Time Series Forecasting with LLM Architectures

Robert W. Smith

May, 2026

?contentsname?

1 Core Concepts

1.1 Natural Language Processing (NLP) Overview

To a statistician, modern NLP can be best understood not as a set of linguistic rules, but as high-dimensional modeling of discrete sequences. The field previously was dominated by rule-based systems, but now is data-driven with probabilistic models more similar to non-parametric estimation.

1.1.1 From Atomic Symbols to Vector Spaces

Historically, words were treated as discrete, independent atomic symbols using a one-hot-encoding style approach. This resulted in sparse, high-dimensional spaces where the distance between "dog" and "cat" was orthogonal, identical to the distance between "dog" and "microscope."

Modern NLP maps discrete tokens to dense, continuous vector spaces $\mathbb{R}^d$ (where d is typically between 512-4096). This is effectively a dimensionality reduction technique where geometric proximity encodes semantic similarity. These vectors are learned parameters, optimized to maximize the likelihood of observing a word give its context.

1.1.2 Modeling Sequences: The Move to Attention

Early statistical NLP used Markov chains (NLP terminology: N-Gram) to predict the next token based on the previous $n-1$ tokens. This strategy suffers heavily from the curse of dimensionality. Recurrent Neural Networks (RNNs) attempted to solve this problem by maintaining a hidden state vector that updated over time, essentially an autoregressive process with memory.

Modern NLP relies on the Transformer architecture. Instead of processing data sequentially, Transformers use an "Attention mechanism." [**vaswani2023attentionneed**] This mechanism calculates a weighted average of all input vectors to determine which parts of the sequence are relevant to the current prediction task. Attention can be thought of as a dynamic kernel smoothing estimator. The model learns to assign weights (importance) to different parts of the input data based on the data itself and not a fixed window.

1.1.3 Language Models as Generative Distributions

Language Models (like OpenAI's GPT-series) are probabilistic generative models trained on massive text corpora. Their fundamental objective is maximizing the joint probability of a sequence of tokens $x = (x_1, x_2, ..., x_T)$.

Using the chain rule of probability, this is factorized as:

$$P(x) = \prod_{t=1}^{T} P(x_t | x_1, ..., x_{t-1})$$

This type of model estimates the conditional probability of the next token x_t over the entire vocabulary, given the context of all previous tokens. This refactors the language problem into a extremely high-capacity non-parametric density estimation problem. Training these models involves minimizing the Kullback-Leiblier (KL) divergence between the model's predicted distribution and the empirical distribution of the training data.

1.1.4 Fine-Tuning and Alignment

While the "pre-training" phase learns the statistical structure of language, the resulting model merely predicts the most likely continuation. To make models helpful assistants, we apply Reinforcement Learning from Human Feedback (RLHF). [**ouyang2022traininglanguagemodelsfollow**]

Process:

- Supervised Fine-Tuning: The model is trained on specific input-output pairs
- Reward Modeling: A seperate statistical model is trained to predict a scalar reward signal (quality score) based on human preference data
- Policy Optimization: The language model is updated to maximize the expected reward while remaining as close to the original distribution as possible (using KL-regularization)

1.1.5 NLP Summary for the Statistician

Modern NLP has essentially solved the problem of encoding discrete, symbolic data into continuous manifolds where calculus based optimization (Stochastic Gradient Descent) can be applied. It treats language generation as sampling from a learned conditional probability distribution, parameterized by billions of weights and conditioned on an input prompt.

2 Literature Review

2.1 Natural Language Processing Overview

Phase 1: The Statistical Foundation of Language Modeling

Before diving into architecture, we must define the problem space in statistical terms.

The Joint Probability of Sequences

Fundamentally, language modeling is the estimation of a joint probability distribution over a sequence of discrete tokens $X = (x_1, x_2, \ldots, x_T)$, where each x_t belongs to a vocabulary V.

$$P(X) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$$

Statistician's View: This is an autoregressive model. Traditional N-gram models approximated this by truncating the conditional history (Markov assumption) to order n. Transformers aim to model this probability without a fixed horizon constraint, effectively capturing long-range dependencies.

Distributional Semantics and Embeddings

Discrete tokens are mathematically sparse and orthogonal (one-hot vectors). We map these to a continuous vector space $\mathbb{R}^d$.

The Matrix Factorization View: Early methods (like LSA) used SVD on co-occurrence matrices. Modern approaches (Word2Vec, and later the Transformer's input layer) learn a lookup matrix $E \in \mathbb{R}^{|V| \times d}$. Goal: To maximize the dot product (cosine similarity) of vectors that appear in similar distributions of contexts.

Phase 2: The Transformer Architecture (The "Non-Recurrent" Shift)

Prior to 2017, Recurrent Neural Networks (RNNs) processed data sequentially $(t, t+1, \ldots)$. Transformers process the entire sequence in parallel using a mechanism called Self-Attention.

Positional Encodings

Since the model processes the sequence as a set rather than a list, we must inject order information.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Statistician's View: This is essentially adding a deterministic, high-frequency "time covariate" to the input embeddings so the regression function can distinguish between identical tokens at different positions.

Scaled Dot-Product Attention (The Kernel)

This is the core differentiator. For a query matrix Q, key matrix K, and value matrix V:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Statistician's View: QK^T: A similarity matrix (Gram matrix) measuring the correlation between every token and every other token. Softmax: Creates a probability distribution (weights sum to 1). This acts as a kernel smoother or a Nadaraya-Watson estimator. We are computing a weighted average of the "Values" (V) based on the similarity between "Queries" and "Keys." $\sqrt{d_k}$: A scaling factor to prevent gradients from vanishing (keeping the variance of the dot products stable).

Multi-Head Attention

Instead of one attention mechanism, we run h mechanisms in parallel on projections of the data, then concatenate results.

Statistician's View: This is analogous to ensemble methods or mixture models. Different "heads" learn to focus on different syntactic or semantic relationships (e.g., Head 1 tracks subject-verb agreement; Head 2 tracks pronoun coreference).

2.2 Training Paradigms (Inference and Estimation)

Modern NLP relies on a two-step procedure: Pre-training (Unsupervised/Self-supervised) followed by Fine-tuning (Supervised).

Pre-training Objective: Masked Language Modeling (BERT)

Given a sequence X, we corrupt it by masking 15

$$\mathcal{L}MLM = -\sum \bar{x} \in m \log P(\bar{x} \mid \tilde{X}; \theta)$$

Statistician's View: This is a denoising autoencoder objective. It is also conceptually similar to imputation of missing data using the conditional distribution defined by the observed data.

Pre-training Objective: Causal Language Modeling (GPT)

We maximize the likelihood of the next token given previous tokens.

$$\mathcal{L}CLM = -\sum t \log P(x_t \mid x_{<t}; \theta)$$

Statistician's View: Standard Maximum Likelihood Estimation (MLE) on an autoregressive process.

Fine-Tuning

We take the parameters θ learned from pre-training and treat them as a warm start for a specific task (e.g., classification, sentiment analysis). We usually add a small linear layer on top and optimize a supervised loss (e.g., Cross-Entropy).

Statistician's View: Transfer learning. We are using the massive unlabelled dataset to learn a high-dimensional manifold (the prior), then using the labeled data to find the decision boundary within that manifold.

2.3 Modern Refinements and Evaluation

Regularization Techniques

Deep Transformers are prone to overfitting.

Dropout: Randomly zeroing out activations (Bernoulli noise) during training. Layer Normalization: Standardizing inputs to mean 0 and variance 1 across the feature dimension for each sample. Note: Unlike Batch Norm, Layer Norm is independent of the batch statistics, crucial for variable-length sequence modeling.

Decoding Strategies (Sampling)

When generating text, we sample from the output probability distribution $P(x_t \mid x_{<t})$.

Greedy Search: Always pick $\arg\max$. (Can get stuck in loops). Beam Search: Keep the top k most probable paths. Temperature Sampling: Rescale logits by $1/T$ before softmax. $T < 1$: Peaked distribution (low variance, conservative). $T > 1$: Flat distribution (high variance, creative). Top-P (Nucleus) Sampling: Sample from the smallest set of tokens whose cumulative probability exceeds p. This truncates the "long tail" of the distribution dynamically.

2.4 The "Black Box" Interpretability Challenge

For statisticians, the lack of distinct coefficients is often troubling.

Attention Maps

We can visualize the attention weights α_{ij} to see which input tokens influenced a specific output.

Caveat: Attention is necessary but not sufficient for explanation. High attention weight does not strictly imply causal importance.

Probing Classifiers

To test if the model "knows" syntax, we freeze the Transformer and train a linear regression on its internal states to predict linguistic features (e.g., part-of-speech).

Statistician's View: Testing for multicollinearity or information leakage between the latent representation and specific linguistic variables.

Summary for the Statistician

NLP with Transformers is essentially kernel-based regression (Attention) combined with hierarchical representation learning, optimized via stochastic gradient descent (SGD) to maximize the likelihood of sequence continuation or reconstruction. The innovation lies not in a new fundamental statistical theory, but in an architectural prior that perfectly suits the discrete, sequential, and contextual nature of human language.

2.5 Transformer Architecture

The transformer can be understood as a nonlinear, data-adaptive operator on sequences that replaces explicit state-space recursion with repeated applications of weighted averaging and pointwise nonlinear transformations. For statisticians, it is useful to view the architecture as a structured sequence of regression-like operations with learned, input-dependent weights. [**vaswani2023attentionneed**]

At its core, a transformer maps an input sequence

$$x_1, \ldots, x_n \in \mathbb{R}^p$$

into a sequence of latent representations

$$h_1^{(L)}, \ldots, h_n^{(L)} \in \mathbb{R}^d$$

through (L) stacked layers. Each layer consists of two components: self-attention and a position-wise feed-forward map, combined with residual connections and normalization.

Self-Attention as Adaptive Linear Smoothing

Self-attention replaces fixed-lag dependence with data-dependent linear combinations across all time indices. Given latent inputs (

$$H = (h_1, \ldots, h_n)$$

), three linear projections are formed:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V$$

, where (W_Q,W_K,W_V) are learned matrices. For each time index (i), the output is

$$\tilde{h}_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

, with weights

$$\alpha_{ij} = \frac{\exp\left(q_i^\top k_j / \sqrt{d_k}\right)}{\sum_{\ell=1}^{n} \exp\left(q_i^\top k_\ell / \sqrt{d_k}\right)}$$

.

From a statistical perspective, this is a soft, normalized kernel smoother, where:

- the kernel is learned rather than fixed,

- similarity is measured in a learned feature space,
- weights depend on the entire observed sequence.

Unlike classical smoothing or autoregression, the effective dependency structure is global and adaptive, not tied to time ordering unless explicitly enforced.

Multi-Head Attention as Parallel Projections

Instead of a single smoother, the transformer uses multi-head attention, running several attention mechanisms in parallel on different linear projections of the data. These heads can be interpreted as learning multiple, complementary notions of dependence (e.g., short-range vs. long-range, trend vs. seasonality). The outputs are concatenated and linearly recombined, analogous to forming a multivariate basis expansion before aggregation.

Feed-forward layers as local nonlinear regression

After attention, each position is passed through a position-wise feed-forward network:

$$h_i \mapsto W_2 \sigma(W_1 h_i + b_1) + b_2$$

, applied independently across time. This can be viewed as a nonlinear regression applied conditionally on the aggregated context produced by attention. Importantly, there is no interaction across time indices at this stage; all cross-time dependence is handled by attention.

Residual Connections and Normalization

Each sublayer is wrapped in a residual connection and layer normalization:

$$h \leftarrow \text{LayerNorm}(h + \text{Sublayer}(h))$$

. Statistically, this stabilizes estimation by preserving a linear identity path while allowing incremental nonlinear corrections, mitigating vanishing gradients and overfitting in deep stacks.

Positional information

Because attention alone is permutation-invariant, the transformer injects positional encodings—deterministic or learned functions of time index—added

to the inputs. These act as fixed regressors indicating temporal location, allowing the model to distinguish ordering while retaining global interactions
.

Encoder–decoder structure (when used for forecasting or translation)

In sequence-to-sequence settings, an encoder constructs latent summaries of the input sequence, while a decoder generates outputs autoregressively. The decoder includes masked self-attention, enforcing causal structure, and cross-attention, which performs regression of future values on learned summaries of the past.

Statistical interpretation

From a statistical standpoint, a transformer is:

- a stacked sequence of adaptive linear smoothers (attention),
- interleaved with pointwise nonlinear regressions,
- where dependence structure, effective dimension, and interaction range are learned from data rather than specified a priori.

Unlike ARIMA or state-space models, dependence is not parameterized explicitly; instead, it emerges through learned similarity kernels. Unlike Gaussian processes, kernels are input-dependent and evolve across layers. This framing clarifies both the expressive power of transformers and why, as noted in recent time-series analyses, they often operate in low-rank regimes, making them highly flexible yet compressible when applied to structured temporal data.

2.6 Chronos

Ansari et al. propose Chronos, a pretrained probabilistic time series forecasting framework that reformulates forecasting as a language modeling problem by discretizing real-valued time series into tokens and training standard transformer-based language models using a categorical cross-entropy objective. [**ansari2024chronoslearninglanguagetime**] The central insight is that, once numerical observations are appropriately scaled and quantized, forecasting can be treated analogously to next-token prediction, allowing off-the-shelf language model architectures to be used without time-series-specific architectural modifications.

Chronos operates by applying mean scaling to each time series and uniform quantization into a fixed vocabulary of bins, augmented with standard special tokens (e.g., PAD, EOS). Forecasting is performed autoregressively over these tokens, and probabilistic forecasts are obtained by sampling multiple future token trajectories and dequantizing them back into real values. This approach effectively performs regression via classification, enabling flexible,potentially multimodal predictive distributions while retaining the simplicity of categorical modeling.

The authors pretrain Chronos models based on the T5 architecture, ranging from 20M to 710M parameters, on a large corpus of publicly available time series datasets. To mitigate data scarcity and improve generalization, they introduce two augmentation strategies: TSMixup, which generates synthetic series via convex combinations of real series from different datasets, and KernelSynth, which produces synthetic time series using Gaussian processes with randomly composed kernels. These augmentations substantially expand the diversity of temporal patterns seen during pretraining.

Extensive experiments across 42 benchmark datasets demonstrate that Chronos significantly outperforms classical statistical models and task-specific deep learning approaches on datasets seen during training, while achieving competitive or superior zero-shot performance on unseen datasets—often without any task-specific fine-tuning. Compared to approaches that adapt large pretrained LLMs through prompting or fine-tuning, Chronos attains strong performance with modest model sizes and lower inference cost.

Overall, the paper shows that minimal adaptations of language modeling techniques—specifically tokenization through scaling and quantization—are sufficient to produce a strong general-purpose, zero-shot time series forecaster. Chronos provides evidence that pretrained language-model-style approaches can substantially simplify forecasting pipelines and serves as a compelling baseline for future research on foundation models for time series forecasting.

2.7 Chronos-2

Ansari et al. introduce Chronos-2, a pretrained time series forecasting model that generalizes zero-shot forecasting from purely univariate settings to univariate, multivariate, and covariate-informed tasks within a single unified framework. [**ansari2025chronos2univariateuniversalforecasting**] The work addresses a key limitation of earlier time-series foundation models, including the original Chronos, which largely ignore inter-series dependencies and exogenous variables despite their central role in real-world forecasting problems.

Chronos-2 is built around a group attention mechanism that enables in-context learning across flexible groupings of time series. Groups may represent independent series, multiple variates of a multivariate system, collections of related items for cross-learning, or combinations of targets with past-only and known covariates. By alternating time attention (along the temporal axis) with group attention (across series within a group), the model learns to infer inter-dependencies dynamically at inference time, without requiring task-specific architectural changes or fine-tuning.

The model adopts an encoder-only transformer architecture inspired by T5 and operates on patched, normalized real-valued inputs rather than discrete tokens. Inputs are robustly standardized using a $sinh^{-}1$ transformation to stabilize heavy-tailed distributions, augmented with explicit time-index and mask meta-features, and embedded via a residual patch encoder. Forecasts are produced through a direct multi-horizon quantile head, yielding probabilistic predictions over a dense grid of quantiles, including extreme tails to better capture rare events.

A central contribution of the paper lies in its training strategy. Because real multivariate and covariate-rich datasets are scarce at scale, Chronos-2 relies heavily on synthetic data generation. The authors introduce multivariatization procedures that impose contemporaneous and sequential dependencies on base univariate series, enabling the model to learn diverse multivariate structures and covariate relationships during pretraining. Empirical ablations show that these synthetic datasets are critical for enabling universal in-context learning and that models trained exclusively on synthetic data remain competitive.

Extensive evaluation on three large benchmarks: fev-bench, GIFT-Eval, and Chronos Benchmark II demonstrates that Chronos-2 achieves state-of-the-art zero-shot performance across probabilistic and point forecasting metrics. The largest gains are observed on covariate-informed tasks, where Chronos-2 substantially outperforms prior foundation models. Additional analyses show that in-context learning improves performance even for univariate tasks via cross-learning, particularly in short-history or cold-start regimes.

Overall, Chronos-2 establishes a universal forecasting paradigm in which a single pretrained model can be deployed "as is" across heterogeneous forecasting scenarios. The work demonstrates that combining group-based attention with large-scale synthetic pretraining enables practical, scalable, and flexible time series foundation models, significantly advancing the applicability of zero-shot forecasting in real-world systems

2.8 Understanding Transformers for Time Series

The paper analyzes transformer models for time series through the lens of numerical rank and low-rank structure, arguing that many architectural choices inherited from language models are theoretically and practically mismatched to time-series data. [**yu2025understandingtransformerstimeseries**]

The central empirical observation is that time-series embeddings are intrinsically low rank compared to embeddings arising from text or vision. Because time series are continuous, smooth, and typically patched from low-dimensional inputs, their embedded representations exhibit rapidly decaying singular value spectra. The authors formalize this using the notion of ϵ-rank and show, both empirically and theoretically, that common time-series embedding strategies—quantization-based tokenization and continuous MLP-based embeddings—map inputs into low-dimensional subspaces of the model's hidden space. For continuous embeddings, they provide approximation-theoretic guarantees: smooth or analytic embedding functions induce polynomial or exponential decay of singular values, implying strong compressibility independent of the ambient model dimension.

Building on this, the paper establishes theoretical links between low-rank inputs and low-rank attention. The authors prove that when the input embeddings lie in a low-rank subspace, the query, key, and value projections in self-attention can be uniformly approximated by low-rank matrices without loss of accuracy on those inputs. Conversely, they show that high-rank inputs (as in language or vision models) make attention layers fundamentally incompressible. This result provides a linear-algebraic explanation for why large attention matrices are often unnecessary in time-series transformers.

A key conceptual contribution is the notion of "flow-of-ranks." While early layers operate on strongly low-rank representations, nonlinearities, residual connections, and multi-head attention gradually increase numerical rank with depth. The authors characterize this rank inflation theoretically and empirically, showing that earlier layers are substantially more compressible than later ones and that the number of attention heads directly influences how quickly rank grows. This explains observed depth-dependent differences in compressibility and clarifies why uniform architectural scaling across layers is inefficient for time-series models.

These insights are applied to model compression and architectural design. Using truncated SVD, the authors compress attention matrices in a pretrained Chronos time-series foundation model, achieving large reductions in memory and inference time with negligible forecasting degradation. They further show that better results are obtained by pretraining compressed models, using layer-dependent low-rank parameterizations informed by the flow-of-ranks analysis, thereby improving the accuracy–efficiency Pareto frontier.

Overall, the paper provides a statistically grounded, linear-algebraic framework for understanding transformers on time series. It demonstrates that low-rank structure is not incidental but intrinsic to the data modality, yielding principled guidance for attention width, depth, and head count, and offering a theoretical justification for why time-series foundation models are far more compressible than their language counterparts.

3 Bibliography