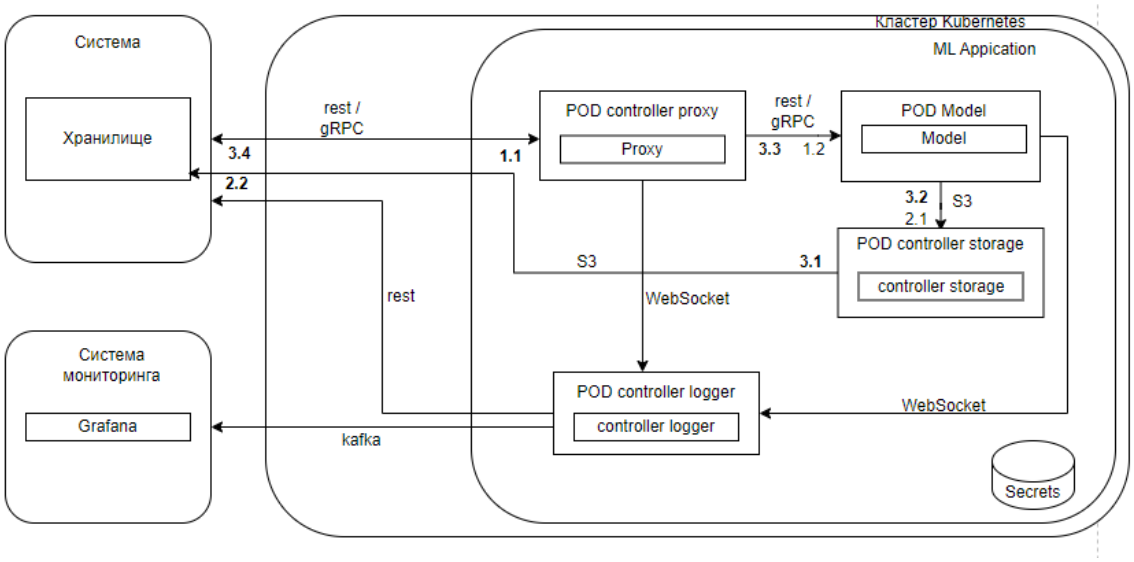

Рис. 4. – Архитектура системы исполнения модели на S3

На представленном рис.4 отражена архитектура системы исполнения моделей, предназначенной для использования в ИТС и интегрированной с объектным хранилищем типа S3. Для взаимодействия с внешним объектным хранилищем данных встроен специализированный сервис — POD controller storage. Этот компонент реализует механизм безопасной передачи больших массивов данных между ML-моделью и S3-хранилищем, используя стандартный протокол S3 как для входящих, так и для исходящих соединений. Сервис выступает единой точкой доступа для запроса данных в рамках выполнения задач машинного обучения и обеспечивает передачу необходимых данных в соответствующем формате из хранилища непосредственно в модель.

Аспекты информационной безопасности реализованы через централизованное хранение чувствительных данных (например, учетных

данных для доступа к S3-хранилищу) в сервисе Kubernetes Secrets, что предотвращает несанкционированный доступ и обеспечивает целостность инфраструктуры. Все взаимодействия между компонентами кластера отграничиваются с помощью современных средств сетевой фильтрации и протоколирования.

Данная архитектура демонстрирует комплексный подход к организации распределенной системы исполнения моделей в рамках интеллектуальных транспортных решений, интегрированной с современными облачными сервисами хранения и мониторинга.

Исполнение модели с использованием баз данных в оперативной памяти

В современных системах поддержки интеллектуальных транспортных решений все большую значимость приобретает использование высокопроизводительных баз данных в оперативной памяти (in-memory database – IMDB) для своевременной обработки и хранения больших потоков данных в рамках приложения машинного обучения. Особый интерес в данном контексте представляет реализация, основанная на технологии Apache Ignite, выступающей в качестве базовой IMDB-платформы. Внедрение IMDB предусмотрено как опциональный функциональный модуль, активация которого выполняется на этапе конфигурирования целевого ML-приложения. Кластер IMDB развертывается строго изолированно в пространстве конкретного приложения, что обеспечивает безопасность и сегментацию пользовательских данных, а также исключает несанкционированный доступ со стороны сторонних сервисов.

Формирование схемы памяти и ключевых таблиц осуществляется автоматически в момент запуска ML-приложения, а устойчивость системы повышается за счет механизма автоматического избрания нового координатора в случае выхода текущего из строя с последующим перезапуском упавших узлов. Особое внимание уделено вопросам

отказоустойчивости и сохранности оперативных данных: применяется трехкратная репликация (одна основная и две резервных копии), что практически сводит к нулю риск утери информации даже при массовых сбоях отдельных компонентов. Подтверждение транзакций в IMDB реализовано в асинхронном режиме, что способствует повышению производительности и уменьшению времени отклика под высоконагруженными сценариями.

Особенно важно, что IMDB полностью работает в оперативной памяти, не прибегая к внешним постоянным хранилищам, а долговечность данных обеспечивается исключительно механизмами внутренней репликации.

Рабочие компоненты системы имеют возможность прямого асинхронного доступа к данным IMDB, а внешние автоматизированные системы через контроллер взаимодействия с Kafka — при этом поддерживается режим тонкого клиента и сетевое взаимодействие только по протоколу управления передачей (Transmission Control Protocol – TCP) [26]. Для обеспечения безопасности предусмотрено логирование всех обращений, реализация строгой маршрутизации запросов через контроллер логирования и поддержка mTLS v1.2. Компоненты развертываются с использованием DeploymentConfigs и доступны только внутри пространства имен Kubernetes, что дополнительно сегментирует инфраструктуру и минимизирует риски стороннего вмешательства.

Подобная архитектурная организация обуславливает высокую скорость операций, гибкость масштабирования, устойчивость к сбоям и отвечает строгим требованиям по безопасности и управляемости.

В представленной на рис.5 архитектуре входным инициатором процессов выступает внешняя система, которая взаимодействует с ядром ML-приложения посредством унифицированных протоколов REST/gRPC и направляет свои запросы в маршрутизатор.

В кластере Kubernetes сосредоточены основные функциональные компоненты: несколько моделей машинного обучения, реализованных в отдельных моделях POD, и масштабируемый кластер кэширования в оперативной памяти, представленный координирующим узлом (Chache Node Coordinator) и набором рабочих узлов (Chache Node 1, Chache Node 2 и пр.) [27]. Архитектурно связь между моделями и кластером баз данных в оперативной памяти (In-Memory) организована по протоколу TCP. Такой подход позволяет эффективно поддерживать сценарии с высокими требованиями к оперативности обработки информации и быстрым доступом к актуальному кэшированному состоянию.

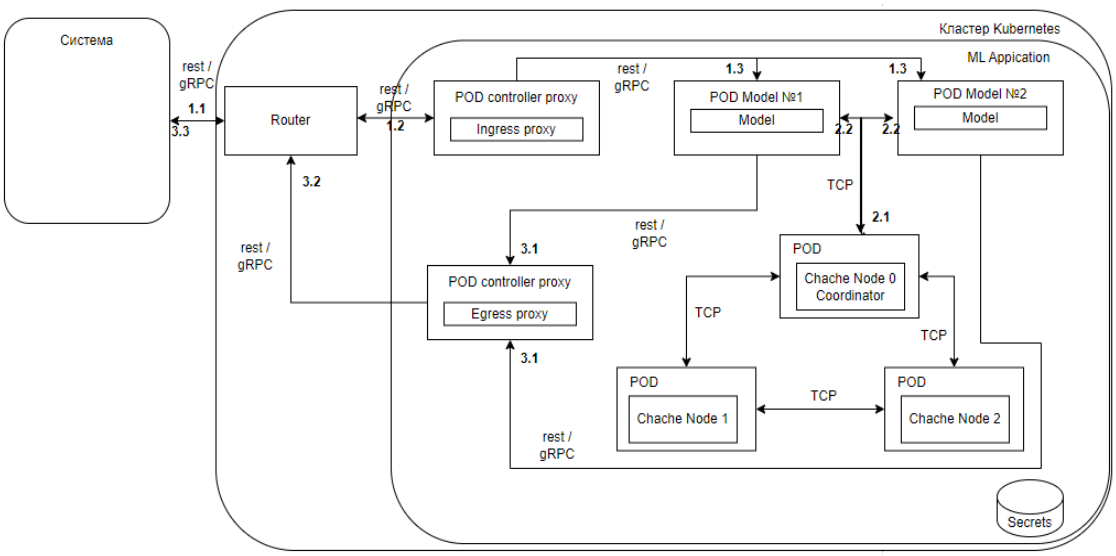

Рис. 5. – Архитектура системы исполнения модели In-Memory

Каждая модель машинного обучения способна обращаться к распределённому In-Memory хранилищу напрямую по защищённым каналам связи, что облегчает реализацию сложных вычислительных цепочек, требующих синхронной работы между множеством компонентов. Для обеспечения согласованности, отказоустойчивости и балансировки нагрузки в кластере In-Memory предусмотрен специальный координирующий узел, который принимает управляющие функции и реализует алгоритмы переназначения ролей в случае сбоя одного из компонентов.

Взаимодействие между компонентами, реализованное на современных протоколах и опирающееся на внутренние ресурсы быстродействующей памяти, обеспечивает максимальное быстродействие при работе с большими объемами транспортных данных, а также гарантирует масштабируемость и гибкость всей архитектуры в соответствии с динамическими требованиями ИТС.

Обращение к модели по Kafka

В архитектуре современных распределённых вычислительных систем для интеллектуальных транспортных решений важнейшее значение приобретает надёжный, масштабируемый и защищённый обмен сообщениями между компонентами, осуществляемый посредством брокеров событий, таких как Apache Kafka. Ключевым элементом подобной архитектуры выступает сервис controller-kafka, интегрированный в окружение машинного обучения и предназначенный для организации защищённых коммуникаций между автоматизированной системой, экземплярами моделей и вспомогательными сервисами среды исполнения. Данный сервис реализует механизм публикации и подписки (pub/sub) для широкого спектра топиков, обеспечивая асинхронное взаимодействие системы с моделью и другими компонентами, что позволяет гибко обрабатывать высоконагруженные потоки событий в режиме реального времени.

Controller-kafka развёртывается в единственном экземпляре для всего ML-Application, но архитектурно поддерживает масштабирование для повышения отказоустойчивости и пропускной способности. Аутентификация потребителей и продюсеров сообщений реализуется за счёт обязательной проверки цифровых сертификатов, защищённых с использованием протокола mTLS. Такое решение также предусматривает специализированные механизмы для работы с большими объёмами данных: ответы моделей,

превышающие заданный порог по размеру, фрагментируются и впоследствии собираются на стороне принимающей системы для корректной реконструкции исходного сообщения.

Для обеспечения высокой производительности чтение сообщений и исполнение вычислений допускают многопоточность — до 10 параллельных потоков на под (pod) — что критически для обеспечения необходимого уровня масштабируемости в транспортных системах [28]. При возникновении ошибок выполнения моделей специализированные сообщения о сбоях и метрики публикуются в отдельные выходные топики Kafka.

Вся среда передачи событий защищается с помощью политик ACL, изолированных топиков Kafka и комплексных средств аутентификации и авторизации. Дополнительно controller-kafka поддерживает сквозную трассировку транзакций за счёт включения уникального идентификатора запроса в каждый транспортируемый пакет, что облегчает аудит операций и повышает прозрачность архитектуры.

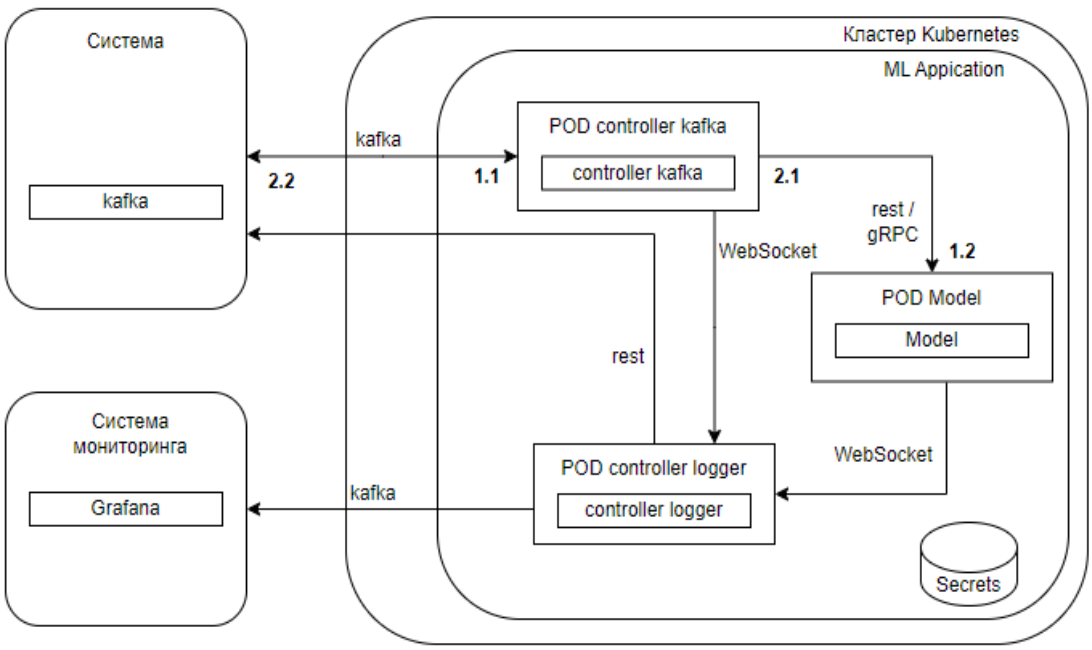

Рис. 6. – Архитектура системы исполнения модели по Kafka

На рис.6 представлена архитектура исполнения моделей, предназначенная для применения в ИТС с опорой на событийно-ориентированный обмен сообщениями посредством Apache Kafka. В качестве инициатора процесса выступает внешняя система, которая взаимодействует с приложением машинного обучения, развёрнутым в кластере Kubernetes, через защищённый Kafka-канал. Таким образом, все входящие сообщения поступают непосредственно в POD controller kafka — специализированный компонент, отвечающий за публикацию и подписку событий, маршрутизацию данных между внешней системой и моделью, а также реализацию механизмов отказоустойчивости и масштабируемости обмена.

Внутри кластера архитектурно заложено разделение функций между вычислительными и сервисными компонентами. Модуль controller kafka осуществляет доставку сообщений в модель (POD Model) с помощью WebSocket-каналов и по протоколам REST/gRPC, обеспечивая двустороннюю асинхронную коммуникацию, необходимую для масштабируемой обработки транспортных данных в режиме pub/sub. Модель, функционирующая в изолированном контейнере, может инициировать обратную отправку результатов и событий также через controller kafka, сохраняя целостность потоков данных и корректный аудит всех операций.

Архитектурное решение обеспечивает адаптивность, горизонтальную масштабируемость и устойчивость к сбоям за счёт отделения транспортного уровня обмена сообщениями от вычислительной логики моделей. Использование событийно-ориентированного стека с поддержкой многопоточности и асинхронного взаимодействия позволяет обрабатывать большие потоки транспортных данных практически в реальном времени — это критично для задач ситуационного реагирования и поддержки принятия

решений в ИТС. Таким образом, схема поддерживает гибкое интеграционное взаимодействие между различными транспортными и аналитическими сервисами, а также обеспечивает непрерывную загрузку и обработку данных с высоким уровнем отказоустойчивости, безопасности и прозрачности работы.

Заключение

Разработка и внедрение архитектурных решений для обработки данных в интеллектуальных транспортных системах требует комплексного подхода, сочетающего передовые методы автоматизированного управления жизненным циклом моделей машинного обучения (MLOps), гибкую организацию потоковой обработки и высокую степень технологической независимости используемой инфраструктуры. Представленные в статье концепции и примеры архитектур — от интеграции с высокопроизводительными вычислительными кластерами (GPU, S3, Hadoop, IMDB) до использования брокеров сообщений Kafka и современных средств оркестрации на базе Kubernetes — демонстрируют богатый набор инструментов, позволяющих эффективно адаптировать ИТС к специфике транспортных данных и требованиям к надежности, масштабируемости и отказоустойчивости.

Особое внимание уделено вопросам организации защищенного обмена данными при помощи протоколов TLS, централизованного управления секретами через Kubernetes, строгая изоляция и контроль доступа к компонентам ИТС формируют отраслевой стандарт для устойчивой промышленной эксплуатации. Интеграция потоковой аналитики, управляемого развертывания моделей, масштабируемых средств мониторинга и логирования обеспечивает возможность быстрой адаптации архитектуры под изменяющиеся прикладные задачи транспортной среды. Эти решения позволяют не только реализовать сквозную цифровизацию

управления транспортной инфраструктурой, но и существенно повысить автономность и технологическую независимость отрасли в современных условиях.

Предложенные подходы к построению архитектур обработки данных обеспечивают необходимый фундамент для создания интеллектуальных транспортных систем нового поколения, способных функционировать в режиме реального времени, поддерживать высокий уровень безопасности и адаптироваться к вызовам городской мобильности.

Литература

1. Gorodnichev M., Nigmatulin A. Technical and Program Aspects on Monitoring of Highway Flows (Case Study of Moscow City). Advances in Intelligent Systems and Computing. 2013. Vol 224. Pp. 195-204. Doi: 10.1007/978-3-319-00945-2_17.

2. Yuan T., Da Rocha Neto W., Esteve Rothenberg C., Obraczka K., Barakat C., Turletti T. Machine Learning for Next-Generation Intelligent Transportation Systems: A Survey. Transactions on Emerging Telecommunications Technologies. 2022. Vol. 33. №4. Doi: 10.1002/ett.4427. URL: researchgate.net/publication/356613364_Machine_Learning_for_Next-Generation_Intelligent_Transportation_Systems_A_Survey

3. Ouyang L., Wu J., Jiang X., Almeida D., Wainwright C. L., Mishkin P., Zhang C., Agarwal S., Slama K., Ray A., Schulman J., Hilton J., Kelton F., Miller L., Simens M., Askell A., Welinder P., Christiano P., Leike J., Lowe R. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems (NeurIPS 2022). 2022. Vol. 36. Doi: 10.48550/arXiv.2203.02155. URL: arxiv.org/abs/2203.02155.

4. Скворцова И.В., Нурулин Ю.Р., Сомов А.Г. Исследование влияния искусственного интеллекта на качество и скорость

принятия решений // Инженерный вестник Дона. 2025. №10. URL: ivdon.ru/ru/magazine/archive/n10y2025/10417.

5. Florence S., C. Shyamala Kumari. Big Data and IoT in Smart Transportation System. International Journal of Innovative Technology and Exploring Engineering. 2019. Vol. 8. (9) Pp. 1230-1232. Doi: 10.35940/ijitee.I7597.078919.

6. Bulatova O. Using big data in smart cities transportation systems. E3S Web of Conferences. International Scientific Conference “Fundamental and Applied Scientific Research in the Development of Agriculture in the Far East” (AFE-2022). Tashkent: EDP Sciences, 2023. Vol. 371. Doi: 371. 10.1051/e3sconf/202337106009. URL: doi.org/10.1051/e3sconf/202337106009.

7. Bhanu P. Reddy R. MLOps and DataOps Integration for Scalable Machine Learning Deployment. International Journal For Multidisciplinary Research. 2022. Vol. 4 (1). URL: ijfmr.com/research-paper.php?id=39278.

8. Oladimeji D., Gupta K., Kose N. A., Gundogan K. K., Ge L., Liang F. Smart Transportation: An Overview of Technologies and Applications. Sensors. 2023. Vol. 23 (8), 3880. Doi: 10.3390/s23083880. URL: pubmed.ncbi.nlm.nih.gov/37112221/.

9. Danasingh A.A., Tamizhpoonguil B., Jebamalar L. E. A Survey on Big Data and Cloud Computing. International Journal on Recent and Innovation Trends in Computing and Communication. 2016. Vol. 4, Pp. 273 - 277. Doi: 10.1109/WCCCT.2016.36.

10. Kune R., Pramod K., Arun A., Rao C., Rajkumar . The Anatomy of Big Data Computing. Software: Practice and Experience. 2015, Vol. 46. Pp. 79 -105. Doi: 10.1002/spe.2374.

11. Azeem M., Bassam M.A., Priyadarshana D. Mobile Big Data Analytics Using Deep Learning and Apache Spark. Mesopotamian Journal of Big Data. 2023. Vol. 2023. Pp. 16-28. Doi: 10.58496/MJBD/2023/003.

12. Kekevi U., Aydin A. Real-Time Big Data Processing and Analytics: Concepts, Technologies, and Domains. Computer Science. 2022. Vol. 7 (2). Pp. 111-123. Doi: 10.53070/bbd.1204112.

13. Al-Jaroodi J., Mohamed N. and Jiang H. Distributed systems middleware architecture from a software engineering perspective. Proceedings FifthIEEE Workshop on Mobile Computing Systems and Applications, Las Vegas, NV, USA, 2003. Pp. 572-579. Doi: 10.1109/IRI.2003.1251467.

14. Almeida F. Prospects of Cybersecurity in Smart Cities. Future Internet. 2023. Vol. 15 (9), 285. Doi: 10.3390/fi15090285. URL: mdpi.com/1999-5903/15/9/285.

15. Hui T., Li X., Quan H., Chang C.-H., Baker T. A Lightweight Attribute-Based Access Control Scheme for Intelligent Transportation System With Full Privacy Protection. IEEE Sensors Journal. 2020. Vol. 20 (14). Doi: 10.1109/JSEN.2020.3030688. URL: ieeexplore.ieee.org/document/9222175.

16. Jagatheesaperumal S.K., Bibri S.E., Huang J. et al. Artificial intelligence of things for smart cities: advanced solutions for enhancing transportation safety. Computational Urban Science. 2024. Vol. 4. №10. Doi: 10.1007/s43762-024-00120-6. URL: link.springer.com/article/10.1007/s43762-024-00120-6.

17. Hewage N., Meedeniya D. Machine Learning Operations: A Survey on MLOps Tool Support. Arxiv.2202.10169.2022. Pp. 1-12. Doi: 10.48550/arXiv.2202.10169.

18. Цай Ж., Цзинь Ц., Бобков А.В. Система многообъектной визуально-инерциальной одометрии беспилотного автомобиля // Инженерный вестник Дона. 2025. №9. URL: ivdon.ru/ru/magazine/archive/n9y2025/10387.

19. Gorodnichev M., Erokhin S., Polyantseva K., Moseva M. On the Problem of Restoring and Classifying a 3D Object in Creating a Simulator of a Realistic Urban Environment. Sensors. 2022. Vol. 22 (14): 5199. Doi: 10.3390/s22145199. URL: pubmed.ncbi.nlm.nih.gov/35890879/.

20. Gómez-Bombarelli R., Wei J. N., Duvenaud D., Hernández-Lobato J. M., Sánchez-Lengeling B., Sheberla D., Aguilera-Iparraguirre J., Hirzel T. D., Adams R. P., Aspuru-Guzik A. Automatic chemical design using a data-driven continuous representation of molecules. ACS Central Science. 2018. Vol. 4(2). Pp. 268-276. Doi: 10.1021/acscentsci.7b00572.

21. Singh S., Singh N. Containers & Docker: Emerging roles & future of Cloud technology. 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Bangalore, India, 2016. Pp. 804-807. Doi: 10.1109/ICATCCT.2016.7912109.

22. Sheluhin O. I., D. Kostin V., Gorodnichev M. G. Monitoring Anomalous States of Computer Systems by Intellectual Analysis of Data of System Journals. Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia. Institute of Electrical and Electronics Engineers Inc. 2020. Pp. 1-6. Doi: 10.1109/IEEECONF48371.2020.9078632.

23. Cai H., Chen T., Zhang W., Yu Y., Wang J. Efficient architecture search by network transformation. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018. Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32. Pp. 2787-2794. Doi: 10.1609/aaai.v32i1.11709.

24. Gorodnichev M., Moseva M. On the Problem of Developing a Fault-Tolerant High-Loaded Cluster of Support for an Intelligent Transportation System. Wave Electronics and its Application in Information and Telecommunication Systems (WECONF), St. Petersburg, Russian Federation, 2023. Pp. 1-6. Doi: 10.1109/WECONF57201.2023.10148007.

25. Tan M., Chen B., Pang R., Vasudevan V., Le Q. V. Mnasnet: Platform-aware neural architecture search for mobile. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019. Pp. 2815-2823. Doi: 10.1109/CVPR.2019.00293.

26. Zhong Z., Yan J., Wu W., Shao J., Liu C. -L. Practical Block-Wise Neural Network Architecture Generation. IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018. Pp. 2423-2432. Doi: 10.1109/CVPR.2018.00257.

27. Zoph B., Vasudevan V., Shlens J., Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018. Pp. 8697-8710. Doi: 10.1109/CVPR.2018.00907.

28. Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes. In IEEE Cloud Computing. 2014. Vol. 1, № 3, Pp. 81-84. Doi: 10.1109/MCC.2014.51.

References

1. Gorodnichev M., Nigmatulin A. Advances in Intelligent Systems and Computing. 2013. Vol 224. Pp. 195-204. Doi: 10.1007/978-3-319-00945-2_17.

2. Yuan T., Da Rocha Neto W., Esteve Rothenberg C., Obraczka K., Barakat C., Turletti T. Transactions on Emerging Telecommunications Technologies. 2022. Vol. 33. №4. Doi: 10.1002/ett.4427. URL: researchgate.net/publication/356613364_Machine_Learning_for_Next-Generation_Intelligent_Transportation_Systems_A_Survey.

3. Ouyang L., Wu J., Jiang X., Almeida D., Wainwright C. L., Mishkin P., Zhang C., Agarwal S., Slama K., Ray A., Schulman J., Hilton J., Kelton F., Miller L., Simens M., Askell A., Welinder P., Christiano P., Leike J., Lowe R. Advances in Neural Information Processing Systems (NeurIPS 2022). 2022. Vol. 36. Doi: 10.48550/arXiv.2203.02155. URL: arxiv.org/abs/2203.02155.

4. Skvortsova I.V., Nurulin Yu.R., Somov A.G. Inzhenernyj vestnik Dona. 2025. №10. URL: ivdon.ru/ru/magazine/archive/n10y2025/10417.

5. Florence S., C. Shyamala Kumari. International Journal of Innovative Technology and Exploring Engineering. 2019. Vol. 8. (9) Pp. 1230-1232. Doi: 10.35940/ijitee.I7597.078919.

6. Bulatova O. E3S Web of Conferences. International Scientific Conference “Fundamental and Applied Scientific Research in the Development of Agriculture in the Far East” (AFE-2022). Tashkent: EDP Sciences, 2023. Vol. 371. Doi: 371. 10.1051/e3sconf/202337106009. URL: doi.org/10.1051/e3sconf/202337106009.

7. Bhanu P. Reddy R. International Journal For Multidisciplinary Research. 2022. Vol. 4 (1). URL: ijfmr.com/research-paper.php?id=39278.

8. Oladimeji D., Gupta K., Kose N. A., Gundogan K. K., Ge L., Liang F. Sensors. 2023. Vol. 23 (8), 3880. Doi: 10.3390/s23083880. URL: pubmed.ncbi.nlm.nih.gov/37112221/.

9. Danasingh A.A., Tamizhpoonguil B., Jebamalar L. E. International Journal on Recent and Innovation Trends in Computing and Communication. 2016. Vol. 4, Pp. 273 - 277. Doi: 10.1109/WCCCT.2016.36.

10. Kune R., Pramod K., Arun A., Rao C., Rajkumar . Software: Practice and Experience. 2015, Vol. 46. Pp. 79 -105. Doi: 10.1002/spe.2374.

11. Azeem M., Bassam M.A., Priyadarshana D. Mesopotamian Journal of Big Data. 2023. Vol. 2023. Pp. 16-28. Doi: 10.58496/MJBD/2023/003.

12. Kekevi U., Aydin A. Computer Science. 2022. Vol. 7 (2). Pp. 111-123. Doi: 10.53070/bbd.1204112.

13. Al-Jaroodi J., Mohamed N. and Jiang H. Proceedings FifthIEEE Workshop on Mobile Computing Systems and Applications, Las Vegas, NV, USA, 2003. Pp. 572-579. Doi: 10.1109/IRI.2003.1251467.

14. Almeida F. Future Internet. 2023. Vol. 15 (9), 285. Doi: 10.3390/fi15090285. URL: mdpi.com/1999-5903/15/9/285.

15. Hui T., Li X., Quan H., Chang C.-H., Baker T. IEEE Sensors Journal. 2020. Vol. 20 (14). Doi: 10.1109/JSEN.2020.3030688. URL: ieeexplore.ieee.org/document/9222175.

16. Jagatheesaperumal S.K., Bibri S.E., Huang J. et al. Computational Urban Science. 2024. Vol. 4. №10. Doi: 10.1007/s43762-024-00120-6. URL: link.springer.com/article/10.1007/s43762-024-00120-6.

17. Hewage N., Meedeniya D. Arxiv.2202.10169.2022. Pp. 1-12. Doi: 10.48550/arXiv.2202.10169.

18. Czaj Zh., Czzin Cz., Bobkov A.V. Inzhenernyj vestnik Dona. 2025. №9. URL: ivdon.ru/ru/magazine/archive/n9y2025/10387/.

19. Gorodnichev M., Erokhin S., Polyantseva K., Moseva M. Sensors. 2022. Vol. 22 (14): 5199. Doi: 10.3390/s22145199. URL: pubmed.ncbi.nlm.nih.gov/35890879/.

20. Gómez-Bombarelli R., Wei J. N., Duvenaud D., Hernández-Lobato J. M., Sánchez-Lengeling B., Sheberla D., Aguilera-Iparraguirre J., Hirzel T. D., Adams R. P., Aspuru-Guzik A. ACS Central Science. 2018. Vol. 4(2). Pp. 268-276. Doi: 10.1021/acscentsci.7b00572.

21. Singh S., Singh N. 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Bangalore, India, 2016. Pp. 804-807. Doi: 10.1109/ICATCCT.2016.7912109.

22. Sheluhin O. I., D. Kostin V., Gorodnichev M. G. Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia. Institute of Electrical and Electronics Engineers Inc. 2020. Pp. 1-6. Doi: 10.1109/IEEECONF48371.2020.9078632.

23. Cai H., Chen T., Zhang W., Yu Y., Wang J. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018. Proceedings

of the AAAI Conference on Artificial Intelligence, Vol. 32. Pp. 2787-2794. Doi: 10.1609/aaai.v32i1.11709.

24. Gorodnichev M., Moseva M. Wave Electronics and its Application in Information and Telecommunication Systems (WECONF), St. Petersburg, Russian Federation, 2023. Pp. 1-6. Doi: 10.1109/WECONF57201.2023.10148007.

25. Tan M., Chen B., Pang R., Vasudevan V., Le Q. V. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019. Pp. 2815-2823. Doi: 10.1109/CVPR.2019.00293.

26. Zhong Z., Yan J., Wu W., Shao J., Liu C. -L. IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018. Pp. 2423-2432. Doi: 10.1109/CVPR.2018.00257.

27. Zoph B., Vasudevan V., Shlens J., Quoc V. Le. IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018. Pp. 8697-8710. Doi: 10.1109/CVPR.2018.00907.

28. Bernstein D. In IEEE Cloud Computing. 2014. Vol. 1, № 3, Pp. 81-84. Doi: 10.1109/MCC.2014.51.

Авторы согласны на обработку и хранение персональных данных.

Дата поступления: 16.09.2025

Дата публикации: 26.10.2025