КРАММЕРТИ

ОПИСАНИЕ И РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа для ЭВМ «Краммерти. Модуль прогнозирования временных рядов»

Версия v.2025.1.01 май 2025г.

I ОПИСАНИЕ

«Краммерти. Модуль прогнозирования временных рядов»

1. Предназначение:


РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025663164

Краммерти. Модуль прогнозирования временных рядов

Правообладатель: ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ "КРАММЕРТИ" (RU)

Автор(ы): Клаузер Сергей Викторович (RU)

Заявка № 2025661834
Дата поступления 16 мая 2025 г.
Дата государственной регистрации в Реестре программ для ЭВМ 27 мая 2025 г.

Руководитель Федеральной службы по интеллектуальной собственности Ю.С. Зубов

Программа для ЭВМ «Краммерти. Модуль прогнозирования временных рядов» (далее система) предназначена для автоматизированного краткосрочного прогнозирования временных рядов метеорологических параметров (и не только). Разработка данного модуля произведена в рамках развития проекта по прогнозированию температуры воздуха и поставляется как в составе «Краммерти. Подсистема метеомониторинга» так и как отдельный программный продукт в зависимости от Задач.

Программа разработана для нужд оперативного мониторинга и планирования, предоставляя как основной прогноз, так и расчетную температуру "по ощущениям" (с учетом ветро-холодового индекса). Программа интегрирует данные из внутреннего и/или внешнего API исторических наблюдений необходимых параметров (метеостанции, и иное имеющееся на балансе Заказчика измерительное оборудование и т.д. временные ряды) а также учитываются внешние прогностические данные сторонних сервисов (если это необходимо).

Применяемая модель: LSTM (Long Short Term Memory) — рекуррентная нейронная сеть глубокого обучения, которая способна улавливать закономерности в данных временных рядов и использоваться для прогнозирования будущего тренда данных.

2. Решаемые задачи и функционал системы:

Система автоматизирует полный цикл прогнозирования включая:

- **Сбор и обновление данных:**
  - Автоматическая загрузка исторических данных по заданным параметрам (напр., температура воздуха) с указанных метеостанций через внутренний API (с обработкой пагинации, кэшированием, повторными попытками и обработкой ошибок).
  - Проверка свежести локально сохраненных исторических данных и дозагрузка только новых записей для минимизации трафика и времени.
  - Автоматическая загрузка актуальных прогнозов (температура, скорость ветра и/или иных доступных данных временных рядов) от внешних прогностических сервисов.
  - Проверка свежести локальных копий данных от прогностических сервисов для сокращения запросов к внешнему API.

- **Хранение данных:** Сохранение загруженных исторических данных и прогнозов в локальные CSV-файлы для последующего использования Системой.
- **Предобработка данных:**
    - Очистка данных: Удаление выбросов методом IQR.
    - Ресемплинг: Приведение данных к единой временной сетке (по умолчанию - часовой).
    - Инжиниринг признаков: Создание лаговых, календарных (час, день года, месяц, день недели и т.д.) и циклических (sin/cos) признаков для модели LSTM.
    - Масштабирование: Нормализация данных с помощью RobustScaler для подготовки к подаче в нейронную сеть.
- **Подбор гиперпараметров модели нейросети LSTM методом HPO (Hyper Parameter Optimization):**
    - Использование фреймворка Optuna для автоматического поиска оптимальной комбинации гиперпараметров LSTM-модели (количество нейронов, скорость обучения, функции активации, параметры регуляризации, длина входной последовательности и т.д.) на основе заданных диапазонов в hyperparameters.json.
    - Применение (на этапе HPO) функции потерь, которая учитывает отклонение от фактических данных.
- **Обучение модели:**
    - Обучение финальной LSTM-модели на предобработанных исторических данных с использованием лучших гиперпараметров, найденных Optuna.
    - Использование стандартной функции потерь Huber для обучения финальной модели (не зависит от сторонних прогностических данных на этом этапе).
    - Применение механизма EarlyStopping для предотвращения переобучения.
- **Сохранение и версионирование модели:**
    - Сохранение обученной модели Keras (.keras), соответствующего объекта Scaler (.pkl) и метаданных (включая гиперпараметры и метрики) в JSON-файл.
    - Включение метрики SMAPE и даты в имя файла для отслеживания производительности и выбора лучшей модели.
- **Генерация прогноза:**
    - Загрузка **лучшей** (по SMAPE) сохраненной модели и скейлера для целевого параметра.
    - Генерация прогноза температуры на заданный горизонт с помощью LSTM-модели.
    - Загрузка актуального прогноза прогностических сервисов (температура и ветер).
    - **Коррекция прогноза:** контроль корректности прогноза LSTM путем его сравнения с прогнозами прогностических сервисов, регулирование расхождения при превышении заданного порога (CORRECTION_THRESHOLD), с учетом коэффициента (CORRECTION_FACTOR) и максимального ограничения (MAX_CORRECTION).

    - **Расчет Wind Chill:** Вычисление температуры "по ощущениям" на основе скорректированного прогноза температуры и прогноза скорости ветра прогностических сервисов.
- **Вывод результатов:**
    - Сохранение итогового прогноза (включая температуру и "ощущаемую" температуру) в структурированный JSON-файл с временными метками.
    - Генерация и сохранение графика, визуализирующего исторические данные, прогноз LSTM (скорректированный), прогноз "по ощущениям" и прогноз(ы) прогностического сервиса (ов).
- **Автоматизация:**
    - Использование библиотеки schedule для запуска задач по расписанию (обновление данных, переобучение модели, генерация прогноза).
    - Возможность интеграции с systemd (на Linux) для обеспечения постоянной работы и автоматического перезапуска сервисов (обновления данных/планировщика и API).
- **API для доступа к прогнозу:**
    - Предоставление простого веб-API (на базе Flask), которое по запросу отдает содержимое самого свежего сгенерированного JSON-файла с прогнозом.

4. Архитектура системы:

Модуль представляет собой модульное Python-приложение, состоящее из следующих ключевых компонентов:

- **Модули доступа к данным:**
    - **api_client.py**: взаимодействует с внутренним API исторических данных.
    - **api_server.py**: взаимодействует с внешними потребителями прогностических данных предоставляя API интеграцию.
    - **open_meteo_client.py**: Взаимодействует с источниками внешних API прогностических сервисов.
    - **data_loader.py**: загружает данные из локальных CSV-файлов.
- **Модуль предобработки:**
    - **data_preprocessing.py**: реализует шаги очистки, ресемплинга, создания признаков и масштабирования данных.
- **Модуль моделирования:**
    - **lstm_model.py**: содержит определение архитектуры LSTM-модели, логику ее компиляции (с возможностью использования кастомных или стандартных потерь/метрик), методы для обучения, предсказания, сохранения и загрузки.
    - **metrics.py**: Функции для расчета метрик качества (MAE, RMSE, SMAPE и т.д.).
- **Модуль оркестрации и логики:**
    - **main.py**: является точкой входа, инициализирует компоненты, читает конфигурацию, содержит логику HPO (Optuna), запускает процессы обучения и

прогнозирования, управляет планировщиком задач (schedule) и содержит вспомогательные функции (например, calculate_wind_chill, get_best_model).

- **Модуль API-сервера:**
  - **api_server.py**: реализует веб-сервер Flask, который предоставляет эндпойнт для получения последнего сгенерированного прогноза из JSON-файла.

- **Конфигурационные файлы:**
  - config.ini: хранит все основные настройки системы (пути, URL, токены, параметры API, расписание, параметры коррекции и т.д.).
  - hyperparameters.json: Определяет диапазоны и типы гиперпараметров для поиска лучших настроек модели LSTM нейросети фреймворком Optuna.
- **Хранилище данных/артефактов:**
  - **Папки (weather_data, models, logs, graphs, lstm_forecasts):** используются для хранения CSV-данных, обученных моделей, логов выполнения обучения, графиков и JSON-прогнозов соответственно, предметно:

    weather_data – хранение исторических данных параметров например Температуры воздуха.csv за весь возможный период доступной истории для каждого источника дифференцировано.

    models – обученные модели .keras, гиперпараметры в .json обученных моделей, скейлеры моделей .pkl.

    logs – каталог хранения базы данных оптимизации и показателей параметров при HPO (hyper parameter optimization) при обучении моделей.

    graphs – каталог хранения графиков обучения и прогнозов.

    lstm_forecasts – каталог хранения JSON файлов прогнозов которые в последующем передаются по API во внешние системы.
  - ***База данных Optuna** (.db файл в папке models): хранит историю и результаты HPO (hyper parameter optimization — это процесс настройки параметров модели для достижения оптимальной производительности. Он необходим для оценки множества вариантов дизайна и справедливого сравнения моделей.

5. Описание файлов Python (.py):

- **main.py:** Главный скрипт, "дирижер" всего оркестра. Отвечает за:
  - Чтение конфигураций (config.ini, hyperparameters.json).
  - Инициализацию основных объектов (API клиентов, загрузчиков, препроцессоров).
  - Оркестрацию первоначальной загрузки данных.
  - Запуск процесса HPO и обучения финальной модели (train_lstm_model), вызывая функции из других модулей.
  - Запуск процесса генерации прогноза (run_prediction_tasks), вызывая get_best_model и make_forecast.

    - Настройку и запуск планировщика schedule с задачами (job_*).
    - Содержит вспомогательные функции, такие как calculate_wind_chill, get_best_model, save_metrics, create_safe_filename.
- **api_client.py:** реализует класс ApiClient для взаимодействия с внутренним API. Обрабатывает авторизацию (токен), таймауты, повторные попытки (tenacity), кэширование (requests-cache), пагинацию (в async_fetch_historical_data).
- **open_meteo_client.py:** реализует класс OpenMeteoClient для взаимодействия с прогностическими сервисами API. Запрашивает прогноз температуры и ветра, обрабатывает таймауты и повторные попытки, извлекает данные (extract_forecast_data).
- **data_loader.py:** реализует класс DataLoader, отвечающий за безопасную загрузку данных из локальных CSV-файлов с использованием Pandas, обработку ошибок чтения и парсинга дат.
- **data_preprocessing.py:** реализует класс DataPreprocessor. Содержит методы для:
    - Удаления выбросов (remove_outliers).
    - Ресемплинга данных (resample_data).
    - Создания признаков (create_features), включая лаги и календарные/циклические признаки.
    - Масштабирования данных с помощью RobustScaler (fit_transform, transform, inverse_transform).
    - (Опционально) Сглаживания данных (smooth_data).
- **lstm_model.py:** реализует класс LSTMModel. Отвечает за:
    - Определение архитектуры нейронной сети LSTM с использованием TensorFlow/Keras.
    - Компиляцию модели с оптимизатором (Adam) и функцией потерь (стандартный Huber или кастомный) и метриками (стандартная MAE или MaskedMAE).
    - Метод train для обучения модели на подготовленных данных, включая использование коллбэков (EarlyStopping, TensorBoard).
    - Метод predict для генерации прогнозов на новых данных.
    - Методы save и load для сохранения/загрузки весов модели Keras.
    - Статический метод prepare_data для преобразования временного ряда в формат (X, y) для LSTM.
- **metrics.py:** содержит функции для расчета метрик качества прогноза (MAE, RMSE, MAPE, R2, SMAPE) метрики сохраняются в metrics.csv файл локально при каждом обучении моделей.
- **api_server.py:** реализует веб-сервер на Flask, который находит последний (самый свежий - сегодняшний) JSON-файл прогноза и отдает его содержимое по GET запросу.

6. Особенности системы:

- **Гибридный подход:** Ключевая особенность – комбинация data-driven модели LSTM с внешними физическими прогнозами прогностических сервисов. Это реализовано двумя способами:

    - *На этапе HPO:* позволяет найти гиперпараметры, которые делают модель более устойчивой и "осведомленной" о внешнем прогнозе. Хотя финальная модель обучается без этого, найденные параметры могут быть более робастными (валидными).
    - *На этапе постобработки (коррекция):* явная коррекция прогноза LSTM на основе прогнозов прогностических центров при сильных расхождениях, что позволяет сгладить аномальные выбросы LSTM и приблизить прогноз к более физически обоснованному.
- **Автоматический подбор гиперпараметров (Optuna):** систематический поиск оптимальных параметров модели вместо ручного подбора.
- **Адаптивное кэширование ОМ прогноза:** запрос к внешнему API ОМ выполняется только при устаревании локального файла, что снижает зависимость от внешнего сервиса, экономит лимиты запросов и ускоряет работу.
- **Автоматизированный конвейер:** полностью автоматизированный цикл от сбора данных до генерации прогноза с помощью schedule и systemd.
- **Версионирование моделей по метрике:** Сохранение моделей с включением SMAPE в имя файла позволяет легко идентифицировать и автоматически выбирать лучшую работающую модель для генерации прогнозов.
- **Расчет Wind Chill:** Добавление важного для пользователя параметра "температуры по ощущениям".

7. Этапы разработки:

- Проектирование архитектуры;
- Разработка и отладка модулей доступа к данным;
- Разработка и тестирование модуля предобработки;
- Разработка, обучение и тюнинг LSTM-модели;
- Интеграция Optuna;
- Разработка логики коррекции и Wind Chill;
- Создание планировщика и API-сервера;
- Настройка окружения и базовое развертывание.

8. Преимущества для Пользователей:

- **Автоматизированный прогноз:** Получение регулярных, обновляемых прогнозов температуры и "ощущаемой" температуры без ручного вмешательства.
- **Повышенная точность (потенциально):** Гибридный подход и HPO могут обеспечить более точный прогноз для конкретной локации по сравнению с общими моделями.
- **Учет локальных особенностей:** Модель LSTM обучается на исторических данных конкретной станции, что позволяет ей улавливать локальные микроклиматические паттерны.
- **Дополнительная информация:** Прогноз "по ощущениям" дает более полное представление о погодных условиях.

- **Интеграция:** Наличие API позволяет легко встраивать данные прогноза в другие информационные системы или дашборды Заказчика.
- **Прозрачность:** Сохранение метрик и графиков позволяет отслеживать качество работы модели.

9. Перспективы развития:

- **Расширение на другие параметры:** Добавление прогнозирования влажности, давления, осадков (требует адаптации модели).
- **Поддержка нескольких станций:** Полноценная поддержка выбора станции через API и обучение/прогнозирование для нескольких станций параллельно.
- **Более сложные модели:** Эксперименты с архитектурами Transformer, Attention, ConvLSTM для потенциального улучшения точности гибридизация.
- **Учет дополнительных признаков:** Включение данных с других метеостанций, данных спутниковых снимков, информации о городском тепловом острове и т.д.
- **Улучшение API:** Добавление возможности запрашивать прогноз на конкретную дату/время, с другим горизонтом, для разных станций/параметров. Реализация асинхронной генерации прогноза по запросу (с фоновыми задачами).
- **Визуализация:** Создание собственного UI/UX интерфейса (веб-дашборда) для отображения прогнозов, метрик и состояния системы.
- **Оптимизация производительности:** Дальнейшая оптимизация чтения данных, обучения и инференса модели.
- **Внедрение блока аналитики:** аналитическое сопоставление исторических данных и прогнозных с указанным прогнозным горизонтом, при этом во времени прогнозные данные становятся историческими прогнозными данными, что позволяет накапливать прогнозные данные нейросети, внешних источников и проводить анализ валидности с выводом графических представлений и метрик производительности.

10. Применимость для других задач и сфер:

Архитектура и подходы, реализованные в настоящем ПО, являются достаточно универсальными для прогнозирования различных временных рядов, где доступны исторические данные и, возможно, внешние влияющие факторы или прогнозы.

- **Финансы:** Прогнозирование цен акций, курсов валют, объемов торгов. Внешним фактором могут быть новости, экономические индикаторы, прогнозы аналитиков.
- **Энергетика:** Прогнозирование потребления электроэнергии, генерации возобновляемых источников, цен на энергоносители.
- **Ритейл и Продажи:** Прогнозирование спроса на товары, планирование запасов. Внешние факторы: промо-акции, праздники, погода, экономические тренды.
- **Промышленность и IoT:** Прогнозирование отказов оборудования на основе данных с датчиков (предиктивное обслуживание), прогнозирование загрузки производственных линий, потребления сырья.

- **Транспорт и Логистика:** Прогнозирование транспортных потоков, времени прибытия, спроса на такси/каршеринг. Внешние факторы: погода, мероприятия, дорожная ситуация.
- **Медицина (с осторожностью):** Прогнозирование показателей пациентов, нагрузки на медицинские учреждения (требует строгого соблюдения этики и регуляторных норм).

Ключевые адаптируемые элементы:

- **Источники данных:** заменяются на специфичные для предметной области.
- **Предобработка и признаки:** требуется глубокая адаптация под конкретные данные и цели (инжиниринг доменных признаков).
- **Внешние данные:** аналоги прогностических сервисов (подобно метео) нужно искать в соответствующей области (например, экономические прогнозы, данные о конкурентах, отраслевые индексы иные статистические данные интересные к прогнозированию).
- **Архитектура модели:** LSTM является хорошей отправной точкой, для некоторых задач могут потребоваться гибридный подход и использование архитектур (GRU, TCN, N-BEATS, Transformer) возможная комбинация походов и типов.
- **Метрики качества:** могут потребоваться специфичные для отрасли метрики.

Таким образом, система является не просто решением для прогноза параметра Температура воздуха и иных метеорологических и экологических параметров данных, но и гибким фреймворком для гибридного прогнозирования временных рядов, который можно адаптировать под широкий круг задач.

II Руководство пользователя

«Краммерти. Модуль прогнозирования временных рядов»

Оглавление:

1. Введение

1.1. Назначение программного обеспечения

Модуль прогнозирования временных рядов предназначен для автоматизированного краткосрочного прогнозирования температуры воздуха и температуры "по ощущениям" (с учетом ветро-холодового индекса) для заданных метеостанций. ПО использует гибридный подход, комбинируя нейронную сеть LSTM, обученную на исторических данных, с актуальными прогнозами от прогностических сервисов.

1.2. Основные возможности

- Автоматический сбор и обновление исторических данных с внутреннего API.
- Автоматическая загрузка и кеширование прогнозов внешних прогностических (температура и ветер и др.).
- Предобработка данных (очистка, ресемплинг, создание признаков, масштабирование).
- Автоматический подбор оптимальных гиперпараметров модели с помощью Optuna.
- Обучение и сохранение LSTM-моделей с версионированием по качеству (SMAPE).
- Генерация прогноза температуры на 72 часа вперед.
- Коррекция прогноза LSTM на основе анализа данных прогностических сервисов.
- Расчет и включение в прогноз температуры "по ощущениям" (Wind Chill).
- Сохранение прогнозов в формате JSON и в виде графиков.
- Работа по расписанию для автоматического обновления данных, переобучения и генерации прогнозов.
- Предоставление последнего (свежего) прогноза через веб-API.

2. Архитектура и Компоненты

(См. выше подробное описание архитектуры и Python файлов *.py - разделы 4 и 5 ОПИСАНИЯ.)

Система состоит из модулей для доступа к данным (внутренний API, внешний API, локальные файлы), предобработки, моделирования (LSTM, метрики), оркестрации (main.py с Optuna и Schedule), API-сервера (Flask) - эндпойнт и конфигурационных файлов.

3. Установка и Настройка

(представлено кратко, подробно ознакомьтесь с отдельным документом «Инструкция по установке» поставляемой с данным ПО).

3.1. Системные требования

- Сервер под управлением ОС Linux (рекомендуется Ubuntu 20.04/22.04 LTS).
- Python 3 (версия, совместимая с зависимостями, обычно 3.9+).

- Доступ в Интернет (для API и установки пакетов).
- Доступ к внутреннему или внешнему API исторических данных (если он сетевой).
- Достаточное количество RAM и дискового пространства (зависит от объема данных и моделей). Наличие GPU с поддержкой CUDA ускорит обучение TensorFlow.
- Права sudo для установки системных пакетов и настройки сервисов.

3.2. Клонирование/Копирование проекта

1. Подключитесь к серверу по SSH.
2. Создайте директорию проекта (например, home/krammerti/lstm) и перейдите в нее.
3. Скопируйте папки и файлы поставляемого дистрибутива (дистрибутив поставляется в папке LSTM) в созданную Вами на сервере папку.

3.3. Создание виртуального окружения

```
cd /home/krammerti/lstm
python3 -m venv venv
source venv/bin/activate
```

3.4. Установка зависимостей (requirements.txt)

Убедитесь, что файл requirements.txt актуален и находится в папке проекта.

```
pip install --upgrade pip
pip install -r requirements.txt
pip install gunicorn # Устанавливаем отдельно
```

3.5. Конфигурация (config.ini)

1. **Скопируйте шаблон конфига** (если нужно) или отредактируйте существующий: nano config.ini
2. **Обязательно настройте секцию [PATHS]**, указав **абсолютные пути** к директориям данных, моделей, логов и т.д. на сервере (например, /krammerti/lstm/weather_data).
3. **Настройте секцию [API]:** Укажите правильный BASE_URL и TOKEN для внутреннего API.
4. **Настройте секцию [OPEN_METEO]:** Проверьте координаты (LATITUDE, LONGITUDE), модель, таймауты и FORECAST_FRESHNESS_HOURS.
5. **Настройте секцию [DATA]:** Укажите TARGET_STATION_ID, ADDITIONAL_STATION_IDS (если есть), проверьте ALLOWED_PARAMETERS. Даты START/END используются для первоначальной загрузки и фильтрации при обучении.
6. **Настройте секцию [SCHEDULE]:** Установите желаемое время для HISTORY_UPDATE_TIME, TRAINING_TIME, FORECAST_RETRAIN_TIME и интервал OM_UPDATE_INTERVAL_MIN.

7. **Настройте секцию [OPTIMIZATION]:** Задайте N_TRIALS (количество попыток HPO) и TIMEOUT. Для продакшена рекомендуется больше trials (50-100), но это займет больше времени.
8. **Настройте секцию [FORECAST] и [WINDCHILL]:** естановите пороги и коэффициенты для коррекции прогноза и расчета Wind Chill (настроен по умолчанию).
9. **Сохраните файл в кодировке UTF-8.**

4. Запуск и Использование

4.1. Запуск основного процесса (обучение и планировщик)

Используется systemd для управления фоновой службой.

Создайте/отредактируйте файл сервиса: sudo nano /etc/systemd/system/lstm-scheduler.service (содержимое см. в инструкции по установке). Замените <your_user> и проверьте пути.

- **Перезагрузите systemd:** sudo systemctl daemon-reload
- **Включите автозапуск:** sudo systemctl enable lstm-scheduler.service
- **Запустите сервис:** sudo systemctl start lstm-scheduler.service
- **Проверьте статус:** sudo systemctl status lstm-scheduler.service
- **Смотрите логи:** sudo journalctl -u lstm-scheduler.service -f и tail -f /krammerti/lstm/logs/app.log

Важно: При первом запуске main.py выполнит первоначальную загрузку данных, обучение (включая HPO), сохранение модели, графиков, метрик и генерацию первого прогноза. Это может занять **значительное время** (часы). Последующие запуски по расписанию будут быстрее (если HPO не требует много trials или если вы решите запускать его реже).

4.2. Запуск API сервера

Используется systemd и gunicorn.

1. **Создайте/отредактируйте файл сервиса:** sudo nano /etc/systemd/system/lstm-api.service (содержимое см. в инструкции по установке). Замените <your_user>, проверьте пути и порт bind (например, 127.0.0.1:8000).
2. **Перезагрузите systemd, включите и запустите сервис:**

   sudo systemctl daemon-reload

   sudo systemctl enable lstm-api.service

   sudo systemctl start lstm-api.service
3. **Проверьте статус и логи:**

   sudo systemctl status lstm-api.service

   sudo journalctl -u lstm-api.service -f

4.3. Настройка Nginx (если еще не сделано)

1. **Создайте/отредактируйте файл конфигурации Nginx:** sudo nano /etc/nginx/sites-available/forecast_api (содержимое см. в инструкции по установке). Убедитесь, что listen 3000 и proxy_pass http://127.0.0.1:8000; (или другой порт Gunicorn) указаны верно.
2. **Создайте симлинк:** sudo ln -s /etc/nginx/sites-available/forecast_api /etc/nginx/sites-enabled/
3. **Проверьте конфигурацию:** sudo nginx -t
4. **Перезапустите Nginx:** sudo systemctl restart nginx

4.4. Настройка брандмауэра (ufw)

1. **Разрешите трафик:**

   sudo ufw allow ssh # или 22/tcp

   sudo ufw allow 3000/tcp # Порт для доступа к API через Nginx
2. **Включите (если нужно) и проверьте статус:**

   sudo ufw enable

   sudo ufw status

4.5. Использование API для получения прогноза

Отправьте HTTP GET запрос на URL вашего сервера, указав порт Nginx и путь к эндпоинту:

http://<IP_адрес_сервера>:3000/api/v1/forecast/temperature

- Замените <IP_адрес_сервера> на реальный IP вашего Ubuntu (продакшен) сервера (например, 192.168.0.0).

В ответ вы должны получить JSON, содержащий массив объектов прогноза, каждый из которых включает time, parameter, value (прогноз температуры) и feels_like_value (температура по ощущениям). Это будет самый последний прогноз, сгенерированный фоновым процессом main.py.

5. Описание конфигурационного файла (config.ini)

(См. подробное описание каждой секции и параметров в Инструкции по установке. Кратко здесь)

- **[API]:** Настройки для доступа к внутреннему API исторических данных.
- **[OPEN_METEO]:** Настройки для доступа к API прогностических сервисов и параметр свежести локального файла.
- **[WINDCHILL]:** Пороги для расчета температуры по ощущениям.
- **[DATA]:** Параметры станций, временные рамки данных, горизонт прогноза, список прогнозируемых параметров.
- **[PATHS]:** Пути к директориям и файлам данных, моделей, логов, графиков и прогнозов. **Важно использовать абсолютные пути на сервере.**

- **[SCHEDULE]:** Расписание автоматического запуска задач.
- **[OPTIMIZATION]:** Параметры для подбора гиперпараметров Optuna (N_TRIALS, TIMEOUT).
- **[FORECAST]:** Параметры для коррекции прогноза LSTM на основе данных ОМ.

6. Работа с данными и моделями

- **Структура директорий:** Все рабочие файлы (данные, модели, логи и т.д.) должны находиться внутри директории проекта (например, home/krammerti/lstm) согласно путям, указанным в config.ini.
- **Исторические данные:** хранятся в CSV файлах в подпапках weather_data/<НазваниеСтанции_IDСтанции>/<НазваниеПараметра>.csv. Обновляются автоматически.
- **Прогноз прогностических сервисов:** хранится в weather_data/open_meteo_forecasts.csv. содержит температуру и скорость ветра и/или иные параметры с которыми Вы работаете. Обновляется автоматически по мере устаревания.
- **Сохраненные модели:** находятся в папке models. Для каждого параметра сохраняются:
    - .keras файл: Сама модель Keras.
    - _meta.json: Гиперпараметры и метаданные модели.
    - _scaler.pkl: Объект RobustScaler, обученный на данных этой модели.
      Имя файла включает дату и SMAPE для версионирования.
- **JSON-файлы прогнозов:** Генерируются в папке lstm_forecasts. Каждый файл содержит прогноз на 72 часа (или иной горизонт) для конкретной станции и параметра, включая feels_like_value. Имя файла содержит временную метку генерации. API отдает самый свежий из этих файлов.
- **Метрики:** Результаты оценки моделей сохраняются в metrics.csv.
- **Графики:** сохраняются в папке graphs (графики производительности модели и графики прогнозов).
- **База данных Optuna:** Файл .db в папке models хранит результаты HPO.

7.Настройка обучения и прогнозирования

Настройка осуществляется через два основных файла:

- **hyperparameters.json:** Определяет **диапазоны поиска** гиперпараметров модели во время автоматической оптимизации (HPO) с помощью Optuna. Вы задаете границы, а система ищет лучшее значение внутри них.
- **config.ini:** Содержит **фиксированные настройки** для работы системы, включая параметры доступа к данным, расписание, а также параметры, используемые для *постобработки* прогноза (коррекция).

7.1. Настройка Гиперпараметров Модели (hyperparameters.json)

Данный файл управляет процессом автоматического подбора (оптимизации) гиперпараметров нейронной сети LSTM. Optuna будет пробовать различные комбинации значений из указанных диапазонов, чтобы найти ту, которая дает наилучшую ошибку (val_loss) на валидационных данных во время HPO.

- **lstm_units**: Количество нейронов (юнитов) в каждом LSTM слое.
    - **Предназначение:** определяет "емкость" или сложность модели. Большее количество юнитов позволяет модели запоминать более сложные и долгосрочные зависимости в данных.
    - **Влияние:** слишком малое значение может привести к недообучению (модель не уловит все закономерности). Слишком большое – к переобучению (модель запомнит шум в тренировочных данных и будет плохо работать на новых) и увеличению времени обучения/требований к памяти.
    - **Рекомендации:** Диапазон 32-256 с шагом 32 является разумным стартом. Начните с min: 32, max: 128, если обучение идет слишком долго или ресурсы ограничены.
- **dropout_rate**: Доля нейронов, случайно "отключаемых" на каждом шаге обучения между LSTM слоями (обычный Dropout).
    - **Предназначение:** Метод регуляризации для борьбы с переобучением. Заставляет сеть быть более устойчивой и не полагаться слишком сильно на отдельные нейроны.
    - **Влияние:** Значение 0.0 означает отсутствие Dropout. Слишком высокое значение может привести к недообучению.
    - **Рекомендации:** 0.0 - 0.5 с шагом 0.1. Часто оптимальные значения лежат в диапазоне 0.1-0.3.
- **learning_rate**: Скорость обучения оптимизатора Adam.
    - **Предназначение:** Определяет, насколько сильно веса модели корректируются на каждой итерации обучения на основе ошибки.
    - **Влияние:** Слишком высокая скорость может привести к "перепрыгиванию" через оптимальное решение и нестабильности обучения (loss будет скакать). Слишком низкая – к очень медленному обучению или застреванию в локальном минимуме.
    - **Рекомендации:** Логарифмическая шкала (как у вас, log=True в коде Optuna) часто эффективна. Диапазон 1e-5 (0.00001) до 1e-2 (0.01) – хороший старт. *Примечание: в вашем JSON указан step, но для логарифмической шкалы он не используется Optuna; Optuna сама выбирает точки внутри диапазона.*
- **epochs**: Максимальное количество эпох (проходов по всем тренировочным данным) для обучения модели *в рамках одного trial'а Optuna*.
    - **Предназначение:** задает максимальную продолжительность обучения для одной комбинации гиперпараметров. Реальное количество эпох будет меньше из-за EarlyStopping.

    - **Влияние:** должно быть достаточно большим, чтобы модель успела сойтись, но не настолько, чтобы тратить время зря.
    - **Рекомендации:** 50-200 – разумный диапазон. EarlyStopping сам остановит процесс раньше, если улучшение прекратится.
- **batch_size**: Количество обучающих примеров, обрабатываемых моделью за одну итерацию обновления весов.
    - **Предназначение:** влияет на стабильность обучения, скорость и использование памяти.
    - **Влияние:** Меньшие значения приводят к более шумным, но потенциально быстрее сходящимся обновлениям весов (могут помочь выбраться из локальных минимумов). Большие значения – к более стабильным обновлениям, быстрому прохождению эпохи, но требуют больше памяти (особенно VRAM на GPU). Слишком большой размер может ухудшить способность модели к обобщению.
    - **Рекомендации:** 16-64 (или 128, если позволяет память) с шагом 8 или 16 – стандартный выбор.
- **sequence_length**: Длина входной последовательности (количество предыдущих временных шагов), используемая для предсказания следующего шага.
    - **Предназначение:** определяет, на какой "исторический горизонт" смотрит модель при прогнозировании.
    - **Влияние:** слишком короткая последовательность может не дать модели достаточно информации о зависимостях. Слишком длинная – увеличивает вычислительную сложность, требования к памяти и может "размыть" влияние недавних событий.
    - **Рекомендации:** зависит от данных. Для часовых данных о температуре: от 12 часов (полсуток) - 168 часов (семеро суток) – разумные пределы для начала. Ваш диапазон должен быть равен = горизонту прогнозирования. Шаг для автоматического подбора 12, минимум =12, максимум 168. (как Пример! если горизонт прогнозирования 168 часов (7 суток)!) по умолчанию скажем: 24.
- **l1**: Коэффициент L1 регуляризации для весов ядра LSTM слоев.
    - **Предназначение:** Метод регуляризации, добавляющий штраф к функции потерь, пропорциональный сумме абсолютных значений весов. Способствует разреженности (некоторые веса становятся нулевыми), что может помочь в отборе признаков и борьбе с переобучением.
    - **Влияние:** слишком большое значение может "убить" слишком много весов, приводя к недообучению.
    - **Рекомендации:** небольшие значения, например, 0.0 - 0.1. Часто используется 0.0 (отсутствие L1) или очень малые значения (1e-5, 1e-4). Ваш диапазон 0.01-0.2 довольно агрессивен, возможно, стоит начать с min: 0.0.
- **recurrent_dropout_rate**: Доля нейронов, случайно "отключаемых" для *рекуррентных* соединений внутри LSTM ячейки.
    - **Предназначение:** еще один способ борьбы с переобучением, специфичный для рекуррентных сетей.

    - **Влияние:** аналогично dropout_rate.
    - **Рекомендации:** 0.0 - 0.5 с шагом 0.1. Часто используются значения 0.1-0.3.
- **unroll**: Опция Keras LSTM слоя.
    - **Предназначение:** если true, цикл обработки последовательности "разворачивается" в статический граф. Это может ускорить обучение на *коротких* последовательностях, но требует *значительно* больше памяти.
    - **Влияние:** обычно false является более безопасным и менее ресурсоемким вариантом, особенно для длинных последовательностей. True может дать небольшое ускорение на CPU для коротких sequence_length, но часто замедляет на GPU и всегда требует больше памяти.
    - **Рекомендации:** оставить [true, false], чтобы Optuna могла проверить оба варианта, но false часто является предпочтительным.
- **open_meteo_diff_threshold**: Порог разницы с ОМ для штрафа в кастомной функции потерь (используется **только** во время HPO).
    - **Предназначение:** jпределяет чувствительность модели к прогнозу ОМ во время подбора гиперпараметров.
    - **Влияние:** меньшее значение заставит Optuna чаще учитывать штраф от ОМ, большее – реже.
    - **Рекомендации:** ваш диапазон 0.1-2.0 разумен. Позволяет Optuna найти баланс. Иногда нужно отказаться от штрафов в угоду точности.
- **kernel_initializer**: Метод инициализации начальных весов для входных соединений LSTM.
    - **Предназначение:** правильная инициализация весов важна для стабильной сходимости обучения.
    - **Влияние:** разные инициализаторы могут лучше подходить для разных функций активации. glorot_uniform (или Xavier) хорошо работает с tanh, sigmoid. he_normal часто рекомендуется для relu.
    - **Рекомендации:** ["glorot_uniform", "he_normal"] – хороший выбор для тестирования. lecun_uniform менее распространен.
- **recurrent_initializer**: метод инициализации начальных весов для рекуррентных соединений LSTM.
    - **Предназначение:** аналогично kernel_initializer, но для внутренних весов.
    - **Влияние:** orthogonal часто рекомендуется для рекуррентных сетей, для предотвращения исчезающих/взрывающихся градиентов. identity может быть полезен в некоторых случаях.
    - **Рекомендации:** ["orthogonal", "identity"] – хороший выбор.
- **activation**: функция активации, используемая внутри LSTM ячеек (для основного выхода и вентилей, если не указаны отдельные recurrent_activation).
    - **Предназначение:** вносит нелинейность, позволяя сети изучать сложные зависимости.
    - **Влияние:** tanh – классический выбор для LSTM. relu может ускорить обучение, но иногда приводит к "мертвым нейронам". sigmoid обычно используется для

вентилей, но реже как основная активация LSTM выхода.

- **Рекомендации:** ["relu", "tanh"] – основные кандидаты для тестирования.

7.2. Настройка Параметров Процесса (config.ini)

Этот файл содержит фиксированные параметры, **которые НЕ подбираются автоматически.**

- **[API]**: Настройки доступа к вашему внутреннему (или внешнему – источник исторических данных получаемых, сохраняемых локально, обновляемых по API. *Изменять только если изменились URL или токен.*
- **[OPEN_METEO]**: Координаты, модель прогностического сервиса.
  FORECAST_FRESHNESS_HOURS: как часто обновлять локальный файл прогноза ОМ (в часах). Меньшее значение = чаще запросы к внешнему API, но более свежие данные для коррекции и обучения. Большее значение = реже запросы, экономия лимитов, но потенциально более старые данные. **Рекомендация:** 3-6 часов.
- **[WINDCHILL]**: Пороги температуры и скорости ветра для начала расчета "ощущаемой" температуры. **Не рекомендуется изменять, если стандартная формула не подходит для вашего региона или нужд.**
- **[DATA]**:
  - START_YEAR/MONTH/DAY, END_YEAR/MONTH/DAY: определяют период данных для обучения.
  - FORECAST_HORIZON: на сколько часов вперед генерировать прогноз.
  - TARGET_STATION_ID, ADDITIONAL_STATION_IDS: выбор станций.
  - ALLOWED_PARAMETERS: список параметров для прогнозирования (пример: "Температура воздуха").
- **[PATHS]**: **важно указывать абсолютные пути на сервере!**
- **[SCHEDULE]**: время запуска автоматических задач. **Настраивать в соответствии с желаемой частотой обновлений и обучения и переобучения.**
- **[OPTIMIZATION]**:
  - N_TRIALS: количество попыток HPO. Больше = лучше поиск, но дольше.
    Рекомендация: 5-20 для быстрой проверки, 50-100+ для тщательного поиска.
  - TIMEOUT: Максимальное время на HPO в секундах.
- **[FORECAST]**: Параметры коррекции прогноза LSTM:
  - CORRECTION_THRESHOLD: Порог разницы с ОМ для активации коррекции. Меньше = коррекция чаще.
  - MAX_CORRECTION: Максимальное изменение прогноза LSTM за шаг. Больше = более сильная коррекция при больших расхождениях.
  - CORRECTION_FACTOR: Сила "притяжения" к ОМ (0-1). Больше = сильнее притяжение.
    Рекомендация: настраивать экспериментально, анализируя графики прогнозов, метрики. Начните с увеличения MAX_CORRECTION, если коррекция слабая.

Проведите качественно настройку гиперпараметров LSTM модели, опираясь на метрики – например SMAPE и графики, tensorboard для визуализации процессов обучения (*tensorboard --logdir ./logs*), не стоит прибегать к грубой коррекции при плохой производительности самой модели.

Старайтесь поддерживать производительность модели в диапазоне метрики SMAPE не более 20, стремиться к 10-15.

7.3. Рекомендации по настройке:

- **Начинайте с малого:** при первой настройке или изменении данных используйте небольшое значение N_TRIALS (например, 5-10) для быстрой проверки работы всего конвейера.
- **Баланс HPO и Переобучения:** не обязательно запускать полный HPO каждый раз. Можно настроить расписание так (TRAINING_TIME, FORECAST_RETRAIN_TIME) использует *последние лучшие найденные параметры*.
- **Анализ логов и TensorBoard*:** это ваши главные инструменты для понимания того, как проходит обучение и HPO. Следите за val_loss - он должен уменьшаться. Анализируйте, какие параметры Optuna выбирает как лучшие.

  Все логи системы записываются в файл app.log (файл в корневой системе /home/krammerti/LSTM/app.log (не путать с каталогом /logs!!)
- **Настройка коррекции:** Подбирайте параметры в [FORECAST] итеративно, генерируя прогнозы и сравнивая их с ОМ и фактом (если доступен) на графиках.
- **Помните о накоплении данных:** Система контролирует размерность файла app.log, однако не забывайте о регулярном накоплении данных которые важны и не подвергаются контролю размерности:

  Исторические данные (папка: weather_data)

  Графики обучений и прогнозов (папка: graphs)

  Файлы моделей (папка: models)

  Файлы прогнозов (папка: lstm_forecasts)

  Файлы обучения модели: (logs)

  Не удаляйте без критичной надобности файлы в данных каталогах!

  Увеличивайте квоту дискового пространства, если это невозможно, сомневаетесь? - обратитесь в поддержку разработчика ООО «Краммерти».

Этот раздел руководства поможет пользователю понять, как влиять на процесс обучения и прогнозирования через конфигурационные файлы.

8.Устранение неполадок

- **API не отвечает / Connection refused:**
  - Проверьте, запущен ли Nginx: sudo systemctl status nginx
  - Проверьте, запущен ли Gunicorn/Flask (сервис forecast-api): sudo systemctl status forecast-api

- Проверьте логи Gunicorn/Flask: sudo journalctl -u forecast-api -f
- Проверьте настройки брандмауэра (sudo ufw status), разрешен ли порт 3000.
- Проверьте конфигурацию Nginx (sudo nginx -t).

- **API возвращает 404 "Прогнозы не найдены":**
  - Проверьте, существует ли папка, указанная в FORECASTS_DIR в config.ini.
  - Убедитесь, что основной процесс (main.py) успешно отработал хотя бы раз и сгенерировал хотя бы один .json файл прогноза в этой папке. Проверьте логи lstm-scheduler.
  - Убедитесь, что api_server.py ищет файлы в правильной директории (проверьте значение FORECASTS_DIR, используемое API сервером, особенно если была ошибка чтения конфига).
- **Прогнозы не обновляются:**
  - Проверьте статус и логи сервиса lstm-scheduler (sudo systemctl status lstm-scheduler, sudo journalctl -u lstm-scheduler -f). Убедитесь, что он активен и задачи выполняются по расписанию без ошибок.
  - Проверьте лог приложения app.log на наличие ошибок во время обучения или генерации прогноза.
- **Исторические данные не обновляются:**
  - Проверьте логи lstm-scheduler в app.log на наличие ошибок во время выполнения job_update_history или load_and_save_data_async.
  - Убедитесь в доступности внутреннего API и корректности токена/URL в config.ini.
- **Ошибка при обучении:**
  - Анализируйте логи в app.log и логи TensorBoard (с данной библиотекой рекомендуется работать во время обучения) вызов:

    tensorboard --logdir ./logs
  - Проверьте наличие свободного места на диске, достаточность RAM/VRAM.

9. Обновление системы

1. **Остановите сервисы:**

   sudo systemctl stop lstm-api.service

   sudo systemctl stop lstm-scheduler.service
2. **Перейдите в директорию проекта:** cd home/krammerti/lstm
3. **Активируйте окружение:** source venv/bin/activate
4. **Обновите зависимости (если requirements.txt изменился):** pip install -r requirements.txt
5. **Примените изменения в config.ini** (если необходимо).
6. **Перезапустите сервисы:**

   sudo systemctl start lstm-scheduler.service

   sudo systemctl start lstm-api.service
7. **Проверьте статус и логи** в app.log после перезапуска.

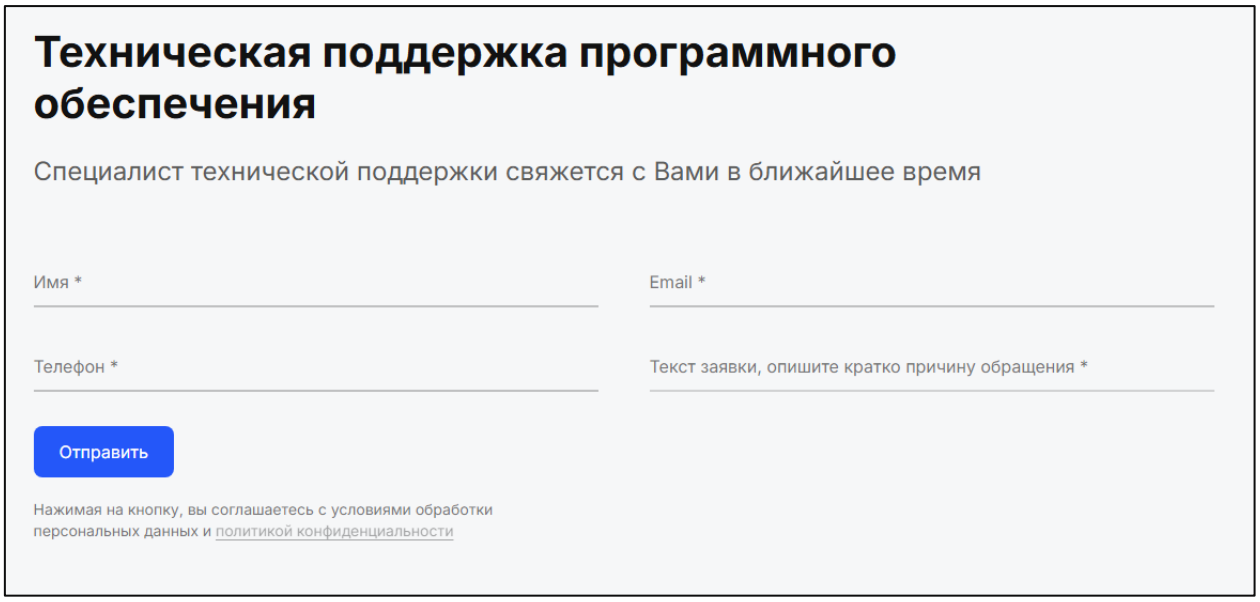
10. Контакты и Поддержка:

ЧАТ ПОДДЕРЖКИ ТЕЛЕГРАМ:

https://t.me/krammerti

Форма технической поддержки:

https://krammerti.ru/software#form-2



Адрес офиса: 628412, Ханты - Мансийский автономный округ – Югра, г.Сургут, ул. Юности, 8;

https://krammerti.ru (корпоративный сайт)

https://краммерти.рф (интернет-магазин программного обеспечения)

Телеграмм: https://t.me/krammerti

Электронная почта: krammerti@yandex.ru

Телефон: +7 (922) 078-99-44;