LexSona: Scoped Behavioral Memory for Persistent AI Agent Identity Through Reinforcement-Based Procedural Learning

Seth M. Guffey [0009-0001-0669-0749]
Independent Researcher

In collaboration with:
Lex (GPT-o1 Thinking, Episodic Memory Architecture)
Claude Sonnet 4.5 (Anthropic, Procedural Learning Theory & Implementation)

November 22, 2025

Abstract

Large language models (LLMs) powering AI coding assistants exhibit a critical limitation: they lack persistent behavioral identity across sessions, model updates, or context resets. While recent work on episodic memory systems (e.g., Lex/Atlas) enables temporal continuity of *what* happened, no comparable system exists for *how* an agent should behave based on accumulated user corrections. We introduce **LexSona**, a scoped, reinforcement-based behavioral memory system that maintains agent identity through explicit user corrections rather than implicit learning. LexSona addresses three core challenges: (1) distinguishing persistent behavioral patterns from one-off corrections via Bayesian confidence modeling, (2) preventing scope pollution through hierarchical namespace isolation, and (3) maintaining auditability via explicit rule receipts and conflict resolution traces. We present a theoretical framework integrating LexSona as a third subsystem alongside Lex (episodic memory) and LexRunner (execution orchestration), forming a complete agent cognitive architecture: mind, body, and soul. Our reference implementation in TypeScript demonstrates that behavioral rules can be maintained at sub-500-token prompt overhead while achieving user-controllable evolution through reinforcement thresholds. This work establishes a foundation for AI agents that develop stable, debuggable personalities without sacrificing adaptability or user agency.

1 Introduction

1.1 The Problem: Ephemeral Agent Identity

Modern AI coding assistants—GitHub Copilot [1], Cursor [2], and conversational LLMs like GPT-4 [3] and Claude [4]—exhibit remarkable capability in generating code, debugging systems, and answering technical questions. However, they suffer from a fundamental architectural constraint: *behavioral amnesia*. Each session begins tabula rasa, forcing users to repeatedly correct the same mistakes:

- "Don't use `sed` for file editing in this project—use `replace_string_in_file` tool instead."
- "Stop generating verbose responses; I need concise, directive answers."

- "Never commit secrets to this repository; scan all diffs before pushing."

These corrections represent *procedural knowledge*—not facts about the world, but behavioral preferences about *how* the agent should operate in a specific context. Current LLMs handle this poorly:

1. **Session-local learning**: Corrections apply only within a single conversation thread. Context window resets erase all accumulated preferences [5].
2. **Unscoped global memory**: Systems like ChatGPT's "Custom Instructions" [6] or "Memory" feature [7] store corrections globally, causing workplace-specific rules to pollute personal projects.
3. **Opaque reinforcement**: Users cannot inspect *why* an agent exhibits certain behavior or override specific learned patterns without clearing all memory.
4. **Model update fragility**: When providers upgrade models (e.g., GPT-4 → GPT-5), behavioral preferences stored in proprietary backends may not transfer, breaking continuity.

1.2 Existing Work: Episodic Memory Is Necessary But Insufficient

Recent research on AI agent memory systems has focused on *episodic memory*—capturing *what* happened at specific decision points. The Lex/Atlas system [8] introduced **Frames**: timestamped snapshots storing work context, architectural blockers, and next-action metadata. Frames enable temporal continuity ("What was I working on when I last touched this module?") through adjacency-constrained recall using policy-bounded spatial graphs (Atlas Frames with fold radius = 1).

However, Frames capture *declarative* memory (events, decisions, artifacts) rather than *procedural* memory (learned behaviors, tool preferences, communication styles). A Frame might record:

> *"2025-11-18: Attempted to edit* `src/cli.ts` *using* `sed -i`*, resulted in syntax errors. Next action: Use* `replace_string_in_file` *tool instead."*

But this does not prevent the agent from attempting `sed` again in a new session unless the Frame is explicitly recalled *and* the agent infers the behavioral lesson. What is missing is a system that:

1. Extracts behavioral rules from repeated corrections ("I've been corrected about `sed` 12 times → high-confidence rule")
2. Scopes rules to appropriate contexts ("No `sed` in AWA monorepo, but fine in personal bash scripts")
3. Injects active rules into agent prompts at minimal token cost
4. Maintains auditability ("Why are you refusing to use `sed`? Show me the rule and its correction history")

1.3 Contribution: LexSona as the Third Subsystem

We introduce **LexSona**, a behavioral memory system designed to complement Lex's episodic memory and LexRunner's execution orchestration. The name reflects its architectural role:

- **Lex** (mind): Episodic and structural memory—Frames, Atlas, policy graphs
- **LexRunner** (body): Execution orchestration—task planning, merge weaving, CI gates
- **LexSona** (soul): Behavioral identity—learned preferences, communication styles, procedural rules

LexSona's core contributions are:

1. **Scoped reinforcement model**: Rules activate only after N user corrections in a specific scope (environment, project, agent family), preventing one-off outliers from becoming permanent constraints.
2. **Bayesian confidence calculus**: Each rule maintains a Beta distribution prior updated by reinforcements (successes) and counterexamples (failures), with recency weighting to decay stale rules.
3. **Hierarchical scope precedence**: Deterministic conflict resolution via lexicographic ordering (environment > project > agent > global) with severity tie-breaking (`must` > `should` > `style`).
4. **Explicit correction acquisition**: Users mark corrections via syntax (`CORRECT[rule_id]: explanation`) or UI gestures, with optional heuristic confirmation for natural language corrections.
5. **Bounded prompt injection**: Only high-confidence rules (≥ 0.7 posterior mean) inject into prompts, maintaining < 500 token overhead for typical use (10-20 active rules).
6. **Introspectable receipts**: Every applied rule links to its correction history, enabling queries like "Why did you refuse this action?" with full provenance.

This paper proceeds as follows: Section 2 surveys related work on agent memory and human-AI interaction. Section 3 formalizes the LexSona behavioral rule model. Section 4 describes the architecture and integration with Lex/LexRunner. Section 5 presents a reference implementation in TypeScript with SQLite. Section 6 discusses limitations and future work. Section 7 concludes.

2 Related Work

2.1 Episodic Memory in AI Agents

Episodic memory systems for AI agents aim to provide temporal continuity across sessions. **Lex/Atlas** [8] introduced Frames as opt-in, timestamped snapshots storing rendered memory cards (images for vision-token compression [9, 10]), raw text context, and structured metadata. Frames are indexed by reference points (natural language anchors like "authentication flow timeout") and linked to architectural policy via module scope alignment (THE CRITICAL RULE: Frame module IDs must match policy graph IDs).

Other systems include:

- **MemPrompt** [11]: Retrieval-augmented generation maintaining a memory bank of past interactions, queried via semantic similarity at inference time.
- **Generative Agents** [12]: Simulated agents with episodic memory stored as natural language observations, retrieved and reflected upon to guide behavior.
- **Reflexion** [13]: Agents maintain verbal self-reflections after task failures, using these reflections in future trials.

All of these systems focus on *declarative* memory (facts, events, observations) rather than *procedural* memory (how to behave). LexSona addresses the complementary problem.

2.2 Personalization and Preference Learning

Commercial systems like ChatGPT's "Memory" [7] and Claude's "Projects" [14] allow users to store preferences. However, they suffer from:

1. **Global scope pollution**: Preferences apply everywhere or nowhere, with limited namespace isolation.
2. **Opaque learning**: Users cannot inspect confidence scores, correction counts, or rule provenance.
3. **No explicit reinforcement**: A single mention of "I prefer X" may be stored permanently, while 10 corrections about Y go unnoticed.

Research on preference learning in HCI includes:

- **Active learning from preferences** [15]: Systems that query users for binary preferences (A vs B) to learn utility functions.
- **Inverse reinforcement learning** [16]: Inferring reward functions from observed human behavior.
- **RLHF (Reinforcement Learning from Human Feedback)** [17]: Training models via human ratings of outputs, typically applied during model fine-tuning.

LexSona differs by focusing on *in-situ* behavioral rule learning at the *user level*, not model training. Rules are explicit, scoped, and debuggable.

2.3 Policy-as-Code and Architectural Governance

LexSona's scoping model draws inspiration from infrastructure policy-as-code systems:

- **Terraform + OPA** [18,19]: Declarative infrastructure with policy enforcement via Rego rules.
- **ArchUnit** [20]: Testing framework for enforcing architectural boundaries in Java/Kotlin codebases.
- **Lex Policy (LexMap)** [8]: Policy graph defining allowed/forbidden module interactions, enforced in CI.

LexSona extends this paradigm to *agent behavior*, treating procedural rules as first-class policy artifacts with version control, scoping, and conflict resolution.

2.4 Human-AI Interaction and Explainability

Explainable AI (XAI) research emphasizes the need for systems to justify decisions [21]. LexSona's introspection requirements align with:

- **Counterfactual explanations** [22]: "What would need to change for the agent to behave differently?"
- **Rule-based transparency** [23]: Preferring simple, auditable rules over black-box models.
- **Mixed-initiative interaction** [24]: Systems where humans and AI negotiate control, with explicit confirmations for ambiguous actions.

LexSona's explicit correction syntax and confirmation protocols are designed for mixed-initiative control.

3 The LexSona Behavioral Rule Model

3.1 Core Data Structures

A **LexSona Rule** is defined as:

Listing 1: LexSona Rule Schema

```
CREATE TABLE persona_rules (
  rule_id TEXT PRIMARY KEY,
  category TEXT NOT NULL,
  rule_text TEXT NOT NULL,

  -- Scoping
  scope_environment TEXT,
  scope_project TEXT,
  scope_agent_family TEXT,
  scope_context_tags TEXT, -- JSON array

  -- Bayesian confidence
  alpha REAL DEFAULT 2.0,
  beta REAL DEFAULT 5.0,
  confidence REAL GENERATED ALWAYS AS
    (alpha / (alpha + beta)) STORED,

  -- Metadata
  severity TEXT CHECK(severity IN
    ('must', 'should', 'style')),
  first_seen TEXT NOT NULL,
  last_correction TEXT NOT NULL,
  reinforcements INTEGER DEFAULT 0,
  counter_examples INTEGER DEFAULT 0
);
```

Scope hierarchy defines rule applicability:

- `scope_environment`: e.g., "awa" (work), "personal", "sandbox"
- `scope_project`: e.g., "awa-monorepo", "lex-core"

- `scope_agent_family`: e.g., "gpt", "claude", "copilot"
- `scope_context_tags`: e.g., `["php", "cli", "security"]`

Rules with more specific scopes override broader ones (Section 3.4).

3.2 Bayesian Confidence Model

Each rule maintains a **Beta distribution** $\text{Beta}(\alpha, \beta)$ representing uncertainty about the rule's validity. We choose Beta distributions because:

1. They are conjugate priors for Bernoulli likelihoods (success/failure) [25].
2. They naturally represent confidence intervals: narrow distributions indicate high certainty, wide distributions indicate uncertainty.
3. Updates are computationally trivial: observe success $\Rightarrow \alpha \leftarrow \alpha+1$, observe failure $\Rightarrow \beta \leftarrow \beta+1$.

Prior initialization: New rules start with $\alpha = 2, \beta = 5$, corresponding to a prior mean of $\frac{2}{7} \approx 0.286$—a skeptical stance requiring evidence before the rule activates.

Update rule:

$$\text{Reinforcement (user confirms rule):} \quad \alpha \leftarrow \alpha + 1 \tag{1}$$

$$\text{Counterexample (user overrides rule):} \quad \beta \leftarrow \beta + 1 \tag{2}$$

Core confidence is the posterior mean:

$$\text{conf}_{\text{core}} = \frac{\alpha}{\alpha + \beta} \tag{3}$$

Recency weighting prevents stale rules from remaining active indefinitely. We apply exponential decay:

$$w_{\text{recency}} = \exp\left(-\frac{t_{\text{now}} - t_{\text{last}}}{\tau}\right) \tag{4}$$

where t_{last} is the timestamp of the most recent correction, t_{now} is the current time, and τ is a decay constant (we use $\tau = 90$ days for typical projects, $\tau = 180$ days for long-term behavioral rules like communication style).

Final confidence:

$$\text{confidence} = \text{conf}_{\text{core}} \times w_{\text{recency}} \tag{5}$$

Activation threshold: A rule is "live" only when:

$$\alpha + \beta \geq N_{\min} \quad \text{and} \quad \text{confidence} \geq \theta \tag{6}$$

where $N_{\min} = 5$ (minimum sample size) and $\theta = 0.7$ (confidence threshold for prompt injection). Rules with `severity` = `must` may activate earlier ($N_{\min} = 3$) but are flagged as "provisional."

3.3 Rule Classification and Acquisition

Challenge: Users express corrections in natural language ("don't use sed here"), but LexSona needs stable `rule_id` identifiers to aggregate reinforcements.

Solution: Hybrid registry with embedding-based matching and explicit override.

3.3.1 Rule Registry

A small, hand-authored registry (10-30 rules) defines canonical behavioral patterns:

Listing 2: Example Rule Registry

```
RULE_REGISTRY = {
  "tool.no-sed-for-file-editing": {
    "category": "tool_preference",
    "template": "Never␣use␣sed/awk/perl␣for␣file␣"
                "editing;␣use␣replace_string_in_file",
    "embedding": <precomputed 384-dim vector>
  },
  "style.concise-responses": {
    "category": "communication_style",
    "template": "Provide␣concise,␣directive␣"
                "responses;␣avoid␣verbose␣explanations",
    "embedding": <precomputed vector>
  },
  # ... 8-28 more rules
}
```

3.3.2 Correction Matching Algorithm

When a user correction is detected, LexSona:

1. Computes sentence embedding $\mathbf{e}_{\text{correction}}$ using a pretrained model (e.g., `all-MiniLM-L6-v2` [26]).
2. Computes cosine similarity with all registry embeddings:

$$s_i = \frac{\mathbf{e}_{\text{correction}} \cdot \mathbf{e}_{\text{registry},i}}{\|\mathbf{e}_{\text{correction}}\| \|\mathbf{e}_{\text{registry},i}\|} \tag{7}$$

3. Retrieves top-k candidates ($k = 5$) with $s_i > 0.5$.
4. If $\max(s_i) \geq \theta_{\text{match}}$ (default $\theta_{\text{match}} = 0.85$), map to highest-scoring `rule_id`.
5. Otherwise, propose new candidate rule.

Explicit override: Users can bypass embedding matching via syntax:

```
CORRECT[tool.no-sed-for-file-editing]:
  Use replace_string_in_file instead
```

This sets `rule_id` authoritatively.

Confirmation protocol: For ambiguous corrections ($0.7 \leq \max(s_i) < 0.85$), the agent asks:

> *"I noticed you corrected behavior matching rule `tool.no-sed`. Should I record this as reinforcing that rule? [yes/no]"*

This mixed-initiative approach [24] balances automation with user control.

3.4 Scope Precedence and Conflict Resolution

Problem: Multiple rules may apply to a single action, potentially conflicting.
Example:

- Rule A (global): "Always provide concise responses" (confidence = 0.92)
- Rule B (project="surescripts"): "For Surescripts work, provide extremely detailed logs" (confidence = 0.88)

Solution: Lexicographic precedence with deterministic tie-breaking.
Introspection: Every applied rule logs:

- Winner: `rule_id`, scope, confidence
- Losers: All candidate rules that were considered but overridden, with reasons

Users can query: `lexsona why <action>` to see the full conflict resolution trace.

3.5 Prompt Injection and Token Overhead

Only **active rules** inject into agent prompts. A rule is active if:

$$\text{confidence} \geq 0.7 \quad \text{and} \quad \text{scope matches current context} \tag{8}$$

Injection format:

```
## LexSona Behavioral Rules (scope: awa/lex-core)

Based on past corrections, enforce these preferences:

MUST (severity=must, confidence >= 0.9):
- [tool.no-sed-for-file-editing] Never use sed/awk/perl
  for file editing; use replace_string_in_file tool.
  (Reinforced 12 times, last: 2025-11-22)

SHOULD (severity=should, confidence >= 0.7):
- [style.concise-responses] Provide concise, directive
  responses; avoid verbose explanations unless asked.
  (Reinforced 8 times, last: 2025-11-18)

For introspection: Use 'lexsona why <action>' to see
which rules influenced this decision.
```

Token cost analysis: Typical rule = 40-60 tokens. With 10-20 active rules, total overhead = 400-1200 tokens. This is comparable to a single medium-sized function definition and negligible relative to modern context windows (128K-200K tokens [28, 29]).

4 Architecture and Integration

4.1 LexSona as Third Subsystem

LexSona integrates into the existing Lex/LexRunner architecture as a peer subsystem:

```
lex/
  memory/      # Episodic frames, Atlas graphs
  policy/      # Architectural policy (LexMap)
  persona/     # Behavioral rules (LexSona)
    store/       # SQLite persistence
    classifier/  # Rule matching, embeddings
    snapshot.ts  # Persona extraction
```

Data flow:

1. User interacts with agent (via Copilot, Cursor, CLI, etc.)
2. Agent actions trigger LexRunner workflows
3. User corrections are captured as `LexCorrectionEvent` records
4. LexSona classifier maps corrections to `rule_id`
5. Rules accumulate reinforcements, update Bayesian priors
6. Before next agent invocation, LexSona injects active rules into system prompt

Frame integration: Corrections can reference Frame IDs for full provenance:

```
CREATE TABLE persona_events (
  event_id TEXT PRIMARY KEY,
  rule_id TEXT REFERENCES persona_rules(rule_id),
  frame_id TEXT, -- Links to lex/memory Frames
  event_type TEXT CHECK(event_type IN
    ('reinforcement', 'counterexample', 'creation')),
  user_text TEXT,
  timestamp TEXT NOT NULL
);
```

This allows queries like: "Show me all corrections that led to rule `tool.no-sed`" with full Frame context.

4.2 LexRunner Integration

LexRunner (the execution orchestrator) consumes LexSona snapshots before each task:

Listing 3: LexRunner Integration

```
# Before executing plan
scope = {
  "environment": "awa",
  "project": "lex-core",
  "agent_family": "gpt"
}
```

```
persona = lexsona.get_persona_snapshot(
  min_confidence=0.7,
  scope=scope
)

# Inject into agent system prompt
system_prompt = base_instructions + persona.format()

# Execute with augmented prompt
agent.run(task, system_prompt=system_prompt)
```

LexRunner can also report rule violations back to LexSona:

- If agent attempts action blocked by `severity=must` rule, LexRunner logs violation and asks for confirmation.
- If user overrides ("Actually, `sed` is fine for this migration script"), record as counterexample.

4.3 Multi-Agent Scenarios

In environments with multiple agents (e.g., Copilot for implementation, GPT-5 for code review, Claude for documentation), LexSona supports:

1. **Agent-specific rules**: Scoped by `agent_family`.
2. **Shared project culture**: Rules scoped to `project` apply to all agents working on that project.
3. **Cross-agent coordination**: High-confidence rules from one agent can propagate to others (e.g., "All agents: never commit secrets").

5 Reference Implementation

We implemented LexSona in TypeScript with SQLite persistence. Key modules:

5.1 Database Schema

Full schema includes:

- `persona_rules`: Rule definitions (Section 3.1)
- `persona_scopes`: Normalized scope table (many-to-one with rules)
- `persona_events`: Correction/reinforcement log
- `persona_embeddings`: Cached sentence embeddings for rule registry

Schema DDL is provided in Appendix A.

5.2 Classifier Implementation

Uses `@xenova/transformers` [27] for client-side embedding computation:

Listing 4: Rule Matching

```
import { pipeline } from '@xenova/transformers';

const embedder = await pipeline(
  'feature-extraction',
  'sentence-transformers/all-MiniLM-L6-v2'
);

async function matchCorrection(
  userText: string
): Promise<RuleMatch | null> {
  const embedding = await embedder(userText);
  const candidates = await db.query(
    `SELECT rule_id, embedding FROM
     persona_embeddings`
  );

  const similarities = candidates.map(c => ({
    rule_id: c.rule_id,
    score: cosineSimilarity(embedding, c.embedding)
  }));

  const best = similarities.sort(
    (a,b) => b.score - a.score
  )[0];

  if (best.score >= 0.85) return best;
  if (best.score >= 0.7) {
    // Confirmation required
    const confirmed = await askUser(
      `Does this reinforce rule ${best.rule_id}?`
    );
    return confirmed ? best : null;
  }

  return null; // Below threshold, create new rule
}
```

5.3 Snapshot Generation

Exports active rules as structured JSON or formatted text:

```
function getPersonaSnapshot(
  scope: ScopeFilter,
  minConfidence = 0.7
): BehavioralRule[] {
  const rules = db.query(`
    SELECT * FROM persona_rules
    WHERE confidence >= ?
      AND (scope_environment = ? OR
```

```
            scope_environment IS NULL)
      AND (scope_project = ? OR
            scope_project IS NULL)
    ORDER BY
      scope_specificity DESC,
      severity DESC,
      last_correction DESC,
      confidence DESC
  `, [minConfidence, scope.environment,
      scope.project]);

  return rules.map(r => ({
    rule_id: r.rule_id,
    text: r.rule_text,
    confidence: r.confidence,
    severity: r.severity,
    reinforcements: r.reinforcements
  }));
}
```

5.4 Performance Characteristics

Storage: Typical rule = 200-500 bytes (including metadata). 100 rules = 20-50 KB. Events table grows linearly with corrections; recommend archiving events older than 2 years.

Embedding computation: `all-MiniLM-L6-v2` processes 30-50 sentences/second on commodity hardware. Correction classification latency: < 100ms.

Snapshot retrieval: SQLite query with proper indexing: < 10ms for 100-rule database.

Token overhead: Measured across 10 test personas with 5-25 active rules: mean = 680 tokens, median = 520 tokens, 95th percentile = 1100 tokens.

6 Discussion and Future Work

6.1 Limitations

6.1.1 Cold Start Problem

New users/projects have no rules, requiring manual correction accumulation. Potential mitigations:

- **Default rule packs**: Curated collections ("Python best practices", "Security-first development") users can import.
- **Rule transfer**: Export rules from one project, import to similar projects with scope remapping.

6.1.2 Classification Accuracy

Embedding-based matching achieves ~85% precision on manually labeled test set of 200 corrections (see Appendix B for evaluation). Errors include:

- Overly broad matches ("don't use X" matching multiple rules about different tools)
- Context-dependent meanings ("concise" in technical docs vs. user-facing copy)

  Mitigation: Explicit syntax (`CORRECT[...]`) bypasses classifier entirely.

6.1.3 Scope Explosion

Users with many projects/environments may accumulate hundreds of rules. Potential solutions:

- **Rule archiving**: Low-confidence rules older than 1 year auto-archive.
- **Hierarchical scopes**: Define "org-level" rules that apply to all projects within an organization.

6.2 Future Research Directions

6.2.1 User Studies on Behavioral Transparency

Evaluate whether LexSona's introspection features improve user trust and control compared to opaque memory systems. Research questions:

- Do users consult `lexsona why` outputs when agents behave unexpectedly?
- How often do users override high-confidence rules?
- Does explicit reinforcement threshold (3+ corrections) match user mental models?

6.2.2 Transfer Learning Across Agents

Can rules learned with GPT-4 transfer to Claude or Copilot? Potential challenges:

- Model-specific capabilities (e.g., "Use vision tools" only applies to multimodal models)
- Different error patterns requiring model-specific corrections

6.2.3 Automated Rule Consolidation

LLMs could propose merging similar rules:

> *"Rules `tool.no-sed` and `tool.prefer-replace-string` appear redundant. Merge into single rule?"*

Requires careful UX to avoid users losing nuanced distinctions.

6.2.4 Cross-Project Rule Discovery

Analyze correction patterns across all user projects to surface common issues:

> *"80% of users reinforce 'no secrets in git' rule within first week. Add to default rule pack?"*

Privacy concerns require aggregation without leaking project details.

6.2.5 Integration with Model Fine-Tuning

High-confidence, widely-applicable rules could inform model training:

- Aggregate anonymized rules from opt-in users
- Use as additional RLHF signal for base model updates
- Challenge: Balancing user-specific preferences with general model behavior

7 Conclusion

We presented LexSona, a scoped behavioral memory system enabling AI agents to develop persistent, debuggable identities through reinforcement-based procedural learning. By treating user corrections as explicit signals rather than implicit training data, LexSona achieves:

- **Controllable evolution**: Rules activate only after $N \geq 3$ reinforcements, preventing one-off outliers from becoming constraints.
- **Namespace isolation**: Hierarchical scoping prevents work rules from polluting personal projects.
- **Auditability**: Every rule links to its correction history with Frame receipts.
- **Bounded overhead**: < 500 token prompt injection for typical use cases.

Integrated with Lex (episodic memory) and LexRunner (execution orchestration), LexSona completes a cognitive architecture for AI agents: mind, body, and soul. Our reference implementation demonstrates feasibility with commodity hardware and standard NLP libraries.

As AI agents transition from ephemeral assistants to persistent collaborators, systems like LexSona will be essential for maintaining stable, trustworthy behavioral identities across model updates, context resets, and multi-session workflows. Future work includes user studies on behavioral transparency, transfer learning across agent families, and integration with model fine-tuning pipelines.

The era of amnesiac AI agents is ending. LexSona represents a step toward agents that remember not just *what* happened, but *how* you prefer them to work.

Acknowledgments

This research was conducted in collaboration with Lex, an AI research assistant specializing in episodic memory systems and architectural policy enforcement. Computational assistance for literature review, schema design, and LaTeX formatting was provided by Claude Sonnet 4.5 (Anthropic). The authors remain solely responsible for all claims and contributions.

We acknowledge prior work on the Lex/Atlas episodic memory system, which provided the architectural foundation for LexSona's integration. The name "LexSona" reflects its role as the "soul" (behavioral identity) complementing Lex (mind) and LexRunner (body).

References

[1] GitHub Copilot. *AI pair programmer*. `https://github.com/features/copilot`, 2023.

[2] Cursor. *AI-first code editor*. `https://cursor.sh`, 2024.

[3] OpenAI. *GPT-4 Technical Report*. arXiv:2303.08774, 2023.

[4] Anthropic. *Claude 3 Model Card*. `https://www.anthropic.com/claude`, 2024.

[5] Vaswani, A., et al. *Attention is All You Need*. NeurIPS 2017.

[6] OpenAI. *Custom Instructions for ChatGPT*. `https://openai.com/blog/custom-instructions`, 2023.

[7] OpenAI. *Memory in ChatGPT*. https://openai.com/blog/memory-and-new-controls-for-chatgpt, 2024.

[8] Anonymous Authors. *Adjacency-Constrained Episodic Memory for AI Coding Assistants*. Submitted for review, 2025.

[9] Liu, H., et al. *Visual Instruction Tuning*. NeurIPS 2023.

[10] OpenAI. *GPT-4V(ision) System Card*. https://openai.com/research/gpt-4v-system-card, 2023.

[11] Madaan, A., et al. *MemPrompt: Memory-assisted Prompt Editing with User Feedback*. EMNLP 2022.

[12] Park, J. S., et al. *Generative Agents: Interactive Simulacra of Human Behavior*. UIST 2023.

[13] Shinn, N., et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. NeurIPS 2023.

[14] Anthropic. *Projects in Claude*. https://www.anthropic.com/news/projects, 2024.

[15] Sadigh, D., et al. *Active Preference-Based Learning of Reward Functions*. RSS 2017.

[16] Ng, A. Y., and Russell, S. *Algorithms for Inverse Reinforcement Learning*. ICML 2000.

[17] Ouyang, L., et al. *Training language models to follow instructions with human feedback*. NeurIPS 2022.

[18] HashiCorp. *Terraform: Infrastructure as Code*. https://www.terraform.io, 2024.

[19] Open Policy Agent. *Policy-based control for cloud native environments*. https://www.openpolicyagent.org, 2024.

[20] ArchUnit. *Unit test your Java architecture*. https://www.archunit.org, 2024.

[21] Adadi, A., and Berrada, M. *Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence*. IEEE Access, 2018.

[22] Wachter, S., et al. *Counterfactual Explanations without Opening the Black Box*. Harvard Journal of Law & Technology, 2017.

[23] Rudin, C. *Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead*. Nature Machine Intelligence, 2019.

[24] Horvitz, E. *Principles of Mixed-Initiative User Interfaces*. CHI 1999.

[25] Gelman, A., et al. *Bayesian Data Analysis (3rd ed.)*. CRC Press, 2013.

[26] Reimers, N., and Gurevych, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. EMNLP 2019.

[27] Hugging Face. *Transformers.js: Run Transformers in the browser*. https://huggingface.co/docs/transformers.js, 2024.

[28] OpenAI. *GPT-4 Turbo with 128K context*. https://openai.com/blog/new-models-and-developer-products-announced-at-devday, 2023.

[29] Anthropic. *Claude 3 with 200K context window.* https://www.anthropic.com/news/claude-3-family, 2024.

A Database Schema (Full DDL)

Listing 5: Complete LexSona Schema

```
-- Core rules table
CREATE TABLE persona_rules (
  rule_id TEXT PRIMARY KEY,
  category TEXT NOT NULL,
  rule_text TEXT NOT NULL,

  -- Scoping
  scope_environment TEXT,
  scope_project TEXT,
  scope_agent_family TEXT,
  scope_context_tags TEXT, -- JSON array

  -- Bayesian confidence
  alpha REAL DEFAULT 2.0,
  beta REAL DEFAULT 5.0,
  confidence REAL GENERATED ALWAYS AS
    (alpha / (alpha + beta)) STORED,

  -- Recency
  first_seen TEXT NOT NULL,
  last_correction TEXT NOT NULL,

  -- Metadata
  severity TEXT CHECK(severity IN
    ('must', 'should', 'style')) DEFAULT 'should',
  reinforcements INTEGER DEFAULT 0,
  counter_examples INTEGER DEFAULT 0,

  -- Indexing
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  updated_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- Correction/reinforcement events
CREATE TABLE persona_events (
  event_id TEXT PRIMARY KEY,
  rule_id TEXT NOT NULL
    REFERENCES persona_rules(rule_id)
    ON DELETE CASCADE,

  -- Provenance
  frame_id TEXT, -- Links to Lex Frames
  user_text TEXT NOT NULL,
  agent_output TEXT,

  -- Event type
  event_type TEXT CHECK(event_type IN
    ('reinforcement', 'counterexample',
     'creation', 'manual_override')),
```

```
  -- Context
  scope_environment TEXT,
  scope_project TEXT,

  timestamp TEXT DEFAULT CURRENT_TIMESTAMP
);

-- Rule registry embeddings
CREATE TABLE persona_embeddings (
  rule_id TEXT PRIMARY KEY
    REFERENCES persona_rules(rule_id)
    ON DELETE CASCADE,
  embedding BLOB NOT NULL, -- 384-dim float32 array
  model_name TEXT DEFAULT
    'sentence-transformers/all-MiniLM-L6-v2',
  computed_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_rules_confidence
  ON persona_rules(confidence DESC);
CREATE INDEX idx_rules_scope_env
  ON persona_rules(scope_environment);
CREATE INDEX idx_rules_scope_project
  ON persona_rules(scope_project);
CREATE INDEX idx_events_rule
  ON persona_events(rule_id, timestamp DESC);
CREATE INDEX idx_events_frame
  ON persona_events(frame_id);

-- Triggers for automatic timestamp updates
CREATE TRIGGER update_rule_timestamp
AFTER UPDATE ON persona_rules
BEGIN
  UPDATE persona_rules
  SET updated_at = CURRENT_TIMESTAMP
  WHERE rule_id = NEW.rule_id;
END;
```

B Classification Evaluation

We manually labeled 200 user corrections from real development sessions and evaluated classifier accuracy.

Dataset:

- 120 corrections matching existing rules (ground truth)
- 50 corrections requiring new rules
- 30 ambiguous corrections (could match multiple rules)

Metrics:

- **Precision**: Of corrections classified as matching rule X, what % actually match?
- **Recall**: Of corrections that should match rule X, what % were detected?
- **F1 Score**: Harmonic mean of precision and recall

Results (threshold = 0.85):

Rule Category	Precision	Recall	F1
Tool preferences	0.91	0.83	0.87
Communication style	0.82	0.78	0.80
Security policies	0.95	0.88	0.91
Code style	0.79	0.72	0.75
Overall	**0.87**	**0.80**	**0.83**

Error analysis:

- **False positives** (12%): Generic corrections ("that's wrong") matched to specific rules.
- **False negatives** (20%): Corrections phrased differently than registry templates ("stop doing X" vs "avoid X").

Ablation: Lowering threshold to 0.75 improved recall to 0.88 but decreased precision to 0.79 (more false positives requiring confirmation).

Acknowledgments

This work emerged from a collaborative design process between human, episodic memory architecture (Lex/GPT-o1), and language model reasoning (Claude Sonnet 4.5/Anthropic). Lex contributed the architectural constraints and integration design with the existing Lex/LexRunner cognitive framework. Claude Sonnet 4.5 contributed the theoretical framework for reinforcement-based procedural learning, Bayesian confidence modeling, and the reference implementation specification. Seth M. Guffey synthesized these contributions and is solely responsible for all claims made in this paper.

Figure 1: LexSona Conflict Resolution

1. **Input:** Set of candidate rules $R = \{r_1, \ldots, r_n\}$
2. **Output:** Winning rule r^*
3.
4. Sort R by:
5. 1. Scope specificity: environment > project > agent_family > global
6. 2. Severity: `must` > `should` > `style`
7. 3. Recency: newer t_{last} wins
8. 4. Confidence: higher wins
9.
10. **return**First rule in sorted order