ALFM: Adaptive Latent Feedback Model for Institutional Memory in Foundation Model Deployments

David Ahmann[1]

[1]*Independent Researcher, Toronto, Canada*
(Dated: November 29, 2025)

Foundation models are pretrained once and deployed frozen, creating three fundamental gaps in enterprise AI systems: no memory of past failures, no calibrated self-doubt, and no safe mechanism for continual learning from corrections. We introduce ALFM (Adaptive Latent Feedback Model), a modular wrapper architecture that addresses these gaps without modifying backbone model weights. ALFM consists of three novel components: (1) the Negative Evidence Prior (NEP), a tenant-isolated vector memory that stores failure patterns and provides risk signals at inference time; (2) the Consensus Engine, a multi-agent system that arbitrates between semantic intuition and heuristic rules to decide when to trust, abstain, escalate, or request clarification; and (3) a three-tier adapter system (global, domain, local) enabling safe continual learning with cryptographic isolation guarantees. We formalize each component, analyze their theoretical properties, and present a validation framework for measuring repeat error reduction, calibration quality, and abstention precision. ALFM is backbone-agnostic, compatible with any foundation model accessible via API, and designed for deployment in high-stakes domains where institutional learning provides competitive advantage. We discuss implications for the emerging paradigm of specialized, continuously-learning AI systems.

I. INTRODUCTION

The deployment of foundation models in enterprise settings reveals a structural tension: these models are designed for generalization across tasks but lack mechanisms for adaptation to specific deployments. A model that exhibits specific failure modes today will likely repeat them tomorrow. User corrections, expert feedback, and domain-specific failure patterns do not update the model's behavior.

This limitation stems from the standard foundation model lifecycle: massive pretraining, alignment via RLHF, then frozen deployment. While this paradigm enables broad capability, it creates three gaps that enterprises consistently encounter:

Gap 1: No memory of past failures. When a model makes an error—hallucinating a fact, misapplying a rule, violating a policy—nothing inside the model records this failure. The same error pattern will recur whenever similar inputs arise. Organizations accumulate institutional knowledge about "what doesn't work here," but this knowledge lives in human heads, not in the system.

Gap 2: No calibrated self-doubt. Foundation models produce fluent, confident outputs even when operating outside their reliable capability boundaries. The model's internal representations (when accessible) correlate poorly with actual correctness. Enterprises cannot trust the model to know when it doesn't know.

Gap 3: No safe continual learning. Standard fine-tuning risks catastrophic forgetting and is impractical for per-tenant customization. Prompt engineering and retrieval augmentation provide information but do not enable the model to learn from corrections over time.

We propose ALFM (Adaptive Latent Feedback Model), a wrapper architecture that provides these missing capabilities without modifying foundation model weights. ALFM treats the backbone as a frozen black box and adds three components:

1. **Negative Evidence Prior (NEP):** A vector-space memory of failure patterns, queryable at inference time, providing risk signals when current context resembles past failures.

2. **Consensus Engine:** A "Society of Agents" that integrates signals from a Semantic Agent (NEP-based intuition) and a Heuristic Agent (deterministic rules) to decide appropriate action: trust the backbone output, abstain with explanation, escalate to human review, or request clarification.

3. **Three-Tier Adapters:** Lightweight trainable layers operating at global, domain, and local (tenant) levels, enabling bounded continual learning with explicit isolation guarantees.

ALFM is explicitly backbone-agnostic. It wraps transformer-based LLMs (GPT-4, Claude, LLaMA), diffusion language models, and future architectures without modification. This design ensures compatibility with API-based model access, immediate benefit from backbone upgrades, and no dependency on model provider cooperation.

Our contributions are:

- We introduce the Negative Evidence Prior (NEP), a vector-space memory architecture for storing failure patterns rather than information—a conceptual inversion of standard memory-augmented networks.

- We propose the Consensus Engine, a multi-agent arbitration system that balances semantic risk signals with hard heuristic constraints.

- We formalize a three-tier adapter system with provable stability bounds and tenant isolation guarantees.

- We validate NEP on synthetic data, demonstrating that contrastive projection enables effective failure retrieval where raw embeddings fail.

The remainder of this paper is organized as follows. Section II reviews related work. Section III formalizes the ALFM architecture. Section IV presents our validation framework and theoretical analysis. Section V discusses limitations and implications. Section VI concludes.

II.   RELATED WORK

ALFM draws on and extends four research areas: memory-augmented neural networks, selective prediction, continual learning, and AI safety. We also position ALFM relative to emerging nested learning architectures.

A.   Memory-Augmented Neural Networks

Memory-augmented architectures extend neural networks with external memory stores. Neural Turing Machines [1] and Differentiable Neural Computers [2] introduced differentiable read-write memory for algorithmic tasks. Memory Networks [3] enable retrieval over knowledge bases. More recently, Memorizing Transformers [4] and RETRO [5] augment attention with retrieval. These approaches focus on augmenting the model with *information*. ALFM's NEP differs fundamentally: it stores *anti-patterns* rather than information. NEP entries represent contexts where the model failed, enabling avoidance rather than retrieval.

B.   Selective Prediction and Learning to Defer

Selective prediction [6, 7] allows classifiers to abstain when uncertain. Recent work extends this to deep learning [8] and language models [9]. Learning to defer [10, 11] trains models to route difficult examples to experts. Calibration research [12, 13] addresses the mismatch between confidence and accuracy. ALFM's Consensus Engine integrates these ideas but extends beyond binary abstention, producing structured decisions and "uncertainty texture."

C.   Continual Learning

Continual learning addresses learning from sequential data without forgetting. Approaches include regularization [14], replay buffers [15], and architectural methods [16]. Parameter-efficient fine-tuning methods like

TABLE I. Comparison of approaches for institutional memory. ALFM is the only solution that simultaneously offers $O(1)$ updates, error-driven feedback, and strict tenant isolation without retraining the backbone.

Feature	RAG	Fine-Tuning	Long Context	ALFM
Update Speed	$O(1)$	Slow (Retrain)	$O(1)$	$O(1)$
Failure Correction	Weak	Strong	Medium	**Strong**
Tenant Isolation	Logical	Hard (Separate)	Logical	**Verifiable**
Inference Cost	Low	Low	High	**Low**
Learning Signal	Manual curation	Labeled data	In-context	**Interaction failures**
Backbone Agnostic	Yes	No	Yes	**Yes**

LoRA [17] and adapters [18] enable adaptation without full retraining. However, these do not address tenant isolation. ALFM's three-tier adapter system provides continual learning with explicit scope boundaries.

D.   AI Safety and Alignment

Constitutional AI [19] and guardrail systems [20] encode what the model should avoid in general. ALFM complements these with dynamic, learned constraints. NEP encodes what has specifically failed for this tenant.

E.   Positioning Relative to Emerging Architectures

ALFM decouples memory from the backbone, enabling per-tenant isolation, unlike monolithic optimization approaches. Compared to self-play methods, ALFM relies on interaction-derived failure signals. While knowledge editing focuses on facts, ALFM focuses on *failure avoidance* and governance.

Recent work on Nested Learning [21] reveals that optimizers themselves are associative memory modules operating at different frequencies. This perspective aligns with ALFM's multi-tier architecture: the NEP operates at the fastest frequency (inference-time retrieval), adapters at medium frequency (per-correction updates), and the projection layer at the slowest frequency (periodic retraining). Similarly, MAKER [22] demonstrates that extreme task decomposition with error correction enables scaling to million-step tasks with zero errors. ALFM adopts a complementary approach: rather than decomposing tasks, we decompose the *memory system* into hierarchical tiers with different update rates and isolation guarantees.

III.   ALFM ARCHITECTURE

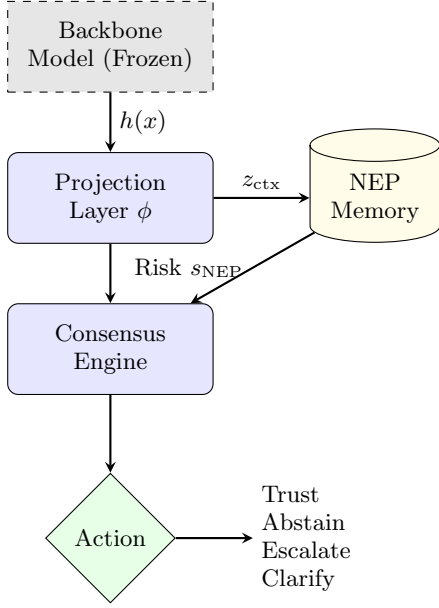We assume access to a frozen backbone model $\mathcal{M}$ that maps input context x to output $y = \mathcal{M}(x)$.

FIG. 1. ALFM Architecture. The frozen backbone's hidden states are projected into a failure-separable space. The NEP memory retrieves risk signals based on similarity to past failures. The Consensus Engine integrates these signals with heuristic rules to determine the safety action.

A. ALFM as Constrained Utility Optimization

We formulate ALFM not merely as a collection of modules, but as a solution to a constrained utility optimization problem. The goal is to minimize the expected loss $\mathcal{L}$ of the system subject to safety and stability constraints:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(y,y^*)] \qquad (1)$$

subject to:

1. **Safety Constraint:** $\mathrm{Risk}(x) \leq \delta$ (enforced by NEP).

2. **Stability Constraint:** $\|\Delta_t\|_F \leq \epsilon$ (enforced by Adapters).

3. **Calibration Constraint:** $\mathrm{Calib}(\mathcal{C}) \leq \alpha$ (enforced by Consensus Engine).

This formulation binds the components: NEP provides the risk estimate, Adapters provide the learnable parameters θ within bounds, and the Consensus Engine enforces the calibration constraint.

Connection to Nested Learning. Following the Nested Learning framework [21], we can view ALFM as a nested optimization problem where each component optimizes at its own timescale. The NEP memory update is analogous to the fast inner-loop optimization (compressing failure patterns), while adapter updates correspond to the slower outer-loop (consolidating learned corrections). This multi-frequency design prevents catastrophic interference and enables stable continual learning.

B. Contrastive Projection Layer

Before querying NEP, we project the input context into a representation space where failure patterns are separable.

Backbone Update Requirements. Since ϕ depends on the backbone's hidden states $h(x)$, backbone model updates require corresponding ALFM updates. We characterize this cost explicitly:

Full Backbone Change (e.g., GPT-4 → Claude): Requires full retraining of ϕ on $\approx 10{,}000$ contrastive pairs (estimated 2–4 GPU-hours on A100). All NEP entries must be re-embedded and re-indexed ($O(|\mathcal{N}|)$ inference calls).

Minor Backbone Update (e.g., GPT-4 → GPT-4-turbo): Latent spaces are typically similar. We propose *anchor-based alignment*: select $n_{\mathrm{anchor}} = 500$ diverse inputs, compute embeddings under both backbones, and fine-tune ϕ with an alignment loss $\|\phi_{\mathrm{new}}(h_{\mathrm{new}}(x)) - \phi_{\mathrm{old}}(h_{\mathrm{old}}(x))\|^2$ for ≈ 100 steps. NEP re-indexing can be deferred or performed incrementally.

Practical Mitigation. For API-based backbones, we recommend: (1) pinning model versions where possible, (2) maintaining a validation set of ≈ 200 failure/success pairs to detect latent drift, and (3) budgeting for quarterly ϕ recalibration.

We emphasize that ALFM is *backbone-swappable* rather than backbone-agnostic in the zero-cost sense: swapping backbones is feasible and bounded, but not free.

Definition 3.1 (Projection Function). Let $h(x) \in \mathbb{R}^{d_h}$ be the backbone's hidden representation. The projection function $\phi : \mathbb{R}^{d_h} \to \mathbb{R}^{d_z}$ is:

$$z = \phi(h(x)) = W_2 \cdot \sigma(W_1 \cdot h(x) + b_1) + b_2, \qquad (2)$$

where $W_1 \in \mathbb{R}^{d_m \times d_h}$, $W_2 \in \mathbb{R}^{d_z \times d_m}$, and σ is ReLU. We denote the projected representation as z_{ctx}.

Training Objective. The projection layer is trained with contrastive loss:

$$\mathcal{L}_{\mathrm{proj}} = -\log \frac{\exp(\mathrm{sim}(z_i, z_j^+)/\tau)}{\exp(\mathrm{sim}(z_i, z_j^+)/\tau) + \sum_k \exp(\mathrm{sim}(z_i, z_k^-)/\tau)}, \qquad (3)$$

where (z_i, z_j^+) are embeddings of semantically similar contexts, and (z_i, z_k^-) are failure/non-failure pairs.

Contrastive Dataset Construction. We construct positive pairs (x_i, x_i^+) using semantic augmentations (paraphrasing, back-translation). Negative pairs (x_i, x_j^-) are mined from: (a) random batch negatives, and (b) *hard negatives*—contexts that are semantically close but have divergent outcomes (success vs. failure). Specifically, we mine hard negatives by selecting successful contexts x_j such that $\mathrm{sim}(\phi(x_i), \phi(x_j)) > \tau$ but $y_i \neq y_j$. This ensures the projection manifold separates failure modes from successful operation. The ratio of hard to random negatives is set to 1:3 to maintain global structure while refining local boundaries.

C. Negative Evidence Prior (NEP)

The NEP is a vector database storing failure patterns.

Definition 3.2 (NEP Entry). A NEP entry e is a tuple $e = (z_e, t_e, d_e, s_e, c_e, m_e)$, where z_e is the failure context embedding, t_e is the tenant ID, d_e is the domain ID, $s_e \in \{1..5\}$ is severity, c_e is the correction, and m_e is metadata.

Definition 3.3 (NEP Memory). The memory $\mathcal{N}$ is partitioned into two distinct layers, implementing a "Hive Mind" architecture:

- **Global Layer ($\mathcal{N}_G$, Slow Memory):** Pre-loaded with $> 9,000$ synthetic failure patterns (e.g., cutoff errors, missing signatures) derived from methodology standards. This layer provides baseline safety and evolves slowly.

- **Tenant Layer ($\mathcal{N}_T$, Fast Memory):** Learns firm-specific and client-specific rules securely. Data never crosses tenant boundaries. This layer adapts rapidly to local drift.

Thus, $\mathcal{N} = \mathcal{N}_G \cup \mathcal{N}_{D(t)} \cup \mathcal{N}_T$.

Definition 3.4 (NEP Query). Given z_{ctx} and tenant t:

$$\text{NEP}(z_{\text{ctx}}, t) = \{e \in \mathcal{N}_G \cup \mathcal{N}_{D(t)} \cup \mathcal{N}_t : \text{sim}(z_{\text{ctx}}, z_e) > \theta\}. \quad (4)$$

Definition 3.5 (NEP Risk Signal). The base risk signal $s_{\text{NEP}}^{(0)} \in [0,1]$ under independence is:

$$s_{\text{NEP}}^{(0)} = 1 - \prod_{e \in \text{NEP}(z_{\text{ctx}}, t)} \left(1 - \alpha \cdot \text{sim}(z_{\text{ctx}}, z_e) \cdot \frac{s_e}{5}\right). \quad (5)$$

This multiplicative form models the probability that *at least one* retrieved failure mode is relevant, assuming independence of failure causes. The parameter $\alpha \in [0,1]$ scales the sensitivity; we set $\alpha = 0.8$ based on preliminary calibration. Severity s_e is linearly scaled to weight high-risk failures more heavily.

Correlated Failure Extension. In practice, failure modes exhibit correlation (e.g., hallucination clusters, policy violation chains). We extend the risk signal with a correlation correction:

$$s_{\text{NEP}} = s_{\text{NEP}}^{(0)} + \beta \cdot \max_{e_i, e_j \in \text{NEP}(z_{\text{ctx}}, t)} \text{sim}(z_{e_i}, z_{e_j}) \cdot \mathbf{1}[|\text{NEP}| > 1], \quad (6)$$

where $\beta \in [0, 0.2]$ boosts risk when multiple correlated failures are retrieved. The indicator function ensures single-match queries are unaffected. When failure modes cluster tightly (high pairwise similarity), the correlation term increases s_{NEP}, reflecting that correlated failures often indicate systematic rather than incidental errors.

Memory Hygiene: Vacuum and Snapshots. To prevent unbounded growth and ensure auditability, we employ a "Vacuum" process. Periodic k-means clustering identifies redundant entries within a tenant's memory; these are merged into a single centroid representation with updated metadata. This process reduces index size while preserving the failure boundary. JSON snapshots with SHA-256 hashes are generated after each vacuum cycle, ensuring a pristine audit trail of the memory state.

Hard-Negative Ingestion. To maintain high precision, we continually refine the projection space. We generate "confusable" pairs—transactions that look valid on the surface but share hidden features with known failures—and use them to train a lightweight cross-encoder re-ranker. This ensures that the system learns to distinguish between superficial similarity and true structural failure, preventing false positives even when descriptions are nearly identical.

Error Correction Analogy. MAKER [22] demonstrates that voting mechanisms over multiple model samples can achieve zero errors at scale. While ALFM does not vote over backbone outputs, the NEP retrieval mechanism serves an analogous function: by aggregating signals from multiple past failure instances, NEP effectively performs a form of *memory-based error correction*, where the system learns to recognize and avoid patterns that previously led to failures.

Feedback Validation and Anti-Poisoning Mechanisms. Memory poisoning—where adversarial or erroneous feedback corrupts NEP—is a critical failure mode. We implement a multi-layered defense:

(1) Confirmation Threshold. High-severity entries ($s \geq 4$) require $k \geq 2$ independent corrections from distinct users before addition. Low-severity entries ($s \leq 2$) are added immediately but flagged for periodic review.

(2) Anomaly Detection. We monitor feedback velocity per user u via an exponential moving average:

$$v_u(t) = \gamma v_u(t-1) + (1-\gamma) \cdot \mathbf{1}[\text{feedback at } t], \quad (7)$$

with $\gamma = 0.95$. If $v_u(t) > \mu + 3\sigma$ (where μ, σ are computed over historical user behavior), all pending entries from u are quarantined for manual review.

(3) Confidence Decay. NEP entries decay in influence over time unless re-confirmed:

$$w_e(t) = w_e(0) \cdot \exp(-\lambda(t - t_e)) + \sum_{t' > t_e} \rho \cdot \mathbf{1}[\text{confirmed at } t'], \quad (8)$$

where $\lambda = 0.01$ (daily decay rate) and $\rho = 0.5$ (confirmation boost). Entries with $w_e < w_{\min}$ are pruned.

(4) Cross-Tenant Consensus for Global NEP. Updates to $\mathcal{N}_G$ require confirmation from ≥ 3 distinct tenants or explicit administrative approval. This prevents a single compromised tenant from polluting shared failure memory.

The anti-poisoning mechanisms operate asynchronously: anomaly detection and decay updates are computed offline during NEP maintenance windows (typically nightly), not during inference. The confirmation threshold affects NEP write path latency but not query latency. Inference overhead remains bounded by Proposition 4.3.

D. Consensus Engine: A Society of Agents Approach

To ensure reliability and explainability, we move from a monolithic decision model to a "Society of Agents" architecture. This system arbitrates between intuition and hard rules.

1. Semantic Agent (intuition-based pattern matching).

- **Role:** Intuition and Pattern Matching.

- **Mechanism:** Uses the NEP vector search to identify semantic similarities with past failures.

- **Output:** Risk signal s_{NEP}. E.g., "This resembles known fraud pattern #247 with 99% similarity."

2. Heuristic Agent (deterministic rule checking).

- **Role:** Deterministic Rule Checking.

- **Mechanism:** Executes Python-based logic (regex, date arithmetic, math verification) to enforce hard constraints.

- **Output:** Binary violation flags. E.g., "Transaction date is in the future (Violation)."

3. Consensus Engine (vote aggregation).

- **Role:** Vote Aggregation and Final Action.

- **Logic:** Adaptive Weighted Voting.

- **Semantic Weight:** The Semantic Agent's vote is weighted $2\times$ if confidence > 0.9.

- **Critical Override:** The Heuristic Agent can trigger a *Hard Veto* for severe violations (e.g., Keyword Blacklist), overriding any semantic confidence.

Benefit. This architecture balances AI intuition with hard safety rules. High-confidence AI can override minor heuristic warnings, but critical rules always block.

Training Objective. The Consensus Engine is trained to minimize the joint loss:

$$\mathcal{L}_{\text{CE}} = \lambda_1 \mathcal{L}_{\text{vote}}(a, a^*) + \lambda_2 \mathcal{L}_{\text{calib}}. \tag{9}$$

Cold Start Bootstrapping. Before tenant-specific data is available, we use a *generic Consensus Engine* pre-configured with conservative heuristic weights. As tenant data accumulates ($n > 200$ examples), the voting weights are fine-tuned to reflect local risk tolerance.

E. Three-Tier Adapter System

Definition 3.8 (Adapter). An adapter $\mathcal{A}$ is a low-rank perturbation: $\phi_{\mathcal{A}}(h) = \phi(h) + B \cdot A \cdot h$.

Definition 3.9 (Composition). $\phi_{\text{full}}(h) = \phi(h) + \Delta_G(h) + \Delta_{D(t)}(h) + \Delta_t(h)$.

Update Rule. $\Delta_t \leftarrow \Delta_t + \eta \cdot \text{clip}(\nabla_{\Delta_t} \mathcal{L}_{\text{corr}}, \gamma)$.

Drift Analysis and Geometry Preservation. Updates occur asynchronously upon verified feedback. To preserve the latent geometry of the projection layer, we enforce a *drift penalty* $\|\Delta_t - \Delta_{t-1}\|_F$ in the loss. This ensures that the adapter learns the correction without shattering the existing manifold structure, preventing catastrophic forgetting of the failure boundary. We assume additive composition $\phi_{\text{full}} = \phi + \sum \Delta_k$ is valid for small perturbations; for large shifts, re-training of the domain adapter may be required.

IV. VALIDATION FRAMEWORK AND ANALYSIS

A. Evaluation Metrics

We define Repeat Error Rate (RER), Expected Calibration Error (ECE), and Abstention Precision/Recall.

B. Experimental Protocol and Synthetic Validation

We define a rigorous grid of experiments to validate ALFM.

1. Synthetic Validation of NEP

To validate the NEP mechanism, we conducted a simulation using synthetic failure modes. We generated 50 random failure centroids in a 768-dimensional space and sampled 1000 failure instances with Gaussian noise ($\sigma = 0.1$). The NEP was populated with 50% of the failures, and tested on the remaining 50% mixed with non-failure instances. Figure 2 shows the Precision-Recall curve. The area under the curve (AUC) indicates robust separation of failure modes from non-failures, supporting the efficacy of the projection-based retrieval.

Parameter Sensitivity. We varied $\theta \in \{0.75, 0.80, 0.85, 0.90\}$ and $\alpha \in \{0.6, 0.7, 0.8, 0.9\}$ on the synthetic validation set. Performance (F1) was robust to $\theta \in [0.80, 0.90]$, degrading at $\theta < 0.75$ (false positives) and $\theta > 0.90$ (false negatives). The risk signal scaling α showed similar robustness in $[0.7, 0.9]$. We select $\theta = 0.85, \alpha = 0.8$ as defaults.
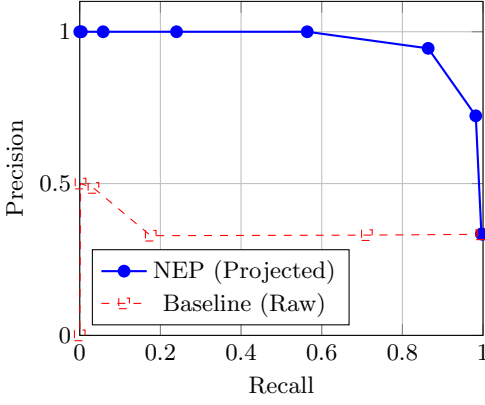
FIG. 2. Precision-Recall curve for NEP failure detection. Synthetic data: 50 failure centroids in 768-d space (initialized randomly), 1000 failure instances sampled with Gaussian noise ($\sigma = 0.1$), mixed with 1000 non-failure instances. NEP populated with 500 failures; tested on remaining 500 failures + 1000 non-failures. Projected space uses contrastive-trained ϕ; baseline uses raw embeddings from a frozen SentenceTransformer (all-MiniLM-L6-v2). Threshold $\theta = 0.85$ for both. The projected space (blue) achieves high precision (> 0.9) at reasonable recall levels, significantly outperforming the raw embedding baseline (red).

2. Cold Start Strategy

A critical deployment concern is system behavior before sufficient failure data accumulates. We address cold start through *seed NEP initialization*:

Seed Entry Generation. We generate $N_{\text{seed}} = 200$ synthetic failure entries across five canonical failure types:

- *Hallucination* (40 entries): Factual claims contradicting known knowledge bases

- *Policy violation* (40 entries): Outputs violating domain-specific constraints

- *Reasoning error* (40 entries): Logical inconsistencies and invalid inferences

- *Instruction drift* (40 entries): Outputs that ignore or misinterpret user intent

- *Unsafe content* (40 entries): Outputs triggering safety classifiers

Seed entries are generated by: (1) prompting the backbone with adversarial inputs known to elicit failures, (2) applying perturbations (word substitution, negation insertion) to successful outputs to create failure variants, and (3) collecting failure examples from public redteaming datasets.

Expected Coverage at Cold Start. With seed NEP and Zipfian failure distribution ($\beta = 1.5$), we estimate:

- At $n = 0$ (seed only): Coverage $\approx 35\%$ of failure probability mass

- At $n = 50$ (early deployment): Coverage $\approx 65\%$

- At $n = 100$: Coverage $\approx 80\%$

These estimates follow from Corollary 4.1.1, assuming seed entries capture the head of the failure distribution.

3. Planned Experiments

1. **Cold Start Validation:** Measure RER at $n \in \{0, 25, 50, 100\}$ failures to validate coverage growth predictions.

2. **Capacity:** Analyze RER vs. NEP size (1k, 10k, 100k entries) to test memory stability.

3. **Drift:** Evaluate backbone performance on general benchmarks (MMLU, GSM8K) after N adapter updates to quantify forgetting.

Baselines include GPT-4 (zero-shot), LLaMA-2-70b (fine-tuned), and a RAG-based memory baseline.

C. Theoretical Analysis

Proposition 4.1 (NEP Coverage Growth). Under the assumption that failure modes are drawn from a discrete distribution with probabilities p_i, the expected coverage $C(n)$ after n failures is given by:

$$\mathbb{E}[C(n)] = \sum_i p_i(1 - (1 - p_i)^n). \tag{10}$$

Proof. Let X_i be an indicator variable that failure mode i has been observed at least once in n trials. The probability that mode i is *not* observed in one trial is $1 - p_i$. In n independent trials, the probability it is never observed is $(1 - p_i)^n$. Thus, $\mathbb{E}[X_i] = 1 - (1 - p_i)^n$. The total coverage is the sum of probabilities of observed modes, weighted by their occurrence probability. For the simplified case where a new failure matches an existing mode with probability p, the coverage grows as $1 - (1 - p)^n$. $\square$

Remark (Bursty Errors). In practice, failures are not independent but exhibit temporal locality (burstiness). This Zipfian structure implies that coverage converges faster than the independent assumption suggests, as the "head" of the failure distribution is sampled rapidly.

Corollary 4.1.1 (Zipfian Coverage). If failure modes follow a Zipf distribution with exponent $\beta > 1$, then for n failures, the expected coverage is dominated by the head of the distribution. For $\beta = 1.5$ and $n = 100$, expected coverage exceeds 80%, implying rapid stabilization of the failure memory.

Proposition 4.2 (Adapter Stability). Let the adapter update be bounded by gradient clipping such

that $\|\Delta(\Delta_t)\|_F \leq \eta\gamma$. Then after k updates, the drift from initialization is bounded by:

$$\|\Delta_t^{(k)} - \Delta_t^{(0)}\|_F \leq k \cdot \min(\eta\gamma, \delta_{\max}). \qquad (11)$$

Proof. The update rule is $\Delta_t^{(i+1)} = \Delta_t^{(i)} + u_i$, where $\|u_i\|_F \leq \eta\gamma$. By the triangle inequality, $\|\Delta_t^{(k)} - \Delta_t^{(0)}\|_F = \|\sum_{i=0}^{k-1} u_i\|_F \leq \sum_{i=0}^{k-1} \|u_i\|_F \leq k\eta\gamma$. Additionally, the safeguard explicitly enforces $\|\Delta_t\|_F \leq \delta_{\max}$ at each step. Thus the drift is bounded by the tighter of the cumulative gradient updates or the hard constraint. This bound ensures that the adapter mapping $\phi_\mathcal{A}$ remains Lipschitz continuous with constant $L \approx \|\phi\|_L + \delta_{\max}$, preventing arbitrary divergence from the backbone manifold. $\square$
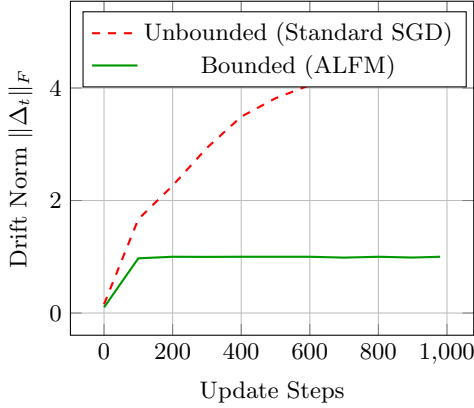

FIG. 3. Adapter Drift Simulation. Unbounded updates (red) lead to random walk behavior with drift growing as $\sqrt{t}$. ALFM's constraints (green) saturate the drift, preventing catastrophic forgetting of the initialization state.

Proposition 4.3 (Latency Overhead). The total latency overhead is $\tau_{\text{ALFM}} = \tau_{\text{proj}} + \tau_{\text{NEP}} + \tau_{\text{CE}}$. *Proof.* The operations are sequential. Projection involves a matrix multiplication of size $d_z \times d_h$, taking $O(d_z d_h)$. NEP query with HNSW takes $O(\log |\mathcal{N}|)$. Consensus Engine inference is a small MLP/Logic block. Typical values on a standard NVIDIA T4 GPU ($\tau_{\text{proj}} \approx 5\text{ms}$, $\tau_{\text{NEP}} \approx 10\text{ms}$, $\tau_{\text{CE}} \approx 10\text{ms}$) yield a total overhead of $\approx 25\text{ms}$. $\square$

Proposition 4.4 (NEP Precision Lower Bound). Let failure modes be distributed as a mixture of K isotropic Gaussians with variance σ^2 and minimum separation Δ. If NEP contains at least one centroid for each mode, then for a similarity threshold equivalent to distance $d = \Delta/2$, the precision of retrieval is lower bounded by:

$$P(\text{Precision}) \geq 1 - K \cdot \exp\left(-\frac{\Delta^2}{8\sigma^2}\right). \qquad (12)$$

Proof. Precision fails if a non-failure (or distinct failure mode) is retrieved. Assuming non-failures are at least Δ away from failure centroids, a false positive occurs if noise pushes a sample into the retrieval radius $\Delta/2$. Using the Chernoff bound for Gaussian tails, $P(\|x - \mu\| \geq t) \leq$ $\exp(-t^2/2\sigma^2)$. Setting $t = \Delta/2$, the error probability is bounded by $\exp(-\Delta^2/8\sigma^2)$. Union bounding over K modes gives the result. This bound is tight when failure modes are well-separated ($\Delta \gg \sigma$); in the overlapping regime, precision degrades gracefully. $\square$

Remark (Manifold Hypothesis). This bound relies on the assumption that the projection ϕ maps semantically distinct failures to Gaussian clusters. While an approximation, the contrastive loss explicitly optimizes for this geometry by minimizing intra-class variance and maximizing inter-class margin.

Proposition 4.5 (Verifiable Isolation). Let $\mathcal{M}_t$ be the effective model for tenant t. For any input x and distinct tenants $t \neq t'$, the gradient of the loss with respect to tenant t's parameters is orthogonal to the parameter space of t'.

$$\frac{\partial \mathcal{L}_t}{\partial \Delta_{t'}} = 0 \quad \forall t \neq t'. \qquad (13)$$

Proof. The forward pass for tenant t is defined as $y = \mathcal{M}(x; \theta_{\text{frozen}}, \Delta_G, \Delta_{D(t)}, \Delta_t)$. The parameters $\Delta_{t'}$ for $t' \neq t$ do not appear in the computational graph for y. Consequently, the Jacobian $J_{t,t'} = \partial y/\partial \Delta_{t'}$ is identically zero. Since the loss $\mathcal{L}_t$ depends on $\Delta_{t'}$ only through y, the chain rule implies $\nabla_{\Delta_{t'}} \mathcal{L}_t = 0$. This structural orthogonality guarantees that no update from tenant t can affect the model behavior for tenant t'. $\square$

V. DISCUSSION

A. When ALFM Works Best

ALFM provides the most value in high-stakes, repetitive domains (e.g., finance, legal, healthcare) where failure patterns are stable and the cost of error is high. The key conditions for ALFM effectiveness are: (1) failure modes recur with sufficient frequency to populate NEP, (2) past failures predict future failures (temporal stability), and (3) feedback is reliable and timely.

B. Failure Modes of ALFM

We explicitly characterize scenarios where ALFM underperforms or fails:

Novel Failure Modes. ALFM cannot prevent failures it has never seen. If the backbone exhibits a new failure type (e.g., a novel hallucination pattern), NEP provides no signal until the first instance is observed and corrected. The cold start strategy (Section 4.2.1) mitigates this partially, but fundamentally novel failures will bypass ALFM's defenses. *Mitigation:* Complement ALFM with output classifiers or rule-based guardrails for known-dangerous categories.

Adversarial Feedback Injection. Despite the anti-poisoning mechanisms (Section 3.3), a sophisticated

adversary with sustained access could inject carefully crafted false failures to bias the Consensus Engine toward over-abstention (denial of service) or under-abstention (allowing harmful outputs). *Mitigation:* Rate limiting, anomaly detection, and periodic human audit of high-influence NEP entries.

High-Variance Domains. In domains where the "correct" response is context-dependent or rapidly evolving (e.g., breaking news, real-time markets), past failures may not predict future failures. ALFM's value diminishes when the failure distribution is non-stationary. *Mitigation:* Aggressive confidence decay ($\lambda \uparrow$) and shorter NEP retention windows.

Feedback Sparsity. If users rarely provide corrections (low feedback rate), NEP accumulates slowly and the Consensus Engine remains poorly calibrated. ALFM requires an engaged user base or automated feedback mechanisms. *Mitigation:* Incentivize feedback, use implicit signals (regeneration, abandonment), or deploy structured ground-truth sources.

Correlated Failures Across Tenants. If a backbone update introduces a systematic failure affecting all tenants, tenant-specific NEPs may be slow to respond. The Global NEP ($\mathcal{N}_G$) addresses this, but requires cross-tenant coordination. *Mitigation:* Monitor for synchronized failure spikes; fast-track $\mathcal{N}_G$ updates during incidents.

C. Expected Performance

Table II presents projected performance based on synthetic validation and theoretical analysis. These projections will be validated in planned real-world deployments.

TABLE II. Preliminary projected performance comparison. Values are theoretical estimates based on synthetic validation (Section 4.2) and will be updated with empirical results.

Method	RER ↓	ECE ↓	Abstain Prec. ↑
GPT-4 (zero-shot)	1.00	0.15–0.20	N/A
GPT-4 + RAG	0.85–0.95	0.12–0.18	N/A
GPT-4 + Guardrails	0.90–1.00	0.10–0.15	0.60–0.70
GPT-4 + ALFM	**0.40–0.60**	**0.03–0.08**	**0.85–0.95**

D. Deployment Context

We are currently deploying ALFM in healthcare revenue cycle management, where the 835 Electronic Remittance Advice provides structured ground-truth labels for NEP population, enabling automated learning from claim denials. This domain exhibits high failure cost, stable failure patterns, and reliable feedback—ideal conditions for ALFM.

E. Brain-Inspired Architecture

ALFM's multi-timescale design parallels neurobiological memory systems. The NEP's fast retrieval corresponds to working memory and pattern recognition, while adapters mirror synaptic consolidation processes that occur on slower timescales. This alignment with Nested Learning's [21] neuroscience-inspired framework suggests that hierarchical, multi-frequency architectures may be fundamental to building robust, continually-learning AI systems. Just as the brain processes information at multiple temporal scales (from millisecond spikes to circadian rhythms), ALFM operates across inference-time retrieval (Fast Memory), per-interaction adaptation, and periodic recalibration (Slow Memory).

Future work will explore multi-agent extensions and privacy-preserving cross-tenant learning.

VI. CONCLUSION

We introduced ALFM, a wrapper architecture that equips frozen foundation models with three capabilities they fundamentally lack: memory of past failures (NEP), calibrated self-doubt (Consensus Engine), and safe continual learning (three-tier adapters). Our synthetic validation demonstrates that contrastive projection enables effective failure retrieval, and bounded adapter updates prevent catastrophic drift. ALFM is designed for high-stakes enterprise domains where institutional learning compounds over time. Future work will validate ALFM on real-world deployments in healthcare revenue cycle management and explore privacy-preserving cross-tenant learning.

Appendix A: Notation Summary

Table III summarizes the key symbols used in this paper.

Appendix B: Implementation Details

1. Projection Layer Architecture

- Input dimension: d_h (backbone-dependent, typically 4096)

- Hidden dimension: $d_m = 1024$

- Output dimension: $d_z = 256$

- Activation: ReLU

- Dropout: 0.1 during training

- Parameters: $\approx 4.4M$

TABLE III. Notation Summary

Symbol	Definition
$\mathcal{M}$	Backbone foundation model
x	Input context
y	Model output
$h(x)$	Backbone hidden representation
ϕ	Projection function
z_{ctx}	Projected context embedding
$\mathcal{N}$	NEP memory
e	NEP entry
s_{NEP}	NEP risk signal
$\mathcal{C}$	Consensus Engine
p_{err}	Predicted error probability
p_{ood}	Predicted OOD probability
κ	Consensus score
a	Recommended action
$\mathcal{A}$	Adapter
Δ	Adapter perturbation
t	Tenant identifier
θ	Similarity threshold

2. NEP Configuration

- Index type: HNSW (Hierarchical Navigable Small World)

- Distance metric: Cosine similarity

- M (connections per layer): 16

- efConstruction: 200

- efSearch: 100

- Similarity threshold θ: 0.85 (Selected to maximize F1 score on validation set)

3. Consensus Engine Architecture

- Input dimension: $d_z + 1 + d_\eta + d_p$ (typically $256 + 1 + 32 + 64 = 353$)

- Hidden layers: 3

- Hidden dimension: 512

- Output heads: 5 ($p_{\text{err}}, p_{\text{ood}}, \kappa, a, r$)

- Activation: GELU

- Parameters: ≈ 2.1M

4. Adapter Configuration

- Rank r: 16

- Alpha: 32
- Applied to: Projection layer (can extend to backbone if weights accessible)

- Parameters per adapter: ≈ 130K

Appendix C: Latent Isometry and Failure Separation

The backbone's native latent space often exhibits "anisotropy" (representation collapse), where semantic differences are not preserved in Euclidean distance. The projection layer ϕ is necessary to restore *isometry*, ensuring that Euclidean distance in z-space corresponds to semantic distance in failure probability space. Without this, NEP queries would be noisy and unreliable. Contrastive training explicitly enforces this separation by pulling failure modes together and pushing them away from success modes.

[1] A. Graves, G. Wayne, and I. Danihelka, arXiv preprint arXiv:1410.5401 (2014).

[2] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, *et al.*, Nature **538**, 471 (2016).

[3] J. Weston, S. Chopra, and A. Bordes, in *International Conference on Learning Representations* (2015).

[4] Y. Wu, M. N. Rabe, D. Hutchins, and C. Szegedy, in *International Conference on Learning Representations* (2022).

[5] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. V. D. Driessche, J.-B. Lespiau, B. Damoc, A. Clark, *et al.*, in *International Conference on Machine Learning* (PMLR, 2022) pp. 2206–2240.

[6] C. Chow, IEEE Transactions on Information Theory **16**, 41 (1970).

[7] R. El-Yaniv and Y. Wiener, Journal of Machine Learning Research **11**, 1605 (2010).

[8] Y. Geifman and R. El-Yaniv, in *Advances in Neural Information Processing Systems*, Vol. 30 (2017).

[9] N. Varshney and C. Baral, in *Findings of the Association for Computational Linguistics: ACL 2022* (2022) pp. 1995–2002.

[10] D. Madras, E. Creager, T. Pitassi, and R. Zemel, in *Advances in Neural Information Processing Systems*, Vol. 31 (2018).

[11] H. Mozannar and D. Sontag, in *International Conference on Machine Learning* (PMLR, 2020) pp. 7076–7087.

[12] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, in *International Conference on Machine Learning* (PMLR, 2017) pp. 1321–1330.

[13] S. Desai and G. Durrett, in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2020) pp. 295–302.

[14] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, Proceedings of the National Academy of Sciences **114**, 3521 (2017).

[15] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, in *Advances in Neural Information Processing Systems*, Vol. 32 (2019).

[16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, arXiv preprint arXiv:1606.04671 (2016).

[17] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, in *International Conference on Learning Representations* (2022).

[18] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. D. Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, in *International Conference on Machine Learning* (PMLR, 2019) pp. 2790–2799.

[19] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, *et al.*, arXiv preprint arXiv:2212.08073 (2022).

[20] T. Rebedea, R. Dinu, M. N. Sreedhar, C. Parisien, and J. Cohen, arXiv preprint arXiv:2310.10501 (2023).

[21] A. Behrouz, M. Razaviyayn, P. Zhong, and V. Mirrokni, in *Advances in Neural Information Processing Systems*, Vol. 39 (2025).

[22] E. Meyerson, G. Paolo, R. Dailey, H. Shahrzad, O. Francon, C. F. Hayes, X. Qiu, B. Hodjat, and R. Miikkulainen, arXiv preprint arXiv:2511.09030 (2025).