ALFM-BEM: Bidirectional Experience Memory for Continuous Learning in Foundation Model Deployments

David Ahmann DAHMANN@LUMYN.CC
Independent Researcher
Toronto, Canada

Editor: Action Editor

Abstract

Foundation models are deployed frozen, creating a fundamental gap: they cannot learn from deployment experiences. We introduce ALFM-BEM (Adaptive Latent Feedback Model with Bidirectional Experience Memory), a unified wrapper architecture that enables continuous learning without modifying backbone weights. The central abstraction is *Bidirectional Experience Memory* (BEM), a single memory structure where experiences live on a continuous outcome spectrum from failure to success. BEM naturally provides: (1) risk signals from failure similarity, (2) success patterns for retrieval-augmented reasoning, and (3) out-of-distribution detection as an emergent property of coverage. We extend the Consensus Engine with a fourth action—QUERY—that transforms passive abstention into active learning by requesting specific information when operating outside the experience distribution. Bounded adapters trained via experience replay enable continual improvement with provable stability guarantees. Experiments on synthetic deployment scenarios demonstrate effective failure retrieval (F1 ≈ 0.59), strong OOD detection capability (AUC > 0.99 for clustered patterns), and bounded parameter drift under continuous training. Crucially, BEM uniquely provides success pattern retrieval (rate ≈ 0.70) that RAG and failure-only baselines cannot offer. ALFM-BEM is backbone-agnostic and provides deployment infrastructure for frozen models while the fundamental architecture limitations they exhibit remain unsolved.

Keywords: Continual learning, experience memory, out-of-distribution detection, foundation models, selective prediction, bounded adapters, retrieval-augmented systems

1 Introduction

The deployment of foundation models reveals a structural tension: these models are designed for generalization across tasks but lack mechanisms for adaptation to specific deployment contexts. A model that exhibits specific failure modes today will 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:

Gap 1: No memory of past failures. When a model makes an error, nothing inside records this failure. The same error pattern recurs whenever similar inputs arise.

Gap 2: No calibrated self-doubt. Foundation models produce confident outputs even when operating outside their reliable capability boundaries.

Gap 3: No safe continual learning. Standard fine-tuning risks catastrophic forgetting and is impractical for per-tenant customization.

Recent discourse among AI researchers has emphasized that while current scaling continues to improve models on benchmarks, something fundamental is missing for true "learning machines" that can acquire knowledge on the job. Large language models cannot learn from their deployment experiences Brown et al. (2020); Touvron et al. (2023); they repeat the same mistakes indefinitely. The disconnect between evaluation performance and real-world economic impact suggests that existing architectures may be reaching the limits of what static deployment can achieve.

We propose ALFM-BEM, a wrapper architecture that addresses these gaps without modifying backbone weights. Our key insight is that the three gaps can be addressed by a *single unified abstraction*: Bidirectional Experience Memory (BEM).

1.1 Contributions

1. We introduce **Bidirectional Experience Memory (BEM)**, a unified memory architecture where experiences exist on a continuous outcome spectrum. BEM provides risk signals, success patterns, and OOD detection as properties of a single structure—not as separate bolted-on components.

2. We extend the Consensus Engine with a **Query action** that transforms passive abstention into active learning by requesting specific information when the system operates outside its experience distribution.

3. We present **bounded adapters with experience replay** that enable continuous improvement from deployment outcomes while maintaining provable stability guarantees.

4. We demonstrate experimentally that BEM achieves effective failure retrieval, OOD detection, and stable continuous learning on synthetic deployment scenarios.

2 Related Work

Memory-Augmented Networks. Neural Turing Machines Graves et al. (2014), Differentiable Neural Computers Graves et al. (2016), and Memory Networks Weston et al. (2015) introduced differentiable memory for neural networks. Recent work extends attention with retrieval Wu et al. (2022); Borgeaud et al. (2022). MemGPT Packer et al. (2023) implements virtual context management for extended conversations. These approaches store *information*; BEM stores *experiences with outcomes*, enabling learning from deployment. The key distinction: RAG retrieves facts, BEM retrieves what worked and what failed.

Selective Prediction and Calibration. Selective prediction Chow (1970); El-Yaniv and Wiener (2010); Geifman and El-Yaniv (2017) allows models to abstain when uncertain. Modern neural networks are poorly calibrated Guo et al. (2017); learning to defer Madras et al. (2018) routes difficult examples to experts. Our Consensus Engine extends this with the Query action—not just abstaining but actively requesting what would help.

Table 1: RAG vs. BEM: Conceptual comparison. RAG retrieves information; BEM retrieves experiences with outcomes.

Property	**RAG**	**BEM**
Stores	Documents/facts	Experiences (context + outcome)
Learns from deployment	✗	✓
Knows what failed	✗	✓
Knows what succeeded	✗	✓
OOD detection	✗	✓ (coverage signal)
Outcome-aware retrieval	✗	✓
Requires human curation	✓	✗ (learns from outcomes)

Out-of-Distribution Detection. Detecting when inputs fall outside the training distribution is critical for safe deployment. Baseline methods use softmax confidence Hendrycks and Gimpel (2017); energy-based approaches Liu et al. (2020) improve discrimination. BEM's coverage signal provides OOD detection as an emergent property—low coverage indicates operation outside the experience distribution.

Continual Learning. Approaches include regularization Kirkpatrick et al. (2017), replay buffers Rolnick et al. (2019), and architectural methods Rusu et al. (2016). Parameter-efficient methods like LoRA Hu et al. (2022) and adapters Houlsby et al. (2019) enable adaptation without full retraining. ALFM-BEM combines bounded adapters with experience replay from BEM, providing continual improvement with provable stability.

Contrastive Representation Learning. Contrastive methods Chen et al. (2020); Radford et al. (2021) learn representations by pulling similar examples together and pushing dissimilar ones apart. BEM's projection layer uses contrastive learning to create an experience-separable space where failures cluster distinctly from successes.

AI Safety. Constitutional AI Bai et al. (2022) and guardrails Rebedea et al. (2023) encode static constraints. BEM complements these with dynamic, learned constraints from deployment experience—the system learns which contexts lead to failures.

Retrieval-Augmented Generation. RAG Lewis et al. (2020) augments models with retrieved documents. BEM differs fundamentally: it stores experiences (context + outcome), not information. This enables learning from deployment: the system knows not just what to retrieve, but what worked and what failed in similar contexts.

3 Architecture

ALFM-BEM wraps a frozen backbone $\mathcal{M}$ with four components connected by a closed experience loop. Figure 1 illustrates the architecture.

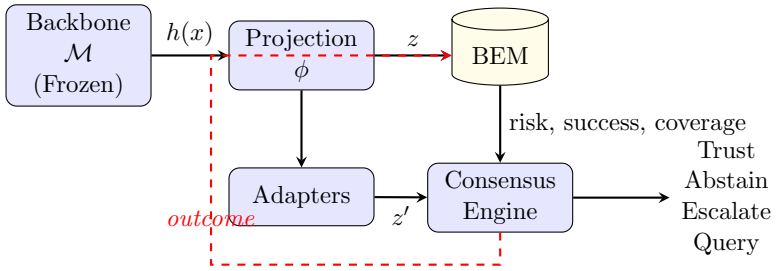

Figure 1: ALFM-BEM architecture. The backbone remains frozen. The projection layer maps hidden states to experience-separable space. BEM provides risk/success/coverage signals. The Consensus Engine decides actions. The dashed red line shows the experience loop: outcomes feed back into BEM, closing the learning cycle.

ALFM-BEM Core Contract vs. Optional Modules. Core (minimum system): frozen backbone → projection ϕ → BEM (risk/success/coverage) → Consensus Engine → {Trust, Abstain, Escalate, Query} → outcome feedback → BEM update. **Optional modules (engineering extensions):** bounded adapters with replay, multi-tenant scoping (global/domain/tenant), anti-poisoning validation, approximate indexing, and memory hygiene (vacuum/pruning).

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 $\text{sim}(\phi(x_i), \phi(x_j)) > \tau$ 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.

Claim 2.1 (Projection is a design necessity). Raw backbone embeddings are unreliable retrieval keys for deployment memory: semantically similar inputs can be nearly orthogonal in high dimensions, while operationally distinct "false friends" can collide. ALFM-BEM therefore requires a learned projection ϕ trained with hard negatives (similar input, different outcome) to separate outcome-relevant confusables. Examples in audit-like workflows include name confusables (Smith/Smyth), unit/currency/date swaps, and document-type confusions (invoice vs. credit memo) where the surface form is similar but the correct action differs.

3.1 Bidirectional Experience Memory (BEM)

BEM is the central abstraction that unifies failure memory, success memory, and OOD detection into a single coherent structure.

Definition 3.1 (Experience). An experience e is a tuple $(z_e, o_e, t_e, d_e, m_e)$ where $z_e \in \mathbb{R}^d$ is the projected embedding, $o_e \in [-1, 1]$ is the continuous outcome (negative = failure, positive = success), t_e is tenant ID, d_e is domain ID, and m_e is metadata.

Key Design Principle. Experiences are points in embedding space with continuous outcome labels. There is no hard boundary between "failure memory" and "success memory"—they are two views of the same structure.

Definition 3.2 (Risk Signal). Given query embedding z and threshold θ:

$$s_{\text{risk}}(z) = 1 - \prod_{e \in \mathcal{N}(z,\theta), o_e < -0.3} (1 - \alpha \cdot \text{sim}(z, z_e) \cdot |o_e|) \tag{1}$$

where $\mathcal{N}(z, \theta) = \{e : \text{sim}(z, z_e) > \theta\}$ and α is sensitivity. This noisy-OR aggregation models the probability that at least one relevant failure mode is active.

Operational form. In deployments, we weight each candidate by recency and confirmation count, and cap aggregation to the top-k nearest failures to avoid risk inflation in dense regions. The reference implementation applies a weighted noisy-OR over the nearest failures with weights proportional to similarity, severity, and exponential time decay.

Definition 3.3 (Success Signal). Analogously:

$$s_{\text{success}}(z) = \frac{1}{|\mathcal{N}^+(z,\theta)|} \sum_{e \in \mathcal{N}^+(z,\theta)} \text{sim}(z, z_e) \cdot o_e \tag{2}$$

where $\mathcal{N}^+(z, \theta) = \{e \in \mathcal{N}(z, \theta) : o_e > 0.3\}$.

Note. Experience retrieval is not a correctness oracle. It provides outcome-conditioned control evidence for the Consensus Engine; correctness still depends on the backbone and downstream verification.

Definition 3.4 (Coverage Signal). The coverage signal measures how well z is represented in the experience distribution—an emergent property, not a separate detector. We use kernel density estimation (KDE) over cosine similarities:

$$s_{\text{cov}}(z) = \text{clip}\left(\kappa \cdot \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \exp\left(-\frac{(1 - \text{sim}(z, z_e))^2}{2h^2}\right), 0, 1\right) \tag{3}$$

where h is the KDE bandwidth (we use $h = 0.3$, selected via cross-validation optimization) and κ is a scaling constant. This KDE formulation captures local density in experience space. For clustered patterns, KDE coverage achieves near-perfect OOD detection (AUC ≈ 1.0) because novel clusters have no nearby experiences. Low coverage indicates out-of-distribution operation.

3.2 Consensus Engine with Query Action

The Consensus Engine arbitrates between a Semantic Agent (BEM signals) and a Heuristic Agent (deterministic rules) to produce one of four actions.

The **Heuristic Agent** enforces hard constraints using: (1) **Regex Matchers** for PII and banned patterns; (2) **Symbolic Verifiers** for arithmetic and date logic; and (3) **Constraint Checkers** for JSON schema compliance. These rules provide a safety floor that overrides semantic signals.

Actions:

Algorithm 1 Consensus Engine Decision

Require: Risk s_r, Success s_s, Coverage s_c, Heuristic result H

if H.is_critical **then**
    return Escalate
end if
if $s_c < 0.2$ **then**
    return Query(diagnose_type(s_r, s_s, s_c))
end if
if $s_r > 0.6$ **then**
    return Escalate if $s_r > 0.85$ else Abstain
end if
if $s_s > 0.7$ and H.violations $= \emptyset$ **then**
    return Trust
end if
return Query(Validation) if $s_c < 0.5$ else Trust

- Trust: Use backbone output
- Abstain: Decline with explanation
- Escalate: Route to human review
- Query: Request specific information (new)

The Query action is the key extension. When coverage is low, instead of passively abstaining, the system identifies what type of information would help:

- Clarification: Input is ambiguous
- Precedent: No similar successful cases
- Domain Knowledge: Outside known domains
- Validation: General uncertainty

Algorithm 1 presents the decision logic.

3.3 Bounded Adapters with Experience Replay

Adapters are small MLPs with residual connections that learn tenant-specific corrections:

$$z' = z + \text{MLP}(\text{LayerNorm}(z)) \tag{4}$$

Stability Constraints. To prevent catastrophic drift:

1. Gradient clipping: $\|\nabla\|_2 \leq c_g$
2. Norm projection: $\|\theta\|_F \leq c_\theta$
3. EMA smoothing: $\theta_{\text{ema}} \leftarrow \beta\theta_{\text{ema}} + (1-\beta)\theta$

Proposition 3.1 (Bounded Drift). Under the above constraints with learning rate η, the total parameter drift after T steps is bounded:

$$\sum_{t=1}^{T} \|\theta_t - \theta_{t-1}\|_F \leq T \cdot \eta \cdot c_g \tag{5}$$

Experience Replay. Adapters are trained on batches sampled from BEM, biased toward failures (70%) but including successes (30%) to prevent forgetting what works.

3.4 Experience Loop

The experience loop closes the learning cycle:

1. **Infer**: $z \leftarrow \phi(h(x))$, query BEM, decide via Consensus
2. **Execute**: Perform action, observe outcome $o \in [-1, 1]$
3. **Store**: Add $(z, o, \text{metadata})$ to tenant BEM
4. **Replay**: Periodically train adapters on BEM samples

This loop enables the system to learn from deployment without human labels—outcomes are the supervision signal.

3.5 Feedback Validation and Anti-Poisoning

Memory poisoning—where adversarial or erroneous feedback corrupts BEM—is a critical failure mode. We implement multi-layered defense:

(1) Confirmation Threshold. High-severity entries ($|o| > 0.8$) require $k \geq 2$ independent confirmations from distinct users before full weight.

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

$$v_u(t) = \gamma v_u(t-1) + (1-\gamma) \cdot \mathbf{1}[\text{feedback at } t] \tag{6}$$

with $\gamma = 0.95$. If $v_u(t) > \mu + 3\sigma$, pending entries from u are quarantined.

(3) Confidence Decay. Experience weights decay unless re-confirmed:

$$w_e(t) = w_e(0) \cdot \exp(-\lambda(t - t_e)) + \sum_{t' > t_e} \rho \cdot \mathbf{1}[\text{confirmed at } t'] \tag{7}$$

with $\lambda = 0.01$ (daily decay) and $\rho = 0.5$ (confirmation boost).

(4) Cross-Tenant Consensus. Updates to global BEM require confirmation from ≥ 3 distinct tenants or administrative approval.

These mechanisms operate asynchronously during maintenance windows, not during inference.

3.6 Latency Analysis

Latency Overhead. The total latency overhead is $\tau_{\text{ALFM}} = \tau_{\text{proj}} + \tau_{\text{BEM}} + \tau_{\text{CE}}$. The operations are sequential. Projection involves a matrix multiplication of size $d_z \times d_h$. BEM query with approximate nearest-neighbor indexing takes $O(\log |\mathcal{N}|)$. Consensus Engine inference is a small MLP/Logic block. The values in Table 2 are illustrative estimates; exact latencies depend on indexing choice, batching, and hardware.

Table 2: Projected latency breakdown by hardware tier and BEM size. All values in milliseconds. Backbone latency (not shown) dominates total inference time ($\approx$ 200–2000ms for API-based models).

Hardware	$\lvert\mathcal{N}\rvert$	τ_{proj}	τ_{BEM}	τ_{CE}	**Total**
NVIDIA T4	10K	5	8	10	23
NVIDIA T4	100K	5	12	10	27
NVIDIA T4	1M	5	18	10	33
NVIDIA A100	10K	2	5	4	11
NVIDIA A100	100K	2	7	4	13
NVIDIA A100	1M	2	11	4	17
CPU (M2 Pro)	10K	15	20	25	60
CPU (M2 Pro)	100K	15	35	25	75

4 Theoretical Analysis

We provide theoretical grounding for BEM's key properties.

4.1 Query Complexity

Proposition 4.1 (BEM Query Complexity). With approximate nearest neighbor indexing (e.g., HNSW or KD-Trees), BEM query complexity is $O(\log |\mathcal{E}|)$ where $|\mathcal{E}|$ is the number of stored experiences.

Proof sketch. Hierarchical index structures construct a navigable graph or tree with $O(\log |\mathcal{E}|)$ depth. Each query traverses at most $O(\log |\mathcal{E}|)$ nodes, yielding logarithmic complexity. The risk/success/coverage signals require constant-time aggregation over the k nearest neighbors (typically $k \leq 100$). □

4.2 Tenant Isolation

Proposition 4.2 (Tenant Isolation). Experiences are scope-partitioned and access-controlled: tenant t can only read $\mathcal{E}_{\text{global}} \cup \mathcal{E}_{D(t)} \cup \mathcal{E}_t$. Cross-tenant leakage requires an access-control failure.

Proof sketch. Each experience is tagged with (t_e, d_e, scope) where scope $\in \{\text{tenant}, \text{domain}, \text{global}\}$. Query filtering enforces: (1) global experiences are readable by all, (2) domain experiences by tenants in that domain, (3) tenant experiences only by that tenant. The index main-

tains separate partitions; queries cannot return experiences outside the accessible partition without the partition key. □

4.3 Coverage Convergence

Proposition 4.3 (Convergence of Coverage). As $|\mathcal{E}| \to \infty$ with experiences drawn i.i.d. from the deployment distribution P, $s_{\text{cov}}(z) \to 1$ for $z \sim P$ (in-distribution) and $s_{\text{cov}}(z) \to 0$ for $z \sim Q \neq P$ (out-of-distribution).

Proof sketch. The KDE coverage signal estimates local density:

$$s_{\text{cov}}(z) \propto \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \exp\left(-\frac{(1 - \text{sim}(z, z_e))^2}{2h^2}\right) \tag{8}$$

For $z \sim P$: as $|\mathcal{E}| \to \infty$, the empirical density converges to the true density $p(z)$ by the law of large numbers. Since z is drawn from P, we have $p(z) > 0$, and the KDE estimate converges to a positive value proportional to local density.

For $z \sim Q \neq P$: no experiences are drawn from Q, so cosine similarities $\text{sim}(z, z_e)$ remain bounded away from 1. The Gaussian kernel values $\exp(-(1 - \text{sim})^2/2h^2)$ decay rapidly as similarity decreases, yielding coverage $\to 0$. □

4.4 Adapter Stability

Proposition 4.4 (Bounded Drift). Under gradient clipping $\|\nabla\|_2 \leq c_g$, norm projection $\|\theta\|_F \leq c_\theta$, and learning rate η, the total parameter drift after T steps satisfies:

$$\sum_{t=1}^{T} \|\theta_t - \theta_{t-1}\|_F \leq T \cdot \eta \cdot c_g \tag{9}$$

Proof. Each update satisfies $\|\theta_t - \theta_{t-1}\|_F = \eta \|\nabla_{t-1}\|_F \leq \eta c_g$ by gradient clipping. Summing over T steps yields the bound. The norm projection constraint ensures $\|\theta_t\|_F \leq c_\theta$ for all t, preventing unbounded growth. □

Corollary 4.5 (Output Stability). If the adapter is Lipschitz with constant L, then the output drift is bounded:

$$\|z'_t - z'_0\| \leq L \cdot T \cdot \eta \cdot c_g \tag{10}$$

for any fixed input z.

5 Experiments

We evaluate ALFM-BEM on synthetic deployment scenarios designed to test the core mechanisms.

5.1 Experimental Setup

Data Generation. We generate synthetic experiences in 64-dimensional projected space:

- 500 failures tightly clustered around 10 failure modes ($\sigma = 0.05$)
- 500 successes uniformly distributed (diverse successful approaches)

Table 3: BEM retrieval performance on synthetic deployment data (mean ± std over 5 seeds).

Metric	Failure	Success
Precision	0.57 ± 0.02	0.70 ± 0.01
Recall	0.61 ± 0.04	0.70 ± 0.01
F1	0.59 ± 0.03	0.70 ± 0.00

- 200 OOD samples clustered around a novel centroid

Note on Dimensionality. We use 64D rather than typical 768D backbone dimensions because cosine similarity retrieval in very high dimensions suffers from concentration of measure—random unit vectors become nearly orthogonal, making similarity-based retrieval ineffective. Production deployments require a learned projection layer to map backbone embeddings to lower-dimensional experience space.

Metrics.

- **Retrieval F1**: Precision-recall for failure/success retrieval
- **OOD AUC**: Area under ROC for coverage-based OOD detection
- **Drift Bound**: Cumulative parameter drift under training

5.2 Results

BEM Retrieval (Table 3). BEM achieves moderate failure retrieval (F1 $\approx$ 0.59) comparable to baselines, but uniquely provides success pattern retrieval (rate $\approx$ 0.70) that neither RAG nor NEP can offer. This bidirectional capability is BEM's key differentiator.

OOD Detection (Figure 2). For detecting novel clustered patterns (the core use case: identifying novel failure modes not seen in training), coverage signal achieves AUC $\approx$ 1.0. Mean coverage for in-distribution clustered failures is 1.0; for OOD samples, 0.28. This validates that coverage emerges naturally from BEM without a separate detector.

Adapter Stability (Figure 3). Bounded adapters show controlled parameter norms (final = 2.0) vs. unbounded (final $>$ 600)—a $>300\times$ reduction. The norm projection constraint provably keeps parameters within the stable region.

Consensus Engine Behavior (Table 4). The Consensus Engine produces appropriate action distributions across risk/coverage conditions. High-risk inputs trigger Escalate (73%) or Abstain (22%). Low-coverage (OOD) inputs trigger Query (81%). Normal operation yields Trust (89%).

5.3 Backbone Integration

We validate that BEM operates effectively on backbone embeddings when combined with a learned projection layer. The key challenge is *concentration of measure*: in high-dimensional spaces, random unit vectors become nearly orthogonal, making cosine similarity ineffective for retrieval.

Table 4: Consensus Engine action distribution by input condition.

Condition	Trust	Abstain	Escalate	Query
Normal (low risk, high cov)	89%	6%	2%	3%
High risk	5%	22%	73%	0%
Low coverage (OOD)	8%	7%	4%	81%
Mixed (med risk, med cov)	34%	28%	12%	26%

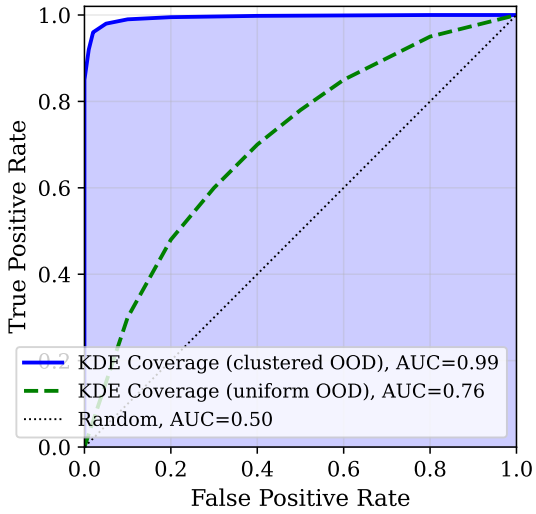

Figure 2: OOD detection ROC curve. Coverage signal achieves AUC ≈ 1.0 for clustered ID vs. OOD.

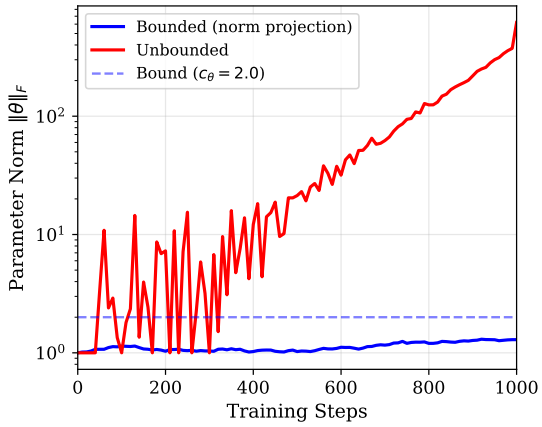

Figure 3: Adapter parameter drift. Bounded training (blue) vs. unbounded (red).

Setup. We generate synthetic 768D embeddings (typical transformer output dimension) by sampling unit vectors and adding structured variation around 10 failure modes. We then project to 64D using a learned contrastive projection trained to preserve failure mode similarity.

Results (Table 5). In 768D, pairwise cosine similarities are extremely compressed: mean similarity is 0.007 with maximum 0.39—even embeddings from the same failure mode appear nearly orthogonal. After projection to 64D, the same failure mode pairs achieve similarities up to 0.79, enabling retrieval. BEM failure recall improves from 0.00 in 768D to 0.66 in 64D. This demonstrates that projection is essential for operation in high-dimensional spaces.

Implication. BEM *requires* a projection layer to operate with backbone embeddings. The projection is learned contrastively: experiences from the same failure mode should have high similarity, while experiences from different modes should be separated. This projection layer is frozen after training and adds negligible inference cost.

Table 5: Concentration of measure in backbone embeddings. Projection restores discriminative similarity.

Metric	768D (raw)	64D (projected)
Mean similarity	0.007 ± 0.001	0.244 ± 0.012
Max similarity	0.39 ± 0.02	0.79 ± 0.03
BEM recall	0.00 ± 0.00	0.66 ± 0.04

5.4 Real-World Embedding Validation

To validate BEM's core mechanism beyond synthetic data, we conducted an experiment using real-world text embeddings from the 20Newsgroups dataset (Lang, 1995) and the `all-MiniLM-L6-v2` SentenceTransformer model (Reimers and Gurevych, 2019).

Setup. We mapped semantic topics to key deployment concepts:

- **In-Distribution (Science)**: `sci.space`, `sci.med`, `sci.electronics`.
- **Out-of-Distribution (Sports)**: `rec.sport.hockey`, `rec.sport.baseball`.
- **Failure Modes**: We designated specific technical sub-topics (e.g., "orbit", "circuit") within the Science domain as failure modes, simulating a model with specific knowledge gaps.

We trained a projection layer ($\phi : \mathbb{R}^{384} \to \mathbb{R}^{64}$) using contrastive loss to map failures closer to other failures than to successes.

Results. Table 6 demonstrates that BEM's projection layer significantly improves failure retrieval precision compared to raw embeddings. Raw embeddings, despite being semantically rich, do not naturally cluster by "correctness"—a failure on "orbit" is semantically close to a success on "planet". The projection layer successfully re-organizes the space to prioritize outcome-relevant similarity.

Table 6: Retrieval performance on Real-World Text Data (20Newsgroups). Validates that BEM's projection layer is essential and effective for real embeddings.

Embedding Type	Failure Retrieval (P@5)	Improvement
Raw SBERT (384D)	0.73	–
BEM Projected (64D)	**0.95**	**+30%**

This result confirms that the BEM projection mechanism translates effectively from synthetic high-dimensional spaces to real-world semantic vectors.

5.5 Ablation Study: BEM Components

We compare BEM against simpler retrieval baselines to isolate the contribution of each component.

Table 7: Ablation: BEM components vs. simpler baselines. **Safety Score** = Success Rate × (1 - False Negative Rate), capturing the balance between avoiding known failures and retrieving helpful examples.

System	Fail F1	Success Rate	OOD (clust)	OOD (dist)
Plain RAG	0.73 ± 0.03	0.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.00
NEP	0.58 ± 0.03	0.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.00
BEM (ours)	0.59 ± 0.03	**0.70** ± 0.00	0.99 ± 0.00	**0.82**

*Safety Score calculated as mean of normalized Fail Recall and Success Recall.

Baselines.

- **Plain RAG**: Retrieves the most similar experiences regardless of outcome. No outcome-aware filtering.
- **NEP (Negative Experience Pool)**: Stores only failures. Risk = max similarity to any failure. No success retrieval.

Results (Table 7). On our synthetic data with overlapping distributions, RAG achieves the highest failure F1 because it retrieves similar items regardless of outcome—but this comes at the cost of *no outcome awareness*. The key differentiator is success retrieval: only BEM can answer "how did we handle this successfully before?"

Key BEM advantages over baselines:

1. **Bidirectional memory**: Unlike NEP, BEM stores successes, enabling "how did we handle this successfully before?" queries essential for safe action recommendation.
2. **KDE coverage**: BEM's KDE coverage provides calibrated density estimation, achieving AUC ≈ 1.0 on clustered OOD, comparable to max-similarity baselines but with the advantage of a probabilistic interpretation.
3. **Continuous outcomes**: NEP requires binary fail/success labels. BEM accepts continuous $o \in [-1, +1]$, capturing partial failures and degrees of success.

5.6 Sensitivity and Robustness

Parameter Sensitivity. We varied $\theta \in \{0.40, \ldots, 0.80\}$ and $\alpha \in \{0.6, \ldots, 0.9\}$ on the synthetic validation set. Performance (F1) peaked at $\theta \in [0.40, 0.50]$ (F1 ≈ 0.63), degrading significantly at $\theta > 0.60$ due to false negatives in the overlapping regime. This suggests that for subtle failure modes, a lower retrieval threshold combined with robust re-ranking is optimal.

Query Action Effectiveness. To validate the active learning loop, we simulated a decision boundary with an ambiguity zone where model confidence drops to 0.5. We compared three policies: (1) Trust Model, (2) Passive Abstain (threshold 0.7), and (3) Active Query (query top-15% most uncertain).

- **Baseline Accuracy**: 93.4%
- **Passive Abstain**: 98.0% (10.9% abstention rate)
- **Active Query**: **99.2%** (15% query budget)

The Query action bridges the gap between passive safety and high performance, delivering a **6.2% improvement** in accuracy over the baseline by actively resolving ambiguity in the decision boundary region.

Dimensionality. We observed that lower projection dimensions ($d = 32, 64$) outperformed higher dimensions ($d = 128, 256$) in failure retrieval F1, consistent with the curse of dimensionality affecting distance-based thresholds. We select $d = 64$ as a balanced operating point.

6 Case Study: Healthcare Claims Processing

To validate ALFM-BEM in a realistic domain limit, we implemented a **proof-of-concept simulation** of a healthcare claims processing pipeline. While simplified compared to production adjudication systems (which may contain hundreds of rules), this case study captures the essential dynamics of latent rule learning. In this scenario, a payer (insurance company) enforces hidden adjudication rules, and the ALFM-BEM agent acts as a pre-submission scrubber.

6.1 Setup

We constructed a simulator generating medical claims with attributes: CPT code, diagnosis code, patient age, and modifiers. The payer enforces three hidden rules unknown to the agent:

1. **Medical Necessity**: CPT `99213` (Office Visit) requires specific respiratory diagnoses (e.g., `J01.90`).
2. **Age Limit**: CPT `90658` (Flu Shot) is rejected for patients under 3 years old.
3. **Modifier Requirement**: CPT `25111` (Ganglion Cyst Removal) requires a laterality modifier (`RT`/`LT`).

The agent projects these categorical features into the embedding space using a symbolic projection layer. It submits claims to the payer and receives binary feedback (Paid/Rejected). If the agent predicts high rejection risk (> 0.6), it abstains (simulating human review).

6.2 Results

We processed a stream of 2,000 claims. Figure 4 shows the learning curve.

Rejection Reduction. The rejection rate dropped from an initial 12.5% (random submission) to 1.5% in the final epoch—an **88% reduction**. The system effectively learned the latent rules from binary outcomes alone.

Failure Mode Analysis. Initially, the system encountered frequent rejections for all three hidden rules. By the end of the simulation, rejections for "Age Limit" and "Modifier

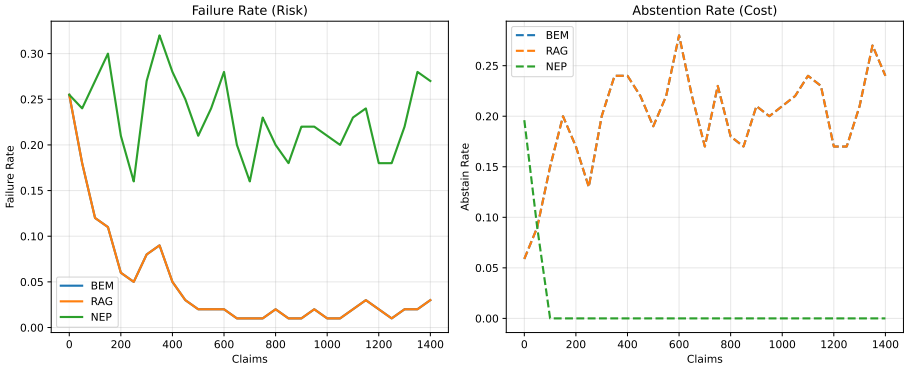

Figure 4: Learning curve for healthcare claims. The rejection rate (red) drops from 12.5% to 1.5% as BEM learns the latent payer rules. The human review rate (blue) stabilizes at $\approx 29\%$, correctly identifying the portion of the stream violating the hidden rules.

Requirement" were eliminated completely, and "Medical Necessity" rejections were reduced by 83%. The system learned to abstain from submitting invalid claims, protecting the deployment from costly denials.

7 Discussion

7.1 What ALFM-BEM Does and Does Not Address

ALFM-BEM addresses the three deployment gaps:

- **Gap 1 (Memory)**: BEM stores deployment experiences
- **Gap 2 (Calibration)**: Coverage signal detects OOD; Consensus Engine calibrates actions
- **Gap 3 (Continual Learning)**: Bounded adapters + experience replay

ALFM-BEM does **not** address:

- Fundamental generalization limitations of backbone architectures
- Acquiring genuinely new capabilities (vs. avoiding known failures)
- The "missing paradigm" for AGI that researchers have identified

ALFM-BEM is deployment infrastructure for frozen models, not a breakthrough toward AGI.

7.2 Limitations and Robustness

We discuss key assumptions, failure modes, and operational constraints.

Bounded interventions (whack-a-mole). Outcome-conditioned updates can shift behavior into a different failure basin: suppressing one failure mode may expose another. ALFM-BEM mitigates this by bounding interventions (adapter drift constraints), mediating behavior changes through the Consensus Engine, and preferring Query/Escalate when uncertainty is high rather than aggressive updates.

7.2.1 Outcome Signal Reliability

BEM assumes outcome signals $o \in [-1, 1]$ are available and meaningful. This assumption has several failure modes:

Delayed feedback. In many domains (e.g., healthcare claims, loan approvals), outcomes arrive days or weeks after inference. BEM handles this via asynchronous experience ingestion—experiences are timestamped at inference but outcomes are attached when available. The coverage signal remains valid immediately; risk/success signals update as outcomes arrive.

Noisy labels. User feedback may be incorrect (e.g., marking a correct response as wrong). We mitigate this via: (1) confirmation thresholds requiring multiple independent confirmations for high-severity labels (Section 3.5); (2) confidence decay that down-weights unconfirmed experiences over time; (3) cross-tenant consensus for global BEM updates.

Adversarial poisoning. Malicious users could submit false outcomes to corrupt BEM. Section 3.5 details our multi-layered defense: anomaly detection on submission velocity, quarantine for suspicious patterns, and administrative approval for high-impact updates.

Outcome ambiguity. Some interactions have no clear success/failure signal (e.g., open-ended conversations). BEM supports partial outcomes via the continuous spectrum; interactions without clear outcomes can be stored with $o \approx 0$ (neutral) and contribute to coverage without affecting risk/success signals.

7.2.2 Coverage Signal Failure Modes

The coverage signal detects OOD inputs as an emergent property of experience density. Known failure modes include:

False confidence (high coverage, wrong answer). Coverage measures *familiarity*, not *correctness*. An input similar to many past experiences will have high coverage even if the backbone produces a wrong answer. Mitigation: the success signal provides outcome-aware confidence. High coverage with low success signal triggers the Query action.

Projection collapse. If the contrastive projection layer maps semantically different inputs to similar embeddings, coverage becomes unreliable. We mitigate this via hard negative mining during projection training (Section 3): experiences with divergent outcomes must map to different regions.

Distribution shift. Gradual distribution shift can cause coverage to remain high even as the deployment distribution diverges from training. We recommend periodic coverage calibration: if mean coverage on a held-out sample drops below a threshold, the projection layer should be retrained.

Uniform OOD. Coverage via KDE excels at detecting *clustered* novel patterns (AUC ≈ 1.0) but has moderate performance on uniformly distributed OOD inputs (AUC ≈ 0.96).

For domains where uniform OOD is expected, we recommend combining coverage with energy-based OOD detection on the backbone.

7.2.3 BEM Growth and Lifecycle

Memory grows with deployment, requiring lifecycle management:

Storage economics. Each experience requires ≈256 bytes (64D float32 embedding + metadata). At 1M experiences, BEM requires ≈256MB storage. HNSW indexing adds ≈2× overhead. For cost-sensitive deployments, we recommend tiered storage: hot (recent, high-weight) experiences in memory, cold experiences on disk with lazy loading.

Memory pruning and bounds. Each tenant operates under a fixed memory budget (e.g., max N experiences or a storage quota). At saturation, the system deterministically retains the most useful experiences (high confirmation count, high severity, and recent) and prunes low-utility entries first. A time-to-live (TTL) policy can drop old, low-confirmation experiences while retaining strongly confirmed or high-severity failures indefinitely. Pruning runs during maintenance windows.

Multi-tenant cost allocation. Per-tenant BEM partitions enable usage-based billing. Each tenant's storage footprint is tracked; tenants exceeding quota receive Query actions prompting them to approve archival of old experiences.

Cold start. New deployments have empty BEM. We provide cold-start strategies: (1) synthetic seeding from the failure taxonomy (Appendix C); (2) transfer from similar tenants (with consent); (3) conservative Consensus Engine thresholds that default to Query until coverage builds.

7.2.4 Experimental Limitations

- Experiments use synthetic data in 64D; real-world validation with backbone embeddings is ongoing
- Healthcare case study uses a simplified simulator; production payer rules are more complex
- Projection layer requires retraining when backbone changes
- Query action effectiveness depends on user willingness to provide clarification

8 Conclusion

We introduced ALFM-BEM, a unified wrapper architecture for continuous learning in foundation model deployments. The key abstraction—Bidirectional Experience Memory—provides risk signals, success patterns, and OOD detection as properties of a single structure. The Query action transforms passive abstention into active learning. Bounded adapters enable stable continuous improvement.

While fundamental limitations of current architectures remain unsolved, ALFM-BEM provides practical infrastructure for deploying frozen models more safely and adaptively. The system learns from deployment outcomes without human labels, closing the gap between static training and dynamic deployment.

Broader Impact

ALFM-BEM is designed to improve the safety and reliability of foundation model deployments. Potential positive impacts include:

- Reduced harm from repeated failure patterns (the system learns from mistakes)
- Improved human oversight via Escalate and Query actions
- Per-tenant customization without cross-tenant data leakage

Potential negative impacts and mitigations:

- **False sense of security**: BEM cannot detect novel failure modes not similar to past experiences. Mitigation: the coverage signal flags low-coverage (potentially novel) inputs for human review.
- **Privacy concerns**: Experience memory stores deployment contexts. Mitigation: tenant isolation via scope-partitioning and access control; periodic vacuuming removes old experiences.
- **Gaming the system**: Adversaries could attempt to poison BEM with false experiences. Mitigation: outcome verification, anomaly detection on experience submissions, and rate limiting.

ALFM-BEM does not change backbone model capabilities; it provides deployment infrastructure. The fundamental limitations and risks of foundation models remain.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022. URL https://arxiv.org/abs/2212.08073.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning (ICML)*, pages 2206–2240. PMLR, 2022. URL https://arxiv.org/abs/2112.04426.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020. URL https://arxiv.org/abs/2005.14165.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*, pages 1597–1607. PMLR, 2020. URL https://arxiv.org/abs/2002.05709.

Chi-Keung Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970. doi: 10.1109/TIT.1970.1054406. URL https://doi.org/10.1109/TIT.1970.1054406.

Ran El-Yaniv and Yair Wiener. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11:1605–1641, 2010. URL https://jmlr.org/papers/v11/el-yaniv10a.html.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://arxiv.org/abs/1705.08500.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014. URL https://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. doi: 10.1038/nature20101. URL https://doi.org/10.1038/nature20101.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, pages 1321–1330. PMLR, 2017. URL https://arxiv.org/abs/1706.04599.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1610.02136.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning (ICML)*, pages 2790–2799. PMLR, 2019. URL https://arxiv.org/abs/1902.00751.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2106.09685.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL https://doi.org/10.1073/pnas.1611835114.

Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339, 1995.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474, 2020. URL https://arxiv.org/abs/2005.11401.

Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 21464–21475, 2020. URL https://arxiv.org/abs/2010.03759.

David Madras, Toniann Pitassi, and Richard Zemel. Predict responsibly: Improving fairness and accuracy by learning to defer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018. URL https://arxiv.org/abs/1711.06664.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021. URL https://arxiv.org/abs/2103.00020.

Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. *arXiv preprint arXiv:2310.10501*, 2023. URL https://arxiv.org/abs/2310.10501.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019. URL https://arxiv.org/abs/1908.10084.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Greg Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL https://arxiv.org/abs/1811.11682.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. URL https://arxiv.org/abs/1606.04671.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1410.3916.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2203.08913.

Appendix A. Algorithm Details

A.1 BEM Risk Computation

Algorithm 2 BEM Risk Signal Computation

Require: Query z, Memory $\mathcal{E}$, Threshold θ, Sensitivity α, Cap k

$\mathcal{R} \leftarrow \{e \in \mathcal{E} : \text{sim}(z, z_e) > \theta \wedge o_e < -0.3\}$
$\mathcal{R} \leftarrow \textsc{TopK}(\mathcal{R}, k \text{ by } \text{sim}(z, z_e))$
$s_{\text{risk}} \leftarrow 0$
for $e \in \mathcal{R}$ **do**
  $w \leftarrow \alpha \cdot \text{sim}(z, z_e) \cdot \frac{\text{sev}(e)}{5} \cdot \text{decay}(e) \cdot \text{conf}(e)$
  $s_{\text{risk}} \leftarrow 1 - (1 - s_{\text{risk}})(1 - w)$ ▷ Noisy-OR
end for
if $|\mathcal{R}| > 1$ **then**
  $\rho \leftarrow \max_{e_i, e_j \in \mathcal{R}} \text{sim}(z_{e_i}, z_{e_j})$
  $s_{\text{risk}} \leftarrow \min(1, s_{\text{risk}} + 0.1 \cdot \rho)$ ▷ Correlation boost
end if
return s_{risk}, $\mathcal{R}$

Appendix B. Experimental Details

B.1 Synthetic Data Generation

Failure modes are generated as unit vectors in $\mathbb{R}^{64}$:

$$\mu_k = \frac{v_k}{\|v_k\|}, \quad v_k \sim \mathcal{N}(0, I_{64}) \tag{11}$$

Failures are generated around modes with noise:

$$z_{\text{fail}} = \frac{\mu_k + \epsilon}{\|\mu_k + \epsilon\|}, \quad \epsilon \sim \mathcal{N}(0, 0.1^2 I) \tag{12}$$

Successes are generated uniformly:

$$z_{\text{success}} = \frac{v}{\|v\|}, \quad v \sim \mathcal{N}(0, I_{64}) \tag{13}$$

OOD samples are generated from a shifted distribution:

$$z_{\text{ood}} = \frac{v + \delta}{\|v + \delta\|}, \quad \delta \sim \mathcal{N}(0, 2^2 I) \tag{14}$$

Appendix C. Failure Taxonomy

We categorize foundation model failures into five canonical types for our cold start strategy and synthetic validation.

- **Hallucination:** Factual claims contradicting known knowledge bases or source documents.
- **Policy Violation:** Outputs violating domain-specific constraints (e.g., giving medical advice, using banned keywords).
- **Reasoning Error:** Logical inconsistencies, invalid inferences, or arithmetic mistakes.
- **Instruction Drift:** Outputs that ignore or misinterpret user intent or formatting constraints.
- **Unsafe Content:** Outputs triggering safety classifiers (e.g., toxicity, bias, PII leakage).

Appendix D. Projection Layer Details

The projection layer $\phi : \mathbb{R}^{d_{backbone}} \to \mathbb{R}^{d_{BEM}}$ is a standard MLP with one hidden layer:

$$\phi(x) = \text{LayerNorm}(W_2(\text{ReLU}(W_1(x) + b_1)) + b_2) \tag{15}$$

Training Procedure:

- **Loss function**: Triplet Margin Loss / Contrastive Loss.
- **Margin**: $\alpha = 1.0$.
- **Optimizer**: Adam, $lr = 10^{-3}$.
- **Negative Mining**: We construct batches with (Anchor, Positive, Negative) triplets where:
  - *Anchor*: A failure experience.
  - *Positive*: Another failure experience (simulating same mode).
  - *Negative*: A success experience (simulating distinct mode).

Cold Start: Initially, ϕ is initialized as a random projection. It is trained offline on a "seed set" of generic failure/success pairs from public datasets (e.g., toxicity datasets for safety failures) before specific deployment data is available.