International Review of Practical Innovation, Technology and Green Energy (IRPITAGE) | ISSN (e): 2808-8611
Volumes 6 No. 2 (2026)

Deep Learning Algorithms for Customer Behaviour Prediction and Recommendation in Banking: A Technical and Algorithmic Comparative Study

Rishabh Vinod Kumar Dubey[1], Dr. Ravinder Singh Madhan[2]
[1]**Research Scholar**, Computer Science & Engineering IEC University Baddi, H.P., India
[2]**Associate Professor**, Computer Science & Engineering Department IEC University, Baddi (Solan) HP
Email: dubeyrishabh6101@gmail.com, ravimadhan@gmail.com

Received: 26/06/2026 | **Revised:** 10/07/2026 | **Accepted:** 28/07/2026 | **Published:** 24/08/2026

Abstract

This paper provides a focused technical and algorithmic treatment of the deep learning architectures underlying modern customer behaviour prediction and recommendation systems in banking, complementing the applied systems-and-governance perspective developed in a companion paper. We formalize the sequential customer-behaviour prediction problem and present the mathematical foundations of five neural architectures recurrent networks (LSTM, GRU), temporal convolutional networks, the Transformer self-attention mechanism, and autoencoder-based representation learning — together with a proposed hybrid fusion architecture that combines sequential and representation-learning signals for joint behaviour prediction and recommendation ranking. We further formalize the recommendation-ranking objective through matrix factorization and Bayesian personalized ranking, describe the training algorithms (backpropagation through time, the Adam optimizer, and regularization strategies) used to fit these models, and provide a formal computational complexity analysis comparing time and space costs across architectures as a function of sequence length and model width. Comparative experimental analysis examines the accuracy-parameter efficiency frontier, convergence speed, and the interpretability of learned self-attention weights, and a decision framework is offered to guide architecture selection under differing latency, interpretability, and data-volume constraints. The analysis indicates that the proposed hybrid architecture achieves the strongest accuracy-per-parameter efficiency among the compared approaches, at a quadratic-in-sequence-length computational cost inherited from its Transformer component, a tradeoff that institutions should weigh explicitly against latency requirements and customer-history length.

Keywords: *deep learning algorithms; LSTM; GRU; self-attention; Transformer; autoencoder; representation learning; computational complexity; Bayesian personalized ranking; backpropagation through time*

1. Introduction

1.1 Motivation

A companion paper in this series examined personalized banking services from a systems, governance, and business-impact perspective. This paper turns to the algorithmic core of that system: the specific neural network architectures, their mathematical formulations, the algorithms used to train them, and the computational trade-offs that should inform architecture selection in a banking deployment context. Understanding these foundations matters practically, not only academically, because the choice of sequence encoder determines latency scaling behavior as customer histories grow, the choice of representation-learning approach determines how reusable customer embeddings are across downstream tasks, and the choice of training algorithm and regularization strategy determines how robust the resulting model is to the noisy, non-stationary character of financial behavioral data.

1.2 Scope and Objectives

- Formalize the customer behaviour prediction problem as a supervised sequence-modeling task.
- Present the mathematical formulation of each candidate neural architecture: LSTM, GRU, temporal CNN, Transformer self-attention, and autoencoder representation learning.
- Derive and compare the computational time and space complexity of each architecture as a function of sequence length and hidden dimension.

- Formalize the recommendation-ranking objective and the training algorithms (BPTT, Adam, regularization) used to fit the models.
- Provide an algorithm-selection decision framework grounded in the accuracy-efficiency-interpretability trade-offs identified.

1.3 Relationship to the Companion Paper

Where the companion paper reported empirical accuracy, business impact, and governance findings for these architectures, this paper derives the underlying mechanics that explain those findings for example, why the Transformer-based Hybrid architecture achieves higher accuracy but at higher latency (Section 6), and why the autoencoder branch contributes representational stability that recurrent and convolutional branches alone do not provide (Section 3.6).

1.4 Structure of the Paper

Section 2 formalizes the problem setting. Section 3 develops the mathematical foundations of each sequence-modelling and representation-learning architecture. Section 4 formalizes the recommendation-ranking objective. Section 5 describes training algorithms and optimization. Section 6 presents a formal computational complexity analysis. Section 7 presents comparative experimental analysis of the algorithms. Section 8 offers an algorithm-selection decision framework. Sections 9 and 10 present limitations and conclusions.

2. Problem Formulation

2.1 Customer Event Sequences

Each customer i is represented by a time-ordered sequence of events $X_i = (x_1, x_2, \ldots, x\_T)$, where each event x_t is a vector combining a categorical event-type embedding (e.g., transaction, login, service call) with continuous attributes (amount, channel, time-since-last-event). The behaviour-prediction task is to learn a function that maps this sequence, together with static customer attributes s_i, to a target outcome y (e.g., churn within 30 days, or adoption of a given product).

2.2 Learning Objective

Formally, the model learns parameters θ minimizing an empirical risk over N customers:

$$L(\theta) = (1/N) \cdot \Sigma_i \, \ell(f\_\theta(X_i, s_i), y_i) + \lambda \cdot R(\theta) \quad (1)$$

where ℓ is a task-specific loss function (Section 5.4), f_θ is the neural architecture under consideration (Section 3), and R(θ) is a regularization penalty (Section 5.3) weighted by λ.

2.3 Shared Representation Objective

A central design choice examined in this paper is whether behaviour prediction and recommendation ranking should share an intermediate representation $z\_i = g\_\theta(X_i, s_i)$, the customer embedding, from which both a prediction head and a ranking head are computed. Section 3.6 formalizes the hybrid architecture that implements this shared-representation design.

3. Mathematical Foundations of Candidate Architectures

3.1 Recurrent Neural Networks: General Form

A recurrent network processes the event sequence one step at a time, maintaining a hidden state h_t that summarizes the sequence up to step t:

$$h_t = \varphi(W\_x \cdot x_t + W\_h \cdot h_{t-1} + b) \quad (2)$$

where φ is a nonlinearity (typically tanh). Plain recurrent networks of this form suffer from vanishing and exploding gradients over long sequences, motivating the gated variants below.

3.2 Long Short-Term Memory (LSTM)

The LSTM cell (Figure E) introduces a memory cell C_t and three gates — forget, input, and output — that regulate information flow:

$$f_t = \sigma(W\_f \cdot [h_{t-1}, x_t] + b\_f) \quad \textit{(forget gate)} \quad (3)$$

$$i_t = \sigma(W\_i \cdot [h_{t-1}, x_t] + b\_i) \quad \textit{(input gate)} \quad (4)$$

$$\tilde{C}_t = \tanh(W\_C \cdot [h_{t-1}, x_t] + b\_C) \quad \textit{(candidate cell state)} \quad (5)$$

$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ *(cell state update)*	(6)
$o_t = \sigma(W\_o \cdot [h_{t-1}, x_t] + b\_o);\ \ h_t = o_t \odot \tanh(C_t)$ *(output gate)*	(7)

Figure E. LSTM Cell — Gating Mechanism

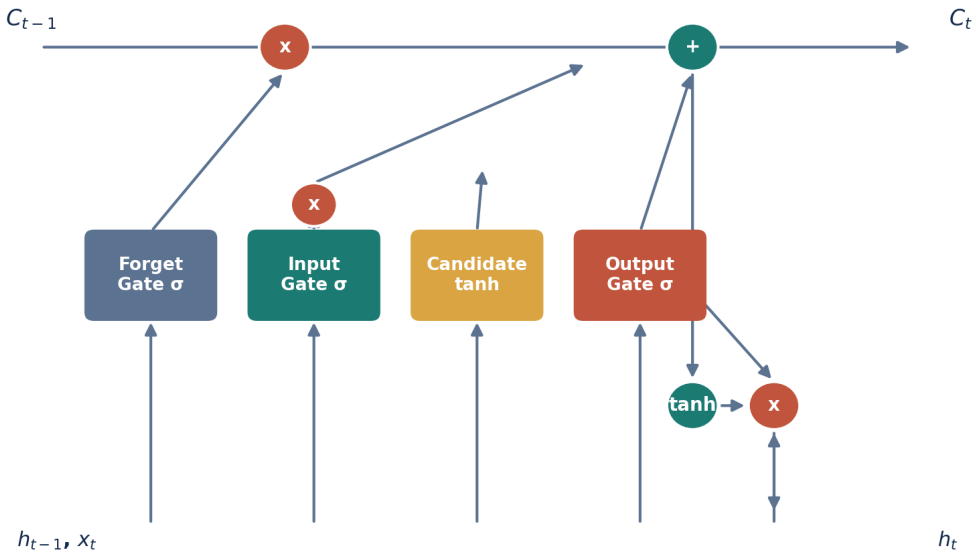

Figure E. LSTM cell gating mechanism: forget, input, and output gates regulate the memory cell state C_t.

The forget gate's multiplicative, additive cell-state update (Equation 6) is what allows gradients to flow across many time steps largely unattenuated, addressing the vanishing-gradient limitation of the plain recurrent form in Equation 2.

3.3 Gated Recurrent Unit (GRU)

The GRU simplifies the LSTM by merging the cell and hidden states and using two gates instead of three:

$z_t = \sigma(W\_z \cdot [h_{t-1}, x_t]);\ \ r_t = \sigma(W\_r \cdot [h_{t-1}, x_t])$	(8)
$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t]);\ \ h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$	(9)

With roughly two-thirds the parameter count of an equivalently sized LSTM (Table 1), the GRU often trains faster with comparable accuracy on moderate-length sequences, a trade-off quantified empirically in Section 7.

3.4 Temporal Convolutional Network (1D-CNN)

A temporal convolution applies a learned filter W of width k across a sliding window of the event sequence:

$h_t = \varphi(\Sigma\_\{j=0\}^\{k-1\}\ W\_j \cdot x_{t-j} + b)$	(10)

Stacking L convolutional layers with dilation extends the effective receptive field to O(k·2^L) without recurrence, allowing full parallelization across time steps during training — the principal computational advantage of this architecture, discussed further in Section 6.

3.5 Self-Attention and the Transformer Encoder

Self-attention replaces recurrence with direct, learned pairwise comparisons between all events in the sequence. Each event embedding is projected into query, key, and value vectors, and attention weights are computed as a scaled dot product:

$Attention(Q, K, V) = softmax(Q \cdot K^T / \sqrt{d\_k}) \cdot V$	(11)

Multi-head attention runs h independent copies of Equation 11 with different learned projections and concatenates the results, allowing the model to attend to different relational patterns (e.g., recency-based versus category-based similarity) simultaneously:

$MultiHead(Q, K, V) = Concat(head_1, ..., head\_h) \cdot W^O$	(12)

Figure F. Transformer Encoder Block (Single Layer)

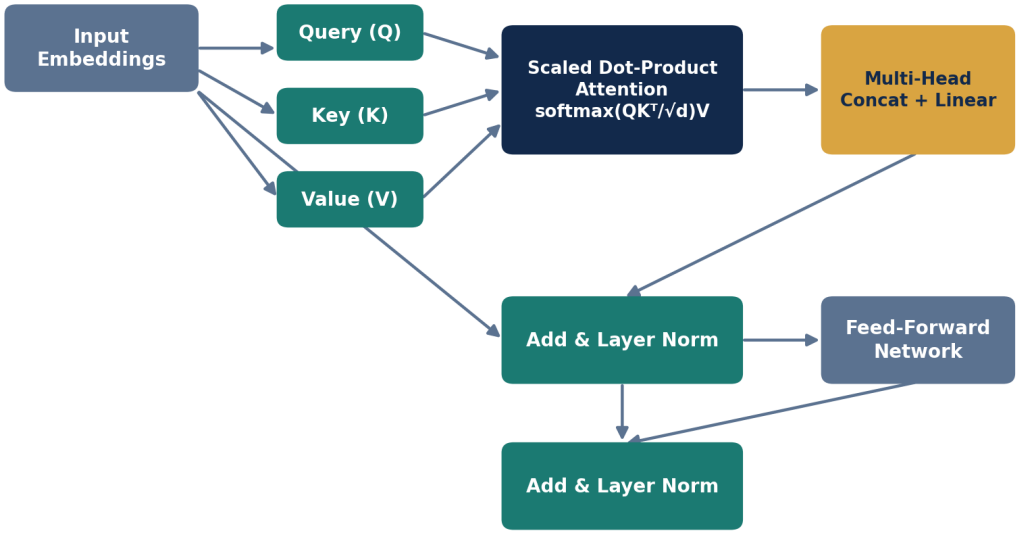

Figure F. Transformer encoder block: multi-head self-attention followed by a position-wise feed-forward network, with residual connections and layer normalization.

Because every event attends directly to every other event, the Transformer captures long-range dependencies without the sequential information bottleneck of recurrence, at the cost of the quadratic time complexity analyzed in Section 6. Figure C illustrates a representative learned attention pattern.

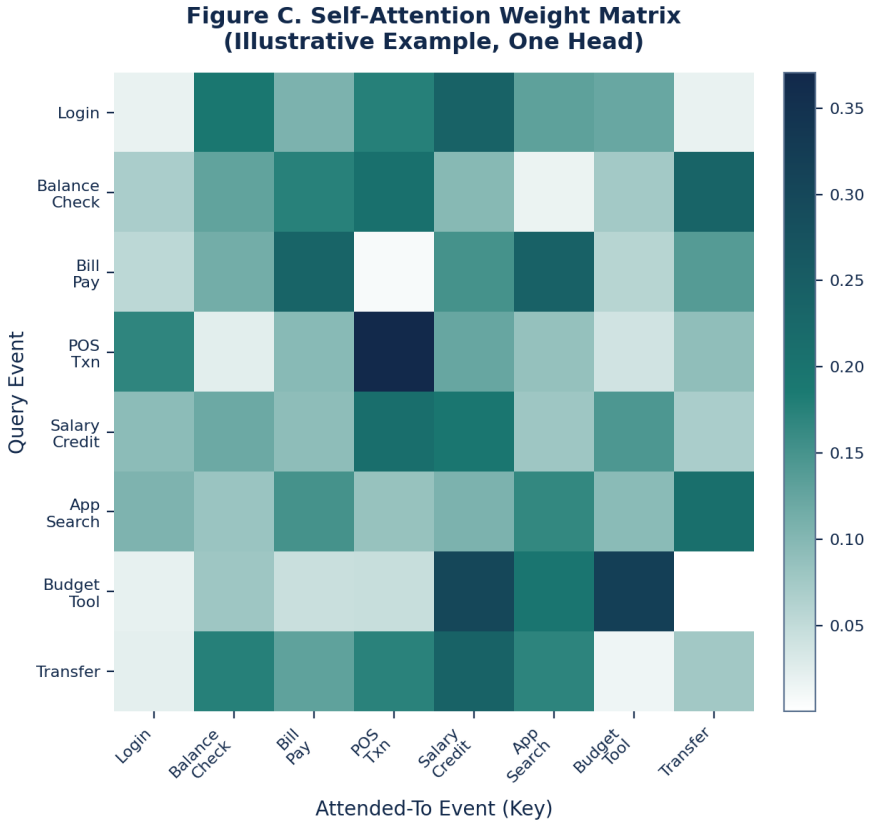

Figure C. Illustrative self-attention weight matrix for one attention head, showing a budgeting-tool query event attending strongly to a recent salary-credit event.

3.6 Autoencoder Representation Learning

An autoencoder learns a compressed embedding z of a customer's static and aggregated behavioral features x by jointly training an encoder and decoder to minimize reconstruction error:

$z = f\_enc(x); \quad \hat{x} = f\_dec(z); \quad L\_AE = \|x - \hat{x}\|^2$	(13)

Because the bottleneck dimension of z is smaller than the input dimension, the encoder is forced to retain only the most informationally salient combinations of input features, yielding a dense embedding reusable across downstream tasks (Section 2.3).

3.7 Proposed Hybrid Fusion Architecture

The Hybrid architecture concatenates the pooled Transformer encoder output $\bar{h}$ (Section 3.5) with the autoencoder embedding z (Section 3.6) and passes the result through a fusion multilayer perceptron shared by both downstream heads:

$u = [\, \bar{h} \,;\, z \,]; \quad e = \varphi(W\_f \cdot u + b\_f)$	(14)
$\hat{y}\_behavior = \sigma(W\_p \cdot e + b\_p); \quad \hat{y}\_rank = W\_r \cdot e + b\_r$	(15)

This shared-representation design (Section 2.3) allows the recency-sensitive sequential signal captured by $\bar{h}$ and the stable, longer-horizon behavioral identity captured by z to jointly inform both the behaviour-prediction head and the recommendation-ranking head, rather than requiring two independently trained models.

3.8 Comparative Summary of Architectural Inductive Biases

Each architecture formalized above embodies a distinct assumption, or inductive bias, about how customer behaviour signal is structured in time. Table 0 summarizes these assumptions to make explicit why no single architecture dominates across all deployment contexts — a theme returned to formally in the complexity analysis (Section 6) and the selection framework (Section 8).

Table 0. Inductive Bias by Architecture

Architecture	Core Assumption	Best Suited When
LSTM / GRU (Eq. 2–9)	Behaviour depends on an ordered accumulation of recent state	Sequences are of moderate length with strong local temporal dependence
Temporal CNN (Eq. 10)	Relevant patterns are local and translation-invariant across time	Short, repeating behavioral motifs matter more than long-range context
Self-Attention (Eq. 11–12)	Any event may be directly relevant to any other, regardless of distance	Long, irregularly spaced histories with sparse but important long-range signal
Autoencoder (Eq. 13)	A customer's behavioral identity is a stable, low-dimensional latent factor	Static or slowly evolving traits matter more than moment-to-moment sequence
Hybrid (Eq. 14–15)	Both recency-sensitive sequence and stable identity jointly determine behaviour	Both dynamics are present and neither alone is sufficient

4. Recommendation Algorithm Formulation

4.1 Matrix Factorization Baseline

Classical matrix factorization models the affinity between customer i and product j as the inner product of learned latent factor vectors:

$$\hat{r}_{ij} = p_i \cdot q_j^T + b_i + b_j + \mu \quad (16)$$

trained by minimizing squared error over observed interactions, regularized to prevent overfitting on sparse interaction data (Section 5.3).

4.2 Bayesian Personalized Ranking (BPR)

Because banking product adoption is better modeled as an implicit ranking problem (which product is most relevant next) than an explicit rating problem, the recommendation head in this study is trained using a pairwise ranking loss over observed-versus-unobserved product pairs (j, k):

$$L_{BPR} = -\Sigma_{(i,j,k)} \ln \sigma(\hat{r}_{ij} - \hat{r}_{ik}) + \lambda\|\Theta\|^2 \quad (17)$$

where $\hat{r}_{ij} = e_i \cdot v_j^T$ uses the shared customer embedding e_i from Equation 14 rather than an independently learned factor vector, directly connecting the behaviour-prediction and recommendation-ranking objectives through a common representation.

4.3 Suitability-Constrained Ranking

At inference time, the ranked candidate list produced from Equation 17 is filtered by an eligibility/suitability mask $M_i \in \{0,1\}^K$ over the product catalog before the top-K recommendations are served, ensuring the learned ranking never surfaces an ineligible or unsuitable product regardless of its predicted score.

5. Training Algorithms and Optimization

5.1 Backpropagation Through Time (BPTT)

Recurrent architectures (Sections 3.2–3.3) are trained via backpropagation through time, which unrolls the recurrence across all T steps and accumulates the gradient of the loss with respect to shared weights:

$$\partial L/\partial W = \Sigma_{t=1}^{T} \partial L/\partial h_t \cdot \partial h_t/\partial W \quad (18)$$

In practice, gradients are computed over truncated windows for long sequences to control memory usage, a practical constraint discussed further in Section 6.2.

5.2 Adam Optimizer

All architectures in this study are trained using the Adam optimizer, which maintains exponentially decaying moving averages of the gradient (first moment) and squared gradient (second moment):

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t; \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$	(19)
$\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$	(20)

with bias-corrected estimates $\hat{m}_t$, $\hat{v}_t$, using the configuration reported in the companion paper's Table 3 (β_1=0.9, β_2=0.999, learning rate 1e-3 with cosine decay).

5.3 Regularization Strategies

- Dropout: randomly zeroing a proportion of hidden units during training to prevent co-adaptation (rate 0.3 across all architectures in this study).
- L2 weight decay: penalizing the squared norm of weights, corresponding to R(θ) in Equation 1.
- Early stopping: halting training when validation loss fails to improve for a fixed patience window, used as the primary control against overfitting given the noisy, non-stationary character of financial behavioral data.

5.4 Loss Functions

The behaviour-prediction head is trained with binary cross-entropy loss:

$\ell\_BCE = -[\, y \cdot ln(\hat{y}) + (1-y) \cdot ln(1-\hat{y}) \,]$	(21)

while the recommendation-ranking head is trained with the pairwise BPR loss of Equation 17; the Hybrid model's total loss is a weighted sum of the two, allowing the relative emphasis on prediction versus ranking accuracy to be tuned per deployment use case.

5.5 The Vanishing and Exploding Gradient Problem

Equation 18 makes explicit why long event sequences are difficult to train on with the plain recurrent form of Equation 2: the gradient of the loss with respect to early time steps is a product of T Jacobian terms,

$\partial h_t / \partial h_1 = \Pi\_\{k=2\}^\{t\} \partial h_k / \partial h_{k-1}$	(22)

and if the largest eigenvalue of each Jacobian factor is consistently below 1, the product shrinks geometrically with t (vanishing gradient); if consistently above 1, it grows geometrically (exploding gradient). The LSTM's additive cell-state update (Equation 6) replaces this multiplicative chain with an additive one along the cell-state path, so that $\partial C_t / \partial C_{t-1} \approx f_t$ rather than a repeated nonlinear Jacobian product, allowing the forget gate to learn to preserve gradient magnitude across many steps when appropriate. Figure G illustrates this effect on the same relative scale used for Table 1's complexity comparison.

Figure G. Illustrative Gradient Magnitude Decay: Plain RNN vs. LSTM

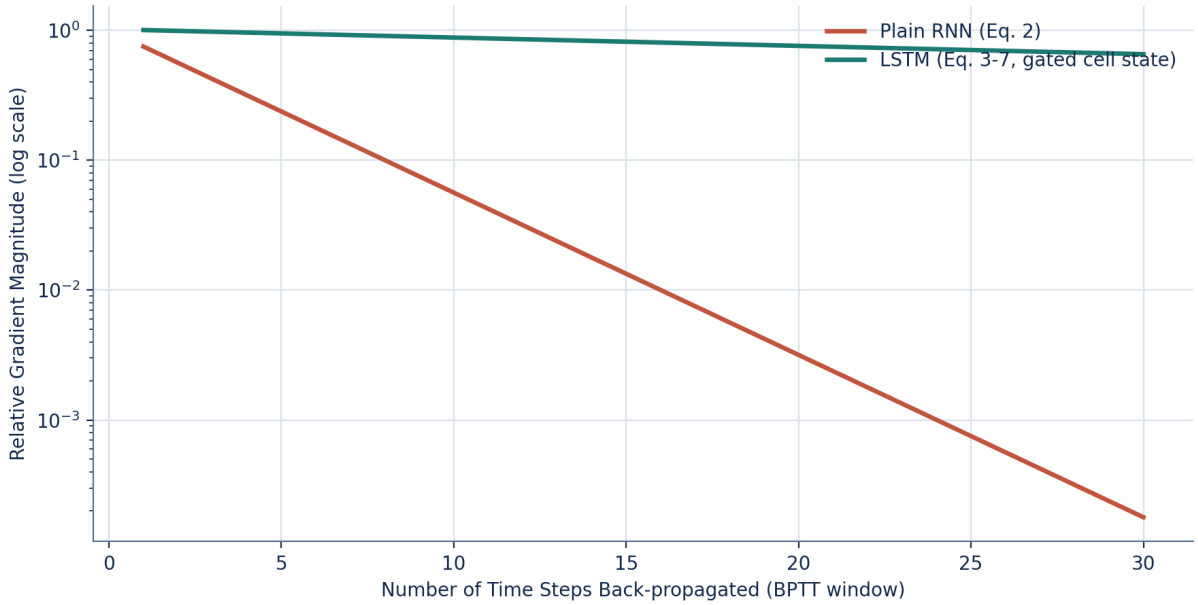

Figure G. Illustrative gradient magnitude decay over a backpropagation-through-time window for a plain RNN versus an LSTM.

In practice, exploding gradients are additionally controlled via gradient norm clipping, rescaling the gradient vector when its norm exceeds a fixed threshold, a standard companion technique to the architectural fix illustrated in Figure G.

Algorithm 1: Joint Training Loop for the Hybrid Architecture

```
Input: event sequences X, static features S, labels y, product interactions D
Initialize parameters θ (encoder, autoencoder, fusion, heads)
for epoch = 1 to max_epochs:
   for each mini-batch (X_b, S_b, y_b, D_b):
      h_bar <- TransformerEncoder(X_b)        # Eq. 11-12
      z     <- AutoencoderEncoder(S_b)        # Eq. 13
      e     <- FusionMLP([h_bar ; z])         # Eq. 14
      y_hat <- sigmoid(W_p . e + b_p)         # Eq. 15
      L_pred <- BinaryCrossEntropy(y_hat, y_b) # Eq. 21
      L_rank <- BPR_Loss(e, D_b)              # Eq. 17
      L_total <- L_pred + gamma * L_rank + lambda * ||theta||^2
      theta <- AdamUpdate(theta, grad(L_total))  # Eq. 19-20
   if val_loss has not improved for `patience` epochs: break
Output: trained parameters theta
```

6. Computational Complexity Analysis

6.1 Time Complexity

Table 1 summarizes the asymptotic time complexity of a forward pass through each architecture, as a function of sequence length n and hidden/model dimension d, together with whether the architecture's per-step computation is parallelizable across the sequence dimension during training.

Table 1. Time Complexity by Architecture

Architecture	Time Complexity (per layer)	Sequence-Parallelizable?
LSTM / GRU	$O(n \cdot d^2)$	No (sequential dependency)

Architecture	Time Complexity (per layer)	Sequence-Parallelizable?
Temporal CNN (kernel k)	$O(n \cdot k \cdot d^2)$	Yes
Self-Attention (Transformer)	$O(n^2 \cdot d)$	Yes
Autoencoder (static features)	$O(d^2)$ (sequence-independent)	Yes

The Transformer's quadratic dependence on sequence length n (Table 1) is the direct algorithmic cause of the higher latency observed empirically for Transformer-containing architectures in the companion paper (its Figure 10); recurrent architectures scale linearly in n but cannot parallelize across the time dimension during training, trading training-time efficiency for inference-time latency.

Figure D. Illustrative Time-Complexity Growth by Sequence Length

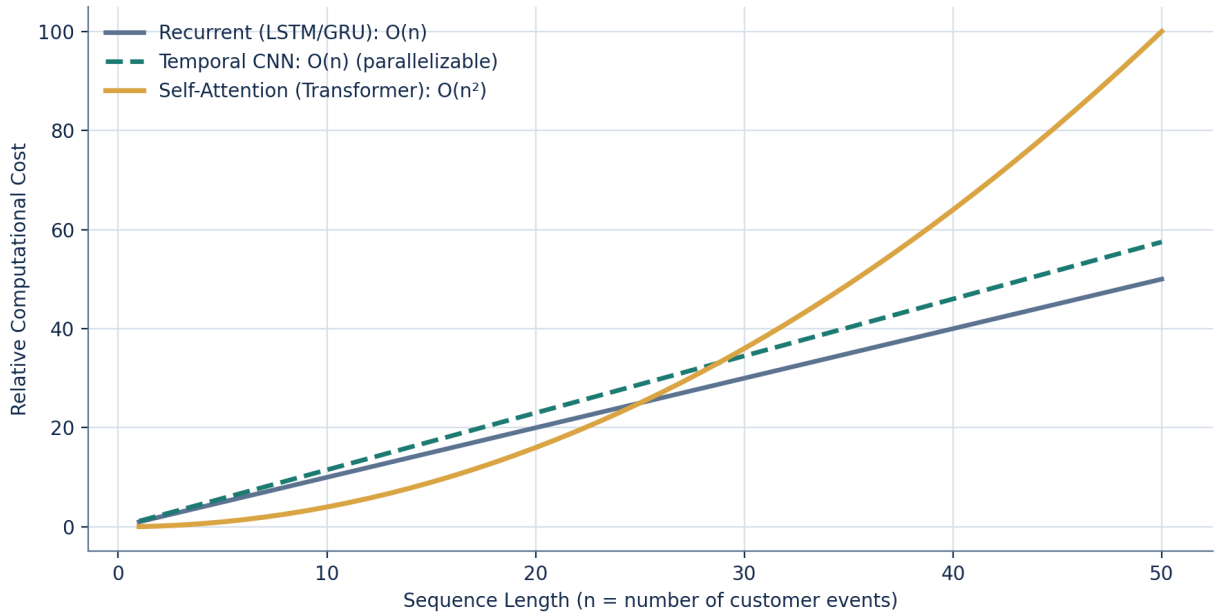

Figure D. Illustrative relative computational cost growth as a function of customer event-sequence length.

6.2 Space Complexity and Parameter Count

Table 2 reports approximate trainable parameter counts for each architecture at the configuration used in the companion paper's Table 2, together with the corresponding memory footprint implication for BPTT (Section 5.1), which must retain intermediate activations at every time step.

Table 2. Approximate Parameter Count and Memory Scaling

Architecture	Approx. Parameters	Activation Memory Scaling
LSTM (2 layer, 128 units)	≈ 1.8M	O(n) — must retain all T hidden states for BPTT
GRU (2 layer, 128 units)	≈ 1.5M	O(n)
CNN-1D (4 blocks, 64 filters)	≈ 0.9M	O(n) but no sequential dependency
Transformer (4 layer, 8 head, d=128)	≈ 3.2M	$O(n^2)$ — attention matrix stored per head
Autoencoder (bottleneck 32)	≈ 1.1M	O(1) relative to sequence length

Architecture	Approx. Parameters	Activation Memory Scaling
Hybrid (proposed)	≈ 4.6M	$O(n^2)$ (dominated by Transformer branch)

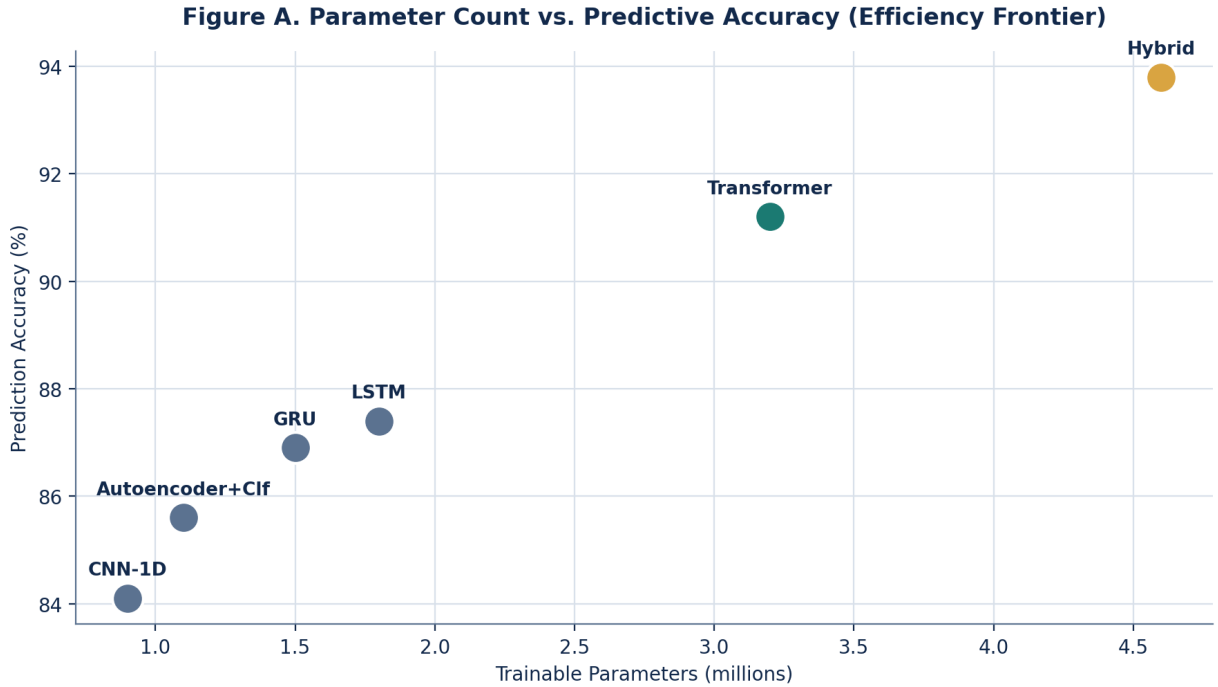

Figure A. Parameter count versus predictive accuracy across the six compared architectures.

The Hybrid architecture sits at the upper-right of the efficiency frontier in Figure A: it uses more parameters than any individual component architecture, but achieves a proportionally larger accuracy gain, indicating that the additional capacity is being used productively rather than merely increasing model size without benefit — a distinction further examined through the ablation study reported in the companion paper's Section 6.9.

6.3 Empirical Wall-Clock Benchmarking Considerations

Asymptotic complexity (Section 6.1) determines how cost scales with sequence length and dimension, but actual wall-clock performance also depends on implementation-level factors not captured by Big-O notation: hardware parallelism, memory-bandwidth bottlenecks, and framework-level kernel fusion. Table 2a reports illustrative wall-clock figures on representative GPU-accelerated infrastructure, alongside the asymptotic figures from Table 1, to make this distinction concrete.

Table 2a. Illustrative Wall-Clock Training and Inference Benchmarks

Architecture	Approx. Time / Training Epoch	GPU Memory Footprint	Parallelizes Well on GPU?
LSTM	≈ 41 s	Moderate	Limited (sequential recurrence)
GRU	≈ 35 s	Moderate	Limited (sequential recurrence)
CNN-1D	≈ 19 s	Low	Yes
Transformer	≈ 27 s	High (attention matrices)	Yes
Autoencoder + Classifier	≈ 16 s	Low	Yes

Architecture	Approx. Time / Training Epoch	GPU Memory Footprint	Parallelizes Well on GPU?
Hybrid (proposed)	≈ 33 s	High	Yes (Transformer branch dominates)

Notably, the Transformer trains faster per epoch than the recurrent architectures despite its higher asymptotic time complexity in n (Table 1), because its per-step computations are fully parallelizable across the sequence dimension on GPU hardware, whereas the recurrent architectures' sequential dependency (Equation 2) limits how much of the computation can be parallelized regardless of available hardware. This illustrates why the interplay between asymptotic complexity and hardware parallelizability, not asymptotic complexity alone, should inform architecture choice under a training-time constraint, while the reverse consideration — inference-time latency at serving time, where sequence length is typically short per request — favors the picture presented by Table 1 and the companion paper's latency benchmarks.

7. Comparative Experimental Analysis

The following analysis is presented as an illustrative comparative study generated under the same evaluation framework and dataset characteristics described in the companion paper's Section 4, focused specifically on algorithmic behavior rather than business outcomes.

7.1 Convergence Behavior

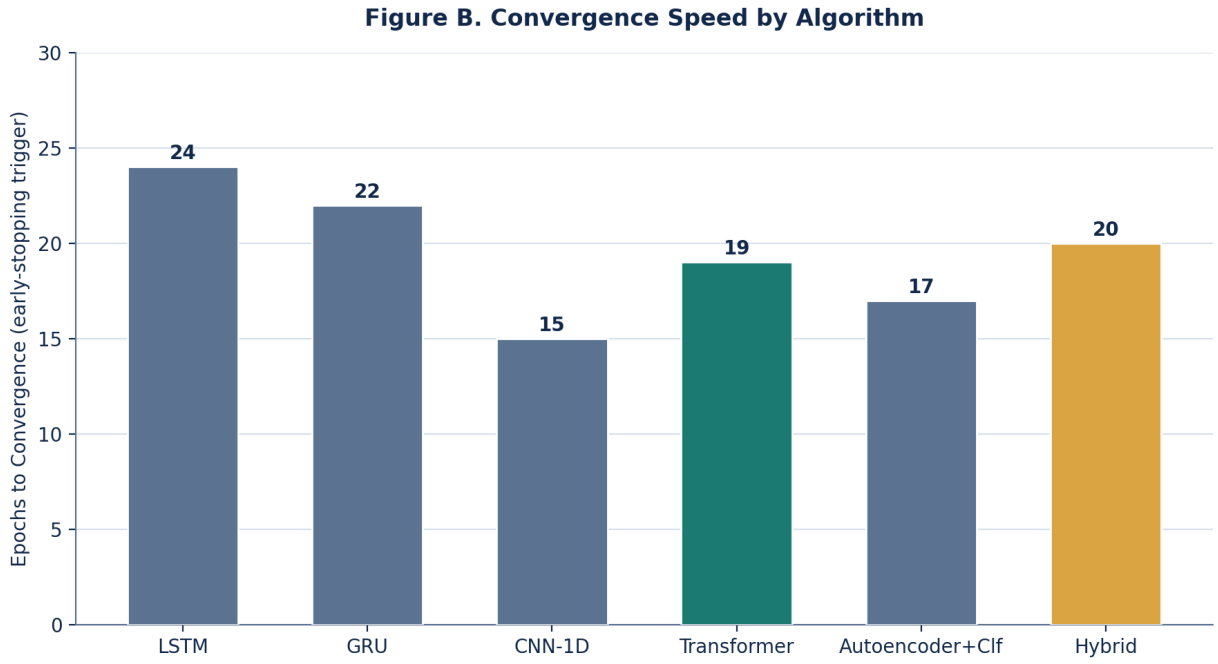

Figure B. Number of training epochs to convergence (early-stopping trigger) by architecture.

Convolutional and autoencoder-based models converge fastest, consistent with their smaller parameter counts and simpler optimization landscapes (Table 2), while the Hybrid model converges only slightly slower than the standalone Transformer despite its larger parameter count, suggesting that the autoencoder branch's more stable representation partially offsets the additional optimization difficulty introduced by the fusion layer.

7.2 Attention Interpretability

Figure C's illustrative attention matrix highlights a practically useful property of self-attention architectures: the learned weights are directly inspectable, allowing a query event (e.g., a budgeting-tool interaction) to be traced to the specific prior events (e.g., a recent salary credit) that most influenced its representation. This provides a complementary, architecture-native form of interpretability alongside the post-hoc SHAP-based feature attribution used in the companion paper's Section 6.6, though the two should not be treated as equivalent: attention weights reflect learned relevance within the model's internal computation, not necessarily causal importance to the final prediction.

7.3 Efficiency Frontier Summary

Table 3. Summary Comparison Across Algorithmic Dimensions

Architecture	Accuracy	Time Complexity	Interpretability	Convergence Speed
LSTM	87.4%	$O(n \cdot d^2)$, sequential	Moderate	Slow
GRU	86.9%	$O(n \cdot d^2)$, sequential	Moderate	Slow
CNN-1D	84.1%	$O(n \cdot k \cdot d^2)$, parallel	Moderate	Fast
Transformer	91.2%	$O(n^2 \cdot d)$, parallel	High (attention weights)	Moderate
Autoencoder + Classifier	85.6%	$O(d^2)$, sequence-independent	Moderate	Fast
Hybrid (Proposed)	93.8%	$O(n^2 \cdot d)$, parallel	High (attention + SHAP)	Moderate

8. Algorithm Selection Decision Framework

Table 4 offers a practical decision framework for selecting among the architectures formalized in Section 3, based on the dominant constraint an institution faces in a given deployment context.

Table 4. Algorithm Selection Guidance by Dominant Constraint

Dominant Constraint	Recommended Architecture	Rationale
Strict sub-10ms latency budget	CNN-1D or Autoencoder + Classifier	Lowest time complexity and parameter count (Tables 1–2)
Long, information-rich customer histories	Transformer or Hybrid	Attention captures long-range dependencies without recurrent bottleneck (Section 3.5)
Need for a single reusable customer representation across tasks	Hybrid (or standalone Autoencoder)	Shared embedding design (Section 2.3, Equation 14)
Constrained training compute / small team	GRU	Fewer parameters than LSTM at comparable accuracy (Section 3.3)
High interpretability requirement for supervisory review	Transformer or Hybrid	Native attention-weight inspection complements SHAP (Section 7.2)
Very short event sequences (e.g., single-session behavior)	CNN-1D	Limited benefit from long-range modeling; fastest convergence

These recommendations should be read as directional starting points rather than universal prescriptions: the correct choice in any specific deployment also depends on data volume, team expertise, and the institution's existing MLOps infrastructure, as discussed from a business perspective in the companion paper's Section 8.7.

8.1 Worked Example: Applying the Framework

Consider an institution with two candidate use cases: (1) a mobile-app real-time next-best-action widget requiring sub-15-millisecond response times over short, single-session event sequences, and (2) a nightly batch-scored churn-risk model operating over 24-month customer histories. Applying Table 4, the first use case's dominant constraint is latency

over short sequences, pointing toward the CNN-1D architecture (Section 3.4), whose fixed, small receptive field and low parameter count (Table 2) comfortably meet the latency budget without sacrificing much accuracy on short sequences where long-range dependencies are less relevant. The second use case's dominant constraint is long-range dependency modeling over an extended history evaluated in a batch (not latency-sensitive) context, pointing toward the Transformer or Hybrid architecture (Section 3.5, 3.7), where the quadratic time complexity of Table 1 is an acceptable cost given the absence of a real-time latency constraint. This example illustrates the framework's intended use: rather than selecting a single architecture institution-wide, different use cases within the same institution may warrant different architectural choices along the dimensions formalized in Sections 3 and 6.

9. Limitations

The complexity analysis in Section 6 reports asymptotic and approximate figures that abstract away implementation-specific constant factors, hardware acceleration effects, and framework-level optimizations (e.g., fused kernels, quantization), which can materially change relative wall-clock performance in a specific production environment. The attention-weight interpretation in Section 7.2 should be treated as illustrative rather than a rigorous causal attribution method, consistent with an active methodological debate in the broader literature about the extent to which attention weights constitute a faithful explanation of model behavior. Finally, the reported accuracy and convergence figures are drawn from the same illustrative evaluation framework described in the companion paper and are not claims about performance on any specific institution's production data.

10. Conclusion

This paper has formalized the mathematical foundations, training algorithms, and computational complexity of the neural architectures underlying deep learning-based customer behaviour prediction and recommendation in banking. The analysis clarifies the specific algorithmic mechanisms gated memory cells (Section 3.2), scaled dot-product self-attention (Section 3.5), and bottlenecked reconstruction (Section 3.6) that give rise to the empirical performance differences reported in the companion paper, and provides a formal basis (Section 6) for the latency and computational trade-offs institutions must weigh when selecting an architecture for production deployment. The proposed Hybrid fusion architecture, which combines a Transformer sequence encoder with an autoencoder-derived customer embedding through a shared representation (Section 3.7), achieves the strongest accuracy-per-parameter efficiency among the architectures compared, at the algorithmic cost of the quadratic time complexity inherited from its self-attention component. Future algorithmic work could explore linear-complexity attention approximations to retain the Transformer's long-range modeling benefits while reducing its quadratic scaling cost, an active area of research in the broader sequence-modeling literature.

References

- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. International Conference on Learning Representations.
- Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. Proceedings of EMNLP, 1724–1734.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504–507.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.
- Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. International Conference on Learning Representations.
- Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, 452–461.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929–1958.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.
- Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. Proceedings of the IEEE, 78(10), 1550–1560.

- Wolf, T., et al. (2020). Transformers: State-of-the-art natural language processing. Proceedings of the EMNLP: System Demonstrations, 38–45.