СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные города сталкиваются с серьёзными вызовами в сфере управления дорожным движением. Постоянный рост количества транспортных средств и динамика перемещений граждан требуют более гибких и интеллектуальных подходов к регулированию транспортных потоков. Одним из наиболее критичных узлов в улично-дорожной сети остаются регулируемые перекрёстки, эффективность работы которых напрямую влияет на пропускную способность магистралей, уровень заторов и безопасность участников движения.

Особенно важным становится вопрос адаптации режимов светофорного управления к реальной интенсивности движения в течение суток и недели. Использование статичных режимов приводит к неоптимальному распределению времени фаз, излишним задержкам и повышенному потреблению энергии. В условиях ограниченных ресурсов появляется потребность во внедрении решений, способных повысить эффективность управления даже в оффлайн-режиме.

Развитие технологий обработки данных и элементов машинного обучения открывает возможности для создания интеллектуальных систем, способных анализировать статистику движения и предлагать обоснованные решения по оптимизации режимов работы светофоров.

Цель данного проекта – разработка интеллектуальной системы прогнозирования и планирования интенсивности движения на регулируемых перекрёстках. Система должна обеспечивать автоматизированный анализ исторических данных о транспортных потоках и формировать обоснованные рекомендации по управлению режимами светофорного регулирования.

Предлагаемая система нацелена на повышение точности в принятии решений и снижение энергопотребления за счёт более рационального распределения временных ресурсов светофорного оборудования, представляя собой шаг к созданию более устойчивой городской транспортной среды.

1 СИСТЕМНЫЙ АНАЛИЗ И ПОСТАНОВКА ЗАДАЧИ

1.1 Изучение предметной области

Организация дорожного движения на регулируемых перекрёстках является одной из ключевых задач городского транспортного планирования и непосредственно влияет на качество жизни горожан. Эффективность функционирования светофорных объектов определяет не только пропускную способность улично-дорожной сети, но и уровень загруженности магистралей, число задержек на маршрутах, а также безопасность всех участников движения, включая пешеходов, автомобилистов и пользователей общественного транспорта.

В современном городе с высокой плотностью движения управление транспортными потоками приобретает особую значимость. Неправильно настроенные режимы светофоров приводят к образованию заторов, неравномерной нагрузке на транспортные артерии и, как следствие, к дополнительным выбросам загрязняющих веществ в атмосферу. В этой связи особое внимание уделяется вопросу оптимального распределения времени сигналов светофора в зависимости от интенсивности движения в разные периоды суток.

На практике, особенно в условиях отсутствия адаптивных светофорных систем, режимы работы светофоров задаются вручную специалистами дорожно-эксплуатационных служб. Этот процесс требует анализа больших массивов исторических данных о трафике, таких как суточные и недельные графики интенсивности движения, и значительного опыта специалистов. Обычно такой анализ осуществляется вручную – сотрудники визуально просматривают графики, определяют пиковые и непиковые интервалы, а затем на основе собственного опыта составляют расписания смены фаз, длительности циклов и пропорций зелёного времени.

Однако такой экспертный подход обладает рядом существенных недостатков. Он подвержен субъективности и не всегда обеспечивает стабильную эффективность при изменяющихся транспортных условиях. Например, строительство новых жилых кварталов, ремонт дорог или изменение маршрутов общественного транспорта могут существенно

повлиять на паттерны движения, и ручная настройка может не успевать за этими изменениями. Кроме того, ручной анализ сложно масштабировать: при большом числе перекрёстков и постоянной потребности в пересмотре режимов нагрузка на специалистов становится чрезмерной.

Современные технологии анализа временных рядов, машинного обучения и визуализации данных позволяют частично или полностью автоматизировать процесс, повысив как скорость, так и объективность принимаемых решений. Особенно актуальной становится задача создания интеллектуальных систем прогнозирования и планирования интенсивности движения, которые могли бы на основе имеющихся данных выявлять закономерности трафика, формировать обоснованные рекомендации по смене режимов светофорного регулирования и, таким образом, обеспечивать эффективную работу перекрёстков даже в условиях отсутствия онлайн-адаптации.

Такие системы позволяют перейти от ручного экспертного подхода к полуавтоматизированной поддержке принятия решений, что особенно ценно в условиях ограниченного ресурса и высокой динамичности городской среды. В перспективе они могут служить промежуточным звеном между традиционным и полностью адаптивным управлением движением.

1.2 Описание интеллектуальной системы

Интеллектуальная система прогнозирования и планирования интенсивности движения предназначена для автоматизации анализа исторических данных о транспортных потоках на регулируемых перекрёстках и формирования обоснованных рекомендаций по изменению режимов работы светофоров. Её основная функция заключается в автоматическом выделении характерных временных интервалов в течение суток или недели, в которых транспортные потоки демонстрируют существенно различающиеся характеристики. Это позволяет определять, в какие периоды необходимо использовать различные фазы и циклы светофорного регулирования, чтобы обеспечить максимальную эффективность работы перекрёстка.

Система представляет собой программное приложение с пользовательским интерфейсом, в которое инженер по организации движения может загрузить входные данные – например, статистику интенсивности по направлениям за определённый временной период (один день, рабочая неделя, совокупность аналогичных дней и т. д.). Система автоматически приводит эти данные к единому формату, обрабатывает их, анализирует структуру изменения потока и выявляет временные интервалы с различной степенью загруженности.

На выходе пользователь получает карту времени – визуализацию дня, разбитую на интервалы, для каждого из которых определены режимы светофорного управления, отражающие реальную транспортную нагрузку. Кроме того, система предоставляет сводные статистические показатели по каждому интервалу, которые можно использовать для составления проектной документации, утверждения режимов или мониторинга эффективности принятых решений. Система также поддерживает экспорт результатов и формирование отчётных материалов.

Таким образом, интеллектуальная система выступает как инструмент поддержки принятия решений в сфере организации дорожного движения. Её применение позволяет сократить зависимость от субъективных оценок, повысить качество планирования, а также масштабировать процессы анализа на большое количество объектов. Система ориентирована на оффлайн-режим прогнозирования и наиболее актуальна для использования в городах и регионах, где внедрение адаптивных светофорных систем ещё не произведено. Она может стать важным промежуточным звеном на пути к внедрению полноценных интеллектуальных транспортных систем, повышающих эффективность городской мобильности.

1.3 Обзор существующих методов

В области прогнозирования и управления дорожным движением существует множество решений, ориентированных как на работу в реальном времени, так и на постобработку исторических данных. В данной работе основной акцент сделан на анализ

исторической информации об интенсивности движения с целью последующего планирования режимов работы светофорных объектов.

Среди существующих подходов к анализу временных последовательностей интенсивности движения можно выделить две ключевые группы методов:

1. Методы кластеризации, которые позволяют объединять схожие по характеристикам участки временного ряда, выявляя повторяющиеся паттерны в трафике (утренние и вечерние часы пик, ночное снижение интенсивности и т.д.) [1]
2. Методы обнаружения точек изменений (change point detection), направленные на автоматическое выявление моментов, в которых происходит смена статистических свойств потока, таких как среднее значение или дисперсия [2]. Это позволяет делить день на интервалы с различными режимами движения без предварительного знания их структуры.

Такие методы позволяют формировать гибкие и адаптивные схемы управления движением, опираясь исключительно на накопленные эмпирические данные. Это особенно актуально в условиях, где динамическое управление в реальном времени затруднено или невозможно.

Существует также отдельный класс решений, основанных на адаптивных алгоритмах управления дорожным движением, таких как SCOOT (Split Cycle Offset Optimization Technique) и SCATS (Sydney Coordinated Adaptive Traffic System). Эти интеллектуальные транспортные системы способны в реальном времени регулировать работу светофоров на основе текущих условий на дорогах.

SCOOT, разработанная в Великобритании, функционирует на основе данных, получаемых от детекторов транспортных средств, установленных на подходах к регулируемым перекрёсткам на расстоянии примерно 40-50 метров от стоп-линии [3]. Детекторы фиксируют прохождение автомобилей и передают информацию о количестве и скорости транспортных средств в центральную систему управления.

Алгоритм SCOOT работает по принципу непрерывной оптимизации трёх ключевых параметров светофорного регулирования. Во-первых, система корректирует длительность зелёных фаз для различных направлений движения, анализируя текущую загрузку

каждого подхода к перекрёстку и перераспределяя время между фазами в зависимости от реальной потребности. Во-вторых, происходит оптимизация общего цикла светофора – система может увеличивать или уменьшать продолжительность полного цикла в зависимости от общей интенсивности движения. В-третьих, осуществляется корректировка временных смещений между соседними перекрёстками для создания «зелёной волны» и обеспечения координированного движения транспортных потоков.

SCOOT использует концепцию «степени насыщения» для каждого подхода к перекрёстку, которая рассчитывается как отношение фактического потока к пропускной способности. На основе этих данных система каждые несколько секунд принимает решения о необходимости корректировки параметров регулирования. Уникальной особенностью SCOOT является её способность объединять группы перекрёстков в регионы и оптимизировать их работу совместно, что позволяет учитывать взаимосвязь между соседними участками сети и минимизировать время задержек на всём маршруте.

Практическая эффективность системы подтверждена многочисленными внедрениями: в Лондоне SCOOT управляет более 6 000 светофоров, что позволило сократить задержки на 13%, а в Сиэтле внедрение системы на 32 перекрёстках вдоль улицы Мерсер привело к снижению задержек и улучшению надёжности движения в часы пик [4,5].

SCATS, разработанная в Австралии, представляет собой более сложную иерархическую систему управления, построенную по принципу распределённой архитектуры [6]. На нижнем уровне располагаются локальные контроллеры светофорных объектов, оснащённые собственными процессорами и способные функционировать автономно при потере связи с центром. Каждый контроллер анализирует данные от детекторов транспорта, установленных на всех подхода к перекрёстку, и формирует локальные стратегии управления.

Центральный компьютер системы SCATS выполняет роль координатора, получая информацию от всех подключённых перекрёстков и принимая решения о необходимости объединения или разделения светофорных объектов в координированные группы. Система способна динамически формировать «подсистемы» – группы перекрёстков, работающих синхронно для обеспечения непрерывного движения транспортных потоков. Размер

и состав таких подсистем может изменяться в течение дня в зависимости от направления и интенсивности основных транспортных потоков.

Особенностью SCATS является использование концепции «степени насыщения» для каждого подхода, которая рассчитывается на основе данных о количестве прибывающих и обслуженных транспортных средств. Когда степень насыщения превышает определённые пороговые значения, система автоматически корректирует длительность зелёных фаз или общий цикл светофора. Для обеспечения приоритета общественному транспорту SCATS использует специальные детекторы, которые распознают приближение автобусов или трамваев и могут продлевать зелёную фазу или досрочно активировать её.

Дополнительным преимуществом SCATS является возможность интеграции с другими транспортными системами, включая управление въездами на автомагистрали, системы информирования участников движения и центры управления городским транспортом. Это позволяет создавать комплексные решения для управления транспортными потоками на уровне всего города или региона.

SCATS широко применяется в Австралии, Новой Зеландии, Китае, Польше и других странах [7]. В Польше, например, система внедрена в 10 городах и управляет около 600 перекрёстками, демонстрируя свою адаптивность к различным условиям дорожной инфраструктуры [8]. В Дублине SCATS используется для адаптивного управления движением в центре города, позволяя оперативно реагировать на изменения в трафике и обеспечивать приоритет для общественного транспорта [9].

Однако внедрение таких адаптивных алгоритмов в широком масштабе сопряжено с рядом существенных ограничений. Прежде всего, это необходимость значительных финансовых вложений на установку детекторов, модернизацию светофорных контроллеров и создание централизованной инфраструктуры управления. Кроме того, эксплуатация подобных систем требует высокой технической подготовки персонала и сложного технического сопровождения, а их эффективность критически зависит от качества и стабильности поступающих данных от различных датчиков и устройств.

В текущих условиях большинство городов и населённых пунктов региона не располагают необходимой инфраструктурой для реализации адаптивного управления в

реальном времени. На территории Беларуси отсутствует широкомасштабное внедрение подобных систем – большинство светофорных объектов функционируют по фиксированным расписаниям, заданным вручную, без возможности динамической адаптации к текущим условиям на дорогах. Лишь в рамках отдельных проектов трансграничного сотрудничества, например в городе Брест, были реконструированы семь перекрёстков с установкой элементов адаптивной системы управления движением.

Учитывая изложенные ограничения, подход на основе анализа исторических данных представляется наиболее целесообразным и применимым в существующих условиях. Он позволяет повысить эффективность организации движения без необходимости радикальной перестройки инфраструктуры и значительных капитальных вложений.

В связи с этим далее будут рассмотрены два класса методов, наиболее подходящих для решения поставленной задачи: методы кластеризации и алгоритмы обнаружения точек изменений. Эти подходы позволяют автоматически структурировать временные ряды интенсивности движения и формировать интервалы для назначения различных режимов светофорного регулирования на основе накопленных эмпирических данных.

1.3.1 Методы кластеризации

Кластеризационные методы представляют собой класс алгоритмов, применяемых для группировки объектов на основе их сходства. В контексте анализа временных рядов интенсивности движения они позволяют автоматически выявлять участки с однородными характеристиками потока – например, часы пик, периоды низкой загрузки или нестандартные аномальные интервалы.

Ключевая идея кластеризации заключается в том, чтобы разделить данные на непересекающиеся подмножества (кластеры или сегменты), внутри которых объекты (вектора интенсивностей по каждому из направлений) схожи между собой по выбранным признакам, а между кластерами – различаются.

Ниже перечислены и кратко охарактеризованы основные подходы к кластеризации, применимые для анализа временных рядов интенсивности движения:

Кластеризация на основе экспертных правил – детерминированный метод группировки объектов по заранее сформулированным логическим или математическим условиям. Эти условия разрабатываются экспертами и отражают известные закономерности в данных. Метод не требует обучения и обеспечивает высокую интерпретируемость результатов. Основные недостатки: необходимость ручной разработки правил для каждого случая, сложность масштабирования при увеличении числа признаков, потребность в переформулировке условий при изменении количества кластеров.

Пороговая кластеризация – разновидность детерминированной кластеризации, где объекты разделяются на основе сравнения признаков с пороговыми значениями. Каждый кластер определяется диапазоном значений признаков. Подход прост в реализации и хорошо подходит для систем с чёткими граничными условиями. Недостатки: чувствительность к выбросам и шуму, особенно вблизи пороговых границ. Выбор оптимальных порогов обычно осуществляется вручную, что снижает обобщающую способность. Плохо справляется с высокоразмерными и слабо разделимыми данными.

Алгоритм k-means является одним из наиболее используемых методов кластеризации благодаря простоте и эффективности [10]. Он минимизирует внутрикластерную дисперсию, итеративно распределяя объекты по k кластерам. Преимущества: быстрая сходимость, масштабируемость и лёгкость реализации. Ограничения: предполагает сферическую форму кластеров схожего размера, чувствителен к выбросам. На практике может не обнаружить заданное число кластеров, особенно при нечётко разделённых данных.

DBSCAN – метод кластеризации, основанный на анализе плотности данных [11]. Объединяет объекты в кластеры, если они образуют плотную область в пространстве признаков. Требует два параметра: ε (радиус окрестности) и $MinPts$ (минимальное количество точек). Преимущества: выявляет кластеры произвольной формы, устойчив к выбросам, автоматически определяет количество кластеров. Недостатки: не предоставляет механизма для получения строго заданного числа кластеров. Плохо работает при равномерном или чрезмерно плотном распределении данных, что делает его неприменимым в задачах с высокой плотностью и временной структурой.

GMM предполагает, что данные сгенерированы из комбинации нескольких нормальных распределений [12]. Использует EM-алгоритм для оценки параметров и формирует вероятностное разбиение данных. Позволяет моделировать эллипсоидные формы кластеров и хорошо работает с перекрывающимися группами. Ограничения: требует нормального распределения данных. В задачах с дискретными, асимметричными данными, близкими к распределению Пуассона, использование GMM некорректно.

PMM – вероятностная модель для данных, подчиняющихся распределению Пуассона [13]. Хорошо подходит для счётных величин и частотных признаков. Использует EM-алгоритм и обеспечивает высокую интерпретируемость параметров в терминах интенсивности пуассоновских процессов. Ограничения: предполагает именно пуассоновское распределение признаков, чувствителен к мультиколлинеарности и выбросам. Несмотря на это, для задач со счётными значениями представляет более адекватный подход, чем гауссовская альтернатива.

Иерархический метод, строящий кластеры снизу вверх: каждый объект изначально – отдельный кластер, затем кластеры последовательно объединяются. Метод Уорда, основанный на минимизации приращения внутрикластерной дисперсии, обеспечивает сбалансированное разбиение и подходит для структур с компактностью и раздельностью кластеров [14].

FINCH – агломеративный алгоритм, основанный на связях первого ближайшего соседа [15]. Не требует настройки гиперпараметров, но позволяет контролировать число кластеров через остановку процесса на нужной итерации. Недостатки: не оптимизирует расстояние между кластерами, что может приводить к слабо различимым группам. Чувствителен к локальным структурам и выбросам.

TW-FINCH – модификация FINCH для временных данных с введением временных весов [16]. Сохраняет достоинства базовой версии: не требует задания параметров, но позволяет получить заданное число кластеров. Недостатки: как и FINCH, не максимизирует расстояние между кластерами. Качество зависит от корректного задания временных весов.

Несмотря на разнообразие методов и их способность выявлять кластеры с высокой межкластерной разницей и низкой внутрикластерной дисперсией, включая особенно эффективный в условиях временной структуры алгоритм TW-FINCH, большинство описанных подходов имеют один общий и серьёзный недостаток. Они не предусматривают прямого механизма ограничения размера кластеров, особенно по временной оси. В задачах, где необходимо контролировать длительность сегментов или обеспечить равномерное распределение объектов между кластерами, такие алгоритмы оказываются либо неприменимыми, либо требуют сложных обходных решений. Более того, многие из них не позволяют явно задать желаемое число кластеров, что критично в прикладных сценариях с фиксированными требованиями к выходной структуре. В этом контексте методы кластеризации оказываются ограниченно пригодными для задач планирования и прогнозирования интенсивности движения, где важно не только выявление структур, но и соблюдение чётких ограничений по длительности и количеству временных интервалов.

1.3.2 Методы обнаружения точек изменений

Методы обнаружения точек изменений представляют собой класс алгоритмов, предназначенных для автоматического выявления моментов времени, в которых происходят существенные изменения статистических свойств временного ряда. В задаче анализа интенсивности движения такие изменения могут соответствовать переходам между различными режимами трафика – например, началом утреннего часа пик, снижением нагрузки в обеденный период или резким спадом движения в ночное время.

Основная идея данных методов заключается в том, чтобы разложить временной ряд на последовательность сегментов, внутри которых характеристики потока (такие как среднее значение, дисперсия, частотные свойства и т. д.) сохраняются относительно стабильными, а между сегментами происходят статистически значимые сдвиги. Это позволяет более точно выделять границы между интервалами, требующими различных подходов к управлению движением.

Подобные алгоритмы особенно полезны в ситуациях, когда структура временного ряда заранее неизвестна, и необходим автоматический способ сегментации на основе данных. Они хорошо сочетаются с последующим анализом, включая планирование режимов светофорного регулирования.

В этот обзор включены также подходы, которые не относятся строго к классическим методам обнаружения точек изменений, такие как ClaSP и Ptr-Net. Несмотря на это, их механизмы направлены на автоматическое разбиение временного ряда на однородные участки, что делает их функционально сопоставимыми с методами сегментации. Эти алгоритмы широко используются на практике для идентификации границ между различными режимами в данных и поэтому уместны в контексте поставленной задачи.

Ниже перечислены и кратко охарактеризованы основные подходы, применимые для анализа временных рядов интенсивности движения:

ClaSP – современный метод сегментации временных рядов, основанный на самообучении [17]. Не требует предварительной разметки данных и способен выявлять точки изменений без задания гиперпараметров. Иерархически разделяет временной ряд, обучая бинарный классификатор для каждой возможной точки разделения. Показал превосходство по точности над другими методами на 107 датасетах [18]. Ограничения: не предоставляет прямого механизма управления количеством сегментов, ориентирован на резкие изменения и плохо работает с плавными переходами. Чувствителен к шуму и неэффективен на коротких временных рядах из-за недостатка данных для обучения классификаторов.

Ptr-Net – нейросетевая архитектура для задач с переменным размером выходных данных [19]. Использует механизм внимания для указания на элементы входной последовательности, формируя выход из индексов входных данных. Эффективна для задач выбора позиций в последовательности, обрабатывает входы переменной длины. Недостатки: ориентирована на точное определение позиций, что неэффективно при плавных переходах. Чувствительна к шуму и требует большого объёма размеченных данных для обучения.

KernelCPD – метод обнаружения точек перемены, основанный на ядровых функциях [20]. Сопоставляет сегменты в преобразованном пространстве признаков, выявляя

сложные нелинейные изменения структуры данных. Использование ядер позволяет обнаруживать переходы, незаметные при линейной обработке. Ограничения: высокая вычислительная затратность, требует настройки параметров ядра. Неэффективен на коротких временных рядах – может не обнаружить изменения или зафиксировать их в произвольных местах.

Binseg – простой метод обнаружения точек перемены, реализующий жадную стратегию [21]. На каждой итерации ищет наиболее выраженную точку перемены, разделяет последовательность и рекурсивно применяется к частям. Обеспечивает высокую вычислительную эффективность и широко используется для предварительной разметки. Недостатки: жадная природа делает его чувствительным к ошибкам на ранних этапах. Не оптимизирует глобальную функцию стоимости, может формировать сегменты одинаковой длины, не адаптируясь к реальной структуре данных.

Bottom-Up – иерархический метод, реализующий подход снизу вверх [22]. Изначально разбивает ряд на мелкие равномерные сегменты, затем итеративно объединяет соседние на основе метрики схожести. Гибок, выявляет как крупные, так и мелкие изменения, не требует предварительного задания числа сегментов. Ограничения: чувствителен к выбору начального размера сегмента. Локальный критерий объединения может не соответствовать глобально оптимальному разбиению. Подвержен ошибкам при наличии шума.

WSS – алгоритм на основе скользящего окна фиксированной длины [23]. Сравнивает статистические характеристики соседних окон, фиксируя точку перемены при превышении порога. Обладает низкой вычислительной сложностью, может работать в реальном времени. Недостатки: чувствительность зависит от размера окна. Неэффективен при плавных переходах, требует ручной настройки параметров, не адаптируется к структуре данных.

PELT – популярный алгоритм с глобальной оптимизацией при линейной временной сложности [24]. Использует динамическое программирование с эвристикой «обрезки», работает с различными функциями стоимости. Автоматически определяет количество точек перемены через минимизацию функции с штрафом. Недостатки: качество сегмен-

тации чувствительно к выбору штрафного параметра. При шуме или медленных переходах может давать нестабильные результаты.

Алгоритм динамического программирования реализует подход глобальной оптимизации, перебирая все возможные разбиения с заданным количеством изменений и выбирая оптимальное по функции стоимости [25]. В отличие от жадных методов, не страдает от ошибок локальной оптимизации и гарантирует нахождение глобально оптимального разбиения при заданном числе сегментов. Гибко адаптируется к различным функциям потерь и обеспечивает наиболее точные границы между сегментами. Особенно эффективен в задачах структурного анализа данных, где критична точность сегментации – при выявлении смены режимов или фаз в системах. Детерминированность метода делает результаты воспроизводимыми и предсказуемыми, что важно для практических приложений. Однако метод имеет существенные ограничения по гибкости применения. Требует явной рекурсивной формулировки задачи, что делает его неприменимым при сложных, недифференцируемых функциях стоимости. Добавление нестандартных ограничений (минимальная длина сегмента, исключение разбиения в определённых интервалах, семантические условия) часто нарушает рекурсивную структуру. Слабо приспособлен к многокритериальной оптимизации – в отличие от эвристических методов, не позволяет легко учитывать несколько факторов качества одновременно. Наиболее эффективен в чётко формализованных задачах с простой функцией потерь, теряя универсальность в сложных сценариях анализа.

Таким образом, несмотря на разнообразие подходов и теоретическую привлекательность ряда алгоритмов, большинство методов обнаружения точек изменений сталкиваются с теми же ограничениями, что и методы кластеризации. В частности, они не обеспечивают прямого управления длиной и числом сегментов, что критически важно в прикладных задачах планирования, где требуется предсказуемая и управляемая структура выходных данных. Более того, в отличие от кластеризации, эти методы часто не гарантируют высокой межкластерной разницы и низкой внутрикластерной дисперсии: сегменты могут отличаться по длине, статистическим характеристикам и не иметь чёткой интерпретации, особенно в условиях шума и плавных переходов между режимами.

Наиболее близким к практическим требованиям алгоритмом является метод динамического программирования, поскольку он позволяет задать фиксированное количество сегментов и обеспечивает глобально оптимальное разбиение по функции стоимости. Однако он также не решает задачу в полном объёме, поскольку не позволяет одновременно учитывать все необходимые требования: максимизацию различий между сегментами, минимизацию внутрисегментной дисперсии, а также наличие жёстких ограничений как на количество, так и на длину сегментов.

1.4 Обоснование необходимости создания интеллектуальной системы

Анализ текущей ситуации в области организации дорожного движения на регулируемых перекрёстках выявляет ряд существенных проблем, которые обосновывают необходимость разработки специализированной интеллектуальной системы:

1. Высокая степень субъективности при разработке режимов работы светофоров – процесс определения временных интервалов осуществляется преимущественно на основе экспертных оценок и личного опыта специалистов, что приводит к непоследовательности решений
2. Значительные временные затраты на анализ данных – ручная обработка больших объёмов информации об интенсивности движения требует существенных трудозатрат инженеров дорожных служб
3. Ограниченная масштабируемость существующих подходов – по мере роста числа перекрёстков традиционные методы ручного планирования становятся неэффективными
4. Недостаточная оперативность реакции на изменения городской инфраструктуры, сезонные колебания интенсивности и другие факторы, требующие регулярного пересмотра режимов работы светофоров
5. Отсутствие единой методологии – различные специалисты используют разные критерии и подходы к сегментации данных о трафике, что затрудняет оценку эффективности принимаемых решений

6. Сложность учёта долгосрочных тенденций – человеческий анализ затрудняется при необходимости выявления долгосрочных трендов и сезонных паттернов
7. Неадаптированность существующих алгоритмов к специфике транспортных данных – большинство методов кластеризации и обнаружения точек изменений не предоставляют механизма для прямого задания желаемого числа временных интервалов и их минимальной/максимальной длительности, а также недостаточно точны.

Необходимость создания специализированной интеллектуальной системы продиктована потребностью в инструменте, способном автоматизировать процесс определения оптимальных временных интервалов для применения различных режимов управления движением. Такая система должна учитывать технические ограничения, включая минимально допустимую длительность фаз светофорного цикла, обеспечивать существенные различия между соседними интервалами для обоснованности переходов между режимами, а также минимизировать внутреннюю вариативность данных в пределах одного интервала для повышения стабильности работы светофорных объектов. Важными характеристиками являются высокая интерпретируемость результатов для специалистов, устойчивость к шумам и аномалиям, характерным для данных транспортных детекторов, а также способность к масштабированию, позволяющая эффективно обрабатывать информацию для большого количества перекрёстков и различных временных периодов.

Внедрение интеллектуальной системы также обеспечит:

– экономическую эффективность – оптимизация работы светофоров даст значительную экономию за счёт сокращения заторов и снижения расхода топлива;
– снижение экологической нагрузки – сокращение простоев транспорта приведёт к уменьшению выбросов выхлопных газов;
– повышение безопасности движения – оптимальные режимы работы светофоров снизят вероятность аварийных ситуаций.

1.5 Постановка задачи

Целью данного проекта является разработка интеллектуальной системы прогнозирования и планирования интенсивности движения, предназначенной для поддержки принятия решений при организации светофорного регулирования на городских перекрёстках. Система должна обеспечивать автоматическое выявление характерных закономерностей транспортных потоков и формирование обоснованных рекомендаций по управлению движением на основе анализа и прогнозов временных рядов.

Основная задача системы заключается в объективной оценке текущих и будущих уровней транспортной нагрузки и в определении оптимальных временных интервалов, в течение которых следует применять различные режимы светофорного регулирования. Система должна учитывать реальное распределение интенсивности движения в разные периоды суток и недели, включая особенности будних, выходных и праздничных дней, а также автоматически выявлять временные интервалы, в течение которых целесообразно применять различные режимы светофорного регулирования, исходя из характерных патернов движения.

Разрабатываемая система должна представлять собой приложение для автоматизированного анализа транспортных данных, в которое пользователь загружает информацию об интенсивности движения (например, в формате CSV или PDF), после чего система обрабатывает эти данные, выявляет характерные временные интервалы с разной транспортной нагрузкой и формирует карты времени с рекомендациями по режимам светофорного регулирования. В результате пользователь получает наглядную визуализацию суточной и недельной динамики движения, статистику по каждому интервалу, а также обоснованные предложения по переключению режимов светофора в конкретные временнные промежутки.

Интеллектуальная система прогнозирования и планирования интенсивности движения должна обеспечивать:

1. поддержка загрузки и валидации входных данных в форматах CSV и PDF;

2. преобразование и унификация данных, приведение их к единой структуре временных рядов;
3. реализацию механизмов фильтрации, агрегации и заполнения пропусков;
4. автоматический анализ интенсивности движения по направлениям на перекрестке;
5. выделение оптимальных временных интервалов для режимов светофорного регулирования;
6. расчёт статистических показателей для каждого временного сегмента и их вывод;
7. визуализацию карт времени и суточной интенсивности;
8. реализацию серверной части с REST API для взаимодействия с пользовательским интерфейсом;
9. пользовательский интерфейс для загрузки данных, контроля статуса и экспорта результатов.

Разработка интеллектуальной системы прогнозирования и планирования интенсивности движения позволит создать надёжный инструмент для автоматизированного анализа транспортных потоков и формирования обоснованных рекомендаций по управлению светофорным регулированием. Учитывая временные закономерности и реальные характеристики движения, такая система обеспечит высокую точность планирования, адаптацию к изменениям в трафике и снижение трудозатрат специалистов. Таким образом, интеллектуальная система прогнозирования и планирования интенсивности движения станет важным звеном в повышении эффективности и устойчивости управления городским транспортом.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1 Архитектура системы

Интеллектуальная система прогнозирования и планирования интенсивности движения спроектирована с использованием модульной архитектуры, что обеспечивает гибкость, масштабируемость и возможность независимой разработки и тестирования компонентов. Система состоит из нескольких взаимосвязанных модулей, каждый из которых выполняет строго определённую функцию.

Схема взаимодействия модулей в системе представлена на рисунке 2.1.

Рисунок 2.1 – Схема взаимодействия модулей в системе

Ниже приведено описание ключевых модулей, входящих в состав системы:

1. Модуль графического интерфейса: обеспечивает взаимодействие пользователя с системой. Отвечает за загрузку исходных данных, настройку параметров анализа, визуализацию процесса обработки и отображение конечных результатов в удобном и наглядном виде
2. Серверная часть системы: выполняет функции координации и обработки данных. В её состав входят следующие функциональные модули:

– Модуль обработки входных данных: реализует функции парсинга и предобработки информации, поступающей от пользователя. Поддерживает загрузку данных в различных форматах (например, CSV, JSON), выполняет их очистку, нормализацию, временное упорядочивание и приведение к унифицированной структуре временных рядов, пригодных для последующего анализа
– Модуль анализа трафика: содержит основную вычислительную логику системы. Выполняет сегментацию временных рядов интенсивности движения, идентификацию устойчивых паттернов, определение типовых временных зон, а также формирует рекомендации по оптимизации режимов светофорного регулирования на основе выявленных закономерностей
– Модуль формирования отчётных данных: отвечает за визуализацию и представление результатов анализа. Формирует график интенсивности движения с разбивкой на сегменты, а также таблицу отчёта, включающую ключевые метрики и параметры, выявленные в процессе анализа. Обеспечивает подготовку структурированных данных для вывода в интерфейсе и последующего экспорта.

Взаимодействие между модулями организовано следующим образом: пользователь через графический интерфейс инициирует необходимые процессы, направляя соответствующие запросы на сервер. Серверная часть обрабатывает запросы, передаёт данные в модуль обработки входных данных, затем в модуль анализа трафика, а полученные результаты направляет в модуль формирования отчётных данных. После этого готовая информация возвращается в графический интерфейс для визуализации и экспорта.

Такое распределение обязанностей между модулями способствует высокой устойчивости системы, облегчает масштабирование и обновление отдельных компонентов, а также обеспечивает прозрачность выполнения всех этапов анализа трафика.

2.2 Проектирование модуля графического интерфейса

Пользовательский интерфейс системы может быть реализован с использованием различных платформ: десктопных приложений, мобильных приложений и веб-страниц.

Наиболее целесообразным решением является веб-интерфейс, поскольку он обеспечивает платформонезависимость, универсальность запуска через любой современный браузер и не требует установки дополнительного программного обеспечения. Такой подход упрощает доступ пользователей к системе, расширяя охват и облегчая распространение. В основе создания веб-интерфейсов лежат три ключевых технологии: HTML, CSS и JavaScript.

HTML, или язык гипертекстовой разметки, выполняет роль каркаса веб-страницы, определяя её структуру [26]. С его помощью указываются местоположения элементов интерфейса – текстов, заголовков, изображений, интерактивных компонентов и других визуальных блоков.

CSS, или каскадные таблицы стилей, отвечает за оформление страницы. С его помощью задаются шрифты, цвета, размеры, отступы, выравнивание и другие параметры отображения элементов [27]. Разделение структуры (HTML) и стиля (CSS) позволяет упростить модификацию внешнего вида и поддержание кода в актуальном состоянии. Иерархическая система стилей делает их применение гибким и масштабируемым как к отдельным элементам, так и к их группам.

JavaScript – это язык программирования, обеспечивающий интерактивность интерфейса [28]. Он позволяет реагировать на действия пользователя, изменять содержимое страницы без её перезагрузки, обмениваться данными с сервером и реализовывать визуальные эффекты и анимации. Благодаря своей универсальности и поддержке различных парадигм программирования, JavaScript широко используется как на клиентской, так и на серверной стороне, включая использование через платформу Node.js.

Для реализации графического интерфейса было принято решение отказаться от использования фреймворков, таких как React, Vue.js, Angular или Next.js. Это обусловлено простотой задачи: интерфейс предназначен для загрузки и обработки данных, и использование чистого HTML, CSS и JavaScript полностью соответствует требованиям. Такой подход снижает сложность архитектуры, исключает зависимость от сторонних библиотек, упрощает сопровождение проекта и обеспечивает максимальную гибкость и совместимость.

Организация пользовательского взаимодействия должна строиться как последовательность логически связанных этапов. Каждый следующий шаг должен становиться доступным только после корректного завершения предыдущего, что позволяет пользователю лучше контролировать процесс и снижает вероятность ошибок [29]. Визуальное разделение шагов и наличие понятных заголовков обеспечивают лёгкую навигацию. Необходима также мгновенная и ясная обратная связь: при выборе файла отображается его имя и статус загрузки, а при запуске обработки – ход выполнения [30]. При этом важно предусмотреть возможность возврата к предыдущим шагам без потери данных, что обеспечивает гибкость интерфейса и устойчивость к ошибкам со стороны пользователя.

Визуальное оформление интерфейса должно быть ориентировано на простоту и ясность. Цветовая схема рекомендуется нейтральная, с минимальным количеством акцентов, применяемых только к элементам взаимодействия. Элементы управления должны быть стандартными и визуально предсказуемыми – например, закруглённые блоки с иконками и адекватными отступами способствуют быстрому пониманию их назначения. Чтобы не перегружать новичков, расширенные параметры – такие как количество частей для разбиения или порог слияния – должны быть скрыты по умолчанию и открываться по запросу пользователя. Такой подход позволяет сохранить интерфейс простым для одних и гибким для других [31].

Интерфейс необходимо адаптировать под различные устройства. Вертикальная компоновка блоков и масштабируемость элементов управления позволяют корректно отображать интерфейс как на настольных компьютерах, так и на мобильных устройствах, что расширяет возможности применения системы в различных условиях [32].

Для наглядного представления логики работы системы были разработаны макеты каждого из четырёх этапов взаимодействия. На рисунке 2.2 показан первый шаг – загрузка файла пользователем. Рисунок 2.3 иллюстрирует отображение загруженных данных и их предварительный просмотр. Рисунок 2.4 демонстрирует настройку параметров обработки, включая доступ к расширенным опциям. На рисунке 2.5 представлен итоговый этап – отображение результатов анализа и отчет, содержащий статистические данные каждого из

сегментов. Эти макеты обеспечивают дополнительную ясность при разработке и реализации интерфейса.

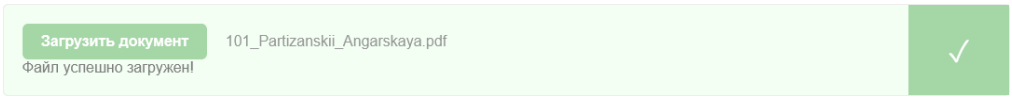

Рисунок 2.2 – Макет первого шага

Рисунок 2.3 – Макет второго шага

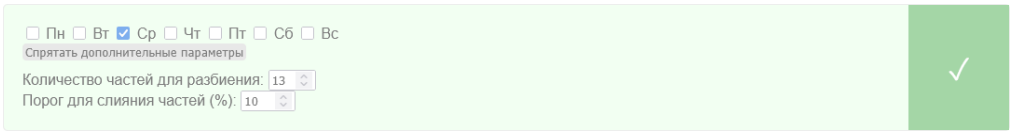

Рисунок 2.4 – Макет третьего шага

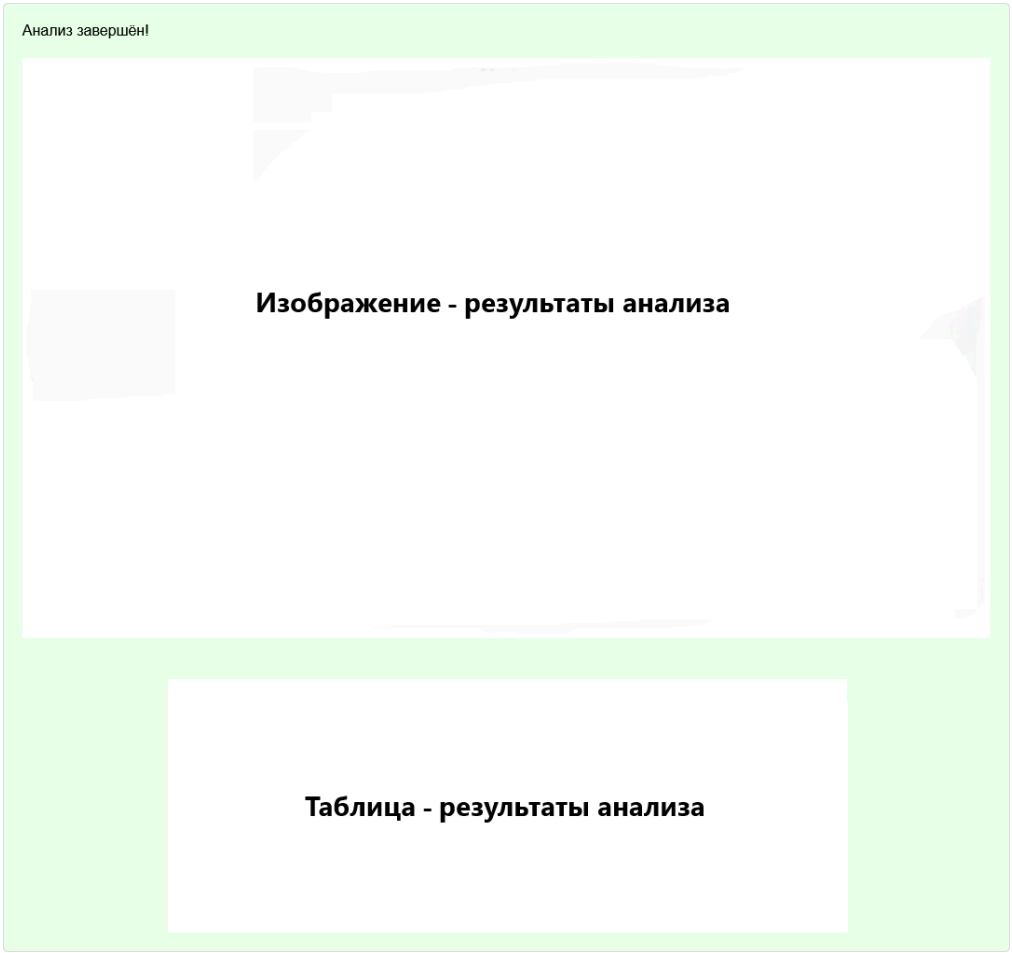

Рисунок 2.5 – Макет четвертого шага

Таким образом, реализация пользовательского интерфейса с опорой на простые и проверенные технологии позволяет достичь высокой функциональности и доступности

без лишней сложности. Чёткая структура, минималистичный дизайн, предсказуемое поведение и адаптивность под разные устройства делают интерфейс интуитивно понятным, надёжным и удобным как для начинающих, так и для продвинутых пользователей.

2.3 Проектирование серверной части

Серверный модуль играет центральную роль в функционировании системы, обеспечивая взаимодействие между пользовательским интерфейсом и другими компонентами. Без него пользователь не смог бы отправлять запросы и получать результаты обработки данных. Серверный модуль выступает в качестве связующего звена, принимая запросы от клиентской части и передавая их на обработку в другие модули, а затем возвращая результаты обратно пользователю.

Для обеспечения взаимодействия с другими модулями серверный модуль предоставляет API (программный интерфейс приложения). API определяет набор данных и доступных функций, используя которые другие модули могут обращаться к серверному модулю с конкретными запросами и получать структурированные ответы в определенном формате. Это позволяет различным частям системы взаимодействовать друг с другом независимо от их внутренней реализации.

Для разработки API могут использоваться различные языки программирования и специализированные библиотеки. Язык Python был выбран в качестве основного инструмента благодаря своей простоте, читаемости и поддержке различных парадигм программирования [33]. Python также обладает обширной экосистемой библиотек для решения широкого круга задач, включая разработку веб-приложений и API. Среди популярных библиотек для создания API на Python можно выделить FastAPI, Flask и Django.

Для реализации API нашего серверного модуля была выбрана библиотека FastAPI [34]. FastAPI представляет собой современный и высокопроизводительный фреймворк для создания API на Python, основанный на стандартах OpenAPI и JSON Schema. Выбор FastAPI обусловлен его скоростью работы, удобством использования, встроенной валидацией данных и автоматической документацией, что делает процесс разработки и

поддержки API более эффективным. Одним из ключевых преимуществ FastAPI является его автоматическая генерация документации API в формате Swagger [35].

Swagger-страница представляет собой интерактивный веб-интерфейс, который содержит подробное описание всех доступных функций API, передаваемых и принимаемых аргументов, доступных версий API и другой полезной информации. Это значительно упрощает процесс интеграции с API как для разработчиков клиентской части, так и для других сервисов, которые могут взаимодействовать с нашей системой. Пример Swagger-страницы приведен на рисунке 2.6.

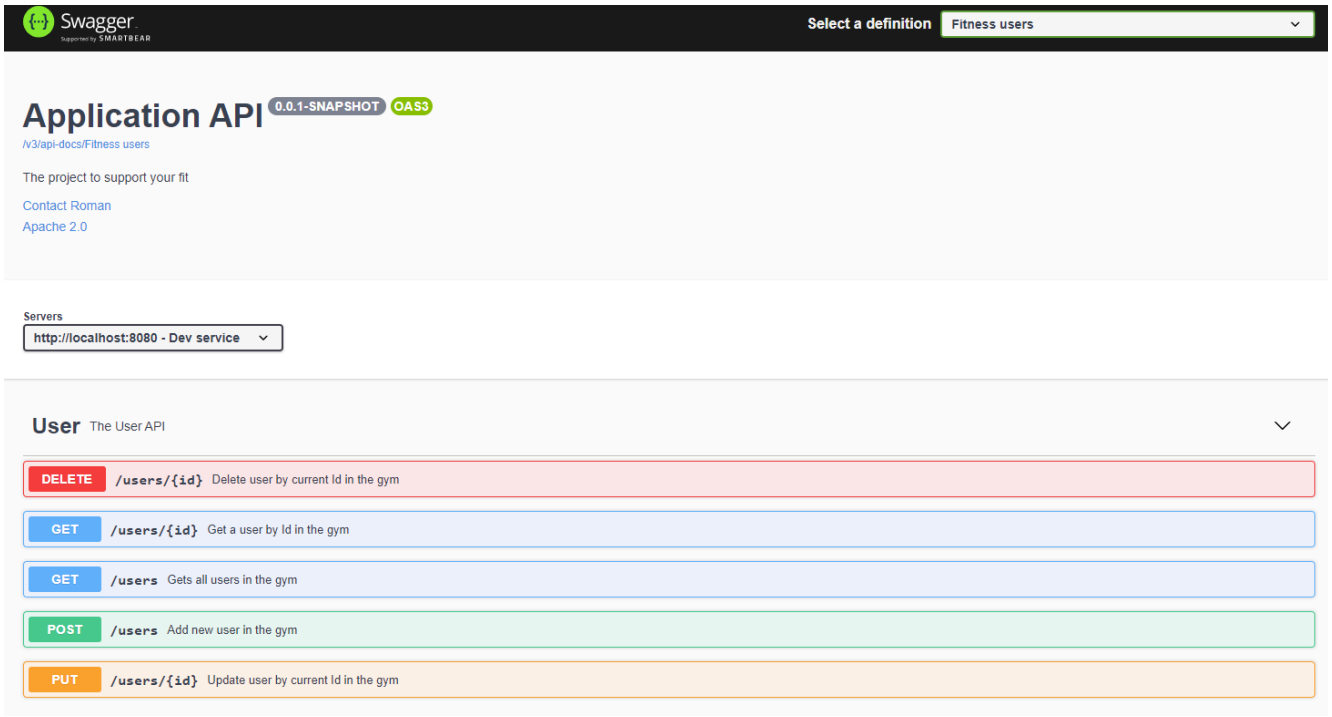

Рисунок 2.6 – Пример Swagger-страницы

Для запуска разработанного API на основе FastAPI необходимо использовать ASGI-сервер, например Uvicorn.

Uvicorn представляет собой сервер, реализованный на языке Python [36]. ASGI является стандартом для асинхронных веб-серверов и приложений. В отличие от более старого стандарта WSGI (Web Server Gateway Interface), который является синхронным, ASGI позволяет обрабатывать асинхронные запросы и ответы, что особенно важно для современных веб-приложений, требующих высокой производительности и способности обрабатывать большое количество одновременных подключений, таких как API, работающие в режиме реального времени или интенсивно использующие операции ввода-вывода.

FastAPI построен на основе асинхронных возможностей Python и использует ASGI для обеспечения своей высокой производительности. Поэтому для запуска приложения FastAPI в production-среде или даже для локальной разработки часто используется именно Uvicorn. Таким образом, Uvicorn выступает в роли «движка», который принимает входящие HTTP-запросы, передает их приложению FastAPI для обработки и затем возвращает ответы клиентам. Использование Uvicorn позволяет FastAPI в полной мере реализовать свои преимущества в плане скорости и эффективности обработки запросов.

Для обеспечения корректного и структурированного обмена данными между клиентским и серверным модулями было принято решение использовать REST API (Representational State Transfer). REST является одним из наиболее популярных архитектурных стилей взаимодействия в распределённых системах благодаря своей простоте, гибкости и совместимости с протоколом HTTP, что делает его особенно удобным для построения клиент-серверных архитектур. Использование REST API позволяет достичь высокой степени разделения ответственности между клиентской и серверной частями системы, обеспечивая при этом масштабируемость и расширяемость архитектуры в будущем.

REST API предоставляет понятные и предсказуемые интерфейсы, построенные на использовании HTTP-методов (таких как GET, POST, PUT, DELETE), что делает его интуитивно понятным для разработчиков и легко тестируемым [37]. В контексте данной системы REST API выступает основным механизмом взаимодействия между пользовательским интерфейсом и логикой серверной обработки, позволяя передавать данные, инициировать вычисления и получать результаты. Иллюстрации принципа взаимодействия между клиентом и сервером в процессе обработки пользовательских запросов представленая на рисунке 2.7.

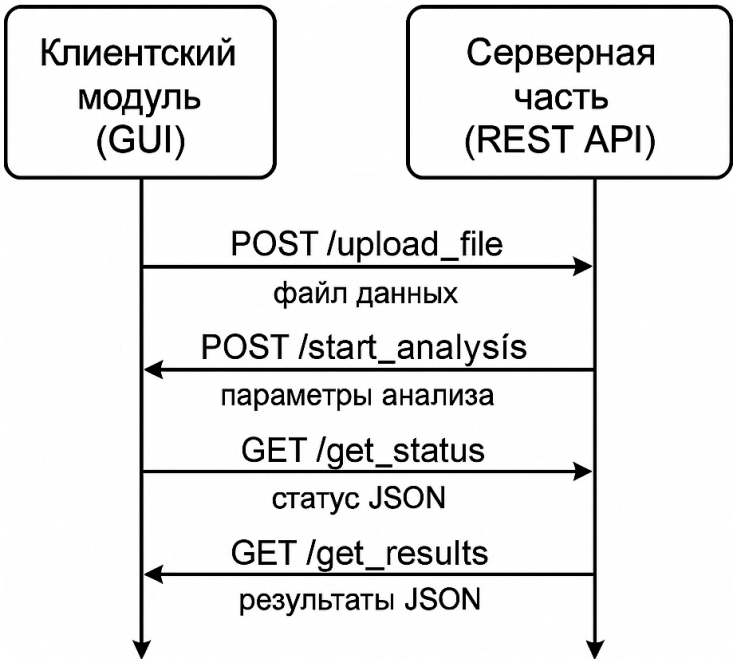

Рисунок 2.7 – Схема взаимодействия клиент-сервер

На рисунке показан типовой цикл взаимодействия: от загрузки данных пользователем до получения результатов анализа. Такой подход позволяет ясно структурировать весь процесс обработки, определить границы ответственности между компонентами и облегчить интеграцию новых функциональных возможностей в будущем.

В рамках проектирования API были определены ключевые эндпоинты, каждый из которых выполняет строго ограниченную задачу. Это упрощает сопровождение системы и повышает её надёжность, поскольку изменение или отказ одного эндпоинта не влияет на работу остальных. Кроме того, наличие четко задокументированных интерфейсов позволяет сторонним разработчикам или другим модулям системы легко подключаться к серверному модулю, что особенно важно в условиях возможной масштабной интеграции.

Ключевые API-эндпоинты, предусмотренные на этапе проектирования:

- `POST /upload_file`: эндпоинт для приема файлов данных трафика (CSV, PDF) от клиентского модуля. В теле запроса передается файл. Разделение загрузки данных от анализа позволяет повысить гибкость и масштабируемость архитектуры
- `POST /start_analysis`: эндпоинт для запуска процесса анализа данных. В теле запроса передаются параметры анализа, заданные пользователем через графический интерфейс (например, желаемое количество сегментов для генетического

алгоритма, параметры алгоритма и т. д.). Серверный модуль передает эти параметры модулю анализирования

- `GET /get_status`: эндпоинт для запроса текущего статуса выполнения анализа (например, «в ожидании», «загружен», «обработка данных», «анализ завершен», «ошибка»). Ответ возвращается в формате JSON и позволяет клиенту отслеживать прогресс и состояние запроса без постоянного взаимодействия с сервером
- `GET /get_results`: эндпоинт для получения результатов завершенного анализа. Ответ содержит сегментированные данные, статистические характеристики сегментов и данные для визуализации (графики), представленные в формате JSON. Это обеспечивает удобство последующей обработки и отображения результатов на клиентской стороне.

Таким образом, проектирование серверного модуля и REST API позволило создать чёткую архитектурную основу для взаимодействия компонентов системы, обеспечивая прозрачность, расширяемость и высокую степень автоматизации при дальнейшей разработке и эксплуатации.

2.3.1 Проектирование модуля обработки входных данных

Модуль обработки входных данных, обеспечивает начальный этап работы интеллектуальной системы прогнозирования и планирования интенсивности движения. Его задача – привести входные данные, полученные от пользователя в формате PDF или CSV, к структурированному и унифицированному виду, пригодному для последующего анализа. Модуль включает два логически связанных компонента: компонент извлечения данных (парсинг) и компонент преобразования (агрегация и фильтрация). Модуль должен поддерживать загрузку исходных данных в двух форматах: PDF и CSV.

PDF – используется в случае, если данные поступают в виде отсканированных или экспортированных из других систем отчётов, содержащих таблицы с временными рядами. Пример формата таких таблиц представлен на рисунке 2.8.

Номер сектора	Номер ДК	Номер направ-ления	Начало интервала накопления	Интервал накопления	Номер характе-ристики	Интенсив-ность (авт/час)
1	101	1	31-01-25 00:15:00	15	1	228
	101	1	31-01-25 00:30:00	15	1	228
	101	1	31-01-25 00:45:00	15	1	232
	101	1	31-01-25 01:00:00	15	1	212
	101	1	31-01-25 01:15:00	15	1	236

Рисунок 2.8 – Пример таблицы в формате PDF

Номер сектора - номер определенной области, где находится данный перекресток. Номер ДК - номер дорожного котроллера для данного перекрестка. Номер направления - индекс направления, по которому проводились измерения. Начало интервала накопления - время, начинаю с которого детектор вел подсчет количества автомобилей. Интервал накопления - время, в течение которого детектор фиксирует количество автомобилей.Номер характеристики - индекс характеристики движения по данному направлению, которая измеряется (в нашем случае интенсивность движения). Интенсивность - количество автомобилей, зафиксированных в течение интервала накопления.

CSV – предпочтительный формат, если данные уже представлены в табличном виде, подготовленном для автоматической обработки. В этом случае этап парсинга опускается, и данные сразу направляются в компонент преобразования. Пример структуры такой таблицы показан на рисунке 2.9.

	Dk Num	Direction Num	Date	Accumulation Start Time	Accumulation Interval	Characteristic Number	Intensity
	101	1	31-01-25	00:15:00	15	1	228
	101	1	31-01-25	00:30:00	15	1	228
	101	1	31-01-25	00:45:00	15	1	232
	101	1	31-01-25	01:00:00	15	1	212
	101	1	31-01-25	01:15:00	15	1	236

Рисунок 2.9 – Пример таблицы в формате CSV

Для парсинга PDF-документов была выбрана библиотека Camelot, которая зарекомендовала себя как инструмент, специально ориентированный на точное и гибкое извлечение табличных данных из PDF [38].

Ключевыми преимуществами Camelot стали её специализация на работе с табличной структурой, что обеспечивает точное позиционирование ячеек даже при слабой разметке и отсутствии явных границ таблиц; высокая гибкость и возможность детальной

настройки параметров извлечения, что позволяет эффективно обрабатывать документы с разнообразной структурой; а также простая интеграция с Python – библиотека написана на этом языке и не требует дополнительных зависимостей, таких как Java, что делает её оптимальным решением в рамках проекта, где весь серверный стек реализован на Python. В качестве альтернативных решений рассматривалась библиотека Tabula-py – Python-обёртка над Java-библиотекой Tabula, которая обеспечивает высокую точность при извлечении простых таблиц, но требует установленной Java и обладает ограниченными возможностями настройки [39]. Библиотека PyPDF2 предоставляет базовые средства для извлечения текста из PDF, однако не имеет встроенных механизмов работы с табличными структурами [40]. PyMuPDF показывает высокую производительность при рендеринге и извлечении различного контента из PDF-документов, но не поддерживает прямое распознавание таблиц [40].

После извлечения таблиц (или прямого получения CSV-файла) данные поступают во второй компонент – преобразования и агрегации. На этом этапе выполняется унификация временных рядов, фильтрация по дням недели и интерполяция пропущенных значений.

Для реализации данного этапа используется библиотека pandas, являющаяся стандартом де-факто в области анализа данных на Python благодаря своей высокой производительности при работе с табличными структурами. Её тесная интеграция с NumPy обеспечивает удобную обработку числовых массивов, а гибкие возможности трансформации данных – такие как группировка, агрегация, интерполяция и работа с временными индексами – делают её особенно подходящей для задач предобработки временных рядов. Дополнительными преимуществами являются широкое сообщество пользователей и обширная документация, что способствует ускоренной разработке и упрощает поддержку кода.

После извлечения или загрузки данных начинается этап преобразования: таблица приводится к формату с индексами по дате и направлению движения, временные метки округляются до ближайшего интервала накопления. Далее выполняется фильтрация по

дням недели, интерполяция пропусков и агрегация в структуру день × направление × интервал, пригодную для анализа.

Таким образом модуль должен обеспечить автоматическую обработку входных данных в форматах PDF и CSV, приводя их к единому структурированному виду. Это упростит последующий анализ и повысит надёжность решения, независимо от в каком формате – PDF или CSV – данных.

2.3.2 Проектирование модуля анализирования

Модуль анализирования, схема которого представлена на рисунке 2.10, представляет собой центральный компонент системы, отвечающий за выполнение анализа временных рядов.

Рисунок 2.10 – Блок-схема генетического алгоритма

Основная задача этого модуля – выявление закономерностей в данных, полученных на предыдущем этапе обработки, и формирование сегментов, которые будут использоваться для дальнейшего прогнозирования и планирования интенсивности движения.

В первой главе был проведён обзор существующих подходов к сегментации временных рядов. Анализ показал, что большинство методов обладают рядом ограничений, которые делают их слабо применимыми для обработки реальных транспортных данных.

Методы кластеризации, несмотря на разнообразие подходов и теоретическую способность выявлять структуры с высокой межкластерной разницей и низкой внутрикластерной дисперсией, на практике сталкиваются с рядом серьёзных ограничений. Во-первых, они не обеспечивают прямого механизма ограничения размера кластеров, особенно по временной оси, что критично для транспортных задач. Во-вторых, многие алгоритмы демонстрируют недостаточную устойчивость к шуму, характерному для реальных данных интенсивности движения, что приводит к снижению точности сегментации и появлению артефактов в виде ложных границ сегментов. В-третьих, качество разграничения сегментов часто оказывается неудовлетворительным – алгоритмы могут создавать сегменты с высокой внутренней вариацией или, наоборот, объединять принципиально разные временные интервалы, не обеспечивая максимального различия между соседними сегментами. В задачах планирования интенсивности движения, где необходимо контролировать длительность сегментов или обеспечить равномерное распределение объектов между кластерами, такие алгоритмы оказываются либо неприменимыми, либо требуют сложных обходных решений. Более того, многие методы кластеризации не позволяют явно задать желаемое число кластеров, что критично в прикладных сценариях с фиксированными требованиями к выходной структуре.

Методы обнаружения точек изменений сталкиваются с аналогичными ограничениями в области точности и качества сегментации. Они не обеспечивают прямого управления длиной и числом сегментов, что критически важно в прикладных задачах планирования, где требуется предсказуемая и управляемая структура выходных данных. В отличие от кластеризации, эти методы часто не гарантируют оптимального разграничения сегментов: результирующие сегменты могут иметь высокую внутреннюю вариацию при

низком различии между соседними интервалами, что противоречит основным принципам качественной сегментации. Особенно остро эта проблема проявляется в условиях шума и плавных переходов между режимами движения, когда алгоритмы либо пропускают реальные точки изменений, либо создают ложные границы. Сегменты могут значительно отличаться по длине и статистическим характеристикам, не имея чёткой интерпретации. Кроме того, многие алгоритмы данной группы требуют жёсткой структуры входных данных или предварительного задания количества сегментов, оказываются неустойчивыми к шуму, плохо масштабируются и сильно зависят от настройки гиперпараметров.

Наиболее близким к практическим требованиям является метод динамического программирования, поскольку он позволяет задать фиксированное количество сегментов и обеспечивает глобально оптимальное разбиение по функции стоимости. Однако он также не решает задачу в полном объёме, поскольку не позволяет одновременно учитывать все необходимые требования: максимизацию различий между сегментами, минимизацию внутрисегментной дисперсии, а также наличие жёстких ограничений как на количество, так и на длину сегментов.

Таким образом, существующие методы не обеспечивают комплексного решения задач сегментации временных рядов интенсивности движения, которые характеризуются высокой неоднородностью, отсутствием явной структуры и варьирующей длиной естественных сегментов, при одновременной необходимости соблюдения строгих ограничений на структуру выходных данных.

Учитывая выявленные ограничения существующих подходов, в настоящей работе предлагается метод на основе генетического алгоритма, который позволяет одновременно решить основные проблемы: обеспечить контроль над количеством и длиной сегментов, максимизировать различия между сегментами при минимизации внутрисегментной дисперсии, а также сохранить устойчивость к шуму.

Для оценки качества работы алгоритмов и их соответствия требованиям были предложены 4 метрики: внутрисегментная вариация (V), межсегментная разность (D), доля сегментов, не удовлетворяющих минимальной длине (SSR), отклонение числа сегментов (SE); а также дополнительно время работы алгоритма (t).

Внутрисегментная вариация представляет собой усреднённую величину, характеризующую степень разброса элементов внутри каждого сегмента относительно его центра. Данная метрика отражает внутреннюю однородность сегментов: чем меньше её значение, тем более однородны данные внутри сегментов.

Вычисление центроида сегмента производится по формуле (2.1):

$$\mu_s = \frac{1}{|I_s|} \sum_{i \in I_s} x_i \ , \tag{2.1}$$

где μ_s – центроид сегмента s;

I_s – множество индексов элементов, принадлежащих сегменту s;

$x_i \in \mathbb{R}^d$ – вектор размерности d для элемента i.

Внутрисегментная вариация вычисляется по формуле (2.2):

$$V = \frac{1}{S} \sum_{s=1}^{S} \frac{1}{|I_s|} \sum_{i \in I_s} ||x_i - \mu_s|| \ , \tag{2.2}$$

где V – внутрисегментная вариация;

S – общее число сегментов;

I_s – множество индексов элементов в сегменте s;

$x_i \in \mathbb{R}^d$ – вектор размерности d для элемента i;

μ_s – центроид сегмента s, вычисленный по формуле (2.1).

Межсегментная разность является усреднённой мерой различий между соседними сегментами и характеризует степень контрастности между ними. Эта метрика отражает различие между сегментами: чем больше её значение, тем сильнее отличаются соседние сегменты друг от друга.

Межсегментная разность вычисляется по формуле (2.3):

$$D = \frac{1}{S-1} \sum_{s=1}^{S-1} \left|\left|\mu_s - \mu_{s+1}\right|\right| \ , \tag{2.3}$$

где D – межсегментная разность;

S – общее число сегментов;

μ_s и μ_{s+1} – центроиды соседних сегментов s и $s+1$, вычисленные по формуле (2.1).

Доля сегментов, не удовлетворяющих минимальной допустимой длине, представляет собой отношение количества сегментов, длина которых меньше $L_{\min}$, к общему числу сегментов:

$$SSR = \frac{|\{s : |I_s| < L_{\min}\}|}{S}\ , \tag{2.4}$$

где SSR – доля сегментов, не удовлетворяющих минимальной длине;

S – общее число сегментов;

$L_{\min}$ – минимальная допустимая длина сегмента.

Отклонение числа сегментов вычисляется по формуле (2.5):

$$SE = \frac{|S - S^*|}{S}\ , \tag{2.5}$$

где SE – отклонение числа сегментов;

S – фактическое общее число сегментов;

S^* – требуемое общее число сегментов.

Без нормализации значений внутрисегментная вариация и межсегментная разность напрямую зависят от масштаба признаков, что затрудняет объективное сравнение. Поэтому используются нормализованные версии метрик, обеспечивающие сопоставимость качества сегментации между различными наборами данных и параметрами.

Нормализованная внтурисегментная вариация вычисляется по формуле (2.6):

$$V_{\text{norm}} = \frac{V}{\max_{i,j}||x_i - x_j||}\ , \tag{2.6}$$

где V_{norm} – нормализованная внутрисегментная вариация;

V – внутрисегментная вариация, вычисленная по формуле (2.2);

$x_i, x_j \in \mathbb{R}^d$ – вектора размерности d для элементов i, j.

Нормализованная внтурисегментная вариация вычисляется по формуле (2.7):

$$D_{\text{norm}} = \frac{D}{\max_{i,j}||x_i - x_j||}\ , \tag{2.7}$$

где D_{norm} – нормализованная внутрисегментная вариация;

D – внутрисегментная вариация, вычисленная по формуле (2.3);

$x_i, x_j \in \mathbb{R}^d$ – вектора размерности d для элементов i, j.

Итак, в таблице 2.1 представлены результаты сравнения различных алгоритмов кластеризации по предложенным метрикам качества: нормализованной внутрисегментной вариации (V_{norm}), нормализованной межсегментной разности (D_{norm}), доле сегментов, не удовлетворяющих минимальной длине (SSR), отклонению числа сегментов (SE) и времени работы алгоритма (t).

Итак, в таблице 2.1 представлены результаты сравнения различных алгоритмов кластеризации по предложенным метрикам: V_{norm}, D_{norm}, SSR, SE и времени работы t.

Таблица 2.1 – Сравнение алгоритмов разделения на сегменты

	V_{norm}	D_{norm}	SSR	SE	t, с
Методы кластеризации					
Экспертные правила	0.06	0.29	0.21	0.39	15.2
Пороговая	0.08	0.41	0.23	0.10	0.1
k-means	0.12	0.49	0.26	0.29	0.2
DBSCAN	0.14	0.36	0.02	0.79	1.0
Гауссовой смесь	0.09	0.46	0.24	0.26	2.0
Пуассоновской смесь	0.11	0.47	0.20	0.21	2.1
Агломеративная	0.07	0.51	0.12	0.09	1.0
Finch	0.09	0.39	0.08	0.16	7.6
TW-Finch	0.08	0.42	0.06	0.12	11.2
Методы поиска точек изменения					
Clasp	0.19	0.26	0.05	0.61	20.3
Pointer Networks	0.18	0.31	0.09	0.59	250.8
Kernel Change Detection	0.19	0.14	0.31	0.39	7.5
Binary	0.16	0.30	0.19	0.31	6.3
Bottom-Up	0.14	0.32	0.18	0.26	6.1
Window Sliding	0.13	0.33	0.20	0.31	6.1
Pelt	0.12	0.41	0.09	0.21	7.0
Dynamic Programming	0.04	0.54	0.00	0.00	210.1
Генетический алгоритм	**0.04**	**0.60**	**0.00**	**0.00**	**180.6**

Как видно из таблицы 2.1, лучшее сочетание качества сегментации демонстрирует предлагаемый генетический алгоритм (GA), который превосходит существующие методы по ключевым показателям.

Генетический алгоритм, схема которого представлена на рисунке 2.11, кодирует разбиение временного ряда в виде хромосом – например, бинарных масок точек разрыва

– и оценивает их по формуле (2.8) и оптимизирует с помощью операторов мутации, скрещивания и селекции. В отличие от жёстких методов (например, Binary Segmentation), GA не требует знания точного количества сегментов и способен эффективно работать в условиях шума, выбросов, разной плотности кластеров и высокой размерности данных. Его стохастическая природа позволяет избегать локальных минимумов, а гибкость в выборе функции стоимости – адаптировать подход под любые метрики качества.

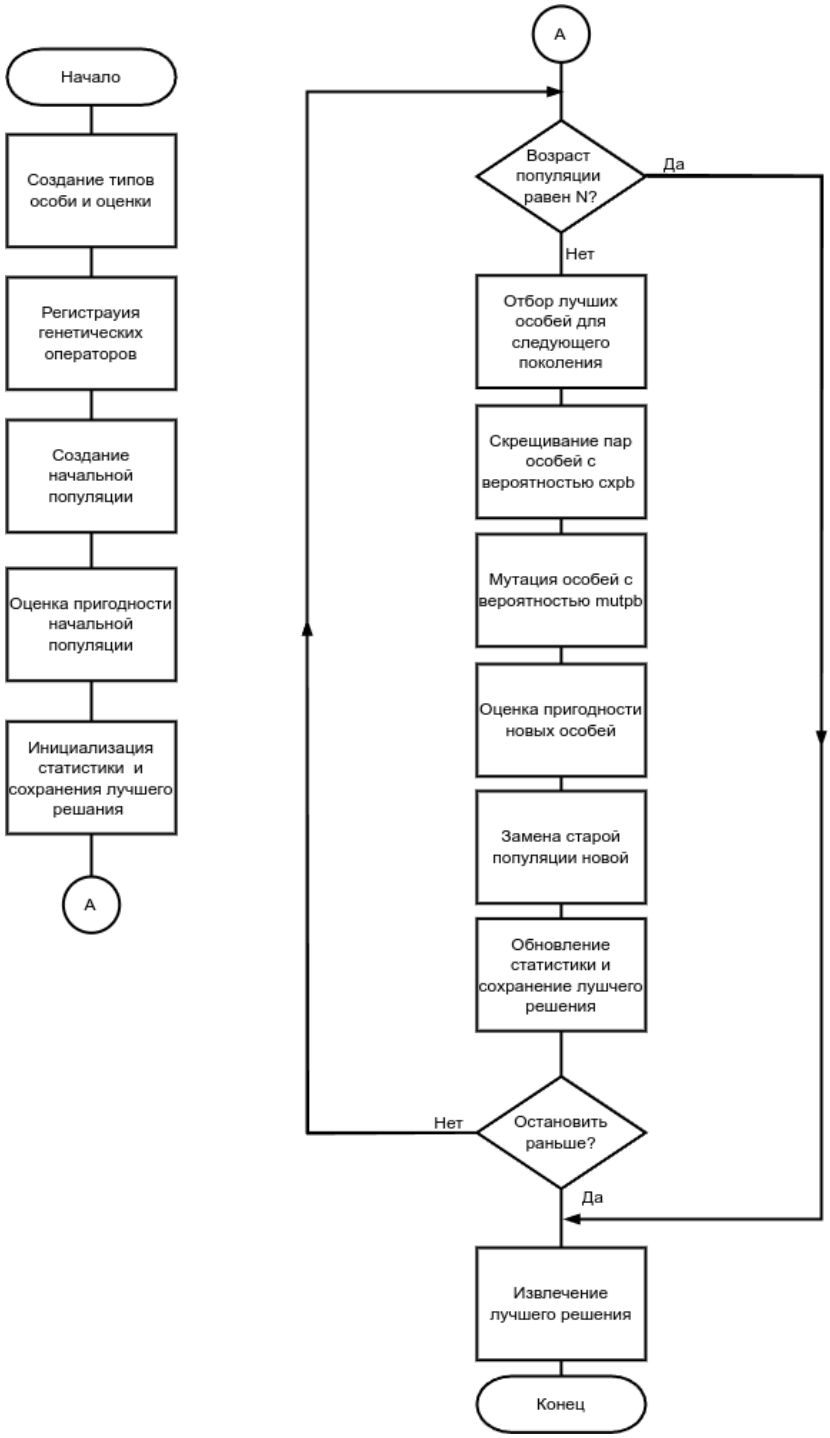

Рисунок 2.11 – Блок-схема генетического алгоритма

Математически функция оценки пригодности (F) выражается формулой (2.8):

$$F = w_1 \cdot D_{norm} - w_2 \cdot P_{size} - w_3 \cdot P_{range} \,, \qquad (2.8)$$

где D_{norm} – нормализованная сумма евклидовых расстояний между средними векторами сегментов;

P_{size} – штраф за неравномерность размеров сегментов;

P_{range} – штраф за неоднородность диапазонов значений в сегментах;

w_1, w_2, w_3 – весовые коэффициенты важности критериев.

Для реализации предлагаемого генетического алгоритма была выбрана библиотека DEAP, предоставляющая удобные механизмы для построения эволюционных алгоритмов. Одним из ключевых факторов выбора стало наличие встроенных инструментов для настройки операторов мутации, скрещивания и отбора, а также поддержка параллельных вычислений, что критично при работе с большими объёмами данных [41]. Гибкость DEAP позволила адаптировать алгоритм под конкретные требования, включая контроль количества кластеров, их размера и однородности. Благодаря этим преимуществам DEAP стал оптимальным выбором для реализации предложенного метода.

2.3.3 Проектирование модуля формирования отчётных данных

Модуль формирования отчётных данных должен обеспечивать как графическое, так и табличное представление результатов кластерного анализа временных рядов. Его основная задача – предоставить пользователю интерпретируемую и наглядную информацию о кластерной структуре данных, полученной на предыдущих этапах обработки.

Проектом предусмотрено использование визуализации временных рядов с выделением кластеров с помощью цветовой кодировки. Такая визуализация должна быть организована по направлениям движения: для каждого направления на графике должны отображаться исходные значения, участки кластеров, а также аннотации со сводной статистикой (средними значениями и диапазонами изменений). Использование различных цветов и адаптивное позиционирование подписей позволит избежать наложений и сохра-

нить читаемость графиков. Это способствует быстрому визуальному анализу структуры кластеров и выявлению типовых шаблонов поведения.

Для дополнения графического отчёта должна быть предусмотрена табличная форма представления данных. Таблица должна содержать для каждого кластера средние значения, диапазоны интенсивности по направлениям и соответствующие временные интервалы. Принципиально важно, чтобы структура таблицы позволяла использовать её как в автономных отчётах (например, PDF или Excel), так и в рамках веб-интерфейса.

Также необходимо, чтобы визуальная часть отчёта преобразовывалась в формат, пригодный для встраивания или хранения без зависимости от файловой системы (например, base64) [42]. Это решение позволит легко передавать отчётные данные через API, сохранять их в базе данных и использовать в других компонентах системы.

Таким образом, модуль отчётности спроектирован как универсальный инструмент визуального и табличного представления результатов анализа. Он должен обеспечивать как техническую точность, так и удобство восприятия для пользователей, не обладающих технической подготовкой.

3 РЕАЛИЗАЦИЯ И ИСПЫТАНИЕ СИСТЕМЫ

3.1 Реализация и тестирование модуля графического интерфейса

Веб-интерфейс реализован в соответствии с проектными решениями с использованием чистых HTML, CSS и JavaScript, без применения сторонних фреймворков. Архитектура приложения базируется на трех основных файлах: `index.html`, в котором размещена структура интерфейса с четырьмя последовательными шагами, `styles.css`, отвечающем за визуальное оформление и адаптивность, и `script.js`, реализующем логику взаимодействия и управления состояниями.

Переходы между шагами управляются динамически за счёт изменения CSS-классов, отражающих текущее состояние каждого этапа: активный, завершённый или неактивный. В интерфейсе предусмотрена возможность возврата к предыдущим этапам с полным восстановлением исходных значений, что позволяет пользователю свободно перемещаться между шагами без потери введённых данных или сбоя в логике работы приложения.

Для отображения прогресса длительных операций реализован механизм опроса состояния, который с заданной периодичностью получает от сервера актуальную информацию. Это обеспечивает обратную связь в реальном времени без блокировки интерфейса, а при завершении процесса или возникновении ошибок опрос автоматически прекращается.

Обмен данными с серверной частью осуществляется через REST API с использованием Fetch API. Каждая операция получает уникальный идентификатор, позволяющий отслеживать её статус на протяжении всего выполнения. Загрузка файлов производится с помощью FormData и включает клиентскую валидацию, что обеспечивает поддержку корректных форматов (например, CSV и PDF) и предотвращает передачу неподдерживаемых типов.

Адаптивность интерфейса обеспечивается с помощью Flexbox и использования относительных единиц измерения, что делает интерфейс удобным для работы как на

мобильных устройствах, так и на больших экранах. Дополнительные параметры анализа, скрытые по умолчанию, можно развернуть по желанию пользователя. Это делает интерфейс простым и понятным при первичном использовании, не лишая при этом гибкости для опытных пользователей. Для предотвращения ошибок ввода в числовые поля применяются защитные механизмы, блокирующие некорректные действия.

Интерфейс организован в виде пошаговой последовательности с интуитивно понятными элементами управления и визуальной обратной связью, как показано на рисунке 3.1.

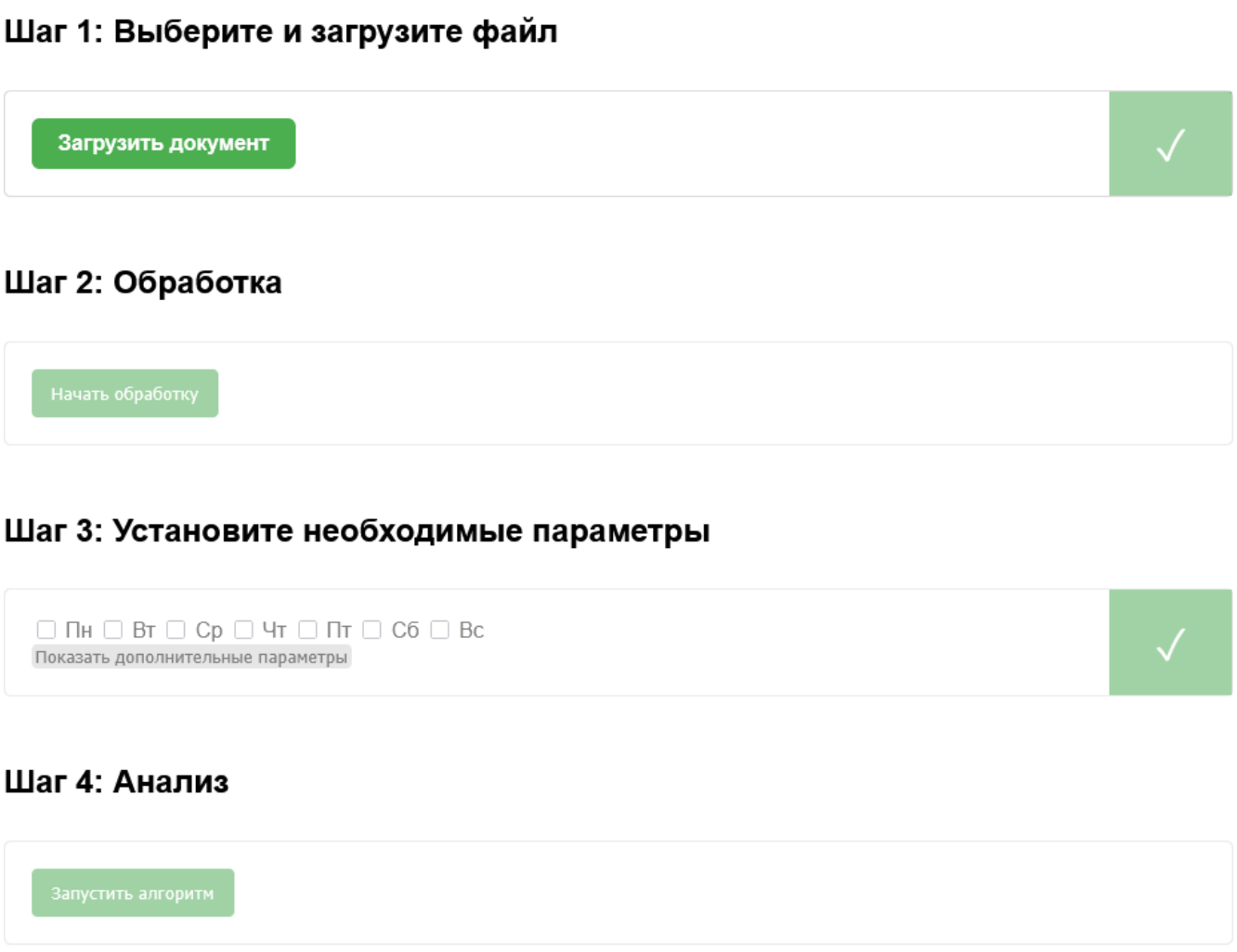

Рисунок 3.1 – Веб-интерфейс

Тестирование проводилось всесторонне: проверялась работа всех шагов пользовательского сценария, валидация файлов, корректность отображения и сохранения параметров, а также получение результатов анализа с отображением визуальных данных.

Интерфейс был проверен на адаптивность и стабильность работы на разных экранах, а также на корректность отображения элементов при переходах между этапами.

Особое внимание уделялось механизму восстановления состояния при возврате на предыдущие шаги, а также обработке ошибок – например, при загрузке неподдерживаемых файлов, сбоях соединения или проблемах на сервере. В рамках кроссбраузерного тестирования была подтверждена корректная работа интерфейса во всех основных браузерах.

В процессе тестирования были выявлены и устранены несколько незначительных проблем, связанных с состоянием интерфейса при быстрой навигации между этапами. Финальная версия системы демонстрирует устойчивую и предсказуемую работу, обеспечивая удобный и понятный пользовательский опыт, полностью соответствующий требованиям проекта по анализу временных данных.

3.2 Реализация и тестирование серверного части

Серверный модуль реализован на Python с использованием фреймворка FastAPI в соответствии с архитектурными решениями, изложенными ранее. В главном файле приложения определены все необходимые маршруты и реализована логика обработки пользовательских запросов.

Для управления загрузкой и обработкой файлов применяется система отслеживания процессов на основе уникальных идентификаторов. Каждый процесс проходит через пять последовательных этапов: выбор типа файла, загрузка, предварительная обработка, настройка параметров и анализ. Состояния процессов сохраняются в оперативной памяти сервера, что обеспечивает быстрый доступ к текущей информации и позволяет в реальном времени отображать прогресс на клиентской стороне.

Обработка входящих файлов сопровождается проверкой форматов и сохранением на сервере. Для повышения отзывчивости системы используется асинхронная обработка – анализ и преобразование данных выполняются в фоновом режиме, позволяя серверу сразу отправлять подтверждение получения и не блокировать интерфейс пользователя.

Поддерживается работа с CSV и PDF. Для PDF-документов реализована автоматическая обработка табличных данных с помощью библиотеки Camelot, обеспечивающей извлечение таблиц и их конвертацию в удобный для анализа формат. Дополнительно выполняется очистка и структурирование информации, включая корректное разбиение столбцов и приведение полей к требуемой структуре.

Для проверки корректности входных данных используются модели, описанные с помощью Pydantic. Они обеспечивают строгую типизацию и валидацию параметров анализа, а также формируют понятные сообщения об ошибках. Настройки анализа, такие как выбор дней недели, количество сегментов и пороговые значения, передаются серверу и сохраняются для дальнейшего использования в процессе вычислений.

Анализ данных запускается асинхронно: параметры преобразуются во внутренний формат, совместимый с аналитическим модулем, и передаются в алгоритм, выполняющий основной расчет. Результаты возвращаются в виде JSON, включая изображения графиков и таблицы в HTML-формате.

Механизмы обработки ошибок реализованы комплексно: все ключевые операции защищены от сбоев с помощью конструкции обработки исключений. В случае ошибок статус процесса обновляется соответствующим образом, что позволяет клиенту корректно отобразить ситуацию и, при необходимости, повторить операцию.

Тестирование серверного модуля проводилось поэтапно. Каждый маршрут API проверялся как на корректную работу с допустимыми данными, так и на устойчивость при ошибках, включая нарушения порядка вызова, передачу некорректных файлов и параметров. Тестирование охватывало также взаимодействие всех компонентов системы: загрузка, промежуточная обработка, анализ и возврат результатов.

Особое внимание было уделено проверке фоновых задач и корректности обновления статуса процессов во время выполнения. Сценарии с параллельной обработкой запросов позволили оценить производительность и выявить узкие места при работе под нагрузкой. Также была протестирована совместимость с реальными данными различного объема и структуры, включая PDF-файлы с нетипичными таблицами.

Встроенная Swagger-документация прошла верификацию на соответствие фактическому поведению API, что обеспечило удобную и точную платформу для тестирования запросов. Результаты тестирования подтвердили стабильность, производительность и готовность серверного модуля к эксплуатации в составе полной системы анализа временных данных.

3.2.1 Реализация и тестирование модуля обработки входных данных

Модуль обработки входных данных реализован на языке Python с использованием специализированных библиотек для работы с табличными и полуструктурированными данными. Обработка PDF-документов осуществляется с помощью библиотеки Camelot, которая позволяет извлекать таблицы с каждой страницы документа. Используется алгоритм распознавания структуры `lattice`, обеспечивающий точное определение границ таблиц при наличии четкой разметки. Извлеченные таблицы объединяются в единый DataFrame, очищаются от артефактов и приводятся к унифицированной структуре с ожидаемыми именами столбцов. Результат сохраняется в CSV-формате, что позволяет применять универсальную логику дальнейшей обработки независимо от исходного типа файла.

Обработка CSV-файлов реализована в функции `process_csv()`, которая выполняет весь цикл преобразования данных об интенсивности движения. Входными параметрами являются путь к файлу и список дней недели, по которым необходимо отфильтровать данные. Основной задачей является агрегация интенсивностей в заданных временных интервалах, фильтрация по дате и направление движения, а также интерполяция пропущенных значений. В результате функция возвращает структурированный словарь, содержащий идентификатор контроллера, список уникальных дат, направления движения, временные интервалы и трехмерный массив интенсивностей.

```
def process_csv(file_path, days):
    df = pd.read_csv(file_path)
    id = int(df['dkNum'].iloc[0])
    df['date'] = pd.to_datetime(df['date'], format="%d-%m-%y")
```

```
accumulationInterval = df['accumulationInterval'].iloc[0]
df = df.pivot(index=['date', 'directionNum'],
              columns='accumulationStartTime', values='intensity')

def floor_to(time_str):
    dt = pd.to_datetime(time_str, format='%H:%M:%S')
    minute = dt.minute - (dt.minute % accumulationInterval)
    floored = dt.replace(minute=minute, second=0)
    return floored.strftime('%H:%M:%S')

timestamps = [col for col in df.columns if col not in [
    'accumulationStartTime', 'date', 'directionNum']]
rename_mapping = {col: floor_to(col) for col in timestamps}
time_df = df[timestamps].rename(columns=rename_mapping)

aggregated_time = pd.DataFrame(index=time_df.index)
for new_col in time_df.columns.unique():
    subset = time_df.loc[:, time_df.columns == new_col]
    aggregated_time[new_col] = subset.sum(axis=1, min_count=1)
df = aggregated_time
timestamps = [col for col in df.columns if col not in [
    'accumulationStartTime', 'date', 'directionNum']]

weekdays_df = df[df.index.get_level_values('date').dayofweek.isin(days)]
weekdays_df = weekdays_df.interpolate(
    method='linear', axis=1, limit_direction='both')

indices = list(weekdays_df.index)
dates, directions = zip(*indices)
dates = np.unique(np.array([d.strftime('%Y-%m-%d') for d in dates]))
directions = np.unique(np.array(directions))
n_directions = directions.shape[0]

weekdays_df = weekdays_df.to_numpy()
n_rows, n_cols = weekdays_df.shape
intensities = weekdays_df.reshape(
    n_rows // n_directions, n_directions, n_cols)
```

```
    return {
        "id": id,
        "dates": dates,
        "timestamps": timestamps,
        "directions": directions,
        "intensities": intensities
    }
```

Интерполяция пропущенных значений реализована встроенным методом с линейным подходом и направлением заполнения в обе стороны. Это позволяет восстанавливать значения даже при наличии разрывов на границах временных интервалов, не нарушая при этом общий тренд в данных. Финальное преобразование обеспечивает получение массива фиксированной размерности: по числу дней, направлений и временных меток.

Тестирование модуля проводилось на наборах реальных данных с различной структурой и степенью полноты. Были проверены сценарии с корректными CSV-файлами, файлами с пропущенными значениями и файлами, содержащими нестандартные временные интервалы. Валидация обработки PDF-документов проводилась на примерах с различным качеством сканирования и структурой таблиц: от четко размеченных до слабоструктурированных. Библиотека Camelot показала высокую устойчивость к разнообразию входных документов, при необходимости обеспечивая настройку параметров извлечения.

Проверка результатов обработки включала контроль за сохранением исходного количества наблюдений, корректностью формирования временных осей и соответствием размерностей получаемого массива ожидаемым. Интерполяция проверялась на искусственно созданных пропусках разной протяженности, показав эффективность заполнения при интервалах до 2–3 временных отсчетов. Отдельная проверка фильтрации по дням недели подтвердила правильность исключения выходных при соответствующих настройках.

Производительность модуля оценивалась на крупных входных файлах объёмом до 50 МБ с данными за продолжительные периоды. Тестирование показало устойчивую и предсказуемую работу при последовательной и параллельной обработке, что подтверждает готовность компонента к использованию в составе полной аналитической системы.

3.2.2 Реализация и тестирование модуля анализирования

Центральным компонентом модуля является функция `partition_array_ga`, реализующая генетический алгоритм для оптимального разбиения временных рядов на сегменты. Алгоритм основан на библиотеке DEAP и использует специализированную функцию оценки пригодности особей. Каждая особь в популяции представляет собой список границ разбиения, где каждый элемент определяет позицию разделения временного ряда. Функция `create_individual` генерирует случайные, но валидные границы сегментов с учетом ограничения на минимальную ширину:

```
def create_individual():
    # Распределить дополнительные столбцы случайным образом
    extra = N - num_parts * min_width
    dividers = sorted(random.sample(range(1, extra + 1), num_parts - 1))

    # Преобразовать в фактические граничные положения
    boundaries = []
    cum_sum = 0
    for i in range(num_parts - 1):
        if i == 0:
            cum_sum += min_width + dividers[0]
        else:
            cum_sum += min_width + (dividers[i] - dividers[i-1])
        boundaries.append(cum_sum)

    return boundaries
```

Качество разбиения оценивается многокритериальной функцией `evaluate`, учитывающей три ключевых фактора: максимизацию различий между сегментами, равномерность их размеров и однородность диапазонов значений внутри сегментов по формуле (2.8):

```
def evaluate(individual):
    partitions = get_partitions(individual, N)

    # Вычисление векторов средних значений для каждой части
    mean_vectors = []
```

```
    range_vectors = []
    partition_sizes = []

    for start, end in partitions:
        block = array[:, start:end]
        mean_vector = np.mean(block, axis=1)
        range_vector = np.max(block, axis=1) - np.min(block, axis=1)

        mean_vectors.append(mean_vector)
        range_vectors.append(range_vector)
        partition_sizes.append(end - start)

    # Максимизация различий между частями
    total_diff = 0
    for i in range(len(mean_vectors)):
        for j in range(i+1, len(mean_vectors)):
            total_diff += np.linalg.norm(mean_vectors[i] - mean_vectors[j])

    # Штраф за неравномерность размеров
    ideal_size = N / num_parts
    size_penalty = (np.std(partition_sizes) / ideal_size) * size_penalty_weight

    # Штраф за неоднородность диапазонов
    mean_ranges = [np.mean(range_vec) for range_vec in range_vectors]
    range_cv = np.std(mean_ranges) / (np.mean(mean_ranges) + 1e-10)
    range_penalty = range_cv * range_uniformity_weight

    # Итоговая оценка пригодности
    fitness = diff_weight * (total_diff / (num_parts * (num_parts - 1) / 2)) -
size_penalty - range_penalty

    return (fitness,)
```

Для эволюции популяции используются стандартные генетические операторы. Оператор скрещивания `tools.cxOnePoint` выполняет одноточечный кроссовер, объединяя границы разбиения от двух родительских особей. Оператор мутации `mutate_boundaries` случайным образом изменяет позиции границ с учетом ограничений минимальной ширины сегментов:

```
def mutate_boundaries(individual, indpb=0.3):
    for i in range(len(individual)):
        if random.random() < indpb:
            # Вычисление допустимого диапазона для границы
            if i == 0:
                min_val = min_width
            else:
                min_val = individual[i-1] + min_width

            if i == len(individual) - 1:
                max_val = N - min_width
            else:
                max_val = individual[i+1] - min_width

            if min_val < max_val:
                individual[i] = random.randint(min_val, max_val)

    return (individual,)
```

Селекция осуществляется турнирным отбором `tools.selTournament` с размером турнира 3, что обеспечивает баланс между давлением отбора и разнообразием популяции.

После получения оптимального разбиения применяется иерархическая кластеризация для объединения схожих сегментов. Функция `cluster_parts_by_threshold` использует пороговый алгоритм, основанный на процентилях матрицы различий между сегментами:

```
def cluster_parts_by_threshold(diff_matrix, threshold_percentile=25):
    # Вычисление порога на основе процентиля
    upper_triangle = diff_matrix[np.triu_indices_from(diff_matrix, k=1)]
    threshold = np.percentile(upper_triangle, threshold_percentile)

    # Инициализация кластеров
    num_parts = diff_matrix.shape[0]
    cluster_labels = np.arange(num_parts)

    # Итеративное объединение близких кластеров
    while True:
        merged = False
```

```
        min_diff = float('inf')
        merge_i, merge_j = -1, -1

        for i in range(num_parts):
            for j in range(i + 1, num_parts):
                if (cluster_labels[i] != cluster_labels[j] and
                    diff_matrix[i, j] < threshold and
                    diff_matrix[i, j] < min_diff):
                    min_diff = diff_matrix[i, j]
                    merge_i, merge_j = i, j

        if min_diff < threshold:
            old_label = cluster_labels[merge_j]
            new_label = cluster_labels[merge_i]
            cluster_labels[cluster_labels == old_label] = new_label
            merged = True

        if not merged:
            break

    return normalized_labels, threshold
```

Для каждого выявленного кластера функция `analyze_clusters` вычисляет статистические характеристики, включая средние значения интенсивности и диапазоны колебаний по временным периодам:

```
def analyze_clusters(array, partition_boundaries, cluster_labels):
    cluster_stats = {}

    for i in range(num_parts):
        start, end = partition_boundaries[i], partition_boundaries[i+1]
        cluster_id = cluster_labels[i]

        for row in range(M):
            data = array[row, start:end]
            if cluster_id not in cluster_stats:
                cluster_stats[cluster_id] = {
                    'mean_values': np.mean(data),
                    'range_values': np.max(data) - np.min(data)
```

```
            }

    return cluster_stats
```

Обработка данных также содержит извлечение минимальных и максимальных значений по каждому сегменту, которые затем используются для построения матрицы евклидовых расстояний между векторами признаков. Это позволяет выявить степень схожести между сегментами и служит основой для кластеризации. На заключительном этапе смежные сегменты, отнесённые к одному кластеру, объединяются в непрерывные временные интервалы, что позволяет представить результаты в виде целостных и интерпретируемых фрагментов поведения.

Тестирование генетического алгоритма проводилось на синтетических данных с известной структурой сегментации для проверки корректности кодирования особей, монотонности функции оценки и сходимости алгоритма. Все сгенерированные границы соответствовали ограничениям минимальной ширины сегментов, а решения с более выраженными различиями между сегментами получали более высокие оценки пригодности.

Полный процесс анализа проходил тестирование на реальных данных интенсивности транспортного потока с акцентом на оценку устойчивости кластеризации, адекватности разметки и вычислительной эффективности. При сравнении с экспертной сегментацией метод показал более стабильные и последовательные результаты, особенно на граничных участках. Среднее время обработки одного временного ряда составляет около одной минуты, а показатель стабильности между запусками достигает 0.99, что свидетельствует о высокой воспроизводимости получаемых интервалов.

3.2.3 Реализация и тестирование модуля формирования отчётных данных

Модуль формирования отчётных данных обеспечивает графическое и табличное представление результатов анализа временных рядов. Визуализация реализована в виде многоуровневых графиков, на которых участки, принадлежащие разным кластерам, выделяются различными цветами. Это позволяет наглядно отразить структуру и динамику

сегментированных данных по каждому направлению движения. Для обеспечения визуального различия используется расширенная цветовая палитра, подходящая даже при большом количестве кластеров.

Особое внимание уделено способу отображения статистики кластеров на графиках. Подписи размещаются с учётом плотности данных и доступного пространства: анализируется наличие свободной области над и под каждым кластером, а также исключается перекрытие уже размещённых аннотаций. Благодаря этому даже в случаях с высокой плотностью разбиений сохраняется читаемость визуализации. Границы между кластерами представлены вертикальными линиями различного стиля – сплошными и пунктирными, в зависимости от типа разделения.

На рисунке 3.2 представлен пример графической визуализации результатов анализа.

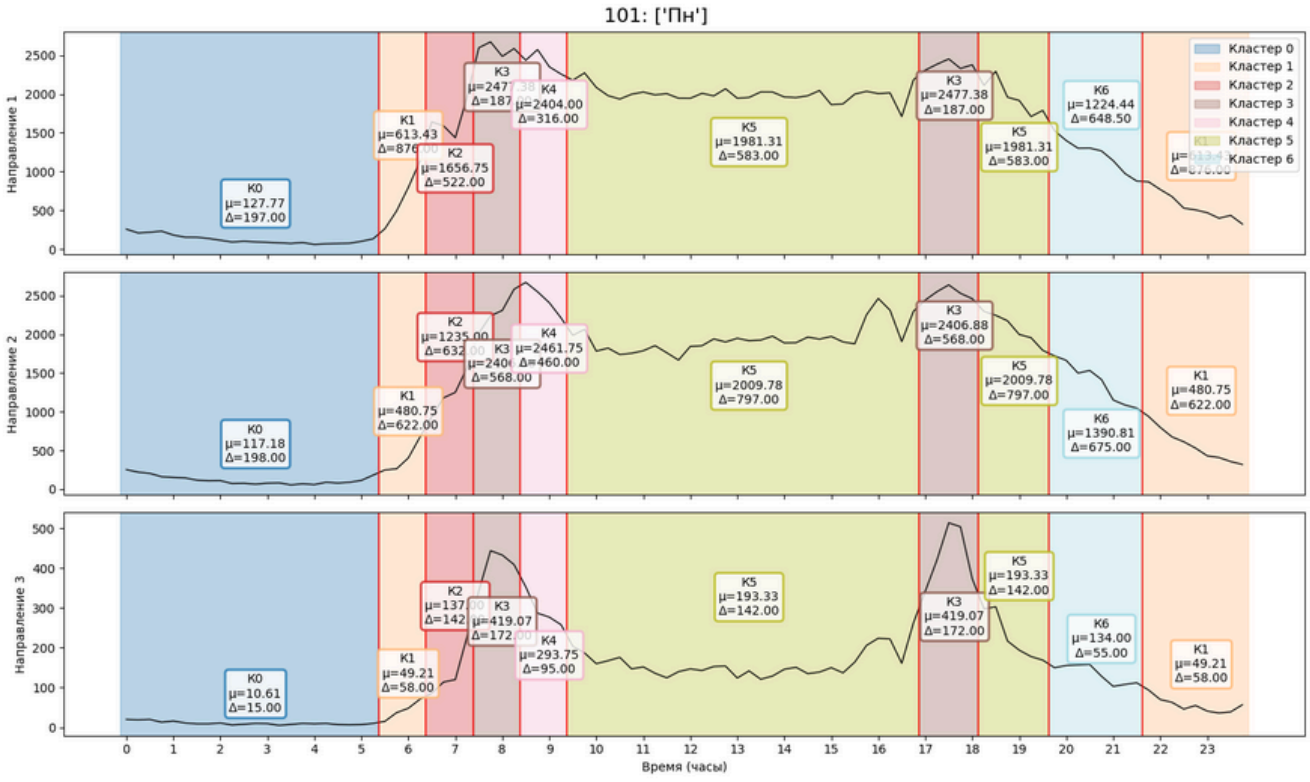

Рисунок 3.2 – График интенсивностей с выделенными сегментами

Помимо графика, формируется подробная таблица с характеристиками каждого кластера: включаются интервалы времени, агрегированные показатели интенсивности по направлениям, а также статистические диапазоны. Таблица построена с логической груп-

пировкой столбцов по типам данных, что облегчает восприятие и делает её пригодной как для отчётности, так и для анализа специалистами. Все численные значения форматируются для улучшения читаемости.

Пример такой таблицы приведён на рисунке 3.3.

Кластер	Среднее значение			Диапазон значений			Временные границы кластера
	1	2	3	1	2	3	
0	128	117	11	197	198	15	00:00:00 - 05:15:00
1	613	481	49	876	622	58	05:15:00 - 06:15:00, 21:30:00 - 23:45:00
2	1657	1235	137	522	632	142	06:15:00 - 07:15:00
3	2477	2407	419	187	568	172	07:15:00 - 08:15:00, 16:45:00 - 18:00:00
4	2404	2462	294	316	460	95	08:15:00 - 09:15:00
5	1981	2010	193	583	797	142	09:15:00 - 16:45:00, 18:00:00 - 19:30:00
6	1224	1391	134	648	675	55	19:30:00 - 21:30:00

Рисунок 3.3 – Таблица со статистикой кластеров

Для интеграции с веб-интерфейсом графики преобразуются в формат base64 без сохранения во временные файлы, что обеспечивает независимость от файловой системы и удобную передачу изображений через API. Табличный отчёт формируется в виде валидной HTML-разметки, готовой к встраиванию в интерфейс пользователя.

Проверка работоспособности модуля проводилась как на синтетических, так и на реальных данных, охватывая различные сценарии: от малых выборок с двумя кластерами до более сложных конфигураций с десятками сегментов и несколькими направлениями движения. Отдельное внимание уделялось корректности отображения границ, соответствию цветовой кодировки меткам кластеров и устойчивости визуализации при масштабировании данных.

Адаптивное размещение подписей прошло проверку в условиях плотной сегментации и сильных колебаний значений, что подтвердило его эффективность в сохранении читаемости графиков. Для таблиц проверялась целостность структуры, правильность формирования временных интервалов и корректность представления данных при изменении числа направлений.

Измерения производительности показали линейную масштабируемость алгоритмов относительно объёма входных данных. Модуль успешно прошёл интеграционное тестирование: выходные форматы корректно обрабатываются другими компонентами системы, изображения стабильно передаются через API, а HTML-таблицы без ошибок встраиваются в пользовательский интерфейс.

Результаты подтвердили соответствие модуля требованиям проекта и его готовность к эксплуатации в составе системы анализа транспортных временных рядов.

3.2.4 Комплексная оценка разработанной системы

В результате выполненной работы была создана полнофункциональная система анализа временных данных транспортной интенсивности, объединяющая современные технологии веб-разработки, серверной обработки данных и алгоритмов машинного обучения. Система демонстрирует высокую степень интеграции между компонентами и обеспечивает решение поставленных задач сегментации временных рядов с применением генетических алгоритмов.

Архитектурные решения, основанные на разделении клиентской и серверной частей, позволили создать масштабируемое и поддерживаемое решение. Веб-интерфейс, реализованный на чистых веб-технологиях без использования сторонних фреймворков, обеспечивает интуитивно понятное пошаговое взаимодействие с системой. Асинхронная обработка данных на серверной стороне гарантирует отзывчивость интерфейса даже при работе с большими объёмами информации.

Модуль обработки входных данных успешно справляется с различными форматами файлов – от стандартных CSV до сложноструктурированных PDF-документов. Автоматическое извлечение табличных данных из PDF с помощью библиотеки Camelot существенно расширяет применимость системы в условиях разнообразных источников данных. Реализованные механизмы очистки, интерполяции и структурирования данных обеспечивают высокое качество входной информации для последующего анализа.

Центральный аналитический модуль, построенный на основе генетического алгоритма, продемонстрировал высокую эффективность в задачах оптимального разбиения временных рядов на сегменты. Многокритериальная функция оценки, учитывающая различия между сегментами, равномерность их размеров и внутреннюю однородность, позволяет получать стабильные и воспроизводимые результаты. Дополнительная иерархическая кластеризация обеспечивает объединение схожих сегментов, что повышает интерпретируемость результатов анализа.

Модуль формирования отчётных данных создаёт наглядные визуализации и структурированные табличные отчёты, адаптированные для различных задач – от оперативного анализа до подготовки презентационных материалов. Адаптивное размещение аннотаций на графиках и логическая группировка данных в таблицах обеспечивают высокое качество представления результатов.

Комплексное тестирование системы подтвердило её надёжность, производительность и готовность к практическому применению. Все компоненты демонстрируют устойчивую работу как в изолированном режиме, так и в составе интегрированного решения. Система успешно обрабатывает реальные данные различного объёма и структуры, обеспечивая стабильные результаты анализа при времени обработки, приемлемом для практических задач.

Реализованная система полностью соответствует поставленным техническим требованиям и готова к развёртыванию в производственной среде для решения задач анализа транспортных потоков и других типов временных данных с аналогичной структурой.

4 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

Задачей данного дипломного проекта является разработка интеллектуальной системы прогнозирования и планирования интенсивности движения, предназначенной для автоматизированного анализа транспортных данных и формирования рекомендаций по светофорному регулированию на основе выявленных закономерностей в транспортных потоках.

Разработка программного продукта предусматривает проведение практически всех стадий проектирования и относится ко второй группе сложности.

Последовательность расчетов:

– Расчёт объёма функций программного модуля

– Расчёт полной себестоимости программного продукта

– Расчёт цены и прибыли по программному продукту.

4.1 Расчёт объёма функций программного модуля

Общий объём ПО определяется по формуле (4.1) исходя из объёма функций, реализуемых программой.

$$V_0 = \sum_{i=0}^{n} V_i \ , \qquad (4.1)$$

где V_0 – общий объём ПО;

V_i – объём функций ПО;

n – общее число функций.

В том случае, когда на стадии технико-экономического обоснования проекта невозможно рассчитать точный объём функций, то данный объём может быть получен на основании прогнозируемой оценки имеющихся фактических данных по аналогичным проектам, выполненным ранее, или применением нормативов по каталогу функций.

По каталогу функций на основании функций разрабатываемого ПО определяется общий объём ПО. Также на основе зависимостей от организационных и технологических условий, был скорректирован объём на основе экспертных оценок.

Уточнённый объём ПО (V_y) определяется по формуле (4.2).

$$V_y = \sum_{i=0}^{n} V_{yi} \ , \qquad (4.2)$$

где V_{yi} – уточнённый объём отдельной функции в строках исходного кода.

Перечень и объём функций ПО приведён в таблице 4.1.

Таблица 4.1 – Перечень и объём функций программного обеспечения

№ функции	Наименование (содержание) функции	Объём функции строк исходного кода	
		По каталогу V_y	Уточнённый V_{yi}
101	Организация ввода информации	130	130
104	Обработка входного заказа и формирование таблиц	630	213
303	Обработка файлов	1050	280
305	Формирование файла	2130	15
506	Обработка ошибочных и сбойных ситуаций	1540	80
507	Обеспечение интерфейса между компонентами	1680	170
701	Математическая статистика и прогнозирование	3780	580
702	Расчет показателей	420	217
707	Графический вывод результатов	420	220

С учётом информации, указанной в таблице 4.1, уточнённый объём ПО составил 1905 строк кода вместо предполагаемого количества 11780.

4.2 Расчёт полной себестоимости программного модуля

Стоимостная оценка программного средства у разработчика предполагает составление сметы затрат, которая включает следующие статьи расходов:

– отчисления на социальные нужды ($Р_{соц}$);

– материалы и комплектующие изделия ($Р_м$);

– спецоборудование ($Р_с$);

– машинное время ($Р_{мв}$);

– расходы на научные командировки ($Р_{нк}$);

– прочие прямые расходы ($Р_{пр}$);

– накладные расходы ($Р_{нр}$);

– затраты на освоение и сопровождение программного средства ($Р_о$ и $Р_{со}$).

Полная себестоимость ($С_п$) разработки программного продукта рассчитывается как сумма расходов по всем статьям с учётом рыночной стоимости аналогичных продуктов.

Основной статьёй расходов на создание программного продукта является заработная плата проекта (основная и дополнительная) разработчиков (исполнителей) $(ЗП_{осн} + ЗП_{доп})$, в число которых принято включать инженеров-программистов, руководителей проекта, системных архитекторов, дизайнеров, разработчиков баз данных, Web-мастеров и других специалистов, необходимых для решения специальных задач в команде.

Расчёт заработной платы разработчиков программного продукта начинается с определения:

– продолжительности времени разработки ($Ф_{рв}$), которое устанавливается студентом экспертным путём с учётом сложности, новизны программного обеспечения и фактически затраченного времени. В данном дипломном проекте $Ф_{рв}$ – 2 месяца;

– количества разработчиков программного обеспечения.

Заработная плата разработчиков определятся как сумма основной и дополнительной заработной платы всех исполнителей.

Основная заработная плата каждого исполнителя определяется по формуле (4.3).

$$ЗП_{осн} = Т_{ст.1р} \cdot \frac{Т_к}{Ф_{эфф.р.в.}} \cdot Ф_{рв} \cdot К_{пр} , \qquad (4.3)$$

где $Т_{ст.1р}$ – базовая ставка 270 рублей, утверждённая согласно ЕТС РБ на дату написания дипломного проекта);

$Т_к$ – тарифный коэффициент согласно разряду исполнителя;

$Ф_{эфф.р.в.}$ – среднее количество рабочих дней;

$Ф_{рв}$ – фонд рабочего времени исполнителя (продолжительность разработки программного модуля, дни);

$К_{пр}$ – коэффициент премии, $К_{пр} = 1,5$.

Рассчитаем основную заработную плату инженера-программиста и техника-программиста согласно формуле (4.3). Тарифный коэффициент согласно 11 разряду инженера-программиста $Т_к = 1,91$. Продолжительность разработки программного продукта – 44 дня.

Дополнительная заработная плата каждого исполнителя ($Н_{доп.зп.}$ – 20%). Она рассчитывается от основной заработной платы по формуле (4.4).

$$ЗП_{доп} = ЗП_{осн}\frac{Н_{доп.зп}}{100\%}\ , \qquad (4.4)$$

Результаты вычислений внесём в таблицу 4.2.

Таблица 4.2 – Расчёт заработной платы

Категория работников	Разряд	Тарифные коэффициент (T_k)	$Ф_{эфф.р.в.}$, (дн.)	Коэффициент премии (K_p)	$Т_к$ (час.)	Заработная плата, бел.руб.		
						Основная	Дополнительная	Всего
Техник-программист	11	1.91	44	1.5	8	1547.1	309.42	1856.52
Всего	-	-	-	-	-	1547.1	309.42	1856.52

Таким образом, как видно из таблицы, заработная плата инженера-программиста составляет 1856.52 бел. руб.

Отчисления на социальные нужды ($Р_{соц}$) определяются по формуле (4.5) в соответствии с действующим законодательством по нормативу (29% - отчисления в ФСЗН + 6% отчисления по обязательному страхованию):

$$Р_{соц} = (ЗП_{осн} + ЗП_{доп}) \cdot \frac{35}{100}, \quad (4.5)$$

Расходы по статье «Спецоборудование» ($Р_с$) включает затраты на приобретение технических и программных средств специального назначения, необходимых для разработки методического пособия, включая расходы на проектирование, изготовление, отладку и другое.

В данном дипломном проекте для разработки интеллектуальной системы прогнозирования и планирования интенсивности движения приобретение какого-либо специализированного оборудования не предусматривалось. Так как спецоборудование не было приобретено, данная статья не рассчитывается.

По статье «Материалы и комплектующие изделия» ($Р_м$) отражаются расходы на бумагу, картридж и красящие ленты для принтера, необходимые для разработки ПП. Норма расхода материалов в суммарном выражении определяются в расчете на 100 строк исходного кода. В данном дипломном проекте не рассчитывается.

$Н_м$ – норма расхода материалов в расчете на 100 строк исходного кода программного продукта. Принимаем равной 0.4 бел. руб.

По статье «Машинное время» ($Р_{мв}$) включают оплату машинного времени, необходимого для разработки и отладки программного продукта. Они определяются в машиночасах по нормативам на 100 строк исходного кода машинного времени. $Р_{мв}$ определяется по формуле (4.6).

$$Р_{мв} = Ц_{мвi} \cdot \frac{V_0}{100} \cdot Н_{мв}, \quad (4.6)$$

где $Ц_{мвi}$ – цена одного машинного часа (50 бел. руб.);

V_0 – уточнённый общий объём машинного кода. Согласно расчётам пункта (4.2) данное значение равно 1905 строк;

$Н_{мв}$ – норматив расхода машинного времени на отладку 100 строк кода, машино-часов. Принимается в размере 0,6.

Расходы по статье «Научные командировки» ($Р_{нк}$) берутся либо по смете научных командировок, разрабатываемой на предприятии, либо в процентах от основной заработ-

ной платы исполнителей (10-15%). Так как в данном проекте научные командировки не предусмотрены, данная статься не рассчитывается.

Расходы по статье «Прочие затраты» ($Р_{пр}$) включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются по нормативу в процентах к основной заработной плате исполнителей. Так как специальная научно-техническая информация и специальная литература не приобреталась, то данная статься не рассчитывается.

Затраты по статье «Накладные расходы» ($Р_{нр}$) связаны с содержанием вспомогательных хозяйств, а также с расходами на общехозяйственные нужды. Определяется по нормативу в процентах к основной заработной плате по формуле (4.7).

$$Р_{нр} = \frac{Н_{нр}}{100} \cdot ЗП_{осн} , \qquad (4.7)$$

где $Н_{нр}$ – норматив накладных расходов, в данном дипломном проекте норматив накладных расходов равен 40%.

Сумма вышеперечисленных расходов по статься на программный продукт служит исходной базой для расчёта затрат на освоение и сопровождение программного продукта. Они рассчитываются по формуле (4.8).

$$СЗ = ЗП_{осн} + ЗП_{доп} + Р_{соц} + Р_{м} + Р_{мв} + Р_{с} + Р_{нк} + Р_{пр} + Р_{нр} , \qquad (4.8)$$

Затраты на освоение программного продукта ($Р_{о}$). Организация-разработчик участвует в освоении программного продукта и несёт соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Затраты на освоение определяются по установленному нормативу от суммы затрат по формуле (4.9).

$$Р_{о} = СЗ \cdot \frac{Н_{о}}{100} , \qquad (4.9)$$

где СЗ – сумма вышеперечисленных расходов по статьям на разработку программного продукта;

$Н_{о}$ – установленный норматив затрат на освоение. Для данного дипломного проекта принимается равной 5%.

Затраты на сопровождение программного продукта ($Р_{со}$). Организация-разработчик осуществляет сопровождение программного продукта и несёт расходы, которые оплачиваются заказчиком в соответствии с договором и сметой на сопровождение. Эти расходы рассчитываются по формуле (4.10).

$$Р_{со} = СЗ \cdot \frac{Н_{со}}{100}, \qquad (4.10)$$

где СЗ – сумма вышеперечисленных расходов по статьям на разработку программного продукта;

$Н_{со}$ – установленный норматив затрат на сопровождение программного продукта. Для данного дипломного проекта принимается равной 5%.

Полная себестоимость (СП) разработки программного продукта рассчитывается как сумма расходов по всем статьям. Она определяется по формуле (4.11).

$$СП = СЗ + Р_{о} + Р_{со}, \qquad (4.11)$$

Результаты вычислений занесём в таблицу 4.3.

Таблица 4.3 – Себестоимость программного продукта.

Наименование статей затрат	Норматив, %	Сумма затрат, бел.руб.
Заработная плата	-	1856.52
Основная заработная плата	-	1547.1
Дополнительная заработная плата	-	309.42
Отчисления на социальные нужды	35%	649.78
Спецоборудование	Не применялось	-
Материалы	Не применялись	-
Машинное время	-	571.5
Научные командировки	Не планировались	-
Прочие затраты	Не применялись	-
Накладные расходы	40%	618.84
Сумма затрат на разработку программного продукта	-	3696.64
Затраты на освоение	5%	184.83
Затраты на сопровождение	5%	184.83
Полная себестоимость	-	4066.3

В результате всех расчётов полная себестоимость программного продукта составила 4066,30 бел.руб.

4.3 Расчёт цены и прибыли по программному продукту

Для определения цены программного продукта необходимо рассчитать плановую прибыль, которая рассчитывается по формуле (4.12).

$$П = СП \cdot \frac{R}{100}, \quad (4.12)$$

где СП – полная себестоимость программного модуля, бел. руб;

R – уровень рентабельности программного модуля. В данном дипломном проекте уровень рентабельности равен 30%.

После расчета прибыли от реализации по формуле (4.13) определяется прогнозируемая цена программного продукта без налогов.

$$Ц_п = СП + П, \quad (4.13)$$

где СП – полная себестоимость программного модуля управления ESP32 на базе JavaScript, бел. руб;

П – плановая прибыль от реализации программного модуля, бел. руб.

Отпускная цена (цена реализации) программного продукта включает налог на добавленную стоимость и рассчитывается по формуле (4.14).

$$Ц_о = СП + П + НДС_{пп}, \quad (4.14)$$

где СП – полная себестоимость программного модуля управления ESP32 на базе JavaScript, бел. руб;

П – плановая прибыль от реализации программного модуля, бел. руб.;

$НДС_{пп}$ – налог на добавленную стоимость.

Для данного программного продукта $НДС_{пп}$ рассчитывается по формуле (4.15).

$$НДС_{пп} = Ц_п \cdot \frac{НДС}{100}, \quad (4.15)$$

где $Ц_п$ – прогнозируемая цена, бел. руб.;

НДС – налог на добавленную стоимость. В настоящее время он составляет 20%.

Прибыль от реализации программного продукта за вычетом налога на прибыль является чистой прибылью (ПЧ). Чистая прибыль остаётся организации-разработчику и

представляет собой экономический эффект от создания нового программного продукта. Она рассчитывается по формуле (4.16).

$$ПЧ = П \cdot \left(1 - \frac{Н_п}{100}\right), \quad (4.16)$$

где П – плановая прибыль от реализации программного модуля, бел. руб.;

$Н_п$ – ставка налога на прибыль. В настоящее время он равен 20%.

Все расчёты цены и прибыли по программному продукту сведены в таблицу 4.4.

Таблица 4.4 – Расчёт отпускной цены и чистой прибыли программного модуля

Наименование статей затрат	Норматив, %	Сумма затрат, бел. руб.
Полная себестоимость	-	4066.3
Прибыль	30	1219.89
Цена без НДС	-	5286.19
НДС	20	1057.24
Отпускная цена	-	6343.43
Налог на прибыль	20	243.98
Чистая прибыль	-	975.91

В ходе произведенных расчетов определены основные экономические показатели: полная себестоимость – 4066.30 бел.руб.; прогнозируемая цена – 5286.19 бел. руб.; чистая прибыль – 975.91 бел.руб.

Разработанная система имеет малое количество конкурентов с более высокими ценами на их услуги. Таким образом, рассчитанная отпускная цена на программный продукт, разрабатываемой в рамках данного дипломного проекта, является конкурентоспособной. При расчете цены учтены отчисления в фонд социальной защиты, а также налоги, необходимые к уплате. К конечному итогу получаем окончательную цену продукта, равную 6343.43 белорусских рубля.

5 ЭНЕРГОСБЕРЕЖЕНИЕ

Вопрос энергосбережения становится всё более актуальным в условиях роста потребления ресурсов в городском хозяйстве и стремления к устойчивому развитию. Одной из сфер, где существует существенный потенциал для повышения энергоэффективности, является организация дорожного движения, в частности – работа светофорных объектов. Светофоры, особенно расположенные на крупных перекрёстках, работают круглосуточно и потребляют электрическую энергию как на управление сигналами, так и на обеспечение вспомогательных функций, включая питание контроллеров, модемов, датчиков и подсветки.

На первый взгляд энергопотребление одного светофорного объекта может показаться незначительным, однако в масштабах города, где количество регулируемых перекрёстков может исчисляться сотнями, общее энергопотребление становится весьма ощутимым. Более того, неэффективные режимы работы светофоров могут приводить к косвенным энергетическим потерям: увеличению времени простоя транспортных средств с работающими двигателями, частым остановкам и ускорениям, росту расхода топлива и, как следствие, снижению общей энергоэффективности транспортной системы.

Одним из путей снижения прямого и косвенного энергопотребления является рационализация режимов светофорного регулирования. Если распределение фаз и длительность циклов светофора не соответствуют текущей или характерной для определённого времени суток интенсивности движения, это приводит к ненужным задержкам и снижению пропускной способности, заставляя транспортные средства расходовать больше топлива. Кроме того, чрезмерно длинные зелёные фазы при отсутствии потока или, наоборот, частая смена сигналов в часы с высокой нагрузкой создают неэффективные условия для проезда перекрёстка и увеличивают число остановок.

Для достижения энергосбережения необходимо обеспечить согласованность работы светофорных объектов с реальными транспортными потоками. В этом контексте современные интеллектуальные системы, способные автоматически анализировать исторические данные и выявлять закономерности в трафике, становятся эффективным инстру-

ментом планирования. Они позволяют точно настраивать режимы работы светофоров в зависимости от времени суток, дня недели и уровня загруженности, минимизируя как ненужное электропотребление самих объектов, так и косвенные потери, связанные с простоем и неравномерным движением транспорта.

Предлагаемая интеллектуальная система прогнозирования и планирования интенсивности движения ориентирована в том числе на достижение целей энергосбережения за счёт повышения эффективности работы светофорных объектов. Анализируя временные ряды транспортной активности, система способна автоматически выявлять периоды, в течение которых отдельные направления движения характеризуются низкой интенсивностью. В таких случаях она предлагает сократить длительность зелёных фаз, тем самым уменьшая общее время включения сигнальных ламп и снижая прямое энергопотребление оборудования.

Кроме того, система формирует интервалы с устойчивыми характеристиками потока, что позволяет избежать излишне частых переключений режимов и, как следствие, уменьшить нагрузку на контроллеры и снизить энергозатраты, связанные с их работой. Более точная настройка фаз светофора в соответствии с реальной загруженностью позволяет сократить число ненужных остановок и запусков транспортных средств, что в свою очередь снижает расход топлива и связанные с ним энергетические потери. Особенно заметный эффект достигается в ночные и малонагруженные часы, когда система может предложить более длительные и стабильные интервалы работы светофоров, исключающие лишние циклы мигания и переходов, тем самым снижая нагрузку на оборудование.

Таким образом, применение разработанной интеллектуальной системы в городских условиях, где ещё не внедрены адаптивные светофорные комплексы, обеспечивает существенное повышение энергоэффективности. За счёт автоматизированного анализа данных и обоснованного планирования режимов светофорного регулирования система позволяет минимизировать избыточное энергопотребление и внести вклад в устойчивое развитие транспортной инфраструктуры.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была рассмотрена актуальная задача повышения эффективности работы регулируемых перекрёстков в условиях городской транспортной сети. Изучение предметной области показало, что традиционные методы светофорного регулирования, основанные на ручной настройке режимов, обладают существенными недостатками – высокой трудоёмкостью, зависимостью от субъективных оценок и низкой гибкостью при изменении транспортной ситуации.

Для решения поставленной цели была разработана интеллектуальная система прогнозирования и планирования интенсивности движения, которая автоматизирует процесс анализа исторических данных о транспортных потоках. Система обеспечивает автоматическое выделение временных интервалов с различной загруженностью и формирование рекомендаций по режимам работы светофоров. Пользователь получает наглядную карту времени с визуализацией смены нагрузки в течение суток и сводные статистические данные для проектирования и оптимизации светофорных циклов.

Результаты работы продемонстрировали возможность значительного повышения эффективности планирования в области организации дорожного движения. Применение разработанной системы позволяет сократить влияние человеческого фактора и повысить масштабируемость процесса на большое количество перекрёстков. За счёт точного соответствия режимов реальной интенсивности движения достигается сокращение времени ожидания участников движения и снижение энергопотребления светофорных объектов.

Созданная интеллектуальная система может быть успешно применена в качестве эффективного инструмента поддержки принятия решений. Её внедрение способно повысить общую эффективность функционирования улично-дорожной сети, сократить заторы и задержки, а также внести вклад в энергосбережение и устойчивое развитие транспортной инфраструктуры.

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

SCOOT – Split Cycle Offset Optimization Technique.

SCATS – Sydney Coordinated Adaptive Traffic System.

DBSCAN – Density-Based Spatial Clustering of Applications with Noise.

GMM – Gaussian Mixture Model.

EM – Expectation-Maximization.

PMM – Poisson Mixture Model.

FINCH – Fast and INterpretable Clustering of High-dimensional data.

TW-FINCH – Time Windowed FINCH.

ClaSP – Closed Sequential Patterns algorithm.

Ptr-Net – Pointer Network.

KernelCPD – Kernel Change Point Detection.

WSS – Window Sliding Segmentation.

PELT – Pruned Exact Linear Time.

AIC – Akaike Information Criterion.

BIC – Bayesian Information Criterion.

REST – Representational State Transfer.

API – Application Programming Interface.

HTML – HyperText Markup Language.

CSS – Cascading Style Sheets.

WSGI – Web Server Gateway Interface.

ASGI – Asynchronous Server Gateway Interface.

HTTP – Hypertext Transfer Protocol.

CSV – Comma-Separated Values.

PDF – Portable Document Format.

JSON – JavaScript Object Notation.

DEAP – Distributed Evolutionary Algorithms in Python.

ПО – программное обеспечение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Time Series Clustering // AI Sia [Электронный ресурс]. – 2023. – Режим доступа: https://sia-ai.medium.com/time-series-clustering-b84bcaaa63ac. – Дата доступа: 04.05.2025.
2. Change Point Detection in Time Series // AI Dataman in [Электронный ресурс]. – 2023. – Режим доступа: https://medium.com/dataman-in-ai/change-point-detection-in-time-series-c0b507752889. – Дата доступа: 04.05.2025.
3. SCOOT - Intelligent Signal Control // Software TRL [Электронный ресурс]. – 2025. – Режим доступа: https://trlsoftware.com/software/intelligent-signal-control/scoot/. – Дата доступа: 04.05.2025.
4. SCOOT Traffic Control System // Authority Greater London [Электронный ресурс]. – 2024. – Режим доступа: https://www.london.gov.uk/who-we-are/what-london-assembly-does/questions-mayor/find-an-answer/scoot-traffic-control-system. – Дата доступа: 04.05.2025.
5. Mercer SCOOT // Transportation Seattle Department of [Электронный ресурс]. – 2025. – Режим доступа: https://www.seattle.gov/transportation/projects-and-programs/programs/technology-program/mercer-scoot. – Дата доступа: 04.05.2025.
6. SCATS - Sydney Coordinated Adaptive Traffic System // NSW Transport for [Электронный ресурс]. – 2025. – Режим доступа: https://www.scats.nsw.gov.au/home. – Дата доступа: 04.05.2025.
7. Signal Coordination and Automation (SCATS) // Transport Department of, Planning Victoria [Электронный ресурс]. – 2024. – Режим доступа: https://transport.vic.gov.au/business/road-and-traffic-management/traffic-lights/signal-coordination-and-automation/scats. – Дата доступа: 04.05.2025.
8. System Sterowania Ruchem // S.A. Sprint [Электронный ресурс]. – 2025. – Режим доступа: https://sprint.pl/en/services/system-sterowania-ruchem. – Дата доступа: 04.05.2025.

9. DCC SCATS Detector Volume Jan-Jun 2024 // Dublin Smart [Электронный ресурс]. – 2024. – Режим доступа: https://data.smartdublin.ie/dataset/dcc-scats-detector-volume-jan-jun-2024. – Дата доступа: 04.05.2025.
10. K-Means Clustering // IBM [Электронный ресурс]. – 2024. – Режим доступа: https://www.ibm.com/think/topics/k-means-clustering. – Дата доступа: 04.05.2025.
11. DBSCAN Clustering Algorithm // DataCamp [Электронный ресурс]. – 2024. – Режим доступа: https://www.datacamp.com/tutorial/dbscan-clustering-algorithm. – Дата доступа: 04.05.2025.
12. Understanding Gaussian Mixture Models: A Comprehensive Guide // Olamendy Juan Carlos [Электронный ресурс]. – 2024. – Режим доступа: https://medium.com/@juanc.olamendy/understanding-gaussian-mixture-models-a-comprehensive-guide-df30af59ced7. – Дата доступа: 04.05.2025.
13. Mixture Models Beyond Gaussians // DataCamp [Электронный ресурс]. – 2024. – Режим доступа: https://campus.datacamp.com/courses/mixture-models-in-r/mixture-models-beyond-gaussians?ex=6. – Дата доступа: 04.05.2025.
14. Agglomerative Clustering: Theory and Practice // Habr [Электронный ресурс]. – 2024. – Режим доступа: https://habr.com/ru/articles/798331/. – Дата доступа: 04.05.2025.
15. Efficient Parameter-free Clustering Using First Neighbor Relations // Sarfraz M. Saquib, Sharma Vivek, Stiefelhagen Rainer [Электронный ресурс]. – 2019. – Режим доступа: https://openaccess.thecvf.com/content_CVPR_2019/papers/Sarfraz_Efficient_Parameter-Free_Clustering_Using_First_Neighbor_Relations_CVPR_2019_paper.pdf.
16. Temporally-Weighted Hierarchical Clustering for Unsupervised Action Segmentation // Sarfraz M. Saquib, Murray Naila, Sharma Vivek, Diba Ali, Van Gool Luc, Stiefelhagen Rainer [Электронный ресурс]. – 2021. – Режим доступа: https://arxiv.org/pdf/2103.11264.

17. ClaSP: Parameter-Free Time Series Segmentation // Ermshaus Arik, Schäfer Patrick, Leser Ulf [Электронный ресурс]. – 2024. – Режим доступа: https://arxiv.org/abs/2411.08397.
18. State-of-the-Art: Change Point Detection on TSSB // Code Papers With [Электронный ресурс]. – 2024. – Режим доступа: https://paperswithcode.com/sota/change-point-detection-on-tssb. – Дата доступа: 04.05.2025.
19. Pointer Networks: What Are They? // Kierszbaum Samuel [Электронный ресурс]. – 2024. – Режим доступа: https://kierszbaumsamuel.medium.com/pointer-networks-what-are-they-c3cb68fae076. – Дата доступа: 04.05.2025.
20. Kernel Change Point Detection Reference // Borelli Centre [Электронный ресурс]. – 2024. – Режим доступа: https://centre-borelli.github.io/ruptures-docs/code-reference/detection/kernelcpd-reference/. – Дата доступа: 04.05.2025.
21. Binary Segmentation Change Point Detection Reference // Borelli Centre [Электронный ресурс]. – 2024. – Режим доступа: https://centre-borelli.github.io/ruptures-docs/code-reference/detection/binseg-reference/. – Дата доступа: 04.05.2025.
22. Bottom-Up Change Point Detection Reference // Borelli Centre [Электронный ресурс]. – 2024. – Режим доступа: https://centre-borelli.github.io/ruptures-docs/code-reference/detection/bottomup-reference/. – Дата доступа: 04.05.2025.
23. Window-Based Change Point Detection Reference // Borelli Centre [Электронный ресурс]. – 2024. – Режим доступа: https://centre-borelli.github.io/ruptures-docs/code-reference/detection/window-reference/. – Дата доступа: 04.05.2025.
24. PELT Change Point Detection Reference // Borelli Centre [Электронный ресурс]. – 2024. – Режим доступа: https://centre-borelli.github.io/ruptures-docs/code-reference/detection/pelt-reference/. – Дата доступа: 04.05.2025.
25. Dynamic Programming Change Point Detection Reference // Borelli Centre [Электронный ресурс]. – 2024. – Режим доступа: https://centre-borelli.github.io/ruptures-docs/code-reference/detection/dynp-reference/. – Дата доступа: 04.05.2025.

26. HTML: HyperText Markup Language // Docs MDN Web [Электронный ресурс]. – 2025. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTML. – Дата доступа: 04.05.2025.
27. CSS Documentation // Docs MDN Web [Электронный ресурс]. – 2025. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS. – Дата доступа: 04.05.2025.
28. JavaScript Documentation // Docs MDN Web [Электронный ресурс]. – 2025. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript. – Дата доступа: 04.05.2025.
29. Всё о User Flow: последовательность взаимодействия с интерфейсом // Team Uprock [Электронный ресурс]. – 2023. – Режим доступа: https://www.uprock.ru/education/vse-o-user-flow. – Дата доступа: 29.05.2025.
30. Юзабилити: эвристики Нильсена // Wikipedia [Электронный ресурс]. – 2025. – Режим доступа: https://en.wikipedia.org/wiki/Usability. – Дата доступа: 29.05.2025.
31. Как уменьшить когнитивную нагрузку в UI-дизайне // Team Pixcap [Электронный ресурс]. – 2024. – Режим доступа: https://pixcap.com/ru/blog/cognitive-load-ui. – Дата доступа: 29.05.2025.
32. Адаптивный веб-дизайн // Wikipedia [Электронный ресурс]. – 2025. – Режим доступа: hhttps://en.wikipedia.org/wiki/Responsive_web_design. – Дата доступа: 29.05.2025.
33. Python About // Foundation Python Software [Электронный ресурс]. – 2025. – Режим доступа: https://www.python.org/about/. – Дата доступа: 04.05.2025.
34. FastAPI Documentation // Contributors FastAPI [Электронный ресурс]. – 2025. – Режим доступа: https://fastapi.tiangolo.com/. – Дата доступа: 04.05.2025.
35. Swagger - API Development for Everyone // Contributors Swagger [Электронный ресурс]. – 2025. – Режим доступа: https://swagger.io/. – Дата доступа: 04.05.2025.

36. Uvicorn Introduction // Contributors Uvicorn [Электронный ресурс]. – 2025. – Режим доступа: https://www.uvicorn.org/. – Дата доступа: 04.05.2025.
37. What is a RESTful API? // Contributors RESTful API [Электронный ресурс]. – 2025. – Режим доступа: https://restfulapi.net/. – Дата доступа: 04.05.2025.
38. Camelot Documentation // Contributors Camelot [Электронный ресурс]. – 2025. – Режим доступа: https://camelot-py.readthedocs.io/en/master/. – Дата доступа: 04.05.2025.
39. Tabula Documentation // Contributors Tabula [Электронный ресурс]. – 2025. – Режим доступа: https://tabula-py.readthedocs.io/en/latest/. – Дата доступа: 04.05.2025.
40. PyMuPDF Documentation // Contributors PyMuPDF [Электронный ресурс]. – 2025. – Режим доступа: https://pymupdf.readthedocs.io/en/latest/. – Дата доступа: 04.05.2025.
41. DEAP Documentation // Contributors DEAP [Электронный ресурс]. – 2025. – Режим доступа: https://deap.readthedocs.io/en/master/. – Дата доступа: 04.05.2025.
42. Base64 // Wikipedia [Электронный ресурс]. – 2025. – Режим доступа: https://en.wikipedia.org/wiki/Base64. – Дата доступа: 29.05.2025.

ПРИЛОЖЕНИЕ А. ТЕКСТ ПРОГРАММЫ