Adjacency-Constrained Episodic Memory for Software Workflows

A Local-First Framework for Policy-Grounded AI Continuity

Joseph Gustavson[*]

Independent Researcher

Acknowledgment: GPT-5 Thinking is credited for assistance but not listed as an accountable co-author, consistent with publication norms requiring human responsibility for claims. [?, ?]

November 1, 2025

Abstract

Mainstream AI coding assistants behave like short-term collaborators: they respond to prompts, summarize diffs, or suggest code, but they cannot reliably resume multi-day engineering work without the user re-explaining what they were doing, why it mattered, and what remains blocked. [?, ?] Separately, most production codebases operate under undocumented or tribal architectural rules that are enforced informally ("ask the senior dev if this import is allowed"). Tools like linters and static analyzers can flag localized violations, [?, ?, ?, ?] but they rarely capture system-wide architectural boundaries, feature gating, and permission requirements as an auditable contract.

This paper introduces **Lex**, a unified system that addresses both problems simultaneously.

Lex provides *episodic work memory* through explicit "Frames": timestamped, opt-in snapshots of an engineer's working state at a meaningful moment. Each Frame stores (1) a rendered "memory card" image, (2) the raw high-signal text behind that image, and (3) structured metadata including branch, blockers, next action, feature flags, and `module_scope`. The rendered image exists because modern multimodal models can consume high-density panels of text as vision input at dramatically lower token cost than equivalent raw text, with reported effective compression on the order of 7–20× for long-context reasoning tasks. [?, ?, ?]

Lex also provides *policy-as-code*: a machine-readable file (`lexmap.policy.json`) that encodes module ownership, allowed and forbidden call edges between modules, and kill patterns to be eliminated from the codebase. Language-specific scanners (PHP, TypeScript, Python, etc.) emit factual observations about code usage, which Lex then merges and checks against policy. This follows the pattern of policy-as-code in infrastructure systems like Terraform and OPA/Rego, [?, ?] but is applied to application architecture.

These two subsystems are unified through (a) a shared canonical module vocabulary and (b) a spatial retrieval strategy we call the **fold radius**. The shared vocabulary is formalized as **THE CRITICAL RULE**: every module identifier in episodic Frames must match a module identifier in `lexmap.policy.json`. The fold radius defines how much of the architectural map is "unfolded" for recall: when recalling a past Frame, the assistant retrieves only the touched modules and their immediate adjacency neighborhood, not the entire codebase. This produces policy-grounded continuity ("why was that button still disabled?") without dumping tens of thousands of lines of code into the model.

We argue that this combination — timestamped, opt-in episodic memory aligned to a policy-governed architectural map, with adjacency-bounded recall and high-compression visual context — constitutes a practical path to explainable, auditable AI-assisted continuity for large legacy codebases.

[*]ORCID: 0009-0001-0669-0749

1 Introduction

Large software systems degrade the working memory of the humans maintaining them. An engineer working at 1:30 AM often leaves the codebase in a "half-wired, almost-working" state: one branch, two failing tests, a disabled UI control, and a mental note such as "I need to route this call through the approved service tomorrow." By the next day, that intent is gone. Whoever resumes the work must re-derive not just *what* is broken, but *why it was left that way*.

In parallel, most production codebases have undocumented architectural boundaries. A typical rule might be: "UI code must not call this backend adapter directly; it must go through the approved access layer." These boundaries are often enforced socially ("ask the senior engineer") or via code review. Violations appear late, under pressure.

We claim both problems — human continuity and architectural enforcement — are the same core failure: lack of persistent, queryable, structured memory that (1) captures developer intent at specific decision points and (2) encodes architectural policy in machine-readable form. Existing assistants do not solve (1), because their context vanishes after the session. [**?**, **?**] Existing static analysis tools do not solve (2), because they typically operate on local rules (imports, style, lint) rather than an explicitly versioned map of allowed module interactions. [**?**, **?**, **?**, **?**]

We introduce **Lex**, a unified system with two cooperating subsystems:

- **Memory subsystem**: an episodic memory layer that stores **Frames**. A Frame bundles (a) an opt-in "memory card" image summarizing a high-signal moment (logs, failing tests, rationale for leaving a control disabled), (b) the raw text behind that summary, and (c) structured metadata (branch, blockers, `status_snapshot.next_action`, etc.).
- **Policy subsystem**: a policy layer that represents architectural intent as code. It defines modules, their owned paths/namespaces, and which edges are allowed or forbidden. It also tracks feature flags, required permissions, and kill patterns (anti-patterns scheduled for removal), analogous in spirit to infrastructure policy-as-code systems (e.g. Terraform with OPA/Rego). [**?**, **?**]

Two ideas unify these subsystems:

THE CRITICAL RULE. Every module name used anywhere in the system *must* match the canonical IDs in `lexmap.policy.json`. No ad hoc aliases. If the vocabulary drifts, you lose the ability to align "what happened last night" with "what the architecture is supposed to be."

Fold radius and Atlas Frames. Instead of reloading the entire codebase, the assistant reconstructs only the local neighborhood of modules relevant to a Frame. Starting from the Frame's `module_scope`, we "unfold" just one hop of adjacency in the architectural policy graph. We call that exported slice an **Atlas Frame**, and we call the breadth of that neighborhood the **fold radius**.

This yields recall like:

> `/recall TICKET-123`
> Return: the most recent Frame tagged with that ticket, plus an Atlas Frame of the relevant modules and their direct policy edges.
> Answer: "You left this button disabled because the UI layer was still calling a forbidden module; your declared next step was to route it through the approved service boundary."

The memory card is intentionally image-based. Recent work in high-density optical packing of textual context shows that large textual state can be delivered to a multimodal model as vision tokens at a fraction of the cost of naively inlining that same text in the language context, while preserving semantic recoverability. [**?**, **?**, **?**] This makes multi-day recall economically viable.

2 Related Work

2.1 AI Coding Assistants and Context Windows

Mainstream coding assistants can summarize diffs, suggest fixes, and help navigate recent changes. [?, ?] But they do not keep persistent, queryable memory of *why* a change was deferred, which boundary rule blocked it, or what the human said they would do next. Context is typically ephemeral to the chat session, so the human must restate intent every day.

2.2 Static Analysis and Architecture Testing

Tools such as ESLint, Pylint, SonarQube, and ArchUnit detect style violations, bug patterns, or illegal import-layer crossings. [?, ?, ?, ?] However, they typically do not express cross-language architectural boundaries as a version-controlled policy artifact inside the repo. They rarely encode "which feature flag must be active for this path," "which permission gates this module," or "UI must not call this low-level adapter directly" as a first-class, reviewable contract.

2.3 Policy-as-Code

Policy-as-code is well established in infrastructure and deployment pipelines: configuration is expressed declaratively, and automated systems (e.g. Terraform plus OPA/Rego) enforce organizational rules in CI/CD. [?, ?] Lex applies the same idea to application architecture and runtime boundaries: `lexmap.policy.json` encodes module ownership, allowed/forbidden edges, kill patterns scheduled for removal, and required flags/permissions, and violations can fail CI.

2.4 Long-Context Compression via Vision Tokens

Recent optical / multimodal compression strategies (e.g. Visual Instruction Tuning, DeepSeek-VL, GPT-4V-style vision-language pipelines) encode text-heavy or state-heavy information into structured visual panels and pass those panels through a vision-language model. [?, ?, ?] Reported results indicate effective reductions of $\sim$7–20$\times$ in token cost, while preserving enough structure for tasks like table extraction, historical reasoning over logs, or state reconstruction. Lex adopts this principle for "memory cards": each Frame's high-signal context is rendered once into an image, which can later be fed back to an assistant cheaply instead of replaying megabytes of log text.

2.5 Authorship and AI Contribution

Emerging publication norms require that AI systems be acknowledged but not credited as accountable co-authors, because they cannot take responsibility for claims or consent to publication. [?, ?] We follow that model here: GPT-5 Thinking is acknowledged as providing assistance in system design and technical framing; a human author remains responsible for factual claims.

3 System Overview

Lex is a unified system with two cooperating subsystems: episodic memory and architectural policy. Together, they support recall with receipts: "What was I doing, why was it blocked, and what did I say I'd do next?"

3.1 Memory Subsystem: Episodic Work Memory

The memory subsystem captures **Frames**. A Frame is created intentionally (e.g. via `/remember`) at a meaningful boundary such as:

- after diagnosing a blocker,
- before switching branches,
- before ending the workday,
- when deferring an unfinished fix.

Each Frame stores:

1. Rendered Memory Card (Image). A compact, high-contrast panel that includes recent failure output, a short diff summary or state snapshot, active feature flags, and a human-authored rationale for why work stopped and what must happen next. This panel is designed to be consumed later by a multimodal model at low token cost, leveraging vision-token style compression. [?, ?, ?]

2. Raw Text Payload. The exact text (logs, notes, next steps) that informed the card. This supports high-fidelity recall and diffing.

3. Structured Metadata (Indexable State).

```
{
  "timestamp": "2025-11-01T16:04:12-05:00",
  "branch": "feature/TICKET-123_refactor_policy_gate",
  "jira": ["TICKET-123"],
  "module_scope": ["ui/user-admin-panel", "services/user-access-api"],
  "feature_flags": ["user_admin_panel_enabled"],
  "permissions": ["user_admin_manage"],
  "summary_caption": "UI action remains disabled pending access wiring",
  "status_snapshot": {
    "tests_failing": 2,
    "merge_blockers": [
      "panel still calls forbidden service directly",
      "timeout handling not aligned with policy"
    ],
    "next_action": "Route calls via user-access-api instead of direct adapter"
  },
  "keywords": [
    "admin panel disabled",
    "access wiring",
    "timeout handling",
    "policy violation",
    "TICKET-123"
  ]
}
```

Key fields include:

- `summary_caption`: why this moment mattered.
- `status_snapshot.next_action`: what the human said should happen next.
- `module_scope`: which architectural modules were touched.

Frames are stored locally in a developer-owned database (e.g. SQLite such as `/srv/lex/memory/thoughts.c`
Lex exposes Frames to assistants via a Model Context Protocol (MCP) channel over `stdio` [**?**] (a spawned local process with environment variables), not via an always-on HTTP service. This local-first posture is intentional: no surveillance, no silent upload of work product.

3.2 Policy Subsystem: Architecture as Code

The policy subsystem stores architectural boundaries in `lexmap.policy.json`. Each module describes:

- the file paths or namespaces it owns,
- what it exposes as public surface,
- which modules may call it (`allowed_callers`),
- which modules must *not* call it (`forbidden_callers`),
- which feature flags gate it,
- which permissions are required,
- kill patterns scheduled for removal.

Example excerpt:

```
{
  "modules": {
    "ui/user-admin-panel": {
      "owns_paths": ["web-ui/admin/**"],
      "exposes": [
        "renderAdminPanel",
        "openUserAccessModal"
      ],
      "allowed_callers": ["services/user-access-api"],
      "forbidden_callers": ["services/raw-auth-adapter"],
      "feature_flags": ["user_admin_panel_enabled"],
      "requires_permissions": ["user_admin_manage"],
      "kill_patterns": [],
      "notes": "UI must not call raw-auth-adapter directly; must go
               through user-access-api"
    },

    "services/user-access-api": {
      "owns_paths": ["core/services/access/**"],
      "exposes": [
          "grantUserAccess",
          "revokeUserAccess"
        ],
        "allowed_callers": ["ui/user-admin-panel"],
        "forbidden_callers": [],
        "feature_flags": [],
        "requires_permissions": ["user_admin_manage"],
        "kill_patterns": ["direct_auth_adapter_bypass"],
        "notes": "Approved boundary between UI and low-level auth adapters"
      }
    },

    "global_kill_patterns": [
      "direct_auth_adapter_bypass"
    ]
  }
```

Lex ingests scanner output. Each scanner is language-specific (PHP, TypeScript/JavaScript, Python, etc.) and is **intentionally dumb**: it walks source files and emits observed facts (declarations, imports, feature-flag checks, permission checks). It does *not* enforce architecture or infer intent.

Multiple scanner outputs are merged into a single `merged.json`, then validated:

```
lexmap check merged.json lexmap.policy.json
```

This maps each file to a module (via `owns_paths` / `owns_namespaces`), resolves module-to-module calls, flags forbidden edges, surfaces kill patterns, and exits with CI-friendly codes (0 = clean, 1 = violations, 2 = tool error). The result is a reviewable, enforceable architectural contract living in version control, analogous to infrastructure policy enforcement in Terraform/OPA workflows. [**?**, **?**]

4 THE CRITICAL RULE

THE CRITICAL RULE: Every module name used anywhere in the system MUST match the IDs in `lexmap.policy.json`. No ad hoc naming. If the vocabulary drifts, we lose the ability to align "what happened last night" with "what the architecture is supposed to be."

Concretely:

- `module_scope` in a Frame is an array of module IDs, e.g. `["ui/user-admin-panel", "services/user-access-api"]`.
- Those IDs MUST appear verbatim in `lexmap.policy.json`.
- Violation reports from the policy subsystem must refer to those exact IDs.
- Assistants must use those IDs when reasoning about policy.

This enables automatic answers to questions like "Why is the button still disabled?" by cross-referencing (a) the last saved Frame and (b) the allowed/forbidden edges in policy.

5 Spatial Recall: Fold Radius and Atlas Frames

5.1 Reference Point

Each Frame is anchored at a **reference point** such as "the Save button in the admin panel is still disabled." That reference point maps to one or more modules in `module_scope` (e.g. `["ui/user-admin-panel", "services/user-access-api"]`).

5.2 Atlas Frame

Starting from `module_scope`, the policy subsystem exports a local architectural slice called an **Atlas Frame**:

- All modules in `module_scope`.
- All directly adjacent modules by policy edge:
  - modules they are allowed to call,
  - modules allowed to call them,
  - modules they are forbidden to call (the blockers).

That slice includes ownership, allowed/forbidden call edges, required permissions, feature flags, and kill patterns. Intuitively, it is a single "map page" of the system, not the entire atlas.

5.3 Fold Radius

We define **fold radius = 1** as: export only the immediate neighborhood (one hop) of those modules. Larger radii (2, 3, ...) would unfold progressively more of the system graph.

Radius 1 is the default because it minimizes irrelevant detail, keeps token cost low, matches human intuition ("show me what touches the thing I'm working on"), and lines up with adjacency-bounded vision compression: we only generate memory cards for what matters, not the entire monolith. [?, ?, ?]

Operationally:

1. The user runs `/recall TICKET-123`.
2. The memory subsystem returns the most recent Frame for that ticket, including its memory card image and structured metadata (e.g. `status_snapshot.next_action`).
3. The policy subsystem uses `module_scope` to export an Atlas Frame at fold radius 1.
4. The assistant answers: "You left this disabled because the UI was still calling a forbidden module. Your declared next step was to route through the approved boundary."

This replaces manual archeology across terminals, chat logs, and half-written TODOs.

6 Security, Privacy, and Trust

- **Local-First Storage.** Frames live in a local database on the developer's machine (e.g. SQLite). No automatic upload. No default HTTP service. Access is via MCP over `stdio`,[?] not a background sync.

- **Intentional Capture, Not Surveillance.** Lex does not continuously record keystrokes, shells, or screens. A Frame exists only when the human explicitly triggers it (`/remember`).
- **Explicit Policy, Not Tribal Knowledge.** `lexmap.policy.json` lives in version control. Changes are code-reviewed. Violations can fail CI by team choice. [?, ?]
- **Auditable Narrative.** Because Frame metadata includes timestamp, branch, blockers, and `status_snapshot.next_action`, recall is effectively an audit trail of engineering intent across days. The goal is to help "Future Me" or a teammate, not to provide surveillance data for management dashboards.

7 Limitations and Future Work

Scanner Fidelity. Current language scanners rely on partial AST analysis, heuristics, and pattern matching. They are "dumb by design": they emit facts (imports, calls, flag checks) and do not enforce policy. Over time we expect richer parsers and better detection of feature-flag or permission checks.

Policy Granularity. Real legacy boundaries are messy. `lexmap.policy.json` assumes modules and edges can be described cleanly. We expect incremental bootstrapping: start with one or two modules, expand outward.

Memory Card Rendering Quality. The memory card images must be legible and consistent, not necessarily pretty. The compression win relies on modern multimodal models' ability to decode structured text from high-resolution panels (similar in spirit to recent vision-language methods). [?, ?, ?]

Social Adoption. Engineers must trust that Frames will not be used against them. Local-first storage and explicit capture are non-negotiable.

Authorship and Disclosure. Publication norms require disclosing AI assistance while identifying a human author who takes responsibility for claims. [?, ?]

8 Conclusion

We present an integrated model for AI-assisted continuity in software development that does not depend on passive surveillance or LLM guesswork. Our contributions are:

1. **Frames (Memory Subsystem).** Deliberate, timestamped, opt-in episodic memory units that store a compressed, vision-friendly memory card image, raw text context, and structured metadata (branch, blockers, `status_snapshot.next_action`, `module_scope`).
2. **Policy-as-Code (Policy Subsystem).** A version-controlled `lexmap.policy.json` that encodes architectural ownership, allowed/forbidden edges, required permissions, feature flags, and kill patterns. Enforced in CI, borrowing ideas from infrastructure policy-as-code. [?, ?]
3. **THE CRITICAL RULE.** Module IDs in Frames *must* match module IDs in `lexmap.policy.json`. Without that alignment, episodic memory cannot be reconciled with architectural law.

4. **Fold Radius + Atlas Frames.** An adjacency-bounded recall mechanism. When you `/recall` a ticket, the system rehydrates only the modules you touched and their one-hop neighborhood, not the entire monolith.

5. **Vision-Compressed Context.** By storing a rendered memory card image, we can hand extremely dense state back to a multimodal assistant at a fraction of the token cost, leveraging recent advances in vision-language compression. [**?**, **?**, **?**]

This produces an assistant that can answer, "Why is this still blocked?" using timestamped evidence, architectural policy, and the human's own declared next step — instead of guessing.

References

[1] GitHub Copilot. *AI pair programmer.* GitHub, Inc., 2021–2025. `https://github.com/features/copilot`

[2] Cursor. *The AI Code Editor.* Anysphere, Inc., 2023–2025. `https://cursor.sh`

[3] ESLint. *Find and fix problems in your JavaScript code.* OpenJS Foundation, 2013–2025. `https://eslint.org`

[4] Pylint. *Python code static checker.* Python Code Quality Authority, 2003–2025. `https://pylint.org`

[5] SonarQube. *Continuous code quality inspection.* SonarSource SA, 2007–2025. `https://www.sonarqube.org`

[6] ArchUnit. *Unit test your Java architecture.* TNG Technology Consulting GmbH, 2017–2025. `https://www.archunit.org`

[7] HashiCorp. *Terraform: Infrastructure as Code.* HashiCorp, Inc., 2014–2025. `https://www.terraform.io`

[8] Open Policy Agent (OPA). *Policy-based control for cloud native environments.* Cloud Native Computing Foundation, 2016–2025. `https://www.openpolicyagent.org`

[9] Liu, H., et al. *Visual Instruction Tuning.* arXiv:2304.08485, 2023.

[10] DeepSeek-AI. *DeepSeek-VL: Towards Real-World Vision-Language Understanding.* arXiv:2403.05525, 2024.

[11] OpenAI. *GPT-4 Vision Technical Report.* OpenAI, 2023.

[12] Model Context Protocol (MCP). *Open protocol for LLM-application integration.* Anthropic PBC, 2024. `https://modelcontextprotocol.io`

[13] Nature Portfolio. *Authorship: AI tools and services.* Springer Nature, 2023. `https://www.nature.com/nature-portfolio/editorial-policies/authorship`

[14] Science Magazine. *Generative AI and scientific publishing.* AAAS, 2023.

Acknowledgments

Conceptual development of fold radius, Atlas Frames, and the integration of episodic Frame recall with architectural policy enforcement was performed collaboratively in dialogue between the human author and GPT-5 Thinking. GPT-5 Thinking is acknowledged here as an assisting system but is not listed as an accountable co-author, consistent with norms requiring human responsibility for scientific claims. [?, ?]