ChatGPT

Обзор репозитория IBM Granite TSFM

Общее описание и архитектура проекта

Granite TSFM (Time Series Foundation Models) – это открытый проект IBM для работы с **фундаментиальными моделями для временных рядов**. Цель репозитория – предоставить сообществу готовые предобученные модели и инструменты для прогнозирования по временным рядам, аналогично тому, как **фундаментиальные LLM-модели** работают с текстом [1]. Granite TSFM ориентирован на задачи прогнозирования значений временных рядов (например, прогнозирование спроса, финансовых показателей, энергетических нагрузок и т.д.), а также поддерживает классические задачи **классификации и регрессии на данных временных рядов** [2].

В основе проекта лежат современные архитектуры на базе Transformer-подходов, адаптированные для временных рядов. В частности, реализованы и предобучены следующие модели:

- **PatchTST** – трансформер для временных рядов, разбивающий серию на «патчи» (блоки точек) и обрабатывающий их подобно тому, как Vision Transformer работает с изображениями. Эта модель была представлена в работе IBM и соавторов на ICLR 2023 [3] и обеспечивает точное долгосрочное прогнозирование временных рядов за счёт механизма self-attention на патчах.
- **PatchTSMixer** – облегчённая модель на основе архитектуры MLP-Mixer, адаптированная для многомерных временных рядов (KDD 2023) [4] [5]. Она миксирует информацию **по патчам, каналам (переменным ряда) и скрытым признакам**, и дополнена легковесными механизмами внимания (gated attention и опционально self-attention) [2]. PatchTSMixer предназначен для эффективного моделирования без громоздких self-attention, что ускоряет обучение и инференс, но при этом достигает качества, сопоставимого или превосходящего более тяжёлые трансформеры [6].
- **TinyTimeMixer (TTM)** – семейство **ультра-компактных моделей** на основе идей Mixer, которое является первым в своём роде **«tiny» фундаментиальным моделям** для временных рядов [7]. Каждая модель TTM имеет всего порядка 1–5 млн параметров, но обучена на колоссальных объёмах данных (сотни миллионов до миллиарда временных точек) [8]. Благодаря этому TTMs достигают **state-of-the-art** качества **в нулевом или малошотовом прогнозировании** (zero-shot/few-shot) – они **превосходят модели с миллиардами параметров** на ряде бенчмарков [9]. Модели TTM обучены *канально-независимым* способом (каждый временной ряд обучается без смешивания каналов, что повышает обобщающую способность), но при дообучении могут включать и механизм смешивания каналов для учёта корреляций между переменными [10] [11]. Архитектурно TTM относится к семейству **MLP-Mixer** (полностью MLP-основанная архитектура без стандартного self-attention) с рядом улучшений, специально разработанных для временных рядов (например, специальные «reconciliation heads» для учёта иерархичности серий и гибридный подход к моделированию каналов) [12] [6].

Все модели проекта реализованы с использованием **PyTorch** и интегрированы с экосистемой **Hugging Face Transformers**. Это значит, что конфигурации и классы моделей совместимы с интерфейсами Transformers, а сами модели можно загружать через `from_pretrained` и

использовать как стандартные трансформеры. IBM в сотрудничестве с Hugging Face добавила эти модели непосредственно в библиотеку Transformers [13], поэтому ключевые архитектуры (PatchTST, PatchTSMixer, TTM) доступны пользователям через привычный API Hugging Face [2].

Архитектура решения предполагает три основных компонента:

- **Предобученные модели** – весовые параметры моделей (разных масштабов и настроек контекста/горизонта прогноза), выложенные в репозитории Hugging Face (`ibm-granite/*`). Они обучены на больших наборах временных рядов (Monash Forecasting Repository, LOTSA и др., в сумме до ~1 млрд точек) и покрывают различные диапазоны: от минутных и часовых рядов (версии r1, r2) до дневных и недельных (версия r2.1) [14] [15]. Все модели **открыты под лицензией Apache 2.0** – их можно свободно использовать в исследованиях и коммерческих приложениях [16].
- **Инструменты подготовки данных и инференса** – специальный пайплайн для подготовки данных и получения прогнозов. Вместо текстовых промптов (как в NLP) здесь используется мета-информация о временном ряде (идентификаторы, типы столбцов, частота измерений и т.п.) и окно недавних наблюдений в качестве «контекста» модели [17]. Репозиторий предоставляет класс `TimeSeriesPreprocessor` для обработки сырых данных: масштабирования признаков, кодирования дат и категорий, формирование окон контекста и целевых интервалов [18] [19]. Для инференса предназначен класс `TimeSeriesForecastingPipeline`, который оборачивает модель и автоматизирует все стадии прогнозирования – от препроцессинга pandas DataFrame до выдачи готового прогноза в удобной форме [20] [21].
- **Компоненты сервинга и демонстрации** – в состав проекта входят средства для развёртывания моделей как сервисов и наглядной демонстрации их работы. Например, предоставлен **Streamlit-демо** (папка `tsfmhfdemos`), показанный на NeurIPS 2023, где можно интерактивно задавать данные и наблюдать прогнозы модели [22]. Кроме того, есть модуль `services` для интеграции моделей в производные сервисы – например, в экосистему IBM Watsonx или пользовательские API. Этот компонент позволяет настроить способ вызова модели и формат вывода (через Granite IO framework) [23], упрощая внедрение моделей в приложения.

Итого, Granite TSFM предоставляет целостную архитектуру: готовые предобученные **ML-модели для временных рядов** + **утилиты** для их применения на данных + **примеры сервиса**, что существенно снижает порог входа для практиков. Проект следует современным принципам: все решения опираются на открытые данные, отвечают требованиям этичности IBM (данные с открытыми лицензиями) и ориентированы на **enterprise-применение**, сохраняя при этом исследовательскую прозрачность и доступность сообществу [24] [25].

Структура репозитория и ключевые компоненты

Репозиторий имеет четко организованную структуру директорий и файлов, отражающую разделение на библиотечный код, примеры и служебные материалы:

- `tsfm_public/` – основная директория с исходным кодом **библиотеки TSFM**. Здесь расположен Python-пакет, устанавливаемый как `granite-tsfm`. Внутри можно выделить субмодули:
- `models/` – реализации моделей. Например, там находятся классы для **TinyTimeMixer** и, вероятно, оболочки для PatchTST/TSMixer (если не используются напрямую из Transformers). В модельных классах определены конфигурации (напр.

`PatchTSMixerConfig`) и сами PyTorch-модели (`TinyTimeMixerForPrediction`, `PatchTSTModel` и т.д.), совместимые с API Transformers.

- `toolkit/` – утилиты для подготовки данных и предсказаний. Здесь находятся классы `TimeSeriesPreprocessor` и `TimeSeriesForecastingPipeline`, а также вспомогательные функции. Например, `time_series_preprocessor.py` содержит функциональность масштабирования и группировки данных по ID временных серий, кодирования категориальных столбцов и т.п. [26]. Pipeline-класс инкапсулирует загрузку модели, применение препроцессора и формирование выходного прогноза в виде DataFrame [20] [21]. Эти инструменты снимают с пользователя рутину подготовки данных – достаточно настроить несколько параметров (список колонок-идентификаторов, целевых колонок, длина контекста и горизонта) и передать сырые данные, на выходе вернётся готовый прогноз.
- *Возможны другие подмодули*, например `evaluation/metrics` для вычисления метрик качества прогноза, `datasets/` для работы с датасетом HuggingFace `datasets` (которая подключена как зависимость [27]). В коде присутствуют и другие вспомогательные классы – например, для генерации временных окон, работы с датами (включая поддержку различных частот: минутные, часовые, дневные и т.д.), которые нужны для универсальности моделей [14].
- `notebooks/` – набор Jupiter-нотбуков с руководствами и примерами. Они служат «рецептами» по применению моделей Granite TSFM к реальным задачам:
- *Getting started* с PatchTSMixer – демонстрирует базовое использование модели PatchTSMixer для многомерного временного ряда [28]. В нотбуке показывается, как загрузить предобученную модель, подать данные и получить прогноз.
- *Transfer learning* с PatchTSMixer и с PatchTST – примеры **дообучения (fine-tuning)** моделей на новых данных [29]. Здесь показано, как можно использовать уже предобученные веса и адаптировать их под конкретный набор данных (например, доучить модель на другом промышленном датасете).
- *Getting started* с TinyTimeMixer (TTM) – руководство по использованию компактной модели TTM в режиме нулевого шота и с минимальным обучением [30]. Данный ноутбук иллюстрирует, как TTM сразу из коробки прогнозирует новый ряд и как добавить небольшое количество обучающих данных для повышения точности.
- *TTM full benchmarking* – скрипты и результаты бенчмаркинга TTM на стандартных наборах данных [31]. Вероятно, здесь содержатся коды для воспроизведения экспериментов из статьи по TTM (NeurIPS 2024), включая сравнение с другими моделями.

Нотбуки специально выделены в отдельный **pip-экстра** `notebooks`: то есть можно установить необходимые зависимости (`pip install "granite-tsfm[notebooks]"`) для запуска этих примеров [32]. В зависимостях для ноутбуков указаны `jupyter`, `matplotlib`, `plotly`, `ipywidgets`, `tensorboard` и др. для визуализации и интерактивной работы [33] [34]. - `scripts/` – скрипты автоматизации. Здесь находятся, к примеру, скрипты подготовки окружения, возможно загрузки данных или запуска обучения из командной строки. Точный состав не детализирован в документации, но обычно такие скрипты содержат CLI-интерфейсы для запуска типовых задач (например, `train.py` для обучения модели на своем наборе, `predict.py` для инференса без ноутбука). Они могут быть полезны для интеграции в конвейеры без необходимости писать свой код с нуля. - `services/` – код, связанный с развертыванием моделей как сервиса. Документация указывает, что **“services component”** помогает настроить, как модель вызывается и как выдается ответ [23]. Вполне вероятно, тут реализованы шаблоны API (например, на **FastAPI** или **Flask**) или интеграция с **Granite-IO** – вспомогательной библиотекой IBM для ввода/вывода моделей Granite [23]. С её помощью можно изменить формат ввода пользователя или формат возвращаемого моделью ответа (например, обернуть прогноз в JSON). Таким образом, `services` делает модели Granite готовыми для

включения в микросервисы или веб-приложения с минимумом дополнительного кода. - `tsfmhfdemos/` – папка с **демо-приложением**, представленным на конференции NeurIPS 2023. Это, по сути, **витрина возможностей** Granite TSFM. Демо показывает, как предобученные модели могут применяться на практике: там реализован, например, интерактивный веб-интерфейс для загрузки своего датасета и получения прогноза. Судя по требованиям (`pip install ".[demos]"` устанавливает `streamlit`, `altair`, `plotly`, `streamlit-aggrid` и др. [35]), приложение сделано на **Streamlit** с визуализацией временных рядов и прогнозов. Код, скорее всего, содержит готовые сценарии: загрузка заранее подготовленных моделей (предполагается, что пользователь до этого обучил или скачал веса), применение их к новым данным и отображение результатов в браузере. Этот раздел будет полезен тем, кто хочет быстро понять, как Granite TSFM можно интегрировать в прикладное решение (например, внутренний аналитический инструмент). - `tests/` – набор модульных тестов, покрывающих основные компоненты. Присутствуют тесты для утилит (например, `test_time_series_forecasting_pipeline.py` проверяет корректность работы пайплайна прогнозирования [36], включая сохранение формы выходных данных, инвертирование масштабирования и т.д.), для препроцессора, а также для моделей (возможно, проверка что модель выдаёт прогноз нужной размерности и поддерживает разные режимы – CPU/GPU). Наличие тестов повышает надежность кода и позволяет сторонним разработчикам понимать, как ожидалось использовать функции. - **Служебные файлы**: - `pyproject.toml` – настройки проекта (метаданные пакета и зависимости). Здесь указано имя пакета `granite-tsfm`, версия (на момент обзора – **0.2.27**), автор – IBM, лицензия Apache 2.0 [37]. В разделе зависимостей перечислены основные библиотеки: **pandas ≥2.2**, **numpy**, **scikit-learn**, **urllib3**, а главное – **transformers[torch] ≥ 4.38.0** и **datasets** [38]. То есть проект требует свежую версию библиотеки Hugging Face Transformers (в которой уже включены модели Granite). Отдельно заданы *extra*-группы: `notebooks` (визуализация, Jupyter), `testing` (pytest, coverage), `dev` (linting, pre-commit) и `demos` (Streamlit и проч.) [33] [35]. Это позволяет удобно устанавливать только нужный поднабор зависимостей. - `README.md` – описательный файл (значительная часть его включена в раздел “About” на GitHub). В README приведены инструкции установки и ссылки на ноутбуки, колабы, а также важное **примечание о статусе проекта** [39] (о нём ниже). - Файл `LICENSE` – текст лицензии Apache 2.0. - Конфиги для разработчиков: `.gitignore`, `.isort.cfg`, `pytest.ini` и `Makefile` (например, Makefile может содержать команды для форматирования кода, запуска тестов или сборки пакета). - `wiki.md` – возможно, копия или шаблон содержимого Wiki (на GitHub). Основная документация вынесена на официальные сайты IBM и Hugging Face, поэтому wiki содержит главным образом ссылки на статьи, пресс-релизы и т.п. (например, список публикаций группы TSFM, ссылки на блог-посты, см. раздел **Блог и СМИ** в Wiki [40] [41]).

Зависимости и технологии. Granite TSFM построен на **Python 3.9+** (поддерживает версии вплоть до 3.12 [42]) и интенсивно использует научный стек: PyTorch (для реализации моделей), Transformers и Datasets от Hugging Face, Pandas (манипуляции с таблицами времени), NumPy. Для визуализаций в демо и ноутбуках – Plotly, Matplotlib, Altair. Тестирование – PyTest. В рамках ML-оптимизации могут применяться и сторонние решения: например, упоминается интеграция с **vLLM** (оптимизированный инференс) в блоге [43], но на уровне репозитория основной упор на стандартные библиотеки. Благодаря Apache 2.0 лицензии, проект может легко объединяться с любыми другими Python-библиотеками и встраиваться в существующие ML-пайплайны.

Возможности использования: установка, запуск, обучение, инференс

Установка библиотеки. Проект распространяется через PyPI под именем `granite-tsfm`. Актуальную версию можно установить командой:

```
pip install granite-tsfm
```

Это установит базовый пакет с основными зависимостями (Transformers, pandas и др.). Если планируется запускать прилагаемые ноутбуки или демо-приложение, авторы рекомендуют установить дополнительные группы зависимостей:

- Для ноутбуков: `pip install "granite-tsfm[notebooks]"` [32] – подтянет Jupyter, визуализации и прочее.
- Для демо: `pip install "granite-tsfm[demos]"` [22] – дополнительно установит Streamlit и связанные библиотеки.
- Можно установить сразу всё (для разработки): `pip install "granite-tsfm[all]"`, объединяющий dev, testing, notebooks, demos и др. (опция `all` определена в манифесте пакета [44] [45]).

Альтернативно, можно клонировать репозиторий и установить локально:

```
git clone https://github.com/ibm-granite/granite-tsfm.git
cd granite-tsfm
pip install .
```

(Добавив `[notebooks]` или `[demos]` при необходимости). Проект не требует специальных компиляций – все написано на Python, поэтому установка проходит быстро.

Загрузка предобученной модели. После установки можно сразу пользоваться предобученными моделями IBM Granite. Пакет регистрирует в Transformers необходимые конфигурации, поэтому можно, например, сделать:

```
from transformers import AutoModelForTimeSeriesForecasting

model = AutoModelForTimeSeriesForecasting.from_pretrained("ibm-granite/
granite-timeseries-ttm-r2")
```

Здесь `"ibm-granite/granite-timeseries-ttm-r2"` – имя модели на Hugging Face Hub (TinyTimeMixer, версия r2). Аналогично доступны: - `"ibm-granite/granite-timeseries-ttm-r1"` – ранняя версия TinyTimeMixer. - `"ibm-granite/granite-timeseries-patchtst"` – модель PatchTST. - `"ibm-granite/granite-timeseries-patchtsmixer"` – модель PatchTSMixer. - А также другие размеры (например, TTM tiny/small/etc., если предусмотрены). Полный список моделей и ссылок приведен в документации IBM [46].

Стоит отметить, что **Granite TSFM предоставляет предобученные веса для разных настроек контекста и горизонта**. Модели «focused pre-trained» – каждая оптимизирована под определенную длину контекста (например, 512 точек входа) и длину прогноза (скажем, 96 точек выхода) [47]. Поэтому при выборе модели важно учитывать, какие временные промежутки требуются именно вам (в названиях на Hugging Face это может быть отражено, либо в описании model card).

Подготовка данных. Для использования модели необходимы данные в виде **таблицы (pandas DataFrame)**, содержащей временные ряды. Разработчики рекомендуют убедиться, что: - Имеется **временная метка** (дата/время) в одном столбце – она будет использоваться как `timestamp_column`. - Присутствуют один или несколько столбцов значений, которые нужно прогнозировать – `target_columns`. - Возможные категориальные или идентификационные столбцы (например, идентификатор серии, если в одном датасете множественные временные ряды) должны быть указаны как `id_columns`. - Данные желательно **предварительно масштабировать** (нормализовать) по каждому признаку отдельно [48], поскольку модели в текущей версии не делают автоматического нормирования (они обучались на нормированных данных). Для этого можно воспользоваться утилитой **TimeSeriesPreprocessor**.

Использование препроцессора выглядит так (пример из документации IBM):

```
from tsfm_public.toolkit import TimeSeriesPreprocessor

tsp = TimeSeriesPreprocessor(
    timestamp_column="date",
    id_columns=[],                # если есть идентификатор группы, можно
указать
    target_columns=["value1", "value2"],
    context_length=512,
    prediction_length=96,
    scaling=True                  # применить стандартное масштабирование
(StdScaler)
)
```

Этот объект `tsp` при инициализации запоминает настройки столбцов и вычисляет параметры масштабирования. Затем им можно преобразовать данные:

```
train_df, test_df = tsp.split(dataframe, split_config={"train": 0.7, "test":
0.2})
train_ds = tsp.preprocess(train_df)
```

Здесь `split` может разбить исходный DataFrame на обучающую и тестовую выборки в заданных пропорциях (остальное пойдет в валидацию автоматически). Метод `preprocess` применит масштабирование и организует данные в необходимую структуру (например, создаст PyTorch Dataset или тензоры, в зависимости от режима).

Для **инференса без обучения** (zero-shot) можно обойтись без разделения – просто подать имеющийся ряд как `test_df`, указав `scaling=False` если вы уже нормализовали его самостоятельно. В общем случае **пользователь должен самостоятельно обеспечить масштабирование каждого канала отдельно** перед использованием моделей [48]. TimeSeriesPreprocessor может это сделать автоматически (посредством StandardScaler или MinMaxScaler) при параметре `scaling=True`.

Прогнозирование (инференс). Самый простой путь получить прогноз – воспользоваться `TimeSeriesForecastingPipeline`. Этот класс принимает на вход модель и препроцессор, а при вызове на новых данных выдаёт прогноз. Например:

```
from tsfm_public.toolkit import TimeSeriesForecastingPipeline

# Загрузили модель и создали tsp как выше
pipeline = TimeSeriesForecastingPipeline(model=model, feature_extractor=tsp,
device="cpu")

forecasts = pipeline(test_df)
```

Внутри, `pipeline` выполнит необходимые шаги: 1. Автоматически применит `tsp.preprocess` к `test_df` (включая масштабирование и формирование нужного формата с учетом context_length/prediction_length) [49]. 2. Прогонит подготовленные данные через модель (`model.forward`) и получит прогнозные значения. 3. При необходимости, выполнит **обратное масштабирование** результатов к исходному масштабу (если `inverse_scale_outputs=True`, что по умолчанию включено при передаче `feature_extractor=tsp` и наличии в нем настроек масштабирования [49]). 4. Вернет **pandas DataFrame** с колонками прогноза на требуемый горизонт (например, 96 точек вперёд) [20] [21].

Таким образом, буквально 2–3 строки позволяют получить прогноз с предобученной модели. В официальной документации IBM демонстрируется аналогичный код: создается pipeline и вызывается на данных [50]. Этот высокий уровень абстракции – большое удобство проекта: пользователь может не задумываться о тензорах, формулах и прочем, а работать на уровне привычных DataFrame.

Например, для **полного нулевого шота** (без дообучения) сценарий может быть такой:

```
# 1. Подготовка данных
df = pd.read_csv("my_series.csv", parse_dates=["date"])
tsp = TimeSeriesPreprocessor(timestamp_column="date", id_columns=[],
target_columns=["y"],
                             context_length=168, prediction_length=24,
scaling=True)
# 2. Загрузка предобученной модели (контекст 168, горизонт 24 – подобрать
соответствующую модель)
model = TinyTimeMixerForPrediction.from_pretrained("ibm-granite/granite-
timeseries-ttm-r2",

num_input_channels=tsp.num_input_channels)
# 3. Прогнозирование
pipeline = TimeSeriesForecastingPipeline(model=model, feature_extractor=tsp)
forecast_df = pipeline(df)
```

Где `forecast_df` будет содержать прогноз на 24 шага вперёд для временного ряда *y*. Такой workflow проверен в примерах: **Granite TTM способен мгновенно давать высококачественный прогноз даже без обучения на целевом датасете** – благодаря обобщенным паттернам, выученным на большом корпусе рядов [51]. При необходимости, конечно, можно модель дообучить.

Дообучение (transfer learning). Для обучения своих моделей на основе Granite TSFM репозиторий предлагает воспользоваться примерами-нотбуками или адаптировать предоставленные скрипты. Общий подход: - Предполагается, что у вас есть свой набор временных рядов. Вы разбиваете его на train/val (можно использовать `tsp.split`). - Инициализируете предобученную модель с `from_pretrained`. - Добавляете, если нужно, слои Fine-Tuning. Например, можно заменить или добавить head над выходами модели для специфичной задачи. - Запускаете обучение. Здесь можно писать обычный PyTorch training loop или воспользоваться Hugging Face **Trainer** API, так как модель совместима. Вероятно, ноутбуки демонстрируют использование `Trainer`: создается `TimeSeriesDataset` (с помощью `tsp.preprocess` можно получить Torch Dataset), указывается оптимизатор, лосс (обычно MSE для прогнозирования). - После обучения сохраняете модель (`model.save_pretrained(path)`).

В официальном **Developer Tutorial** IBM показан пример тонкой настройки Granite TTM под задачу с внешними признаками (например, погодные данные) [52] [53]. Там авторы проходят по шагам: загрузка данных, подготовка с `TimeSeriesPreprocessor`, создание `TimeSeriesForecastingPipeline` **для обучения** – однако важно понимать, что Pipeline предназначен в основном для инференса, а для обучения лучше использовать низкоуровневый подход или специальный скрипт. Один из часто задаваемых вопросов – *"Можно ли обучать через TimeSeriesForecastingPipeline?"*. Разработчики отвечают, что **Pipeline предназначен только для предсказания**, а для fine-tuning следует обращаться к ноутбукам и примерам кода [54] [55]. В issue-разделе GitHub также есть обсуждения, как правильно доучивать модели (например, **Issue #53** на GitHub разъясняет, что нужно смотреть шаги Cell 4 соответствующего ноутбука) [56].

Стоит отметить, что фундаментиальные модели Granite требуют относительно небольшой дообучающей выборки. В статьях показано, что **достаточно ~5% данных** для fine-tuning, чтобы достичь качества, сопоставимого с обучением модели с нуля на всем датасете [51]. То есть, если у вас мало исторических данных, модель Granite может извлечь пользу из знаний, полученных на других сериях, и быстро адаптироваться. Это большой плюс в промышленном применении, где данных по конкретному объекту может быть немного.

Пример использования (конец-узел): Представим задачу – прогнозирование спроса на электроэнергию. Имеются почасовые измерения нагрузки за последние 2 года. Нужно прогнозировать на следующую неделю вперед. Как применить Granite TSFM: 1. **Выбор модели:** Почасовые данные, контекст возьмем 168 часов (7 дней), прогноз 168 часов. Выбираем модель TTM, натренированную на почасовых рядах, например `granite-timeseries-ttm-r2` (она поддерживает minutely/hourly, r2.1 – daily/weekly) [14]. Проверяем в документации модели допустимую длину контекста – допустим, 168 входит. 2. **Настройка препроцессора:** `tsp = TimeSeriesPreprocessor(timestamp_column="Time", target_columns=["Load"], context_length=168, prediction_length=168, scaling=True)`. 3. **Прогноз:** Загружаем модель `model = TinyTimeMixerForPrediction.from_pretrained("ibm-granite/granite-timeseries-ttm-r2", num_input_channels=1)` (1 целевой канал). Далее `pipeline = TimeSeriesForecastingPipeline(model, tsp)`. Наконец, `forecast = pipeline(df)`, где `df` – наши данные с колонками Time и Load. 4. **Результат:** DataFrame `forecast` будет содержать 168 прогнозных значений нагрузки (скорее всего, прогноз продолжит временную шкалу после последней отметки исходного df). Можно визуализировать их и оценить качество.

Для более сложных случаев (например, **мультимодельные данные** с несколькими связанными рядами, или наличие экзогенных факторов) Granite TSFM тоже предоставляет решения. Пока что (по состоянию на версию r2.1) **поддержка экзогенных признаков и категориальных переменных ограничена** – zero-shot сценарий не включает их автоматически [57] [58]. Но в TTM

заявлено, что **поддержка exogenous features присутствует** и они могут быть поданы в модель при fine-tuning [10]. В будущем, вероятно, появятся более универсальные пайплайны, учитывающие экзогенные данные.

Обучение "с нуля" (pre-training) своих моделей: репозиторий также содержит код для полного обучения моделей (в том числе PatchTST, PatchTSMixer) на пользовательских данных. Однако это ресурсоёмкий процесс – модели Granite обучались на сотнях миллионов образцов, с различными аугментациями, на мощных GPU. Поэтому разработчики предполагают, что обычные пользователи будут либо использовать предобученные модели, либо дообучать их, но не тренировать с нуля конкурирующую модель. Тем не менее, **ноутбуки pre-training** в папке `notebooks` показывают процедуру и гиперпараметры, использованные IBM. Это полезно для исследователей, желающих воспроизвести результаты или попробовать вариации архитектур (например, обучить PatchTSMixer на своём наборе временных рядов с нуля).

Дополнительные ресурсы: - Для быстрого старта есть готовые **Google Colab**-ноутбуки. Например, "TTM Colab Tutorial" – содержит весь код, чтобы запустить предобученный TTM на примере задачи (энергопотребление) без локальной установки [59]. Достаточно открыть Colab и последовательно выполнить ячейки. - На Hugging Face Hub выложены **Model Card** для каждой модели с описанием, примерами кода и ссылками. Например, Card для TTM r2 описывает архитектуру, различия между релизами r1/r2, и содержит ссылки на **рецепты** (пример кода по применению) [60] [61]. - IBM опубликовала серию блог-постов: "Patch Time Series Transformer – Getting Started" [62], "PatchTSMixer in HuggingFace – Getting Started" [63], "Introducing IBM TinyTimeMixer" на LinkedIn [64] и др., где шаг за шагом объясняется применение данных моделей. Эти статьи могут служить альтернативным (более повествовательным) руководством. - Примеры полного решения: в репозитории **granite-timeseries-cookbook** [65] (от сообщества IBM Granite) можно найти дополнительные сценарии – например, готовые скрипты для популярных наборов данных (Electricity, Traffic, Weather и др.), демонстрирующие воспроизведение результатов из статей с Granite TSFM.

В целом, **порог входа** для использования Granite TSFM достаточно низок: если вы знакомы с Python и Pandas, то сможете применить модели к своим временным рядам буквально за час, используя готовые шаблоны. Это большое преимущество по сравнению с ситуацией, когда ради прогноза приходится обучать с нуля сложные модели или настраивать десятки параметров традиционных алгоритмов.

Сравнение с аналогами: преимущества и недостатки

На поле анализа временных рядов существует несколько популярных open-source инструментов и фреймворков, и Granite TSFM выгодно выделяется на их фоне своей концепцией **фундаментиальной модели**. Сравним с некоторыми аналогами:

- **Классические библиотеки прогнозирования** (такие как **GluonTS** от Amazon, **Darts** от Unit8, **Nixtla NeuralForecast**, **sktime** и др.): Эти фреймворки предоставляют набор моделей (ARIMA, Prophet, LSTM, Transformer и т.п.) и удобный интерфейс для их обучения на конкретных данных. Однако **они, как правило, не дают предобученных весов** – пользователь обучает выбранную модель с нуля на своем датасете. Granite TSFM же предлагает предобученные модели, которые уже усвоили общие паттерны из большого количества серий. Это похоже на разницу между обучением маленькой нейросети с нуля и использованием предобученного ResNet в CV. **Преимущество Granite** – возможность получения качественных прогнозов даже при очень небольшом количестве своих данных

(за счёт transfer learning) [9]. Например, TTM в нулевом шоте часто сразу обгоняет классические модели, а с 5% данных дообучения достигает уровня, сопоставимого с обучением на 100% данных специальной модели [9]. Такой **data-efficiency** недостижим для моделей без предобученных весов.

- **Специализированные реализации моделей трансформеров для временных рядов.** Например, авторы PatchTST выложили свой код на GitHub (`yuqinie98/PatchTST`) – он получил ~1.5k звёзд [66]. Сообщество Nixtla внедрило PatchTST и другие SOTA-модели в свой пакет NeuralForecast [66]. Однако эти реализации ориентированы на **исследовательское воспроизведение** и требуют от пользователя понимания внутренностей (нужно подать тензоры нужной размерности, выполнить тренинг с определенными гиперпараметрами). **Granite TSFM** же оборачивает такие модели в удобный high-level API. Более того, IBM была **непосредственно вовлечена в разработку PatchTST/TSMixer** (авторы этих моделей – сотрудники IBM Research [67] [68]), поэтому Granite TSFM можно считать **референсной имплементацией** этих методов "от авторов" [69]. В Hugging Face Transformers эти модели интегрированы также "from the authors" [70]. Это означает лучшее соответствие описаниям из статей, а также дополнительные улучшения (например, гибкие механизмы внимания, поддержка masked pre-training) [2] [6], которые могли отсутствовать в более простых open-source кодах.

- **Hugging Face Transformers (общий фреймворк)**: До появления Granite TSFM библиотека Transformers почти не охватывала задачи прогнозирования рядов. Были, конечно, Transformer-модели в аналитике временных рядов (например, TFT – Temporal Fusion Transformer, Informer), но они не были частью стандартного релиза Transformers. Благодаря вкладу IBM, теперь **Hugging Face Transformers ≥4.38** включает в себя классы для Time Series Transformers (PatchTST, PatchTSMixer, TinyTimeMixer и возможно другие) [13]. Таким образом, **Granite TSFM фактически стал "плагином" к HF Transformers для временных рядов**. Это выгодно отличает его: пользователи HF могут почти прозрачным образом начать работать с временными рядами, не покидая знакомой экосистемы. В то же время Granite TSFM сохраняет автономность – он предоставляет дополнительные надстройки (Pipeline, Preprocessor), которых нет в чистом Transformers. То есть, **связка Granite TSFM + Transformers** даёт больше, чем каждый из компонентов по отдельности. По сути, IBM создала первый специализированный фреймворк фундаментиальных моделей для временных рядов и через интеграцию с HF сразу вывела его на уровень промышленного стандарта.

- **Производительность и эффективность.** Одним из критериев сравнения является требуемая модельная сложность для достижения данного качества. **Granite TTM** особо подчёркивает свою эффективность: модель на **<1 млн параметров** превосходит модели с **>1 млрд параметров** на ряде задач [9]. Это означает меньшие требования к памяти, возможность выполнять прогнозы на CPU или на обычном ноутбуке, тогда как огромные трансформеры вроде Informer или стандартных GPT-подобных моделей для рядов могут требовать GPU. Кроме того, Granite модели оптимизированы под разные частоты и горизонты – их можно подобрать точно под задачу, вместо тренировки универсальной громоздкой модели. **Nixtla/ETS/Formers**: Например, есть модель FEDFormer, Autoformer, etc., которые позиционировались как SOTA, но TTM показывает, что их можно превзойти меньшими моделями за счёт предобучения [51].

- **Удобство и обучение порог.** В традиционных фреймворках (GluonTS, Darts) пользователю часто надо писать код для цикла обучения, выбора гиперпараметров, и т.д. Granite TSFM значительно упрощает этот процесс: предоставляет готовые конфиги и даже обученные

веса. Это похоже на сравнение: написать архитектуру нейросети самому vs взять готовый `BertModel.from_pretrained`. Конечно, за удобство отвечают **ноутбуки** и **pipeline**, которые сделали авторы. Например, Darts тоже славится простотой (модель.fit(series)), но **Darts не даёт предобученных моделей** – каждую модель (TFT, N-BEATS и др.) она обучает на ваших данных. Granite TSFM же сразу даёт предсказание, "выученное на опыте" многих серий.

- **Многообразие задач.** Некоторые альтернативы (например, **sktime**) охватывают широкий спектр методов: от статистических ARIMA до классических ML. Granite TSFM сосредоточен именно на **нейросетевых трансформерных моделях**. Если задача не подходит для трансформеров (скажем, очень короткие ряды, где проще среднее считать), то Granite может быть избыточен. Однако для **сложных многомерных, нелинейных задач** трансформеры сейчас лидируют, и тут Granite даёт всё самое передовое.

- **Ограничения и слабые стороны:**

- Пока что Granite TSFM ориентирован на **задачи point forecasting** (одноточечный прогноз на каждый шаг). Интервальные прогнозы или вероятностные предсказания не объявлены явно (хотя можно, конечно, тренировать ансамбли или использовать выходные распределения). Конкурирующие библиотеки типа GluonTS уделяли внимание вероятностным прогнозам (DeepAR, etc.). Возможно, будущие версии Granite добавят и **доверительные интервалы**, но на момент обзора упор на точечные предсказания.
- **Экзогенные факторы и аномалии:** поддержка экзогенных переменных в Granite пока ограничена (zero-shot их не учит) [57]. Конкуренты вроде TFT специализируются на учете известных будущих переменных (например, праздники, цены). В Granite планируют эту функциональность (TTM архитектурно может принимать known future inputs [51]), но пользователю придётся дождаться примеров или реализовать самому.
- **Порог вхождения в кастомизацию:** если требуется модифицировать архитектуру, Granite TSFM чуть сложнее, т.к. завязан на HF Transformers (где многое "магически" происходит). В чистом PyTorch или GluonTS изменить модель иногда проще. Но это плата за высокоуровневость.
- **Сообщество и поддержка:** проекты вроде Nixtla активно поддерживаются независимым сообществом и регулярно обновляются новыми моделями. Granite TSFM – инициатива IBM, и хотя она открыта, но вокруг неё сообщество только формируется. Тем не менее, интеграция с Hugging Face привлекает пользователей: модели Granite уже скачаны сотни тысяч раз [71] (например, только TinyTimeMixer r1 имел >2 млн скачиваний на HF к концу 2024 [46]). Это говорит о растущем интересе.

В целом, **Granite TSFM выгодно отличается наличием предобученных моделей и простотой их применения**. Он закрывает нишу, которая ранее не была покрыта: перенос обучения (transfer learning) для временных рядов. По **точности прогнозов** Granite модели занимают лидирующие места: по отчётам IBM, TTM r2 превосходит предыдущие SOTA (PatchTST, DLinear, FiLM и др.) на **8–60% по MSE** на ряде бенчмарков [6], при этом требуя в разы меньше вычислений [72]. То есть **даже лучшие академические модели на базе трансформеров (PatchTST, Autoformer)** теперь могут уступать более «умным» за счёт предобучения моделям TTM.

Если сравнивать **собственно фреймворки**, а не модели, то наиболее близкий по духу аналог – **Hugging Face Transformers**, но Granite TSFM встроился в него, став по сути расширением. Другие библиотеки пока не предоставляют аналогичной функциональности (единственное, компания Nixtla объявила о планах выпускать свои предобученные модели, но это пока на ранних стадиях).

Вывод: Granite TSFM обладает сильными сторонами: - **Предобучение = высокое качество и переносимость**, - **Универсальность** (одна модель – много задач, много данных уже “в голове”), - **Удобство** (интеграция с HF, пайплайн, готовые примеры), - **Открытость** (Apache 2.0, бесплатное коммерческое использование, открытые данные и прозрачность этики).

К потенциальным минусам можно отнести пока что **ограниченную поддержку специфических кейсов** (например, специфичные частоты вне обученных, экзогенные признаки – нужно ждать новых релизов) и **зависимость от поддержки IBM**, о которой – ниже.

Лицензия, развитие и поддержка проекта

Лицензия. Проект Granite TSFM распространяется под лицензией **Apache 2.0** [73]. Это весьма либеральная лицензия: она позволяет свободно использовать код и модели как в научных, так и в коммерческих продуктах, модифицировать и распространять их, при условии сохранения уведомления об авторских правах и лицензии. Apache 2.0 также предоставляет явные гарантии права на патенты, что важно для компаний (они могут быть уверены, что IBM не будет предъявлять патентных претензий по использованию этих моделей). Таким образом, **для коммерческого использования никаких дополнительных ограничений нет** – можно встроить модели Granite в свой сервис, продавать решения, и т.д., **не опасаясь проблем с лицензированием** [16]. Это выгодно отличает Granite от некоторых других AI-моделей с более строгими лицензиями.

IBM явно заявляет, что модели обучены на **публичных датасетах с разрешающими лицензиями** и соблюдением этических норм [74], что снимает вопросы о легальности данных. Кроме того, в релизах Granite нет сдерживающих условий вроде «не использовать для...» – только общие оговорки Apache 2.0.

Активность разработки. Репозиторий был впервые опубликован примерно во второй половине 2023 года (первая демо была на NeurIPS 2023, декабрь). На протяжении 2024 года и начала 2025 он активно развивался: - Коммит-график показывает регулярные обновления кода. За последний год было более **1500 коммитов** – весьма высокий темп [75]. - На май 2025 выпущено **28 релизов** на GitHub [76], что обычно означает фиксирование значимых обновлений (добавление моделей, исправление ошибок, новые функции). Последний релиз 0.2.27 датирован 14 мая 2025 [76], совсем недавно, а репозиторий обновлён 16 мая 2025 [77] – то есть проект находится на острие развития. - **Число участников:** Около 14 разработчиков внесли вклад [78]. Судя по именам в публикациях (Виджай Экамбарам, Ариндам Джати, Нам Нгуен и др.) и по активным участникам в Issue (например, @wgifford – Уилл Гиффорд, вероятно), основную часть разработчиков составляют исследователи IBM Research, работавшие над статьями TSFM. Есть и внешние контрибьюторы, но их пока немного.

Issue tracker и поддержка. На GitHub видно около **39 открытых Issues и 2 открытых PR** [79]. Вопросы охватывают как использование (иногда спрашивают о примерах инференса, fine-tuning), так и сообщения о багах. Команда IBM довольно активно реагировала на Issues в 2024 году – например, давала пояснения по Pipeline vs Trainer [20], исправляла опечатки (был issue про неправильный URL установки pip, его быстро устранили) [80]. Это свидетельствует, что на фазе запуска проекта IBM уделяла внимание поддержке сообщества.

Однако в README содержится важное заявление – **“IBM Public Repository Disclosure”** [81]. В нём указано:

> *IBM предоставила данный код под лицензией open source и не обязуется выпускать улучшения, обновления или поддержку. Разработчики IBM создали этот код как проект с открытым исходным кодом (не как продукт IBM), и IBM не гарантирует уровень качества или безопасности, и не планирует поддерживать этот код в будущем* [82] .

Эта формулировка стандартна для многих исследовательских проектов IBM на GitHub. Она означает, что **официальной техподдержки или долгосрочных обязательств** от IBM не следует ожидать. Проще говоря, проект предоставлен “как есть”. Тем не менее, это не означает немедленного прекращения развития – как мы видим, обновления продолжались активно. Скорее, IBM ограждается от претензий и не гарантирует выпуск Granite TSFM как полностью поддерживаемого продукта с сопровождением.

С другой стороны, наличие таких формулировок часто свидетельствует, что код открыт в научных целях, а основная коммерческая реализация может идти внутри платформы IBM (например, **IBM watsonx.ai**). Действительно, IBM заявляла о доступности моделей Granite в облачном сервисе WatsonX (бета-доступ) [83] , где, вероятно, есть полноценная поддержка. Таким образом, для энтузиастов и исследователей – GitHub репозиторий, а для бизнес-клиентов – платформа IBM.

Коммерческая поддержка. Несмотря на отказ от обязательств, IBM заинтересована в продвижении Granite. Они предоставляют контактную почту (granite@ibm.com) и приглашают задавать вопросы на GitHub [84] . Также команда публикует новости (например, известие о принятии статьи TTM на NeurIPS 2024 было размещено в Wiki [85]) и отвечает на важные запросы. Можно ожидать, что критические баги будут поправлены, по крайней мере, до тех пор, пока IBM Research продолжает проект TSFM.

Сообщество. Проект набрал **~584 звезды и 216 форков на GitHub** за относительно короткий срок [86] [87] , что свидетельствует об интересе. На Hugging Face модели Granite имеют сотни лайков и были скачаны миллионами раз (например, TTM-r1 ~2M+ скачиваний к концу 2024) [71] . Это косвенно говорит, что вокруг Granite начинает формироваться пользовательская база – практики в индустрии, Kaggle-энтузиасты (есть kernels на Kaggle, использующие Granite [88]), исследователи (появляются статьи и медиум-посты с анализом моделей [41]). IBM даже вынесла Granite TSFM в отдельную организацию GitHub `ibm-granite` – там кроме TSFM есть сопутствующие репозитории (vision models, embedding models, code assistant и др.) [89] [90] , что похоже на стратегию развития экосистемы Granite как целого ряда фундаментиальных моделей.

Обновления и планы. На основе публичных данных можно ожидать: - **Новые версии моделей**: упоминается Granite 4.0 (например, Granite 4.0 Tiny Preview) [91] для LLM, вероятно и по TSFM будут новые релизы (r3?). - **Расширение функциональности**: поддержка дневных/недельных рядов уже добавлена в r2.1 [14] , дальше возможно появление поддержки ежемесячных/годовых интервалов, прогнозирование с экзогенами, поддержка аномалий (например, Granite Guardian для текстов – аналог может быть для временных рядов). - **Социальное доказательство:** Granite TTM завоевал награды (принимается в NeurIPS), это будет стимулировать IBM продолжать работу. Команда опубликовала множество статей (4 KDD, 1 NeurIPS, 1 ICLR, 2 AAAI, 1 ICML в портфолио TSFM) [92] – то есть исследование идет полным ходом.

Подводя итог, **уровень поддержки IBM** можно охарактеризовать так: исходный код предоставлен без гарантий, но пока проект молод и важен для репутации IBM в области ИИ, команда будет уделять ему внимание. Для коммерческого использования разработчикам следует полагаться на собственные силы и сообщество, либо рассмотреть платформа-услуги IBM (где

Granite включен). К счастью, благодаря открытой лицензии, ничто не мешает сообществу форкнуть и поддерживать проект, если IBM переключит фокус.

На данный момент Granite TSFM – **жизнеспособный и активно обновляемый проект**, который уже сегодня приносит пользу практикам, и, вероятно, будет дальше развиваться, задавая тренды в применении трансформеров к временным рядам.

Ссылки на ключевые ресурсы и файлы:

- Описание репозитория (README): *"Public notebooks, utilities, and serving components for working with Time Series Foundation Models (TSFM)"* [1] .
- Заявление об отсутствии обязательств поддержки IBM: *"...IBM makes no assertions as to the level of quality nor security, and will not be maintaining this code going forward."* [82] .
- Лицензия Apache 2.0 позволяет коммерческое использование [16] .
- Модель TTM (TinyTimeMixer) – первый «tiny» foundation model для временных рядов, <1M параметров, превосходит более крупные модели [93] [9] .
- Архитектура PatchTSMixer – MLP-Mixer для временных рядов с механизмами внимания и эффективным смешиванием по патчам/каналам [2] .
- Интеграция моделей в Hugging Face Transformers: **модели Granite доступны начиная с версии 4.38** Transformers [13] , ссылки на документацию PatchTST и PatchTSMixer [94] .
- Инструкция по нулевому шоту: шаги 1) препроцессинг, 2) загрузка модели, 3) прогноз через pipeline [95] .
- Пример использования Pipeline из документации IBM [50] .
- Сравнение с другими реализациями: таблица Wiki, показывающая присутствие PatchTST в разных библиотеках (HuggingFace, GluonTS, Nixtla) [66] – Granite реализован "From the authors".
- Статья NeurIPS 2024 (TTM) – описывает успехи модели [7] [96] .
- IBM Blog (Developer) с tutorial по Granite TSFM [97] – пример практического кейса.

---

[1] [13] [22] [28] [29] [30] [31] [32] [39] [42] [59] [73] [75] [76] [78] [79] [81] [82] [86] [87] GitHub - ibm-granite/granite-tsfm: Foundation Models for Time Series
https://github.com/ibm-granite/granite-tsfm

[2] [5] [6] [12] [72] README.md · ibm-granite/granite-timeseries-patchtsmixer at 0d98cf9af0ec5fa1536f6d4453e4991b82040f73
https://huggingface.co/ibm-granite/granite-timeseries-patchtsmixer/blame/0d98cf9af0ec5fa1536f6d4453e4991b82040f73/README.md

[3] [4] [40] [41] [46] [62] [63] [64] [66] [67] [68] [69] [70] [71] [85] [92] [94] Home · ibm-granite/granite-tsfm Wiki · GitHub
https://github.com/ibm-granite/granite-tsfm/wiki

[7] [8] [51] [60] [61] [65] [96] ibm-granite/granite-timeseries-ttm-r2 · Hugging Face
https://huggingface.co/ibm-granite/granite-timeseries-ttm-r2

[9] [47] [93] ibm-granite/granite-timeseries-ttm-r1 · Hugging Face
https://huggingface.co/ibm-granite/granite-timeseries-ttm-r1

[10] [11] [14] [15] [16] [17] [18] [19] [26] [48] [50] [57] [58] [84] [95] IBM Granite Time Series Documentation – IBM Granite
https://www.ibm.com/granite/docs/models/time-series/

[20] [21] [36] [49] How to use it for inference ? · Issue #46 · ibm-granite/granite-tsfm · GitHub
https://github.com/ibm-granite/granite-tsfm/issues/46

[23] [77] [89] [90] IBM Granite · GitHub
https://github.com/ibm-granite

[24] [25] [74] IBM Granite Embedding Documentation – IBM Granite
https://www.ibm.com/granite/docs/models/embedding/

[27] [33] [34] [35] [37] [38] [44] [45] Wheelodex — granite-tsfm
https://www.wheelodex.org/projects/granite-tsfm/

[43] Efficient Inference on ibm/granite model with vLLM - Medium
https://medium.com/towards-generative-ai/efficient-inference-on-ibm-granite-model-with-vllm-4f79aa7a16d0

[52] [97] Using the IBM Granite models for time series forecasting
https://developer.ibm.com/tutorials/awb-foundation-model-time-series-forecasting/

[53] ibm-granite/granite-timeseries-ttm-r1 · Including Exogenous Variables
https://huggingface.co/ibm-granite/granite-timeseries-ttm-r1/discussions/12

[54] How to do finetuning with TimeSeriesForecastingPipeline #53 - GitHub
https://github.com/ibm-granite/granite-tsfm/issues/53

[55] Energy Demand Forecasting with Granite Timeseries (TTM) - Colab
https://colab.research.google.com/github/IBM/granite-workshop/blob/43d9ad9d739e1084ce812fa5e908527e5ba3d56c/notebooks/Time_Series_Getting_Started.ipynb

[56] granite-tsfm/tests/toolkit/test_time_series_forecasting_pipeline.py at ...
https://github.com/ibm-granite/granite-tsfm/blob/main/tests/toolkit/test_time_series_forecasting_pipeline.py

[80] cannot run `pip install git+https://github.com:IBM/tsfm.git` · Issue #9 ...
https://github.com/IBM/tsfm/issues/9

[83] IBM Granite 3.1: powerful performance, longer context and more ...
https://www.linkedin.com/posts/maximiliannitsche_ibm-granite-31-powerful-performance-longer-activity-7277138354838663168-4hZw

[88] IBM Granite - Hugging Face Transformers - Kaggle
https://www.kaggle.com/code/shravankumar147/ibm-granite-hugging-face-transformers

[91] IBM Granite 4.0 Tiny Preview: A sneak peek at the next generation of ...
https://www.ibm.com/new/announcements/ibm-granite-4-0-tiny-preview-sneak-peek