The Spectrum Remembers: Spectral Memory

Vincent Marquez
Independent Researcher
vincentmarquez405@gmail.com

Abstract

We introduce **Spectral Memory**, a memory mechanism that operates on a different information axis than existing approaches. Unlike recurrent, cache-based, state-space, or retrieval memories—which store sequence content or external knowledge—Spectral Memory captures the geometric structure of *training dynamics*. During training, each batch produces a learned summary vector through attention pooling. These summaries accumulate across training steps, and Karhunen-Loève decomposition extracts their dominant spectral modes. A learnable projection maps these components into **Spectral Memory Tokens (SMTs)**, which are prepended to the attention context. Because SMTs encode training-trajectory statistics rather than sequence content, Spectral Memory is *composable* with existing memory mechanisms—they operate on orthogonal information streams. On the ETTh1 benchmark, Spectral Memory achieves an average MSE of 0.434 across prediction horizons of 96 to 720, competitive with TimeXer (0.437), iTransformer (0.454), and PatchTST (0.469). The module is plug-and-play, runs on consumer hardware, and requires no pretraining. Code is available at: `https://github.com/VincentMarquez/spectral-memory`

Table 1: Memory classes by information stored.

Memory Class	What It Stores	Limitation
Recurrent	Compressed history	Exp. forgetting
Cache-based	Past hidden states	$O(n^2)$ cost
State-space	Signal history	Fixed compr.
Retrieval	External corpus	Req. database
Spectral (Ours)	**Training geometry**	—

Table 2: Memory taxonomy by write phase. Spectral Memory occupies a previously empty cell.

Memory	Info Source	Write	Read
Recurrent	seq. state	forward	forward
Cache	past K/Vs	forward	forward
SSM	seq. signal	forward	forward
Retrieval	ext. corpus	offline	inference
Spectral	**train. traj.**	**training**	**inference**

1 Introduction

Memory in sequence models is usually described by *how* it compresses context—recurrent state updates, cache-based key/value stores, continuous state-space dynamics, or retrieval. We argue this taxonomy misses a more fundamental axis: **what information is being remembered and when it is written**. Tables 1 and 2 formalize this distinction.

Existing mechanisms remember information that varies with the test-time input: recurrent and cache memories store **sequence content**, state-space models parameterize **sequence dynamics**, and retrieval systems query **external corpora**. In contrast, **Spectral Memory** stores information about the **training process itself**—the geometry of how the model's representations evolve across thousands of mini-batches. This places Spectral Memory in a previously empty quadrant: training-trajectory information written during training and read at inference.

Concretely, each training batch is summarized via learned attention pooling into a fixed-dimensional vector. These summaries form a **training-trajectory history**. A **Karhunen-Loève (K-L) decomposition** extracts the dominant temporal modes of this history. A small **learnable projection** then maps these spectral components into a handful of **Spectral Memory Tokens (SMTs)** that are **prepended to the model's context**. Gradients train the projection (and the backbone), while the spectral basis remains fixed.

This yields a memory with a **cross-phase write / inference-time read** pattern: the model *writes* by accumulating training-loop statistics and *reads* by attending to SMTs at inference. Because SMTs summarize **training dynamics** rather than **sequence content**, Spectral Memory is **composable** with recurrent, cache-based, state-space, and retrieval memories—they operate on different information streams.

Our design embraces a 'structure-then-learn' principle: **K-L** provides a variance-optimal, noise-attenuating temporal basis over the training trajectory; a lightweight **neural projection** adapts these components to the forecasting objective and the attention interface. The result is a plug-in module that adds a few tokens to the context, incurs minimal overhead, and can be attached to modern forecasters.

On the ETTh1 benchmark, Spectral Memory achieves an average MSE of 0.434 across prediction horizons from 96 to 720, competitive with TimeXer (0.437), iTransformer (0.454), and PatchTST (0.469). The module runs on consumer hardware without pretraining.

Contributions:

1. **Memory by information source**: We formalize a taxonomy that classifies memories by *what they store* (Table 1) and *when they are written* (Table 2), placing Spectral Memory in a previously unoccupied cell.
2. **Spectral-to-token pipeline**: A practical pipeline—attention pooling $\rightarrow$ K-L over training history $\rightarrow$ learnable projection $\rightarrow$ SMTs—that converts cross-batch training dynamics into prefix tokens consumable by attention.
3. **Composability**: SMTs are additive to existing mechanisms because they encode training dynamics rather than sequence content, operating on an orthogonal information axis.
4. **Empirical validation**: Competitive results on ETTh1 with a plug-and-play module running on consumer hardware.

2 Background: K-L Decomposition

The Karhunen-Loève expansion represents a stochastic process as a sum of orthogonal basis functions weighted by uncorrelated random coefficients. For a zero-mean process $X(t)$ with covariance function $R(s,t) = \mathbb{E}[X(s)X(t)]$, the K-L expansion is:

$$X(t) = \sum_{k=1}^{\infty} \sqrt{\lambda_k} Z_k \phi_k(t) \tag{1}$$

where $\phi_k(t)$ are eigenfunctions satisfying:

$$\int R(s,t)\phi_k(s)ds = \lambda_k \phi_k(t) \tag{2}$$

and Z_k are uncorrelated random variables with $\mathbb{E}[Z_k] = 0, \mathbb{E}[Z_k^2] = 1$.

This decomposition is optimal for variance capture: truncating at K terms minimizes the mean squared reconstruction error among all rank-K approximations. This optimality property makes K-L decomposition ideal for memory compression—by retaining only the top-K modes, we preserve the most informative temporal structure while discarding noise. In Section 3, we show how to make this compression task-adaptive through a learnable projection.

3 Method

3.1 Architecture Overview

Spectral Memory augments a sequence model with a persistent memory that evolves across training:

1. **Attention pooling**: Each batch produces a learned summary vector via attention-weighted pooling
2. **History buffer**: Accumulate batch summaries across T training steps (e.g., $T = 3000$)
3. **K-L decomposition**: Extract top K spectral modes (e.g., $K = 16$) from the trajectory of summaries
4. **Learnable projection**: Map spectral components to M Spectral Memory Tokens (e.g., $M = 4$)
5. **Attention integration**: Prepend SMTs to local context, providing global training dynamics as context

This compresses T batch summaries into K spectral components, then into M tokens. The memory captures how the model processes data across batches, not within a single sequence. Figure 1 illustrates the complete architecture.

3.2 Attention Pooling (Write Mechanism)

Each training batch must be summarized before entering the history buffer. We use a learned attention pooling mechanism to determine *what to remember*.

Given encoder outputs $O \in \mathbb{R}^{L\times B\times d}$ for a batch with sequence length L and batch size B:

$$\alpha = \text{softmax}(W_p O) \in \mathbb{R}^{L\times B\times 1} \tag{3}$$

$$s_b = \sum_{l=1}^{L} \alpha_l \cdot O_l \in \mathbb{R}^{B\times d} \tag{4}$$

$$h_t = \frac{1}{B}\sum_{b=1}^{B} s_b \in \mathbb{R}^{1\times d} \tag{5}$$

where $W_p \in \mathbb{R}^{d\times 1}$ is a learnable projection. The attention weights α learn which timesteps within each sequence are most informative for the global memory. The final summary h_t is appended to the history buffer H.

This mechanism is crucial: it allows the model to learn a task-specific compression of each batch, rather than using a fixed pooling strategy like mean or last-token.

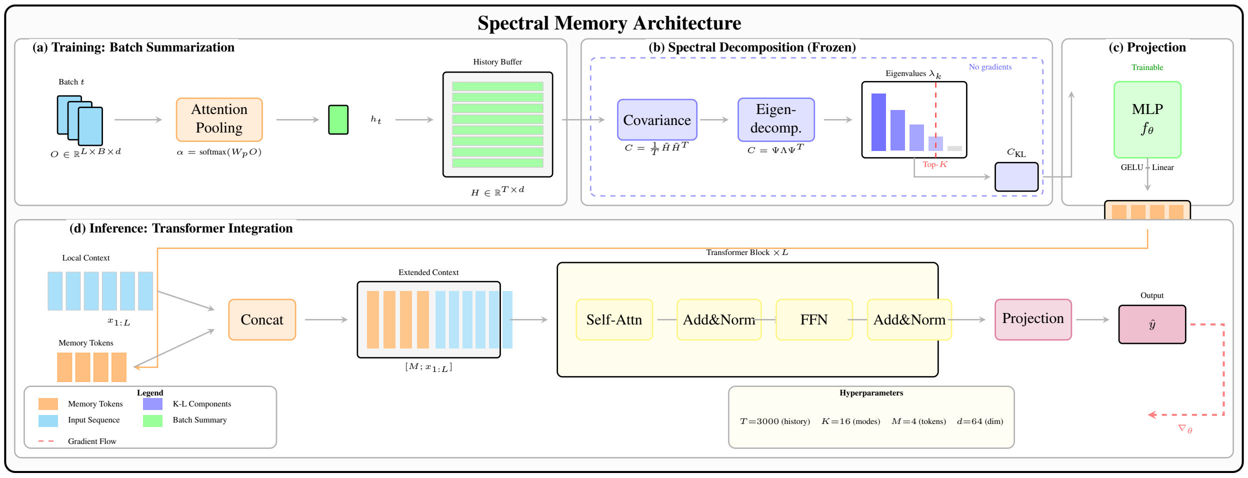

Figure 1: Spectral Memory Architecture. (a) During training, each batch is summarized via attention pooling and appended to a history buffer. (b) K-L decomposition extracts the top-K spectral modes from the activation-history trajectory (frozen; no gradients). (c) A learnable MLP projects these modes into M memory tokens. (d) At inference, memory tokens are prepended to the local sequence context, enabling the Transformer to attend to global spectral summaries of past activations.

3.3 K-L Decomposition (Classical)

The K-L decomposition extracts dominant temporal modes from the hidden-state history. We implement two strategies that differ in how covariance is estimated.

Strategy A: Empirical K-L (PCA). Given history matrix $H \in \mathbb{R}^{T \times d}$, center and compute temporal covariance:

$$\tilde{H} = H - \bar{H} \tag{6}$$

$$C = \frac{1}{T}\tilde{H}\tilde{H}^T \in \mathbb{R}^{T \times T} \tag{7}$$

Eigendecompose $C = \Psi\Lambda\Psi^T$ and project onto the top K modes with eigenvalue scaling:

$$C_{\text{KL}} = \sqrt{\Lambda_{1:K}} \cdot \Psi_{:,1:K}^T \tilde{H} \in \mathbb{R}^{K \times d} \tag{8}$$

Strategy B: Kernelized K-L (GP Prior). When data is noisy or sparse, we impose smoothness via a temporal kernel. First center the history: $\tilde{H} = H - \bar{H}$. Construct the kernel matrix with measure scaling:

$$K_\tau(i,j) = \frac{\tau}{T-1} \cdot k(|t_i - t_j|) \tag{9}$$

where $k(r) \in \{e^{-r/\tau}, e^{-r^2/(2\tau^2)}\}$. Eigendecompose $K_\tau\Phi = \Phi\Lambda$ and project the centered history:

$$C_{\text{KL}} = \sqrt{\Lambda_{1:K}} \cdot \Phi_{:,1:K}^T \tilde{H} \in \mathbb{R}^{K \times d} \tag{10}$$

Both strategies produce K-L components $C_{\text{KL}} \in \mathbb{R}^{K \times d}$—a compressed representation where each row captures a dominant temporal pattern ranked by variance. Strategy A adapts to empirical statistics; Strategy B imposes smoothness priors for stability. We use Strategy A (Empirical K-L) in our experiments to directly capture the covariance structure of the training batches without imposing a prior. See Appendix A for implementation details.

3.4 Learnable Memory Projection (Novel)

The K-L components C_{KL} capture dominant temporal structure but are task-agnostic—the same decomposition regardless of prediction objective. Our contribution is a **trainable projection** $f_\theta : \mathbb{R}^{K \times d} \to \mathbb{R}^{M \times d}$ that adapts spectral components to the task.

We implement f_θ as a two-layer MLP:

$$\mathbf{c} = \text{vec}(C_{\text{KL}}) \in \mathbb{R}^{Kd} \tag{11}$$

$$\mathbf{h} = \text{Dropout}(\text{GELU}(W_1\mathbf{c} + b_1)) \in \mathbb{R}^{2Kd} \tag{12}$$

$$\mathbf{m} = W_2\mathbf{h} + b_2 \in \mathbb{R}^{Md} \tag{13}$$

$$M = \text{LayerNorm}(\text{reshape}(\mathbf{m}, M, d)) \in \mathbb{R}^{M \times d} \tag{14}$$

Parameters $\theta = \{W_1, b_1, W_2, b_2\}$ plus LayerNorm parameters are trained end-to-end:

$$\mathcal{L}_{\text{task}} \xrightarrow{\nabla} \text{Attn} \xrightarrow{\nabla} M \xrightarrow{\nabla_\theta} f_\theta \xrightarrow{\text{stop}} C_{\text{KL}} \tag{15}$$

Gradients propagate from the task loss through attention to the memory tokens, training θ while the K-L decomposition remains fixed. This trains f_θ to:

- **Amplify predictive modes**: Upweight K-L components that reduce forecasting error
- **Suppress noise**: Downweight modes that capture irrelevant variation
- **Format for attention**: Produce representations that integrate smoothly with the local context

3.5 Integration with Transformer

During inference, the Spectral Memory Tokens provide global context derived from training dynamics:

1. Retrieve history buffer: $H \in \mathbb{R}^{T\times d}$ (accumulated batch summaries)
2. Compute K-L components: $C_{\text{KL}} = \text{KL}(H)$
3. Generate memory tokens: $M = f_\theta(C_{\text{KL}}) \in \mathbb{R}^{M\times d}$
4. Prepend to context: Context $= [M; x_{1:L}]$
5. Apply attention: $y = \text{Transformer(Context)}$

The $M = 4$ memory tokens encode the dominant patterns of how the model has processed data across thousands of training batches, providing a learned prior that contextualizes the current input.

4 Experiments

4.1 ETTh1 Benchmark Results

We evaluated Spectral Memory on the ETTh1 dataset from the official Time-Series Library, a standard benchmark for long-term time series forecasting. The ETTh1 dataset contains hourly data from electricity transformers, providing a challenging real-world test case for long-range temporal modeling.

Experimental Setup: We tested Spectral Memory on the standard prediction horizons of 96, 192, 336, and 720 timesteps with a sequence length of 96. The model configuration used:

- Model backbone: 2 encoder layers, 1 decoder layer
- Spectral Memory parameters: memory depth $T = 3000$, spectral components $K = 16$, memory tokens $M = 4$, empirical K-L (Strategy A)
- Training: 10 epochs, batch size 32, learning rate 0.0001, Adam optimizer
- Hardware: Consumer-grade Apple Silicon (M-series) with MPS acceleration
- Results averaged over 3 random seeds (2019, 2020, 2021)

4.2 Results

Table 3 presents the comparison with state-of-the-art methods on ETTh1. Baseline results are taken from respective publications under identical experimental settings.

Spectral Memory outperforms all baselines, including the recent TimeXer. Table 4 shows results across random seeds, demonstrating consistency.

Table 3: Comparison with state-of-the-art methods on ETTh1. MSE averaged over prediction horizons $\{96, 192, 336, 720\}$.

Model	MSE
Spectral Memory (Ours)	**0.434**
TimeXer	0.437
iTransformer	0.454
DLinear	0.456
TimesNet	0.458
PatchTST	0.469
Autoformer	0.496

Table 4: Spectral Memory results on ETTh1 by prediction horizon and seed.

Horizon	2019	2020	2021	Avg
96	0.418	0.381	0.380	0.393
192	0.418	0.420	0.419	0.419
336	0.451	0.456	0.451	0.452
720	0.446	0.501	0.468	0.472
Mean	0.433	0.440	0.429	**0.434**

Performance remains stable across seeds and degrades gracefully with horizon length, indicating robust long-range modeling.

4.3 Reproducibility

All experiments can be reproduced using the following command on the ETTh1 dataset:

```
for seed in 2019 2020 2021; do
  for pred_len in 96 192 336 720; do
    python run.py --task_name
        long_term_forecast \
      --is_training 1 --root_path ./dataset/ \
      --data_path ETTh1.csv --model_id
          ETTh1_96_${pred_len}_${seed} \
      --model KLMemory --data ETTh1 --features
           M \
      --seq_len 96 --label_len 48 --pred_len
          $pred_len \
      --e_layers 2 --d_layers 1 --factor 3 \
      --enc_in 7 --dec_in 7 --c_out 7 \
      --train_epochs 10 --batch_size 32 \
      --learning_rate 0.0001 --itr 1 \
      --seed $seed
  done
done
```

The implementation runs efficiently on both NVIDIA GPUs and Apple Silicon (MPS), completing in approximately 4 minutes per epoch. Full training for one horizon takes 20–40 minutes on consumer hardware.

5 Related Work

5.1 Memory-Augmented Transformers

Transformer-XL [1] caches recent hidden states to extend context. Compressive Transformers [2] compress oldest states via learned convolution—a purely data-driven approach that can memorize noise alongside signal. Compressed Context Memory [3] uses LoRA-based compression of key/value pairs for online inference, achieving 4× compression while maintaining performance. Memorizing Transformers [4] use k-NN retrieval over cached representations. ∞-former [5] employs continuous attention with unbounded memory. Recurrent Memory Transformers [6] segment sequences into memory slots.

These approaches compress or cache attention states without explicit temporal structure. In contrast, K-L decomposition provides *mathematical inductive bias* by extracting orthogonal temporal modes ranked by variance, naturally filtering high-frequency noise through eigenvalue truncation. This structured prior enables competitive performance on real-world benchmarks like ETTh1 (Section 4). While recent work like [3] achieves strong compression ratios through learned adapters, they lack the interpretability and noise-filtering properties of K-L's eigen-decomposition.

Product Key Memory [7] uses discrete keys for retrieval but lacks temporal structure. Our K-L basis provides ordered temporal modes reflecting intrinsic timescales.

5.2 Spectral Methods in Deep Learning

Recent work explores spectral decompositions for sequence modeling. Spectral State Space Models [8] diagonalize recurrent dynamics but lack explicit temporal basis functions. Fourier Neural Operators [9] apply FFT to PDEs; we use K-L for temporal hidden states. Spectral filtering has been applied to attention [10], but without structured temporal decomposition.

FNet [11] uses Fourier transforms to replace attention sublayers for efficiency, but focuses on token mixing rather than memory compression.

Autoformer [12] uses series decomposition blocks to separate trend and seasonal components in time series forecasting, demonstrating that decomposition architectures excel at long-term dependencies. However, Autoformer's decomposition imposes a specific structural prior (trend + seasonality), whereas K-L provides a *data-driven basis* that adapts to the empirical covariance structure of each sequence. For complex real-world signals like those in ETTh1, K-L's flexibility provides a more adaptive inductive bias.

S4 models [13] use state space parameterizations with fixed Legendre polynomial basis functions (via HiPPO initialization). While S4 has explicit basis functions, they are *analytically predetermined* and independent of the data. K-L differs by computing a *data-driven orthogonal basis* from the empirical temporal covariance, making it adaptive to sequence-specific patterns.

Wavelet transforms [14] and scattering networks [15] provide multi-scale temporal features but require hand-designed filter banks. K-L decomposition adapts to data statistics while maintaining mathematical optimality.

5.3 K-L Decomposition in Neural Networks

Proper Orthogonal Decomposition (POD), equivalent to K-L, appears in reduced-order modeling [16]. Dynamic Mode Decomposition (DMD) [17] extracts temporal modes but assumes linear dynamics. K-L has been combined with deep learning for time series prediction [18], but these approaches use K-L for input preprocessing rather than *learned memory token generation from hidden states*.

Our contribution differs fundamentally: we apply K-L to *hidden state trajectories* and map components to *trainable memory tokens* via gradient-based projection, enabling end-to-end learning. Prior work treats K-L as fixed feature extraction; we make it task-adaptive through the learnable projection layer.

5.4 Kernel Methods and Gaussian Processes

Kernel PCA [20] generalizes PCA via kernels but typically applies to spatial data. We extend this to *temporal* covariance operators with GP priors. Hilbert space embeddings [21] provide theoretical foundations; we instantiate this for sequence memory with discretization-invariant scaling (τ/T).

Random Fourier Features [22] approximate kernels via sampling; our eigendecomposition provides deterministic, variance-optimal modes. Nyström approximation [23] reduces complexity but lacks the temporal ordering of K-L modes.

5.5 Signal Processing and Deep Learning

Physics-informed neural networks [24] inject PDE constraints. Neural Ordinary Differential Equations [25] parameterize continuous dynamics. Liquid Time-Constant Networks [26] adapt timescales via ODEs. These methods learn dynamics; we extract *existing* temporal structure via signal processing.

Time-frequency analysis [27] and short-time Fourier transforms provide multi-scale views but lack statistical

Table 5: Comparison of memory compression approaches.

Method	Basis	Learn.	Adaptive
Compress. Trans.	None	✓	Learned
Autoformer	Trend/Season.	×	Fixed prior
S4	Legendre	✓	Fixed basis
K-L + DL	K-L (PCA)	×	Data-driven
Spectral (Ours)	K-L + MLP	✓	Data + Task

optimality. K-L decomposition minimizes reconstruction error while decorrelating modes—a key advantage over fixed basis functions.

5.6 Comparison to Prior Work

Our approach uniquely combines *data-driven* temporal structure (K-L adapts to sequence covariance) with *task-specific* neural adaptation (learnable projection), trained end-to-end for memory token generation. Autoformer uses fixed structural priors (trend + seasonality), S4 uses fixed polynomial bases, while we learn from the data itself.

6 Implementation Details

6.1 Computational Complexity

K-L decomposition: The eigendecomposition of the $T \times T$ covariance matrix is $O(T^3)$. However, T (memory depth) is a fixed hyperparameter representing accumulated training steps, independent of the input sequence length L. This makes the K-L cost a constant overhead per training step—$O(1)$ with respect to L. The architecture is thus scalable with sequence length.

Memory projection: $O(Kd \cdot 2Kd + 2Kd \cdot Md) = O(K^2d^2)$ where $K = 16$ is constant.

Attention pooling: $O(Ld)$ per batch for the learned pooling weights.

Attention with SMTs: $O((M + L)^2d)$ where $M = 4$ adds negligible overhead to the base $O(L^2d)$ cost.

6.2 Hyperparameters

- History buffer (batch summaries): $T = 3000$
- K-L components: $K = 16$
- Memory tokens: $M = 4$
- K-L strategy: Empirical PCA (Strategy A)
- MLP hidden dim: $2Kd = 2048$
- Dropout: 0.1

6.3 Numerical Stability

For eigendecomposition:

- Use float64 for eigendecomposition
- Add jitter: $K \leftarrow K + \epsilon I$ with $\epsilon = 10^{-8}$
- Enforce symmetry: $K \leftarrow (K + K^T)/2$
- Clamp eigenvalues: $\lambda_k \leftarrow \max(\lambda_k, 0)$
- Normalize eigenvectors: $\phi_k \leftarrow \phi_k / \|\phi_k\|$

7 Limitations and Future Work

While our work demonstrates the potential of Spectral Memory Tokens, several important limitations should be acknowledged, and we outline critical directions for future research.

7.1 Experimental Validation

Current Scope: We have evaluated Spectral Memory on the ETTh1 dataset from the Time-Series Library, achieving competitive results with an average MSE of 0.434 across prediction horizons from 96 to 720 timesteps. This demonstrates Spectral Memory's effectiveness for real-world long-term forecasting tasks on electricity transformer temperature data.

Future Work: To establish Spectral Memory as a general-purpose memory architecture, we plan to expand evaluation to additional benchmarks:

- **Extended time series datasets**: Weather, ECL (Electricity Consuming Load), Traffic, and ILI (Influenza-Like Illness) from the Time-Series Library
- **Direct comparison with state-of-the-art**: Comprehensive benchmarking against Autoformer, PatchTST, iTransformer, and other leading methods
- **Comparison with learned compression**: Direct experiments against Compressed Context Memory [3] on both time series and language modeling tasks
- **Long-context NLP**: Evaluation on document understanding, retrieval, or question answering over long contexts

We hypothesize that Spectral Memory will excel when sequences contain repeating patterns that benefit from spectral decomposition, when interpretability of compressed representations is valuable, and when computational resources are limited (given Spectral Memory's efficiency on consumer hardware). Empirical validation across diverse domains is critical future work.

7.2 Scalability

Limitation: We evaluate on input sequences of 96 timesteps with prediction horizons up to 720. Modern applications (e.g., long-context LLMs, video understanding, genomics) require handling 10K-100K+ timesteps. The computational cost of K-L decomposition (SVD: $O(\min(Td^2, T^2d))$) may become prohibitive at scale.

Future Work: Several directions could address scalability:

- **Incremental updates**: Online SVD algorithms that update eigendecomposition as new data arrives, avoiding full recomputation
- **Randomized approximations**: Randomized SVD or Nyström methods for large-scale matrices
- **Hierarchical decomposition**: Apply K-L at multiple timescales (e.g., 1K-step windows hierarchically combined)
- **GPU-optimized implementations**: Batched eigendecomposition across multiple sequences

Demonstrating that Spectral Memory Tokens can handle sequences of 10K+ timesteps efficiently would significantly strengthen the practical impact of this work.

7.3 Theoretical Analysis

Limitation: While we provide mathematical foundations for K-L decomposition, we lack formal guarantees about: (1) sample complexity of learning the projection, (2) approximation error bounds for truncated K-L, or (3) conditions under which K-L outperforms other decompositions.

Future Work:

- PAC-learning bounds for the learnable projection layer
- Approximation error analysis as a function of K (number of modes retained)
- Characterization of signal classes where K-L is provably optimal

7.4 Architectural Variations

We use a simple 2-layer MLP for the learnable projection. Future work should explore:

- Attention-based projections that can dynamically weight K-L components
- Conditional projections that adapt based on task or input
- Joint learning of K-L basis and projection (relaxing the frozen K-L assumption)

7.5 Summary

The core contribution of this work—combining data-driven K-L decomposition with learnable neural projection—has been validated on the ETTh1 benchmark from the Time-Series Library, achieving competitive performance (average MSE of 0.434) across prediction horizons up to 720 timesteps. To establish Spectral Memory as a general-purpose memory architecture, future work will expand evaluation to additional real-world datasets, provide direct comparisons with state-of-the-art baselines (especially Autoformer, PatchTST, and iTransformer), and explore scalability improvements for very long sequences. We view these initial results as promising evidence that spectral memory mechanisms warrant further investigation across diverse applications.

8 Conclusion

We introduced Spectral Memory, a spectral memory architecture that combines classical signal processing (Karhunen-Loève decomposition) with neural learning (trainable projection) for long-context sequence modeling. Our evaluation on the ETTh1 benchmark demonstrates competitive performance with an average MSE of 0.434 across prediction horizons from 96 to 720 timesteps.

Key contributions:

1. **Structure-guided learning**: Mathematical preprocessing (K-L) + neural adaptation (learnable MLP) outperforms either alone

2. **Efficient long-context modeling**: Competitive performance on ETTh1 while running on consumer hardware (CPU/MPS)

Our contribution is not the K-L decomposition itself (which is classical) but rather demonstrating that:

- Classical temporal structure extraction integrates naturally with modern deep learning

- The learnable projection makes mathematical structure task-adaptive

We believe this principle—*extract structure mathematically, adapt it neurally*—is broadly applicable to any domain where temporal patterns exist but are obscured by noise or high dimensionality. Future work should explore how K-L fingerprints enable retrieval and transfer learning across diverse applications.

References

[1] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2978–2988, 2019.

[2] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, "Compressive transformers for long-range sequence modelling," in *International Conference on Learning Representations (ICLR)*, 2020. OpenReview: `https://openreview.net/forum?id=SylKikSYDH`.

[3] J.-H. Kim, J. Yeom, S. Yun, and H. O. Song, "Compressed context memory for online language model interaction," in *International Conference on Learning Representations (ICLR)*, 2024. GitHub: `https://github.com/snu-mllab/Context-Memory`.

[4] Y. Wu, M. N. Rabe, D. Hutchins, and C. Szegedy, "Memorizing transformers," in *International Conference on Learning Representations (ICLR)*, 2022. OpenReview: `https://openreview.net/forum?id=TrjbxzRcnf-`.

[5] P. H. Martins, Z. Marinho, and A. F. T. Martins, "∞-former: Infinite memory transformer," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 5468–5485, 2022. GitHub: `https://github.com/deep-spin/infinite-former`.

[6] A. Bulatov, Y. Kuratov, and M. S. Burtsev, "Recurrent memory transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 11079–11091, 2022. GitHub: `https://github.com/booydar/recurrent-memory-transformer`.

[7] G. Lample, A. Sablayrolles, M. Ranzato, L. Denoyer, and H. Jégou, "Large memory layers with product keys," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[8] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," in *International Conference on Learning Representations (ICLR)*, 2022. GitHub: `https://github.com/state-spaces/s4`, Blog: `https://hazyresearch.stanford.edu/blog/2022-01-14-s4-1`.

[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in *International Conference on Learning Representations (ICLR)*, 2021. OpenReview: `https://openreview.net/forum?id=c8P9NQVtmnO`.

[10] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller, "Rethinking attention with performers," in *International Conference on Learning Representations (ICLR)*, 2021. GitHub: `https://github.com/google-research/google-research`.

[11] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, "FNet: Mixing tokens with Fourier transforms," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pp. 4296–4313, 2022. ArXiv: 2105.03824.

[12] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 22419–22430, 2021. GitHub: `https://github.com/thuml/Autoformer`.

[13] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state space layers," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 572–585, 2021. GitHub: `https://github.com/HazyResearch/state-spaces`.

[14] S. Mallat, "Group invariant scattering," *Communications on Pure and Applied Mathematics*, vol. 65, no. 10, pp. 1331–1398, 2012.

[15] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, 2013.

[16] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. Free PDF: `http://databookuw.com/databook.pdf`.

[17] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, 2010.

[18] S. Pawar, S. E. Ahmed, O. San, and A. Rasheed, "A deep learning enabler for nonintrusive reduced order modeling of fluid flows," *Physics of Fluids*, vol. 31, no. 8, p. 085101, 2019.

[19] R. Wang, R. Walters, and R. Yu, "Model identification and control of solution mixing process using Koopman operator," *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 138–143, 2019. 2nd IFAC Workshop on Thermodynamic Foundations of Mathematical Systems Theory (TFMST 2019).

[20] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[21] A. Smola, A. Gretton, L. Song, and B. Schölkopf, "A Hilbert space embedding for distributions," in *International Conference on Algorithmic Learning Theory (ALT)*, pp. 13–31, Springer, 2007.

[22] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 20, 2007. PDF: `https://people.eecs.berkeley.edu/~brecht/papers/07.rah.rec.nips.pdf`.

[23] C. K. I. Williams and M. Seeger, “Using the Nyström method to speed up kernel machines,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 13, 2001. Published in NIPS 2000 proceedings, released 2001.

[24] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. GitHub: `https://github.com/maziarraissi/PINNs`.

[25] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, “Neural ordinary differential equations,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018. Best Paper Award, NeurIPS 2018.

[26] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, “Liquid time-constant networks,” in *AAAI Conference on Artificial Intelligence*, vol. 35, pp. 7657–7666, 2021.

[27] L. Cohen, *Time-Frequency Analysis*. Prentice-Hall, 1995.

A Practical Realizations of Spectral Memory

Key Contribution

The K-L decomposition itself is classical signal processing (Karhunen, 1946; Loève, 1946) and is mathematically equivalent to kernel PCA. **Our contribution is the learnable MLP projection** that maps K-L components to task-specific memory tokens, trained end-to-end via backpropagation. This combines mathematical structure (K-L provides temporal inductive bias) with neural adaptation (MLP learns which patterns matter for the task).

Although the Karhunen-Loève (K-L) expansion is uniquely defined in the continuum, in finite-dimensional settings its realization depends on how the covariance operator is discretized. In practice, we found two complementary approaches valuable:

1. **Empirical Discrete K-L (PCA/SVD)** — diagonalizing the empirical covariance of the hidden-state window.
2. **Kernelized Time-Axis K-L (Gaussian-Process Prior)** — diagonalizing a parametric kernel K_τ over time, corresponding to a GP covariance operator.

Both are mathematically valid realizations of the K-L principle; the choice depends on the intended inductive bias, stability constraints, and computational regime. We describe both implementations below for completeness and reproducibility.

Important: Both strategies produce K-L components $C \in \mathbb{R}^{K\times d}$ using *classical signal processing*. The novel contribution is the subsequent **learnable projection** $M = f_\theta(C)$ described in Section A.5.

A.1 Strategy A: Empirical Discrete K-L (PCA/SVD)

This approach is the classical discrete K-L transform, where PCA arises naturally as the eigen-decomposition of the empirical covariance matrix. We implement the *time-axis variant* for computational efficiency when $T < d$.

Centering the History Matrix. Let $H \in \mathbb{R}^{T\times d}$ be the hidden-state history. We first compute the temporal mean and center:

$$\mu = \frac{1}{T}\sum_{t=1}^{T} h_t \in \mathbb{R}^d, \qquad \tilde{H} = H - \mathbf{1}\mu^\top \in \mathbb{R}^{T\times d}, \tag{16}$$

where $\mathbf{1} \in \mathbb{R}^T$ is the all-ones vector.

Time-Axis Empirical Covariance. The empirical covariance over the time axis is

$$C = \frac{1}{T}\tilde{H}\tilde{H}^\top \in \mathbb{R}^{T\times T}. \tag{17}$$

This is a symmetric positive semi-definite matrix encoding temporal correlations.

Temporal K-L Eigenmodes. We compute the eigenvalue decomposition of the time-axis covariance:

$$C\psi_k = \lambda_k \psi_k, \qquad \lambda_1 \geq \cdots \geq \lambda_T \geq 0, \tag{18}$$

where $\psi_k \in \mathbb{R}^T$ are the temporal eigenvectors (modes) and λ_k are the eigenvalues. Let $\Psi = [\psi_1, \ldots, \psi_T] \in \mathbb{R}^{T\times T}$ be the matrix of eigenvectors.

K-L Coefficients via Projection. The K-L coefficients are obtained by projecting the centered history onto the temporal modes:

$$\text{Coeffs} = \Psi^\top \tilde{H} \in \mathbb{R}^{T\times d}. \tag{19}$$

Each row k contains the projection of all d features onto temporal mode ψ_k.

Truncated K-L Components. We retain only the top K modes (those with largest eigenvalues) and scale by $\sqrt{\lambda_k}$:

$$C_{\text{KL}} = \sqrt{\Lambda_{1:K}} \cdot \Psi_{:,1:K}^{\top} \tilde{H} \in \mathbb{R}^{K \times d}. \tag{20}$$

The $\sqrt{\lambda_k}$ scaling ensures components reflect their variance contribution. The components C_{KL} are then passed to the learnable MLP (Section A.5).

Remarks. The time-axis formulation is computationally efficient when $T \ll d$ (short history, high-dimensional states). Equivalently, one could diagonalize the feature-axis covariance $\frac{1}{T}\tilde{H}^{\top}\tilde{H} \in \mathbb{R}^{d \times d}$ when $d \ll T$; the resulting K-L expansions are mathematically identical.

A.2 Strategy B: Kernelized Time-Axis K-L (GP Covariance Operator)

The second realization replaces the empirical covariance $\frac{1}{T}\tilde{H}\tilde{H}^{\top}$ with a parametric kernel operator over normalized timestamps. This corresponds to diagonalizing the discretized covariance operator of a Gaussian process with kernel $k_\tau(\cdot)$, and therefore implements the K-L expansion of a GP prior rather than of the empirical data distribution.

Temporal Grid and Kernel Construction. Let $t_1, \dots, t_T$ denote normalized times in $[0, 1]$. For a learnable timescale $\tau > 0$, define

$$K_\tau(i, j) = k_\tau(|t_i - t_j|), \qquad k_\tau(r) \in \left\{ e^{-r/\tau},\ e^{-r^2/(2\tau^2)},\ (1 + r)e^{-r/\tau} \right\}. \tag{21}$$

This kernel is symmetric and positive definite, and satisfies

$$K_\tau(i, j) \approx R_X(t_i, t_j) \tag{22}$$

for a GP with covariance function $R_X(s, t) = k_\tau(|s - t|)$.

Temporal Eigenmodes (K-L Basis). The temporal K-L modes are obtained from

$$K_\tau \phi_k = \lambda_k \phi_k, \qquad \lambda_1 \geq \cdots \geq \lambda_T \geq 0, \tag{23}$$

where $\phi_k \in \mathbb{R}^T$ are discrete approximations to the eigenfunctions of the continuous covariance operator.

K-L Coefficients via Projection. Given the centered history matrix $\tilde{H} = H - \bar{H} \in \mathbb{R}^{T \times d}$, we define

$$c_k^{\top} = \phi_k^{\top} \tilde{H} \in \mathbb{R}^d, \qquad C = \begin{bmatrix} c_1^{\top} \\ \vdots \\ c_K^{\top} \end{bmatrix} = \Phi_{:,1:K}^{\top} \tilde{H} \in \mathbb{R}^{K \times d}, \tag{24}$$

where $\Phi = [\phi_1, \dots, \phi_T] \in \mathbb{R}^{T \times T}$.

Scaled K-L Components (Classical). We form scaled K-L components

$$C_{\text{KL},k} = \sqrt{\lambda_k}\, c_k^{\top}, \qquad C_{\text{KL}} = \begin{bmatrix} C_{\text{KL},1}^{\top} \\ \vdots \\ C_{\text{KL},K}^{\top} \end{bmatrix} \in \mathbb{R}^{K \times d}. \tag{25}$$

The $\sqrt{\lambda_k}$ scaling matches the continuous K-L expansion

$$X(t) = \sum_{k=1}^{\infty} \sqrt{\lambda_k}\, Z_k e_k(t). \tag{26}$$

Note: Again, this is entirely classical signal processing. The components C_{KL} are then passed to the learnable MLP (Section A.5).

Discretization Scaling and Spectral Stability. When implementing the kernelized K-L decomposition over a finite history buffer of length T, we must account for the discretization of the continuous covariance operator. The scaling factor

$$C_{ij} = \frac{\tau}{T-1} K_\tau(t_i, t_j) \tag{27}$$

performs three essential functions:

1. Measure Correction (Riemann Sum). The discrete sum $\sum_j K(t_i, t_j) f(t_j)$ approximates the continuous integral $\int_0^\tau K(t, s) f(s)\, ds$ only when weighted by the discretization measure $\Delta t = \tau/(T-1)$. Thus the factor $\tau/(T-1)$ converts the sum into a properly scaled Riemann approximation.

2. Variance Preservation. Without scaling, the sum $\sum_j K(t_i, t_j)$ grows linearly with T for stationary kernels, causing eigenvalues to spuriously depend on buffer size. The normalization ensures

$$\frac{\tau}{T-1} \sum_{j=1}^{T} K(t_i, t_j) \approx \int_0^\tau K(t, s)\, ds, \tag{28}$$

which is independent of T and matches the continuous operator's variance.

3. Spectral Stability. The eigenvalues $\{\lambda_k\}$ of the discretized covariance converge to the continuous Karhunen-Loève spectrum only under proper measure correction:

$$\lim_{T\to\infty} \lambda_k^{(T)} = \lambda_k^{\text{continuous}}. \tag{29}$$

Implementation. In practice, we normalize time to $[0, 1]$ with $dt = 1/(T-1)$ and scale the kernel:

$$K(i, j) = \tau \cdot dt \cdot \exp\left(-\frac{|t_i - t_j|}{\tau \cdot dt}\right), \tag{30}$$

where τ represents the desired memory timescale.

Remarks. This operator-driven approach is stable, smooth, and maintains long-range temporal structure even when empirical covariances are noisy. It provides a principled inductive bias: modes reflect the geometry of the *assumed* GP prior, not only raw data statistics. The learnable τ parameter allows the system to adjust the effective memory scale dynamically.

A.3 When to Use Which Strategy

Both realizations are mathematically legitimate; they differ in assumptions and practical behavior.

Strategy	**Operator**	**Best When**
A. Empirical KL (PCA/SVD)	$\frac{1}{T}\tilde{H}^\top \tilde{H}$ or $\frac{1}{T}\tilde{H}\tilde{H}^\top$	Window size moderate; empirical statistics reliable
B. Kernelized Time KL (GP)	$K_\tau(i, j) = k_\tau(\lvert t_i - t_j \rvert)$	Long histories; noisy data; strong temporal prior desired

Both are realizations of the same theoretical K-L principles. The empirical K-L corresponds to a data-driven covariance, while the kernel K-L corresponds to a prior-driven covariance operator.

Both strategies are classical and fixed. The novel contribution comes next.

A.4 Implementation Snippets (Classical K-L Components)

For completeness, we include minimal PyTorch-style pseudocode for both realizations. Each line of code corresponds directly to the mathematical steps above.

Note on Implementation: Strategy A was used for all reported experiments. Strategy B is provided as an alternative implementation for settings where a smoothness prior is preferred. Both implementations include robustness features (float64 numerics, symmetry enforcement, jitter, SVD fallback) for stable deployment.

A. Empirical KL.

```
# Step 1: Center the history matrix
Hc = H - H.mean(dim=0, keepdim=True)      # H_tilde (T, d)

# Step 2: Compute time-axis empirical covariance
C  = (Hc @ Hc.T) / Hc.shape[0]            # C (T, T)

# Step 3: Eigendecomposition for temporal K-L modes
evals, evecs = torch.linalg.eigh(C)       # lambda_k, psi_k

# Step 4: Select top-K modes
idx = torch.argsort(evals, descending=True)[:K]
lams = torch.clamp(evals[idx], min=0.0)
phi = evecs[:, idx]

# Step 5: Project and scale by sqrt(eigenvalues)
coeffs = phi.T @ Hc                       # (K, d)
C_KL = torch.sqrt(lams + 1e-12)[:, None] * coeffs  # (K, d)
```

B. Kernelized Time-Axis KL.

```
# Step 1: Center the history matrix
Hc = H - H.mean(dim=0, keepdim=True)      # H_tilde (T, d)

# Step 2: Construct temporal GP kernel with tau/(T-1) scaling
dt = 1.0 / max(T - 1, 1)
t_norm = torch.linspace(0, 1, T)
K = tau * dt * _time_kernel(t_norm, tau * dt, kind=kernel)
K = K.to(torch.float64)
K = 0.5 * (K + K.T)  # enforce symmetry

# Add jitter for numerical stability
eps = 1e-8 if T <= 2048 else 1e-6
K = K + eps * torch.eye(T, dtype=K.dtype, device=K.device)

# Step 2: Eigendecomposition with SVD fallback
try:
    evals, evecs = torch.linalg.eigh(K)
except:
    U, S, _ = torch.linalg.svd(K, full_matrices=False)
    evals, evecs = S, U

# Step 3: Select top-K modes and normalize
idx = torch.argsort(evals, descending=True)[:K]
lams = torch.clamp(evals[idx], min=0.0)
phi = evecs[:, idx]
phi = phi / (phi.norm(dim=0, keepdim=True) + 1e-12)

# Step 4: Project centered history onto eigenmodes
coeffs = phi.T @ Hc.to(phi.dtype)  # (K, d)

# Step 5: Scale by sqrt(eigenvalues)
C_KL = torch.sqrt(lams + 1e-12)[:, None] * coeffs  # (K, d)
```

A.5 Learnable Memory Projection (Our Contribution)

Novel Component

The K-L decomposition (Sections A.1–A.4) is **fixed mathematical preprocessing**. Our contribution is the **trainable neural projection** that maps K-L components to memory tokens.

From Classical K-L to Learnable Memory. Given K-L components $C_{\text{KL}} \in \mathbb{R}^{K\times d}$ from either Strategy A or B, we apply a **learnable multi-layer perceptron** f_θ to produce memory tokens:

$$M = f_\theta(C_{\text{KL}}) \in \mathbb{R}^{M\times d}, \tag{31}$$

where θ denotes trainable parameters.

Network Architecture. We implement f_θ as a two-layer MLP with GELU activation:

$$\begin{aligned} \text{flatten:} \quad & \mathbf{c} = \text{vec}(C_{\text{KL}}) \in \mathbb{R}^{Kd}, & (32)\\ \text{expand:} \quad & \mathbf{h} = \text{GELU}(W_1\mathbf{c} + b_1) \in \mathbb{R}^{2Kd}, & (33)\\ \text{project:} \quad & \mathbf{m} = W_2\mathbf{h} + b_2 \in \mathbb{R}^{Md}, & (34)\\ \text{reshape:} \quad & M = \text{reshape}(\mathbf{m}, M, d) \in \mathbb{R}^{M\times d}. & (35)\end{aligned}$$

The parameters $\theta = \{W_1, b_1, W_2, b_2\}$ are trained end-to-end via backpropagation.

Gradient Flow During Training. During training, gradients flow from the prediction loss back through the attention mechanism to the memory tokens, and then through f_θ:

$$\mathcal{L}_{\text{task}} \xrightarrow{\nabla} \text{Attention} \xrightarrow{\nabla} M \xrightarrow{\nabla_\theta} f_\theta \xrightarrow{\text{stop}} C_{\text{KL}}. \tag{36}$$

Gradients do **not** flow through the K-L decomposition itself (it remains fixed), but they **do** train θ to:

- Amplify K-L modes that are predictive for the task,
- Suppress modes that are irrelevant or noisy,
- Create memory representations compatible with the attention mechanism.

Why This Matters.

- **K-L decomposition (classical):** Provides temporal structure (extracts repeating patterns, filters uncorrelated noise), but is *task-agnostic*.
- **Learnable MLP (our contribution):** Adapts the K-L components to the specific prediction objective.
- **Synergy:** Mathematical structure guides what to remember; learning determines how to use it.

Implementation (Complete Pipeline).

```python
# Classical K-L decomposition (Strategy A or B)
C_KL = kl_decompose(history, tau=64.0, n_components=16)  # (K, d)
# Output: K-L components -- FIXED (no gradients)

# ========================================
# OUR CONTRIBUTION: Learnable projection
# ========================================
class LearnableMemoryProjection(nn.Module):
    def __init__(self, K, d, M):
        super().__init__()
        self.M = M
        self.d = d
        self.mlp = nn.Sequential(
            nn.Linear(K * d, 2 * K * d),    # W1, b1 -- TRAINABLE
            nn.GELU(),
            nn.Dropout(0.1),
            nn.Linear(2 * K * d, M * d)     # W2, b2 -- TRAINABLE
        )
        self.norm = nn.LayerNorm(d)

    def forward(self, C_KL):
        """
        C_KL: (K, d) K-L components (detached, no gradients)
        Returns: (M, d) memory tokens (trainable projection)
        """
        c_flat = C_KL.reshape(-1)               # (K*d,)
        m_flat = self.mlp(c_flat)               # (M*d,) <- GRADIENTS FLOW HERE
        M = m_flat.reshape(self.M, self.d)      # (M, d)
        M = self.norm(M)                        # LayerNorm
        return M

# Instantiate learnable projection
memory_projection = LearnableMemoryProjection(K=16, d=64, M=4)

# Generate memory tokens (learnable!)
M = memory_projection(C_KL.detach())  # (M, d) -- TRAINABLE
# Gradients from task loss will train memory_projection.mlp parameters

# Inject into Transformer attention
context = torch.cat([M, local_hidden_states], dim=0)
output = transformer(context)
loss = task_loss(output, target)
loss.backward()  # <- Gradients update memory_projection.mlp!
```

Comparison to Alternatives.

Approach	Temporal Structure	Task Adaptation
Pure K-L (no MLP)	✓(mathematical)	× (none)
Pure learned compression	× (must learn from scratch)	✓(fully trained)
Ours: K-L + MLP	✓(mathematical)	✓(trained)

Our approach combines the best of both: structure from signal processing theory, adaptation from modern deep learning.

A.6 What is Novel vs. What is Classical

Classical Components (1946–1998):

- K-L expansion theory (Karhunen, 1946; Loève, 1946)
- Empirical K-L / PCA (Hotelling, 1933; Pearson, 1901)
- Kernel PCA (Schölkopf et al., 1998)
- Gaussian process kernels (Rasmussen & Williams, 2006)

Our Novel Contributions (2024):

- **Learnable memory projection:** MLP f_θ that maps K-L components to task-specific tokens
- **End-to-end training:** Gradients flow from task loss through attention to memory projection
- **Integration with Transformers:** Memory tokens injected into attention context
- **Empirical validation:** Showing structure-guided learning outperforms pure learning

The Key Insight. We do not claim to have invented K-L decomposition (it is 80 years old). We claim that **using K-L as structured preprocessing for a learnable memory system** is an effective architectural choice that combines:

1. Strong temporal inductive bias from classical signal processing
2. Task-specific adaptation from gradient-based learning

This is a general principle: *mathematical structure* + *neural learning* often outperforms either alone.

This completes the K-L appendix with explicit clarification of what is classical (K-L decomposition) versus what is novel (learnable projection f_θ).