ChatGPT

Advanced AI-Driven Feature Engineering for Financial Time Series

Financial modeling can benefit from automatically learned features that capture complex patterns beyond hand-crafted indicators. Modern methods include self-supervised representation learning, automated architecture/meta search, symbolic regression, topological analysis, change-point detection, and deep time-series networks. Each can produce rich embeddings from raw market data. Below we explain how each works, why it can beat manual features, how to apply it on market data, and its pitfalls.

Self-Supervised Representation Learning

Self-supervised models learn latent features from unlabeled time series by solving pretext tasks (e.g. predicting masked segments or contrasting augmented views). For example, **TS2Vec** employs hierarchical contrastive learning: it takes overlapping windows of a time series, constructs two augmented "contexts," and trains an encoder so that representations of the *same* sub-series under different contexts are similar, while other segments are dissimilar. This yields timestamp-level embeddings capturing multi-scale context [1] [2]. Likewise, models like **Series2Vec** train on a *similarity prediction* task instead of handcrafted augmentations: given two series, the network predicts if they are similar in temporal/spectral properties [3]. Such self-supervised features often outperform manual indicators because the network can discover subtle patterns (e.g. non-linear correlations or seasonality) automatically, at multiple resolutions, and without human bias [1] [3].

- **How it works:** Architectures like temporal convolutional networks or LSTMs are trained on tasks such as contrastive prediction (e.g. TS2Vec) or autoencoding/masking. For instance, TS2Vec's encoder maps each timestamp to a vector; contrastive losses on augmented windows force the model to capture true temporal context [1]. Series2Vec instead uses a similarity-loss: it embeds series into a space where "similar" time series (in time and frequency content) lie close [3].
- **Why better than manual features:** These methods learn directly from raw prices/volumes (or limit-order-book data) and can capture complex dynamics like multi-scale trends, nonstationary cycles, or hidden factors. They don't rely on pre-specified formulas (e.g. moving averages) so they may find patterns humans miss. In practice TS2Vec embeddings have given state-of-the-art gains on forecasting and anomaly detection, indicating richer representations than handcrafted metrics [4].
- **Implementation with market data:** One typically feeds raw time series (price, volume, indicators) into the self-supervised encoder. For contrastive models, we generate pairs of augmented series (e.g. cropped or scaled versions) from historical windows. We train on abundant historical data (or multiple instruments) and then use the encoder's outputs as features. For example, train TS2Vec on 1-year of OHLC (open/high/low/close) data to produce per-timestamp vectors. These vectors can then feed into downstream tasks (regression/classification).
- **Risks/Overfitting:** Self-supervised models can overfit if the pretext task is not well chosen. For instance, contrastive learning assumes augmentations preserve true semantics – in finance, "cropping" a series may break its meaning, so naive augmentations can mislead the model. Also, if the training data is not diverse, the learned features may latch onto spurious quirks (e.g. calendar

effects) that don't generalize. Finally, high-capacity models might memorize noise in historical data. Careful validation (e.g. on different time periods or assets) is needed to avoid fitting artifacts.

Neural Architecture Search and Meta-Learning

Automated model search techniques can design feature extractors without human tuning. **Neural Architecture Search (NAS)** treats model design as an optimization: given raw inputs, NAS algorithms (e.g. evolutionary search, Bayesian optimization, Hyperband, reinforcement learning) explore combinations of layers (CNNs, RNNs, MLPs) to maximize performance on a task [5]. For financial time series, a study found that simple "chain-structured" search spaces (sequential layers) suffice for small datasets: Bayesian and Hyperband search often find the best networks (often 1D CNNs or RNNs) for forecasting [5]. In effect, NAS can discover architectures that implicitly define complex feature transformations (combinations of convolutions, attention, etc.) that human engineers might not consider. **Meta-learning** (learning to learn) can similarly adapt model hyperparameters or features across tasks, enabling quick adaptation when market regimes shift.

- **How it works:** NAS frameworks iteratively sample network configurations, train them on historical data, and use the results to guide future sampling. For example, one might define a search space of possible CNN kernel sizes, number of layers, and activation functions, then use Bayesian optimization (Tree-structured Parzen Estimator) to pick promising designs [5]. Meta-learning approaches like MAML would train a base model across many simulated markets or time periods so it can fine-tune quickly on new data.
- **Why better than manual tuning:** Humans are limited in the architectures they try. NAS can uncover novel combinations (e.g. mixing RNN and CNN or unusual filter sizes) that yield features tailored to the data's structure. In finance, where relationships can be non-intuitive, NAS may pick up on cross-asset patterns or time-scale hierarchies humans overlook. Meta-learning likewise automates feature/hyperparameter adaptation when the market changes.
- **Implementation with market data:** One could use open NAS libraries (AutoKeras, NNI, etc.) on financial datasets. For example, define a search over LOB-CNN vs. transformer variants with specified hyperranges, and run Hyperband to allocate resources. NAS would be expensive (train many models), but can be done offline. The best-found architecture is then used as a feature extractor: its intermediate layer outputs or logits serve as features. Meta-learning requires defining "tasks" (e.g. forecasting multiple stocks) so the meta-learner learns across them.
- **Risks/Overfitting:** NAS can easily overfit small datasets: it may choose overly complex models that work well on past data but generalize poorly. The search itself may pick architectures that exploit statistical flukes. Also, it is computationally intensive. Meta-learned models may memorize common patterns across training tasks but fail if a new market behaves very differently. Careful cross-validation, ensembling, or limiting architecture complexity is needed to prevent "over-customization" to historical data.

Symbolic Regression (Discovering Equations)

Symbolic regression algorithms (e.g. **PySR**, **AI-Feynman**) search for mathematical formulas that fit the data. Rather than a fixed model form, these tools evolve or optimize expressions composed of operators (.+-*/ etc.) to best explain a target variable. For example, PySR uses genetic programming combined with sparse regression to discover concise symbolic formulas for datasets [6]. In finance, one might apply symbolic regression to returns and fundamentals to uncover new indicator formulas (e.g. a combination of moving

averages that best predicts volatility). Because the output is an explicit equation, the discovered "features" are interpretable formulas. AI-Feynman similarly trains a neural net as a guide, then applies physics-inspired heuristics to piece together a closed-form expression for the data.

- **How it works:** Symbolic regressors treat feature discovery as a search problem. They typically start with a pool of random formulas and iteratively refine them (by mutation, crossover, or gradient-free optimization), scoring each by how well it fits the data. PySR, for instance, can handle multivariate inputs: it will try to find an analytic formula $f(x_1,x_2,\dots)$ approximating a target y. The search continues until a simple or high-scoring formula is found. The result is a new feature: the output of that formula on new data.
- **Why better than manual features:** Symbolic methods can reveal non-obvious functional relationships among variables that humans might miss. They automatically test combinations of variables (and nonlinear functions) and can produce interpretable predictors. In finance, this could mean finding a closed-form relation among prices, volumes, and other signals that captures some latent dynamic. Because the result is a formula, it provides insight and avoids "black box" opacity.
- **Implementation with market data:** Collect input features (e.g. price history, technical indicators) and a target (future return or volatility) for a training period. Run symbolic regression (e.g. PySR in Python) to find an equation that predicts the target from inputs. The discovered equation becomes a new feature to feed into downstream models. Alternatively, one can "distill" a neural network by symbolic regression: train a deep model, then apply symbolic regression to its hidden-layer outputs to extract simpler formulas (see later). AI-Feynman is more specialized (for physics-like laws) but similar in spirit.
- **Risks/Overfitting:** Symbolic regression is prone to overfitting, especially with noisy data like financial series. It may find an equation that fits historical data perfectly but is just curve-fitting noise. Also, search is combinatorially hard, so scalability is limited; too many input variables or allowed operations can make it intractable. Discovered formulas might lack robustness: small changes in data can invalidate them. It's crucial to validate any formula on holdout data and check that it isn't capturing spurious coincidences.

Topological Data Analysis (Persistent Homology)

Topological data analysis (TDA) uses geometry of data to extract features invariant to noise. **Persistent homology** is a key TDA tool that measures topological features (like connected components, loops, voids) across scales. Applied to time series, one common approach is to convert a rolling window of the series into a point cloud (via delay embedding), then compute its homology. For example, a 1-day price history can be mapped into an n-dimensional space (using past values), and persistent homology will quantify cycles or voids in that trajectory. Researchers have found that before market crashes, the topology of price series changes: new loops appear and persist longer [7]. These features are summarized as *persistence diagrams* or *persistence landscapes*.

- **How it works:** First, embed the time series into a point cloud (e.g. via sliding-window embeddings or recurrence plots) [7] [8]. For each scale parameter, build a simplicial complex (connect points within a distance threshold) and compute homology (counts of components, loops, etc.). Track how long each topological feature persists as the scale grows – this produces the persistence diagram/landscape. Finally, convert the diagram to a fixed-size vector (persistence image or landscape coefficients) for ML. For instance, one might use a 10×10 persistence image or statistics of the lifetimes of loops.

- **Why better than manual features:** TDA features capture *shape* of the time series trajectory rather than pointwise statistics. They can detect multi-dimensional cycles, "holes" or recurring patterns that no simple technical indicator would see. In practice, topological features have been shown to spike before crises in stock markets [7], suggesting they capture regime shifts. They are also invariant under certain transformations (e.g. monotonic scalings) that would break traditional features. This robustness and ability to capture global structure can uncover hidden dynamics in price/volume data.
- **Implementation with market data:** Use libraries like Ripser, GUDHI, or Giotto-tda. For a univariate series (e.g. log-prices), form a delay-embedding (e.g. points $(p_t, p_{t-\tau}, ...)$ for τ delays). Compute persistent homology (usually 0D and 1D features) of this cloud. Vectorize the resulting barcode (e.g. persistence landscapes [7] or images [9]). These vectors become features corresponding to the time window. One can do this on rolling windows to feed into a real-time model. Another approach: form a "recurrence plot" matrix and apply PH on that image [10].
- **Risks/Overfitting:** TDA can be computationally heavy (building complexes). With small windows or noisy data, homology estimates can be unstable. Moreover, interpreting topological features is nontrivial – a "loop" in embedding space might or might not correlate with meaningful financial structure. There is also risk of incidental topology (noise forming a spurious loop). Careful choice of window size and validation is needed. Finally, transforming the multi-set of persistence points into a vector (landscape/image) can throw away information, and the vectorized features might still overfit if not regularized or if too many are used.

Change-Point Detection and Regime Modeling

Markets often shift regimes (bull/bear, high/low volatility), so features that capture these shifts can be powerful. Classic methods include **CUSUM**, **Hidden Markov Models (HMMs)**, and **Bayesian online change-point detection (BOCPD)**. For example, a Markov-switching model (a type of HMM) assumes returns follow different distributions in different hidden states (regimes) [11]. CUSUM is a sequential algorithm that signals a change when the cumulative sum of deviations crosses a threshold [12]. BOCPD maintains a posterior over the "run length" since the last change, updating it online as new data comes in [13]. The outputs (regime labels, change probabilities, or alarm signals) can serve as high-level features.

- **How it works:** 1) **HMM/Markov Switching:** Fit a model where each time point's regime is a latent state (e.g. "normal" or "crash" regime), with probabilities to switch states. Estimate by EM or Bayesian methods; the Viterbi algorithm yields most likely regime at each time. 2) **CUSUM:** Compute $S_t = \max(0, S_{t-1} + (x_t - \mu_0 - k))$ (where x_t is the observation and k a slack). When S_t exceeds a threshold, flag a change [12]. 3) **BOCPD:** Use the Adams-MacKay algorithm which, under assumed likelihoods, updates the probability that a change just occurred at each time. It outputs a *posterior* probability of a change at each new point [13].
- **Why better than manual features:** These methods explicitly model structural breaks or regimes that single-value indicators cannot. For instance, instead of a moving average, a regime label (e.g. "volatility high") informs the model that data is coming from a different process. Change-point scores act as signals for unusual events. Because they are adaptive and often nonparametric, they can detect shifts that static human features (e.g. fixed lookback returns) miss. HMMs have been used in finance since Hamilton (1988) and can capture shifts in means or variance effectively [11].
- **Implementation with market data:** Fit a regime model on historical returns or volatility. For HMMs, use existing packages (e.g. hmmlearn in Python) on daily returns to infer 2–3 regimes; then use the inferred state probability (or most likely state) as a feature. For CUSUM/BOCPD, apply online to

streaming returns: CUSUM thresholds can be tuned on historical variance, BOCPD can use an autoregressive likelihood. Many libraries exist (e.g. `ruptures` for offline detection, or `bayesian_changepoint_detection` for online). These produce change flags or a "time since last change" feature. Combine with price/volume inputs in a final model.

- **Risks/Overfitting:** These methods can be too sensitive or miss changes if mis-tuned. CUSUM requires pre-setting a mean shift k and threshold; if chosen poorly, it gives many false alarms or none at all. HMMs assume a fixed number of discrete regimes – real markets may have more complex transitions, so forcing 2–3 states can be an oversimplification. BOCPD relies on assumed data distributions (often Gaussian) and can be fooled by heavy tails or autocorrelation. All can overfit: an HMM might simply label noise as "state changes," or CUSUM might latch onto outliers. Cross-validation is tricky (as change points are endogenous); one should validate on separate time periods or simulate regime shifts to check robustness.

Deep Time-Series and LOB Representation Networks

Recent deep learning architectures for time-series capture raw patterns at multiple scales. Examples include **DeepLOB**, **InceptionTime**, and **Time2Vec** embeddings. DeepLOB is a specialized CNN+LSTM network designed for limit order book data: it applies 2D convolutions to the raw price–volume grid and LSTMs to extract temporal features. Remarkably, DeepLOB learned "universal" features that transfer to new stocks – it outperformed prior models on benchmark LOB datasets and generalized well out-of-sample [14] [15]. Similarly, InceptionTime (a TSC model) uses "Inception" modules of multiple filter sizes in parallel, automatically learning both short- and long-term patterns from raw series. **Time2Vec** is a module that encodes the notion of time itself as a learnable vector (with sinusoids and linear terms) that can be plugged into any model [16]. Together, these learn feature transformations directly from data without manual preprocessing.

- **How it works:** - *DeepLOB:* Takes raw multi-level order book snapshots (prices and volumes at several levels) as a 2D grid. Several convolution layers (with "inception-like" varied filter widths) extract local spatial features (e.g. local patterns across price levels), which feed into LSTM layers to capture dynamics [15]. - *InceptionTime:* An ensemble of CNNs each having inception modules: each module applies 1D convolutions with multiple kernel sizes in parallel, then concatenates the outputs. This lets the network automatically focus on patterns of different lengths in the time series. The ensemble improves stability. - *Time2Vec:* Transforms a scalar time (or timestamp) into a vector $(w_0 t + b_0, \sin(w_1 t + b_1), \sin(w_2 t + b_2), ...)$ with learnable frequencies (w_i) and phases (b_i) [16]. This embeds time-of-day, seasonality, or index features into a richer space that downstream nets can use.
- **Why better than manual features:** Deep CNN/LSTM models learn directly from raw data, so they can find subtle geometric and temporal patterns human features miss. DeepLOB's universal features show that the network discovered invariant LOB motifs predictive of price moves [14]. InceptionTime's multi-scale filters obviate the need to handpick window sizes: the network itself determines which timescales matter. Time2Vec avoids the need to manually encode time aspects (day of week, hour of day) by learning an optimal periodic embedding. All these reduce human bias and can adapt to complex nonstationarity.
- **Implementation with market data:** Preprocess the data as required (e.g. normalize LOB levels, resample OHLC). Use the published architectures or libraries: DeepLOB code is available, InceptionTime has open implementations (e.g. `tsai` Python package), and Time2Vec can be implemented as a simple network layer. Train on historical data: for DeepLOB, use high-frequency

LOB snapshots and future price directions as labels; for InceptionTime, any labeled time-series classification/regression task; for Time2Vec, embed timestamp features and feed into an RNN/transformer. The learned feature outputs (e.g. the last hidden layer of InceptionTime) become input features to your financial model.

- **Risks/Overfitting:** These deep nets have many parameters and can overfit if data is limited. For example, training a DeepLOB on insufficient LOB data may memorize noise. They also assume stationarity to some extent – if market microstructure changes (new tick rules, etc.), features may become invalid. Time2Vec, if misused, can overfit periodicities (e.g. invent a fake daily cycle) or fail if the true time pattern changes (e.g. daylight saving shifts). In all cases, validation on held-out periods or instruments is essential. Interpretability is also harder – unlike simple indicators, it's difficult to know *what* the network's features represent, which complicates trust.

Vector-Database Integration for Similarity Search

Once rich features are generated (from any of the above methods), they can be stored in a vector database for fast similarity search and recall of past signals. The idea is to map each time segment or feature vector into a high-dimensional embedding space, index it, and then use nearest-neighbor queries to find historically similar patterns. For example, one could index TS2Vec embeddings of rolling windows of stock prices. When the model observes a new context vector, it queries the database for "closest" historical vectors and retrieves the corresponding time periods and outcomes. This enables reuse of past signals (e.g. "market patterns similar to today led to X"). Hybrid approaches can combine sparse identifiers (e.g. symbol or regime) with dense vectors.

- **How it works:** Compute feature vectors (e.g. TS2Vec, PH persistence, deep-network embeddings) for many historical time windows. Store them in a vector database like FAISS, Milvus or Pinecone. At runtime, embed the current window and perform an approximate nearest-neighbor search in the vector DB. The results are indices to past periods with similar embeddings, which can provide labels or additional context. For example, if today's TS2Vec vector is close to past vectors from March 2020, the system can flag that similarity.
- **Strategy:** Ensure each stored vector is tagged with metadata (timestamp, asset, regime). Use hybrid search if needed (combine keyword filters with vector query). Periodically update the database as new data arrives or retrained models change the embedding space.
- **Risks:** The notion of "similarity" depends on the feature space – if the encoder is poorly tuned, neighbors may not actually behave alike. Embeddings may drift over time, so a vector DB built on old embeddings might become stale. High-dimensional spaces also suffer from the curse of dimensionality; proper scaling and dimensionality reduction (e.g. PCA) may be needed. Finally, sensitive to alignment: time-series similarity often requires alignment (dynamic time warping) which simple Euclidean vector similarity may not capture.

By combining these advanced feature techniques with vector search, one can create an end-to-end system: deep or topological models extract sophisticated embeddings from raw market data, and a vector database allows quick retrieval of analogous historical patterns. This can enrich financial models with data-driven context that human-crafted features alone would miss [17].

Sources: We drew on recent research in time-series learning and finance to outline each method. For example, TS2Vec's design and benefits are detailed in Yue et al. (AAAI-22) [1] [4]. Series2Vec's contrastive similarity approach is in Foumani et al. (DMKD 2024) [3]. Neural NAS results are from Levchenko et al.

(2024) [5]. Symbolic regression capabilities come from tools like PySR [6]. Topological methods in finance are discussed in Gidea & Katz (arXiv 2017) [7] and Ichinomiya (SciRep 2025) [9] [8]. Change-point techniques follow literature (Page 1954; Adams & MacKay 2007 [12] [13] [11]). DeepLOB, InceptionTime, and Time2Vec are presented in Zhang et al. (DeepLOB) [15] [14], Fawaz et al. (InceptionTime), and Kazemi et al. (Time2Vec) [16]. Finally, Pinecone notes that vector embeddings enable similarity-based time-series analysis [17]. Each citation anchors the above explanations in current research.

---

[1] [2] [4] TS2Vec: Towards Universal Representation of Time Series
https://cdn.aaai.org/ojs/20881/20881-13-24894-1-2-20220628.pdf

[3] Series2vec: similarity-based self-supervised representation learning for time series classification | Data Mining and Knowledge Discovery
https://link.springer.com/article/10.1007/s10618-024-01043-w

[5] Chain-structured neural architecture search for financial time series forecasting
https://ideas.repec.org/p/arx/papers/2403.14695.html

[6] GitHub - MilesCranmer/PySR: High-Performance Symbolic Regression in Python and Julia
https://github.com/MilesCranmer/PySR

[7] [1703.04385] Topological Data Analysis of Financial Time Series: Landscapes of Crashes
https://arxiv.org/abs/1703.04385

[8] [9] [10] Machine learning of time series data using persistent homology | Scientific Reports
https://www.nature.com/articles/s41598-025-06551-3?error=cookies_not_supported&code=6c68c842-ac90-4a41-9623-62fea37af357

[11] [12] [13] Bayesian Autoregressive Online Change-Point Detection with Time-Varying Parameters
https://arxiv.org/html/2407.16376v1

[14] [15] Deeplob: Deep Convolutional Neural Networks For Limit Order Books | PDF | Machine Learning | Deep Learning
https://www.scribd.com/document/659028248/1808-03668

[16] [1907.05321] Time2Vec: Learning a Vector Representation of Time
https://ar5iv.labs.arxiv.org/html/1907.05321

[17] Time Series Analysis Through Vectorization | Pinecone
https://www.pinecone.io/learn/time-series-vectors/