MeridianDB: A Feature-Based Memory System for Continuous AI Learning

Ahmed R. Aldhafeeri
ar.aldhfaeeri11@gmail.com

Abstract

Modern AI systems face a critical challenge: **catastrophic forgetting**—the abrupt loss of previously learned information when acquiring new knowledge. This phenomenon stems from the fundamental **stability-plasticity dilemma**: systems require plasticity to integrate novel information while maintaining stability to preserve consolidated knowledge. Current Large Language Models (LLMs) operate as black boxes without transparent mechanisms for balancing these competing demands, making continuous learning impractical.

MeridianDB introduces a feature-based tabular memory architecture that transforms complex memory operations into efficient, transparent SQL queries while maintaining sophisticated memory consolidation mechanisms inspired by neuroscience. By modeling memory through explicit feature dimensions—semantic, temporal, contextual, and behavioral—rather than abstract graph relationships, MeridianDB achieves the stability-plasticity balance through mathematically rigorous scoring algorithms. Built on Cloudflare's edge infrastructure, the system enables continuous learning that improves agent performance while dramatically reducing computational costs and implementation complexity.

---

1. The Problem: Catastrophic Forgetting and the Stability-Plasticity Dilemma

1.1 Catastrophic Forgetting

Catastrophic forgetting occurs when neural systems abruptly overwrite past knowledge while learning new tasks. In production AI systems, this manifests as:

- Complete loss of previous capabilities during fine-tuning
- Inability to accumulate knowledge over time
- Expensive retraining cycles for each new domain
- AI systems that confidently hallucinate rather than admitting uncertainty

1.2 The Stability-Plasticity Dilemma

Effective memory systems require two opposing forces:

- **Plasticity**: Rapid integration of novel information
- **Stability**: Protection of consolidated knowledge from overwriting

Traditional approaches treat these as mutually exclusive, forcing designers to choose between adaptability and reliability. Current vector databases and RAG systems lack mechanisms to explicitly model and balance these dimensions.

---

2. The Solution: Feature-Based Agent Memory Architecture

2.1 Core Hypothesis

We propose that memory can be effectively modeled as **explicit, measurable feature dimensions** that capture the essential properties of knowledge episodes. This approach makes the stability-plasticity balance not just achievable, but **mathematically tunable** through standard database operations.

2.2 Four-Dimensional Memory Model

MeridianDB represents each memory through four complementary feature spaces:

2.2.1 Semantic Features

Vector embeddings capture the meaning of memory content in machine-understandable form. MeridianDB uses Cloudflare Vectorize for:

- Fast similarity search via cosine distance
- Over-fetching to enable multi-dimensional refinement
- Agent-specific memory isolation through metadata indexing

2.2.2 Temporal Features

Temporal dynamics model memory relevance over time through two key metrics:

Variables:

- f : Access frequency count ($f \geq 0$)
- t : Time elapsed since last access (hours)
- h : Configurable half-life parameter (hours)

- ω_t: Time weight coefficient ($0 \leq \omega_t \leq 1$)
- ω_f: Frequency weight coefficient $0 \leq \omega_f \leq 1$ where $\omega_f + \omega_t = 1$
- S_0: Base score constant (typically 100)

Recency Score Equation:

$$S = min(100, max(0, S_0 \omega_t e^{\frac{-t}{h}} + \omega_f \frac{log_{10}(f+1)}{4}))$$

Components:

- $e^{\frac{-t}{h}}$: Exponential decay with configurable half-life
- $log_{10}(f + 1)$: Logarithmic frequency boost preventing dominance by high-access memories
- ω_t, ω_f: Configurable weights enabling fine-tuned stability-plasticity balance

This algorithm enables:

- **Plasticity**: Recent memories maintain high scores
- **Stability**: Frequently accessed memories resist decay
- **Configurability**: Per-agent tuning via h, w_t, w_f parameters

Special Cases:

- Factual memories: Bypass temporal filtering ($S = 100$ always)
- Irrelevant memories: Automatic exclusion from retrieval

2.2.3 Contextual Features

Structured metadata captures situational applicability:

- Environment context (e.g., coding, research, conversation)
- Task type (e.g., problem-solving, learning, recall)
- Goal category
- User-specific attributes
- Custom developer-defined context

2.2.4 Behavioral Features

Performance tracking enables continuous learning through feedback loops.

Variables:

- s: Number of successful outcomes
- nf: Number of failed outcomes

- $n = s + nf$: Total trials
- z : Z-score for confidence level (1.96 for 95% confidence)

Wilson Score Confidence Interval (Lower Bound):

For n = 0, score is 0. Otherwise:

$$p = \frac{s}{n}$$

$$\omega = max(0, \frac{p + \frac{z^2}{2n} - z\sqrt{\frac{(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}})$$

Properties:

- Conservative scoring for low-sample memories
- Confidence-interval based ranking
- Resistant to manipulation from sparse data
- Enables zero-sum feedback: all memories in a retrieval batch share success/failure outcomes

---

3. System Architecture

3.1 Federated Database Design

MeridianDB employs a hybrid architecture leveraging Cloudflare's edge infrastructure:

Vectorize (Semantic Layer):

- Embedding storage with 768-dimensional vectors
- Fast cosine similarity search
- Agent-specific metadata indexing

D1 (Primary Storage):

- Feature tables storing temporal, contextual, and behavioral dimensions
- SQL-based composite scoring and filtering
- Atomic transactions for data consistency

Workers (Orchestrator layer):

- Enable unmatched performance and developer experience the possibility of the entire auto-RAG running near every single user as well having production grade RAG within minutes.
- Orchestrate SDK integration, authentication, and admin portal.

Schema:

```
CREATE TABLE memory_episodes (
  id TEXT PRIMARY KEY,
  agent_id TEXT NOT NULL,
  organization_id TEXT NOT NULL,
  content TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

  -- Semantic features
  semantic_cluster_id INTEGER,

  -- Temporal features
  recency_score REAL,
  access_frequency INTEGER DEFAULT 0,
  last_accessed_at TIMESTAMP,
  is_factual BOOLEAN DEFAULT FALSE,
  is_irrelevant BOOLEAN DEFAULT FALSE,

  -- Contextual features
  environment TEXT,
  task_type TEXT,
  goal_category TEXT,
  custom_context TEXT,

  -- Behavioral features
  success_count INTEGER DEFAULT 0,
  failure_count INTEGER DEFAULT 0,
  wilson_score REAL DEFAULT 0.0,

  -- Access control
  access_level TEXT DEFAULT 'private'
);
```

3.2 Event-Driven Updates

To maintain low-latency reads (<100ms), MeridianDB uses asynchronous feature updates:

Write Path:

1. User query → Semantic search (Vectorize)
2. Over-fetched results (topK is configurable) → SQL filtering (D1)
3. Access event → Queue message (Cloudflare queue)
4. Periodic processing → Batch feature updates

Queues:

- **Temporal Queue**: Batches access frequency updates
- **Behavioral Webhook**: Processes feedback signals and updates behavioral features of all memory involved in the RAG instance.

Benefits:

- No write-on-read latency penalty
- Efficient batch processing
- Configurable update frequency via cron triggers

3.3 Multi-Agent Memory Sharing

Simple yet powerful access control:

- Single-agent endpoint: Filters by `agent_id`
- Multi-agent endpoint: Omits agent filter, enables cross-agent learning
- Organization-level isolation: All queries filter by `organization_id`

---

4. Retrieval Algorithm

4.1 Multi-Dimensional Query Pipeline

1. Semantic Phase:
  - Embed query → Vectorize search
  - Over-fetch (e.g., top 100) by cosine similarity

2. Temporal Filtering:
  - Calculate recency scores using configured parameters
  - Filter where S ≥ stability_threshold
  - Exclude is_irrelevant = true
  - Always include is_factual = true

3. Behavioral Filtering:
   - Calculate Wilson scores
   - Filter where $\omega^{-} \geq$ success_rate_threshold

4. Contextual Matching:
   - Apply exact matches on environment, task_type, etc.
   - Boost scores for contextual alignment

5. Composite Scoring:
   - Combine semantic, temporal, and behavioral dimensions
   - Return top-k results

4.2 Configurable Stability-Plasticity Balance

Per-agent configuration enables fine-tuned control:

```
{
  // Temporal configuration
  halfLifeHours: 168,        // 7-day half-life (balanced)
  timeWeight: 0.6,           // Favor recency over frequency
  frequencyWeight: 0.4,
  stabilityThreshold: 0.15,  // Filter memories below 15% recency score

  // Behavioral configuration
  successRateThreshold: 0.5, // Require 50%+ Wilson score
  confidenceLevel: 0.95      // 95% confidence interval
}
```

Preset Profiles:

- **Balanced** (default): 7-day half-life, 60/40 time/frequency
- **Aggressive Decay**: 3-day half-life for conversational contexts
- **Long-Term Memory**: 30-day half-life for knowledge bases

---

5. Learning Phases

5.1 Passive Agent Learning Phase

Initial deployment with minimal filtering:

- `stabilityThreshold = 0.0`
- `successRateThreshold = 0.0`
- System accumulates interaction data
- Prevents false positives from sparse data in early usage of agent

5.2 Active Learning Phase

After sufficient data collection:

- Activate temporal filtering with appropriate `stabilityThreshold`
- Enable behavioral filtering with calibrated `successRateThreshold`
- Continuous improvement through feedback loops
- Memories naturally evolve based on utility

---

6. Key Innovations

6.1 Transparent Feature Engineering

Unlike black-box vector databases, MeridianDB makes memory properties explicit and queryable:

- SQL-based scoring enables debugging and optimization
- Feature weights are configurable, not learned
- System behavior is predictable and auditable for temporal and behavioral features.

6.2 Edge-Native Performance

Built on Cloudflare Workers:

- Global deployment with <100ms retrieval latency
- Automatic scaling and geographic distribution
- Cost-efficient serverless architecture

6.3 Eventual Consistency by Design

Acknowledges Vectorize's consistency model:

- Queue-based writes on retrieval to update memories temporal features ensure atomic dual-writes
- Read-optimized architecture tolerates slight lag
- R2-based queue snapshots for fault tolerance

6.4 Developer Experience

Simple three-method API:

- `store()`: Write memories with feature annotations
- `retrieve()`: Multi-dimensional search
- `recordFeedback()`: Log behavioral outcomes

---

7. Limitations and Trade-offs

7.1 Eventual Consistency

Writes propagate asynchronously; reads may lag slightly behind updates.

7.2 Manual Feature Engineering

Developers must supply contextual features (future versions may auto-generate).

7.3 Storage Constraints

D1 has a 10GB database limit (suitable for most agent applications).

7.4 Platform Coupling

Optimized for Cloudflare ecosystem; portability requires adaptation layer.

7.5 Learning Curve

Multi-dimensional retrieval paradigm differs from traditional vector search.

---

8. Conclusion

MeridianDB represents a paradigm shift from opaque vector stores to transparent, feature-based memory systems. By recognizing that effective memory can be modeled through explicit mathematical dimensions rather than abstract embeddings alone, we achieve:

1. **Measurable Stability-Plasticity Balance**: Tunable via configuration, not retraining
2. **Continuous Learning**: Agents improve through feedback without catastrophic forgetting
3. **Production-Ready Performance**: <100ms retrieval on edge infrastructure

4. **Operational Simplicity**: Standard SQL operations replace complex graph traversals

The future of AI memory lies not in increasingly complex architectures, but in **intelligently structured simplicity** that scales. MeridianDB crystallizes neuroscience-inspired principles into queryable, optimizable feature sets—making continuous learning both scientifically rigorous and pragmatically implementable.

For organizations building AI agents that must learn continuously while maintaining reliability, MeridianDB offers a mathematically grounded path to solving catastrophic forgetting through transparent, tunable memory systems.

References

Ebbinghaus, H. (1885/1913). *Memory: A Contribution to Experimental Psychology* (H. A. Ruger & C. E. Bussenius, Trans.). Teachers College, Columbia University.
— The foundational work on the forgetting curve; the half-life decay model for temporal weighting is inspired by these empirical findings.

Randazzo, G., et al. (2022). *Memory Models for Spaced Repetition Systems. arXiv preprint arXiv:2208.14693.*
— Discusses mathematical modeling of memory retention and adaptive frequency adjustments to prevent over-weighting of recent low-utility samples.
https://arxiv.org/abs/2208.14693

Wilson, E. B. (1927). "Probable Inference, the Law of Succession, and Statistical Inference." *Journal of the American Statistical Association, 22*(158), 209–212.
— Introduces the Wilson score interval, often used to derive lower-bound confidence estimates in behavioral scoring and sparse feedback systems.
https://doi.org/10.1080/01621459.1927.10502953

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). *Continual Lifelong Learning with Neural Networks: A Review. Neural Networks, 113*, 54–71.
— A comprehensive overview of lifelong and continual learning strategies in neural networks.
https://arxiv.org/pdf/1802.07569

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). *Distributed Representations of Words and Phrases and Their Compositionality. arXiv preprint arXiv:1310.4546.*
— Introduces word embeddings (word2vec) and compositional representations foundational to modern vector databases and semantic search.
https://arxiv.org/abs/1310.4546

Weaviate Team. (2023). *A Gentle Introduction to Vector Databases. Weaviate Documentation.* — Explains vector representations, embeddings, and similarity search fundamentals in the context of large-scale distributed systems.
https://weaviate.io/developers/weaviate