СОДЕРЖАНИЕ

1 Практическая часть

Abstract

1.1 Цель

С момента своей публикации, модель Трансформера [27] завоевала огромное призвание. Однако у нее есть несколько серьезных проблем, которые усложняют работу с длинными временным последовательностями (LSTF). Последующие исследования предложили различные методы решения данных и связующих проблем (см. Informer [32], Performer [3], Autoformer [30], PatchTST [16], TFT [14] и др.). В данной работе мы сфокусируемся на первых трех.

В данной работе, мы предлагаем заменить слой эмбеддинга в модели Informer компактным двухслойным сверточным блоком с целью повышения эффективности извлечения локальных паттернов, внедрить модуль декомпозиции ряда из Autoformer для явного разделения трендовых и сезонных компонент и заменить ProbSparse-внимание на линейное FAVOR+ из Performer для учёта глобальных зависимостей при низких вычислительных затратах.

1.2 Используемые данные

Прежде чем перейти к методологии нашей работы, рассмотрим модели и понятия, которыми далее будем пользоваться.

1.2.1 Informer

В 2020 году, Zhou et al., опубликовали свою статью «**Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting**», в которой представили новую, основанную на Трансформере [27] модель под названием **Informer** [32].

Informer был создан для решения задачи **Long-sequence time-series forecasting (LSTF)**. Zhou et al. поставили перед собой следующий вопрос: Можем ли мы построить модель, основанную на трансформере, которая

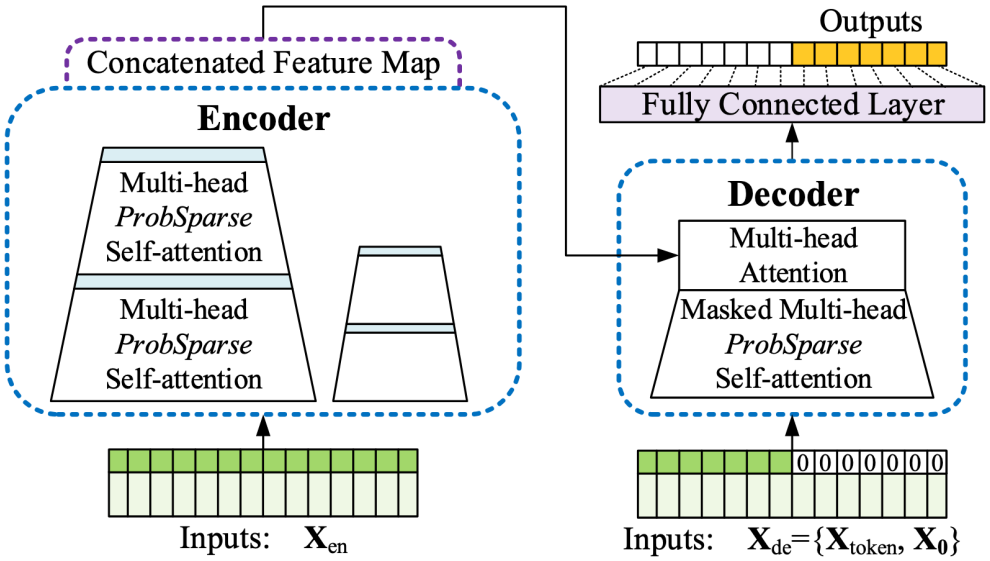


Рис. 1.1: Обзор модели Informer [32].

(a) захватывает очень длинные зависимости

(b) эффективно работает для тысячи временных шагов

Ключевые рассматриваемые слабости трансформера:

- Квадратичная стоимость механизма само-внимания (self-attention) для последовательностей длины L.
- Накладывание слоев умножает эту стоимость, достигая ограничений по памяти.
- Пошаговое (динамичное) декодирование работает медленно и накапливает ошибки.

Информер отвечает на каждый из этих вопросов, реконструируя механизм внимания, кодировщик и декодер.

1.2.1.1 ProbSparse Self-Attention

Зачастую, в длинных временных рядах, большинство скалярных произведений запросов с ключами пренебрежимо малы и лишь некоторые из них достаточно больши. Вместо того, чтобы считать все возможные пары запрос-ключ, informer предлагает следующее:

1. Измерить разброс каждого запроса q_i:

$$M(\boldsymbol{q}_i, \boldsymbol{K}) = \max_j \frac{\boldsymbol{q}_i \boldsymbol{k}_j^T}{\sqrt{d}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{\boldsymbol{q}_i \boldsymbol{k}_j^T}{\sqrt{d}},$$

2. Выбрать u лучших запросов, исходя из $M(\boldsymbol{q}_i, \boldsymbol{K})$, где $u \propto \ln L$.

3. Вычислить полное внимание только для этих u рядов и аппроксимировать остальные через среднее значение.

Что сокращает как время, так и память с $O(L^2)$ до $O(L \log L)$, при этом сохраняя всю важную информацию.

1.2.1.2 Self-Attention Distilling & Кодировщик

Даже после предыдщей операции, каждый слой все равно производит карты признаков длины L, многие из которых повторяют похожие паттерны. Мы можем «дистиллировать» сильнейшие сигналы и сократить последовательность по мере продвижения.

- После каждого блока с механизмом внимания, применяем:

    1. 1-D свертку + ELU активацию,
    2. max-pool со страйдом 2

Что уменьшает размерность в два раза на каждом слое, результируя в пирамиде стэков, чьи выходы в конечном итоге конкатенируют.

Self-attention distilling фокусируется на доминирующих паттернах, при этом сокращая память до $O((2-\varepsilon) L \log L)$.

1.2.1.3 Генеративный декодер

Вместо того, чтобы генерировать генерировать токены по очереди один за одним, заимствуем трюк с начальными токеном из NLP:

- Взять срез ивестной истории (например 5 дней перед целевыми 7ю днями) в качесвте начального токена

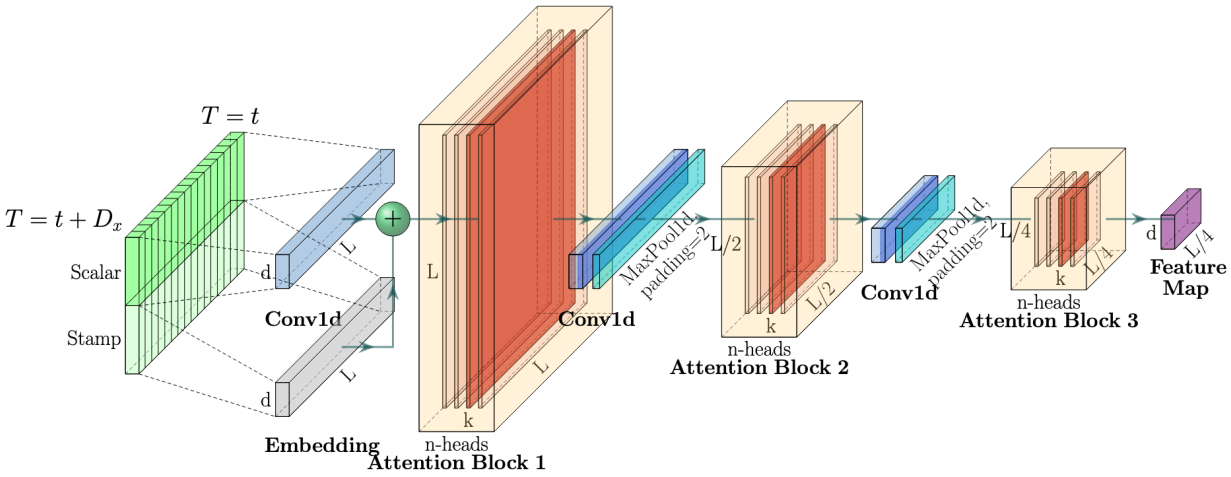


Рис. 1.2: Обзор одного стэка в кодировщике Informer-а [32].

- Разместить плейсхолдеры для всех L_y будущих значение (используя только их временные метки для позиционного контекста)
- За один прямой проход одновременно заполнить все L_y выходов через маскированное ProbSparse внимание.

Что избегает накапливания ошибки и работает гораздо быстрее.

1.2.2 Performer

В 2020 году, Choromanski et al., опубликовали статью «**Rethinking Attention with Performers**», в которой переработали устройство механизма внимания в классических Трансформерах [3].

Как уже было сказано в главе выше, одной из ключевых проблем Трансформера являеется квадратичная сложность расчета внимания: $O(L^2)$. В своей работе, Choromanski et al. задались вопросом:

Можем ли мы добиться той же гибкости глобального внимания, но за линейное время и память?

1.2.2.1 FAVOR+: Fast Attention via positive Orthogonal Random features

Вспомним, что в классическом Трансформере [27] внимание рассчитывается по следующей формуле (без учета нормирующей константы):

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V$$

Заметим, что

$$\exp(q_i^T k_j) = \kappa(q_i, k_j),$$

где κ - softmax ядро.

Согласно «**ядерному трюку**», мы можем представить κ в виде:

$$\kappa(q_i, k_j) = \langle \phi(q_i), \phi(k_j) \rangle,$$

где ϕ - некоторое (возможно, высоко- или бесконечно-мерное) отображение признаков.

Что, в свою очередь, мы можем аппроксимировать согласно **random features**:

$$\kappa(q_i, k_j) \approx \langle \widetilde{\phi}(q_i), \widetilde{\phi}(k_j) \rangle$$

Performer предлагает **FAVOR+**, объединяющий в себе две идеи:

1. **Positive Random Features (PRF)**: Вместо классических тригонометрических random features (которые могут принимать отрицательны значения), предлагается использовать отображение, основанное на экспоненте (таким образом все зачения будут неотрицательными).

2. **Orthogonal Random Features (ORF)**: Выбирать случайные проекции так, чтобы они были взаимно ортогональными, а не независимыми (что уменьшает дисперсию).

Откуда мы получаем выражение для ϕ:

$$\phi(\boldsymbol{x}) = \frac{1}{\sqrt{m}} \exp\left(\boldsymbol{W}\boldsymbol{x} - \frac{1}{2}||\boldsymbol{x}||^2\right),$$

где $\omega_i \sim \mathrm{N}(0, \boldsymbol{I}_d)$ - случайные ортогональные Гауссовы вектора (строки матрицы $\boldsymbol{W} = [\omega_1^T\ \omega_2^T ... \omega_m^T]^T$), m - кол-во случайных признаков (размерность $\phi(\boldsymbol{x})$).

Обозначив $\Phi_Q = \phi(Q) \in \mathbb{R}^{L \times m}$ и $\Phi_K = \phi(K) \in \mathbb{R}^{m \times d}$, получаем:

$$\text{Attention}(Q, K, V) = \Phi_Q(\Phi_K^T V) \in \mathbb{R}^{L \times d},$$

где расчет Φ_Q и Φ_K занимает $O(Lmd)$, расчет $\Phi_K^T V$ занимает $O(Lmd)$, перемножение с Φ_Q занимает $O(Lmd)$.

Итого имеем линейную сложность для расчета внимания (памяти также требуются только эти три матрицы).

1.2.3 Autoformer

В 2021 году, Wu et al., опубликовали статью «**Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting**», в которой представили новую модель, основанную на Трансформере - Autoformer с целью долгосрочного прогнозирования временных рядов. Auformer объединяет в себе автоматизированную декомпозицию временного ряда и новый механизм внимания, основанный на автокорреляции [30].

В данной работе мы рассмотрим только часть с декомпозицией.

1.2.3.1 Декомпозиция временого ряда

В классическом анализе временных рядов (например декомпозиция по сезонным трендам с помощью LOESS), декомпозиция обычно рассматривается как процесс предобработки. Часто применяется скользящее среднее (здесь и далее под скользящим средним подразумеваем Simple Moving Average (SMA)) для извлечения тренда и сезонности из прошлого ряда, после чего остаток подается на вход модели. Однако после предобработки мы теряем возможность позволить модели уточнить или скорректировать эту декомпозицию по мере обработки более глубоких слоев.

Autoformer предлагает внедрить механизм декомпозиции в саму модель. Таким образом она сможет самостоятельно разделять тренд и сезонные компоненты ряда в каждом слое.

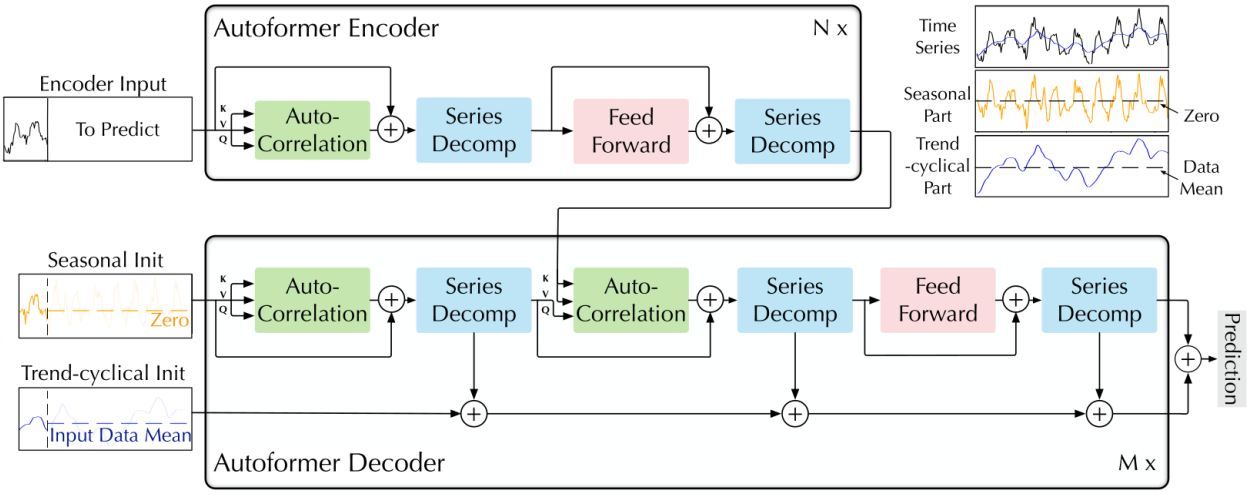


Рис. 1.3: Архитектура Autoformer-а [30].

На каждом слое l берется входная последовательность (которая уже могла быть частитично обработана предыдущими слоями), к которой, внутри слоя, применяется скользящее среднее, что далее разделяется на две части:

- $\text{Trend}^{(l)} = \text{SMA}(\text{Input}^{(l)})$
- $\text{Seasonal}^{(l)} = \text{Input}^{(l)} - \text{Trend}^{(l)}$

После чего мы передаем $\text{Seasonal}^{(l)}$ в наш механизм внимания, а $\text{Trend}^{(l)}$ направляем на вход в следующий слой (via residual connection). Схематично это можно изобразить следующим образом:

Input: $\text{H}^{(l-1)}$

(1) [SeriesDecomp] $\rightarrow \text{Trend}^{(l)}$, $\text{Seasonal}^{(l)}$

where $\text{Trend}^{(l)} = \text{SMA}(\text{H}^{(l-1)})$

$\text{Seasonal}^{(l)} = \text{H}^{(l-1)} - \text{Trend}^{(l)}$

(2) [selft-attention mechanism] on $\text{Seasonal}^{(l)}$

$\rightarrow \text{Seasonal}'^{(l)}$ (+ residual connection, normalization, etc.)

(3) [Feed-Forward] on $\text{Seasonal}'^{(l)} \rightarrow \widetilde{\text{Seasonal}}^{(l)}$

(4) Re-compose $\text{H}^{(l)} = \widetilde{\text{Seasonal}}^{(l)} + \text{Trend}^{(l)}$

1.3 Методология

1.3.1 Повышение эффективности извлечения локальных паттернов

В предлагаемой модификации мы заменяем *TokenEmbedding* внутри слоя *Embedding* модели Informer на два сверточных слоя, оставив без изменений *PositionalEmbedding* и *TemporalEmbedding* (рис. 1.4).

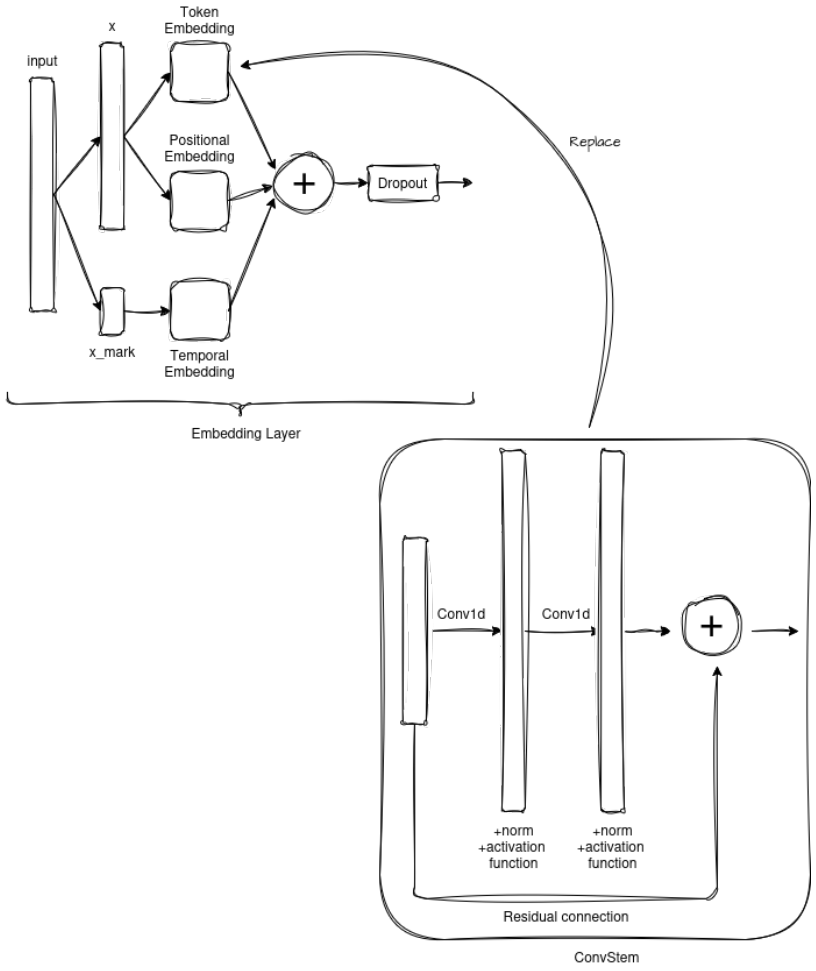


Рис. 1.4: Схема замены TokenEmbedding внутри Embedding слоя в Informer на ConvStem. ConvStem включает две последовательные свёртки: первая проецирует вход из c_{in} в d_{model}, вторая сохраняет размерность. В качестве нормализации была выбрана Instance normalization, в качестве функции активации - GELU. Также было добавлено остаточное соединение. Реализацию см. в листинге 1.

Данную замену мы мотивируем тем, что компактный двухслойный сверточный блок явно кодирует локальные временные паттерны и относительные позиции без дополнительных позиционных кодировок, позволяя глобальному вниманию сконцентрироваться на дальнодействующих зависимостях. Кроме того, ConvStem заметно сокращает число параметров и вы-

числительную нагрузку по сравнению с классическим TokenEmbedding, что критично в условиях ограниченных ресурсов GPU.

1.3.2 Заимствование механизма внимания из Performer

В качестве механизма внимания предлагается использовать FAVOR+ (из Performer) вместо ProbSparse (оригинально применяемого в Informer).

Подобно тому, как в оригинальном Informer механизм FullAttention был заменен на ProbSparse исключительно для вычисления self-attention в слоях кодировщика и декодера, в нашей модификации ProbSparse будет заменен на FAVOR+ в тех же местах, тогда как cross-attention останется реализованным через FullAttention.

Данную замену мы мотивируем тем, что FAVOR+ обеспечивает доказательно несмещённую аппроксимацию softmax-ядра с линейной по времени и памяти сложностью, что гарантирует предсказуемую и масштабируемую работу на очень длинных последовательностях в условиях ограниченных ресурсов GPU. Кроме того, отказавшись от ручной выборки *top-u* запросов, мы позволяем модели изучать любые глобальные зависимости (а не только наиболее сильные периодические), что критично для учёта непериодичных скачков в реальных временных рядах. Реализацию см. в листинге 2.

1.3.3 Внедрение модуля декомпозиции ряда из Autoformer

Чтобы явно разделить тренд и сезонную компоненту, после каждого блока Self-Attention мы вставляем модуль *SeriesDecomp*, позаимствованный из Autoformer.

Последующая обработка устроена так же, как в Autoformer: сезонная составляющая подаётся в слой внимания, а тренд передаётся через остаточное соединение к следующему уровню.

Таким образом, итоговая архитектура объединяет сильные стороны трёх моделей: локальные паттерны ConvStem, линейное глобальное внимание FAVOR+ и эксплицитную декомпозицию ряда из Autoformer, оставаясь при этом полностью совместимой с остальными гиперпараметрами оригинального Informer.

1.4 Эксперимент

1.4.1 Датасет

В данной работе мы проводим эксперименты на датасете (**Electricity Transformer Temperature**) [32]: ETT является важнейшим индикатором долгосрочного развертывания электроэнергетики. Создатели Informer собрали данные за 2 года из двух отдельных уездов Китая. Датасет разбит на несколько частей: { ETTm1, ETTm2 }, где данные записаны по-минутно, а также { ETTh1, ETTh2 }, где данные записаны по-часово для быстрой разработки. Каждый элемент данных состоит из 8 признаков, включая дату измерения, целевую переменную ”температура масла”и 6 различных видов признаков внешней силовой нагрузки.

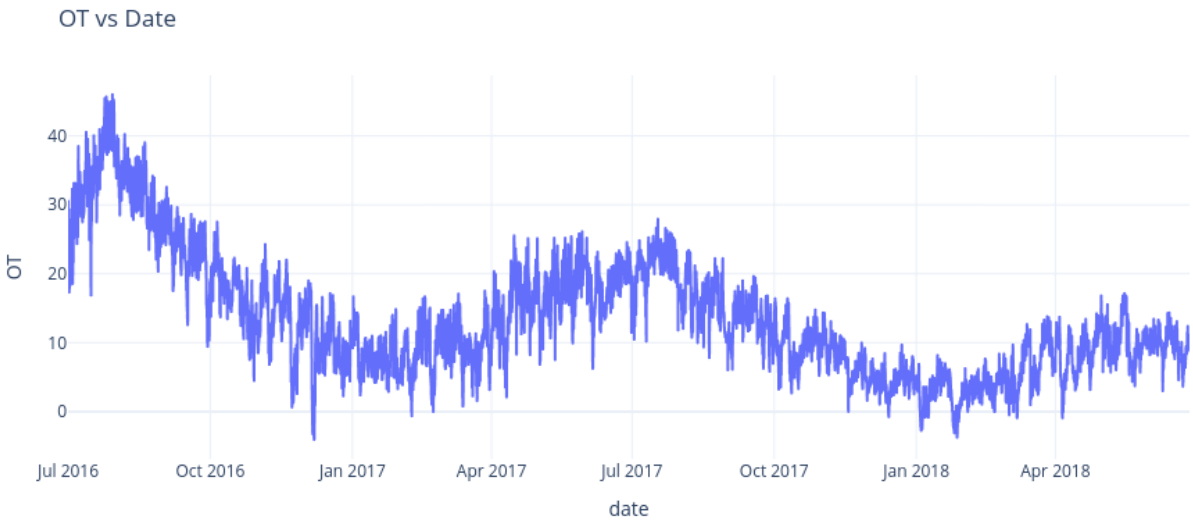


Рис. 1.5: Общий вид «OT» в ETT-small [32].

Field	date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
Description	The recorded **date**	**H**igh **U**se**F**ul **L**oad	**H**igh **U**se**L**ess **L**oad	**M**iddle **U**se**F**ul **L**oad	**M**iddle **U**se**L**ess **L**oad	**L**ow **U**se**F**ul **L**oad	**L**ow **U**se**L**ess **L**oad	**O**il **T**emperature (target)

Таблица 1: Описание каждого столбца.

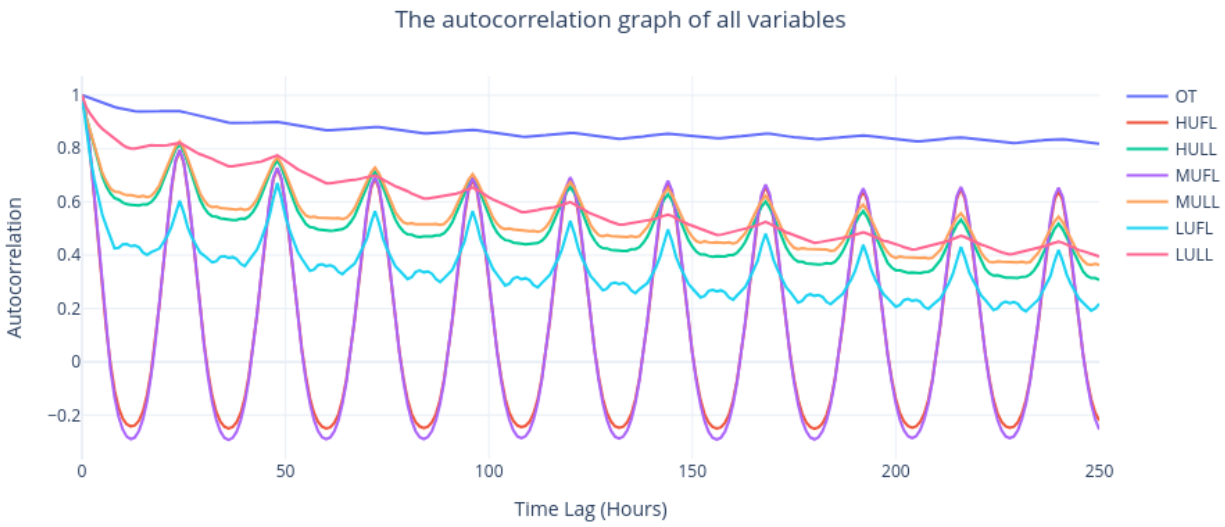


Рис. 1.6: График автокорреляции всех переменных [32].

1.4.2 Benchmark

Сравнение эффективности оригинальной модели Informer и модели Convformer на метриках MSE & MAE & RMSE & MAPE & MSPE (таблица 2).

Horizon	Informer					**Convformer**				
	MSE	MAE	RMSE	MAPE	MSPE	MSE	MAE	RMSE	MAPE	MSPE
24	0.555	0.546	0.744	11.800	**50086.8**	**0.391**	**0.434**	**0.625**	**11.208**	51139.6
48	0.659	0.615	0.811	11.837	**47824.1**	**0.419**	**0.445**	**0.647**	**10.910**	48182.0
168	0.838	0.714	0.915	**10.581**	**37781.6**	**0.430**	**0.456**	**0.656**	11.274	45336.6
336	1.237	0.903	1.111	17.100	108836.9	**0.460**	**0.485**	**0.679**	**12.572**	**57557.9**
720	1.181	0.870	1.086	17.186	113632.0	**0.518**	**0.533**	**0.720**	**12.902**	**60165.5**

Таблица 2: Результаты многомерных предсказаний на датасете ETTh1 с горизонтами предсказаний: { 24, 48, 168, 336, 720 }. Мы фиксируем входную длину последовательностей у моделей как 96. Результаты были найдены в ходе усреднения по 10 отдельным запускам для каждого из горизонтов.

1.4.3 Детали

Baselines: todo **Setup:** Входные данные каждого набора данных нормализованы по нулевому среднему значению. Согласно устройству LSTF, мы постепенно увеличиваем размер горизонта предсказаний, т.е. { 1d, 2d, 7d, 14d, 30d, 40d }. **Metrics:** Мы пользуемся двумя метриками оценки, включая MSE $= \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y} - \hat{\boldsymbol{y}})^2$ и MAE $= \frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{y} - \hat{\boldsymbol{y}}|$ для каждого из горизонтов предсказаний (усредняя для многомерного предсказания), и

прогоняем весь набор с stride=1. **Platform:** Все модели были тренированы/тестированы на единственной NVIDIA GTX 1660 SUPER GPU. Исходный код доступен по ссылке: todo.

1.5 Ablations

1.5.1 Pure ConvStem

Сравнение эффективности оригинальной модели Informer и модифицированной с заменой TokenEmbedding внутри Embdding layer на ConvStem на метриках MSE & MAE (таблица 3).

Horizon	Informer		**ConvStem**	
	MSE	MAE	MSE	MAE
24	0.556	0.547	**0.503**	**0.504**
48	0.657	0.614	**0.578**	**0.548**
168	0.862	0.725	**0.853**	**0.720**
336	1.219	0.892	**1.132**	**0.861**
720	**1.183**	**0.871**	1.247	0.912

Таблица 3: Результаты многомерных предсказаний на датасете ETTh1 с горизонтами предсказаний: { 24, 48, 168, 336, 720 }. Мы фиксируем входную длину последовательностей у моделей как 96. Результаты были найдены в ходе усреднения по 10 отдельным запускам для каждого из горизонтов.

Было установлено, что при коротком горизонте предсказаний ConvStem превосходит Informer; с увеличением длины горизонта их результаты постепенно сравниваются, а при длинном горизонте ConvStem начинает уступать, что ожидаемо для сверточных механизмов.

1.5.2 ProbSparse -> FAVOR+

Сравнение эффективности оригинальной модели Informer и модифицированной с заменой ProbAttention на FAVOR+ на метриках MSE & MAE (таблица 4).

Было установлено, что на всех горизонтах предсказаний различия в результатах статистически незначимы, что ожидаемо, поскольку и ProbSparse ($O(L \log L)$ в среднем) и FAVOR+ (около $O(Lr)$ при $r \ll L$)

Horizon	Informer		with FAVOR+	
	MSE	MAE	MSE	MAE
24	0.556	0.547	0.525	0.528
48	0.657	0.614	0.680	0.626
168	0.862	0.725	0.889	0.751
336	1.219	0.892	1.067	0.842
720	1.183	0.871	1.090	0.842

Таблица 4: Результаты многомерных предсказаний на датасете ETTh1 с горизонтами предсказаний: { 24, 48, 168, 336, 720 }. Мы фиксируем входную длину последовательностей у моделей как 96. Результаты были найдены в ходе усреднения по 10 отдельным запускам для каждого из горизонтов.

направлены на ускорение классического механизма внимания ($O(L^2)$) при контролируемом уровне потерь в точности.

1.5.3 Informer with series decomp

Сравнение эффективности оригинальной модели Informer и модифицированной с добавление модуля декомпозии ряда из Autoformer на метриках MSE & MAE (таблица 5).

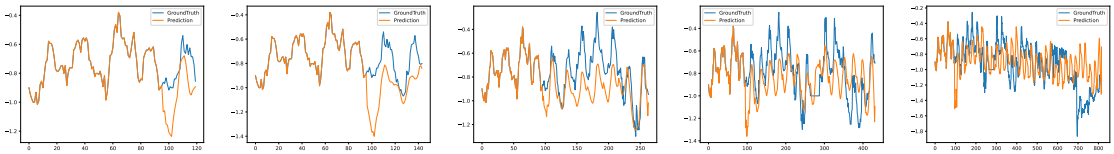
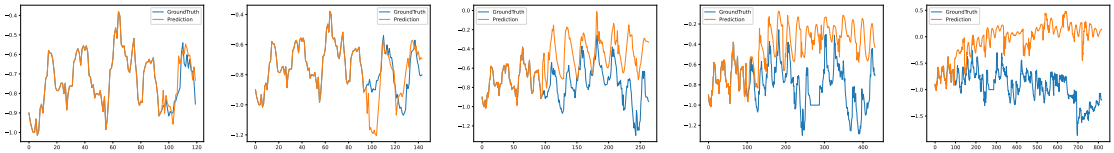
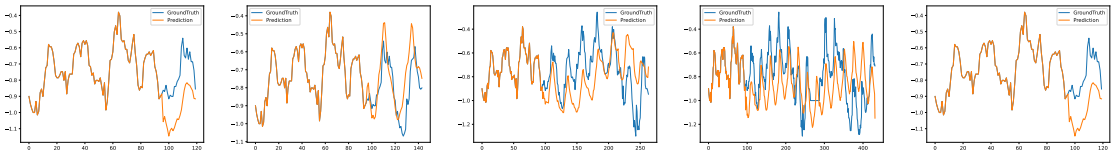
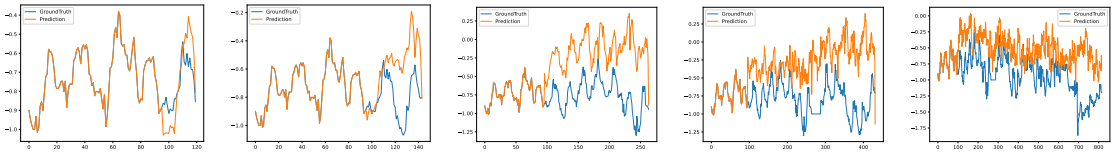
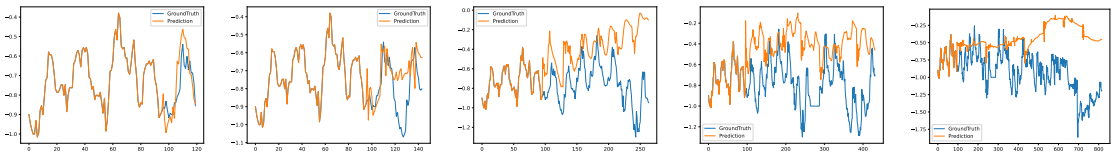
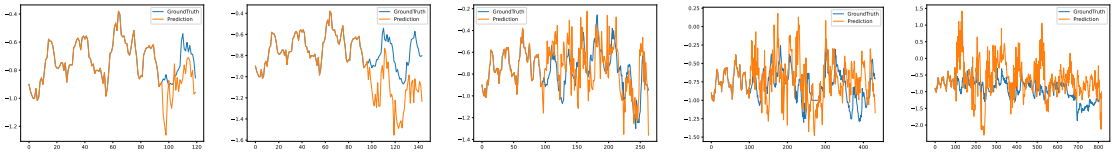
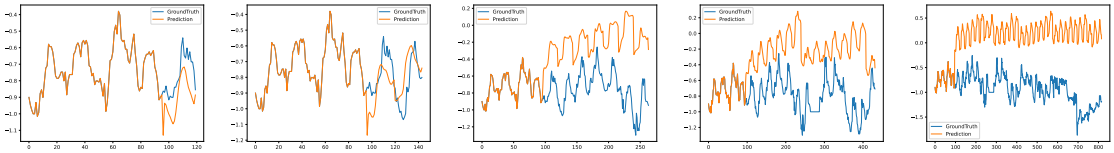
Horizon	Informer		**with decomp**	
	MSE	MAE	MSE	MAE
24	0.556	0.547	**0.457**	**0.484**
48	0.657	0.614	**0.550**	**0.541**
168	0.862	0.725	**0.667**	**0.606**
336	1.219	0.892	**0.951**	**0.746**
720	1.183	0.871	**1.085**	**0.783**

Таблица 5: Результаты многомерных предсказаний на датасете ETTh1 с горизонтами предсказаний: { 24, 48, 168, 336, 720 }. Мы фиксируем входную длину последовательностей у моделей как 96. Результаты были найдены в ходе усреднения по 10 отдельным запускам для каждого из горизонтов.

Было установлено, что ...

1.6 Дополнение к основным результатам

Таблица 6: Performance visualizations of different models across multiple forecast horizons.

Model	Horizon				
	24	48	168	336	720
Convformer					
Informer					
Autoformer					
Reformer					
ConvStem					
Decomp					
FAVOR					

To evaluate the prediction of different models, we plot the last dimension of forecasting results that are from the test set of ETT dataset for qualitative comparison (Figures 8, 9, 10, and 11). Our model gives the best performance among different models. Moreover, we observe that Autoformer can accurately predict the periodicity and long-term variation.

1.7 Результаты

1.8 Заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In International Conference on Learning Representations (ICLR), 2015.

[2] Jason Brownlee. Introduction to Time Series Forecasting with Python: How to Prepare Data and Develop Models to Predict the Future. Machine Learning Mastery, v1.9 edition, 2020.

[3] Krzysztof Choromanski, Valentin Likhosherstov, David Dohan, Xingyou Song, Alex Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Łukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In Proceedings of the International Conference on Learning Representations (ICLR), 2020. arXiv:2009.14794.

[4] Gregory C. Reinsel George E. P. Box, Gwilym M. Jenkins and Greta M. Ljung. Time Series Analysis: Forecasting and Control. Wiley, fifth edition edition, 2015.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence. Springer, 2012.

[7] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems, 28(10):2222–2232, 2017.

[8] James D. Hamilton. Time Series Analysis. Princeton University Press, 1994.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[10] Rob J. Hyndman and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2013.

[11] Brockwell P. J. and Davis R. A. Introduction to Time Series and Forecasting. Springer, third edition edition, 2016.

[12] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. https://karpathy.github.io/2015/05/21/rnn-effectiveness/#:~:text=models%20perform%20this%20mapping%20using,A%20few%20examples%20may, 2015. [Online; accessed 2025-15-05].

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105, 2012.

[14] Bryan Lim, Sercan Ö. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. In Proceedings of the International Conference on Learning Representations (ICLR), 2021. arXiv:1912.09363.

[15] Kevin P. Murphy. Probabilistic Machine Learning: An introduction. MIT Press, 2022.

[16] Wenjie Nie, Di He, Tao Qin, Ming Zhou, and Tie-Yan Liu. A time series is worth 64 words: Long-term forecasting with transformers. In Proceedings of the International Conference on Learning Representations (ICLR), 2023. arXiv:2211.14730.

[17] Michael Nielsen. Neural Networks and Deep Learning. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/.

[18] Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. Distill, 2016.

[19] Christopher Olah. Neural networks, manifolds, and topology. https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/, 2014. [Online; accessed 2025-06-05].

[20] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/#:~:text=Recurrent%20neural%20networks%20address%20this,them%2C%20allowing%20information%20to%20persist, 2015. [Online; accessed 2025-15-05].

[21] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 1958.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. Nature, 323:533–536, 1986.

[23] Grant Sanderson. Attention in transformers, step-by-step | dl6. https://www.youtube.com/watch?v=eMlx5fFNoYc&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=7, 2023. YouTube video, 3Blue1Brown.

[24] Josh Starmer. Statquest: Machine learning series. https://www.youtube.com/playlist?list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF, 2020. YouTube playlist, StatQuest with Josh Starmer.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems (NeurIPS), volume 27, 2014.

[26] Richard E. Turner. An introduction to transformers. Lecture or Technical Report, 2023. Department of Engineering, University of Cambridge and Microsoft Research, Cambridge, UK.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems (NeurIPS), volume 30, 2017.

[28] Brussels WDC-SILSO, Royal Observatory of Belgium. Sunspot number data. https://www.sidc.be/SILSO/datafiles, 2023. Source: WDC-SILSO, Royal Observatory of Belgium, Brussels.

[29] David H. Wolpert and William G. Macready. No free lunch theorems for

optimization. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 1:67–82, 1997.

[30] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with Auto-Correlation for long-term series forecasting. In Advances in Neural Information Processing Systems, 2021.

[31] Haoyi Zhou, Jianxin Li, Shanghang Zhang, Shuai Zhang, Mengyi Yan, and Hui Xiong. Expanding the prediction capacity in long sequence time-series forecasting. Artificial Intelligence, 318:103886, 2023.

[32] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference, volume 35, pages 11106–11115. AAAI Press, 2021.

ПРИЛОЖЕНИЕ В

Программный код, используемый в данной работе.

ConvStem

Листинг 1: Реализация блока ConvStem, состоящего из двух последовательных сверточных слоев с нормализацией и функцией активации.

```
class ConvStem(nn.Module):
    def __init__(self, c_in, d_model):
        super(ConvStem, self).__init__()

        self.conv0 = nn.Conv1d(c_in, d_model, kernel_size=1)

        self.conv1 = nn.Conv1d(in_channels=c_in,
                               out_channels=d_model,
                               kernel_size=5,
                               padding=2)
        self.norm1 = nn.InstanceNorm1d(num_features=d_model, affine=True)
        self.activation1 = nn.GELU()

        self.conv2 = nn.Conv1d(in_channels=d_model,
                               out_channels=d_model,
                               kernel_size=3,
                               padding=1,
                               groups=d_model)
        self.norm2 = nn.InstanceNorm1d(num_features=d_model, affine=True)
        self.activation2 = nn.GELU()

    def forward(self, x):
        x = x.permute(0, 2, 1)
        res = self.conv0(x)

        y = self.conv1(x)
        y = self.norm1(y)
        y = self.activation1(y)

        y = self.conv2(y)
        y = self.norm2(y)
        y = self.activation2(y)
```

```
        out = res + y
        return out.permute(0, 2, 1)


class ConvEmbedding(DataEmbedding):
    def __init__(self, c_in, d_model, embed_type='fixed', freq='h', dropout=0.1):
        super().__init__(c_in, d_model, embed_type, freq, dropout)

        # replace TokenEmbedding with ConvStem
        self.value_embedding = ConvStem(c_in, d_model)

    def forward(self, x, x_mark):
        return super().forward(x, x_mark)
```

FAVOR+ wrapper

Листинг 2: Реализация механизма линейного внимания FAVOR+ на основе библиотеки fast_transformers.

```
class FAVORAttention(nn.Module):
    def __init__(self, mask_flag, d_model, n_heads, n_random_features=256,
                 attention_dropout=0.1, output_attention=False):
        super(FAVORAttention, self).__init__()

        self.d_k = d_model // n_heads
        self.mask_flag = mask_flag
        self.dropout = nn.Dropout(attention_dropout)

        fm_factory = Favor.factory(n_dims=n_random_features)
        if mask_flag:
            self.core = CausalLinearAttention(self.d_k, feature_map=fm_factory)
        else:
            self.core = LinearAttention(self.d_k, feature_map=fm_factory)

    def forward(self, queries, keys, values, attn_mask=None):
        B, L_Q, _, _ = queries.shape
        _, L_K, _, _ = keys.shape
        device = queries.device

        lengths = torch.full((B,), L_K, dtype=torch.int64, device=device)
        len_mask = LengthMask(lengths, max_len=L_K)
```

```
        if self.mask_flag:
            attn_mask = TriangularCausalMask(L_Q, device=device)
            lengths = torch.full((B,), L_Q, dtype=torch.int64, device=device)
            len_mask = LengthMask(lengths, max_len=L_Q)
        else:
            attn_mask = len_mask

        context = self.core(queries, keys, values,
                            attn_mask,
                            len_mask,
                            len_mask)

        # context = self.dropout(context)
        return context.contiguous(), None
```