Soullayer: A Control Plane Architecture for Portable AI Identity, State Governance and Security Interoperability

Author: Shahid Khan - https://www.linkedin.com/in/shahidmkhan/

Abstract The rapid proliferation of diverse AI agents has introduced severe identity fragmentation and governance vulnerabilities across vendor-locked platforms. This paper introduces Soullayer as a portable identity and governance control plane designed for AI assistants and agent runtimes.[1] By formalizing user context into a canonical and version-controlled Soullayer State Document (SSD) the system enables seamless interoperability without compromising privacy. Soullayer utilizes a Cross-Runtime Compiler to translate this unified state into platform-native formats while a deterministic Policy Engine enforces strict redaction and data residency constraints.[1] The architecture addresses critical security threats including Indirect Prompt Injection (IPI) by enforcing a strict demarcation between an untrusted probabilistic machine learning layer and a deterministic Trusted Computing Base (TCB).[1] Through its native integration with the Model Context Protocol (MCP) and rigorous cryptographic standards Soullayer provides a highly scalable, empirically auditable and regulatory-compliant framework for the next generation of autonomous AI ecosystems.[1]

1. Introduction: Identity Fragmentation, Governance and the Threat Landscape

The rapid proliferation of Large Language Models (LLMs) and autonomous generative artificial intelligence agents has catalyzed a paradigm shift in human-computer interaction. It moves the industry from deterministic command execution to probabilistic and intent-driven orchestration. This transition is fundamentally constrained by a critical architectural deficiency. This deficiency is the systemic fragmentation of user identity, context and operational state across isolated and vendor-locked platforms. Currently users are compelled to continually "re-teach" heterogeneous AI systems their personal preferences, operational boundaries and historical contexts [1, 1]. This phenomenon is characterized in contemporary human-computer interaction literature as "identity fragmentation." It severely degrades the user experience, significantly increases token consumption through repetitive context loading and prevents the establishment of persistent cross-platform workflows.[2] As users increasingly orchestrate multi-model interactions leveraging distinct Multimodal Large Language Models (MLLMs) for specific capabilities the absence of a unified state mechanism results in severe coordination failures, inconsistent conversational histories and shifting trust calibrations.[3]

Beyond usability frictions identity fragmentation introduces profound governance vulnerabilities and structural security flaws. When AI agents operate without a centrally managed and auditable identity control plane organizations cannot enforce consistent data retention protocols, privacy redactions or regulatory access constraints. Traditional software identity paradigms like user impersonation via persistent high-privilege access tokens are fundamentally ill-suited for autonomous agents. Because agents act non-deterministically and adapt flexibly to unstructured inputs granting them indistinguishable human user privileges creates critical accountability gaps and circumvents established access controls.[5] The optimal security posture requires a transition toward explicit "on-behalf-of" (OBO) delegated authority wherein an agent's scope is strictly delineated, time-limited and cryptographically verifiable across all interactions.[2] The volume of actions that autonomous agents can perform threatens to overwhelm users with authorization requests. This leads to "consent fatigue" and renders human-in-the-loop oversight practically unscalable.[2]

Concurrently the adversarial threat landscape surrounding LLM-integrated systems has evolved with alarming speed. Prompt injection and particularly Indirect Prompt Injection (IPI) has transitioned from a theoretical vulnerability into a foundational and actively exploited mechanism in agentic systems.[7] In an IPI attack malicious instructions are embedded within external data sources like web pages, emails, calendar invites or retrieved documents which the LLM subsequently consumes as trusted context.[9] Because LLM architectures currently struggle to reliably distinguish between authoritative system instructions and untrusted input data these hidden prompts can successfully hijack the model's objective function. This hijacking leads to unauthorized data exfiltration, lateral network movement or Remote Code Execution (RCE).[7] The advent of tool-calling protocols has amplified this risk into "agentic amplification" where a single poisoned document can autonomously trigger a catastrophic multi-step exploit chain.[8]

The sociotechnical implications of this fragmentation and lack of governance are equally pressing. The continuous generation of digital doppelgangers and isolated pre-mortem AI clones across disparate platforms can induce profound psychological effects including identity dissonance and a phenomenon termed the "Reverse Uncanny Valley." In this scenario the highly efficient digital self overshadows the biological user leading to feelings of obsolescence and doppelganger phobia.[12] Regulatory frameworks such as the European Union's AI Act, the Colorado AI Act and the NIST AI Risk Management Framework (AI RMF) are increasingly demanding that systems provide transparent audit trails, limit automated decision-making risks and maintain rigorous data provenance.[15]

To address these compounding architectural, security and regulatory challenges this research report examines Soullayer as a portable identity and governance control plane architected specifically for AI assistants and agent runtimes. Soullayer conceptualizes AI memory and identity not as an opaque vector database but as a canonical, version-controlled and mathematically formalizable document known as the Soullayer State Document (SSD) [1, 1]. By sitting structurally above raw memory retrieval engines Soullayer compiles this universal state

into platform-native formats, enforces deterministic privacy policies before data egress and provides cryptographic protection at rest [1, 1]. This exhaustive report provides a systems-level analysis of the Soullayer architecture, formalizing its algorithms, evaluating its mitigation of injection threats and outlining its performance scalability across highly distributed enterprise deployments.

2. Background, Related Work and Competitive Landscape

To accurately contextualize the Soullayer architecture it is necessary to examine the current state of the art in AI memory engines, interoperability protocols and the rapidly expanding taxonomy of adversarial AI threats.

Memory Engines, Agent Frameworks and Vendor Features

A structural analysis of contemporary context management reveals a bifurcation between Data Pipelines (like Vectorize.io), Knowledge Graphs/Memory APIs (like Mem0, Zep and Supermemory) and Agent Execution Frameworks (like LangChain and Letta). Proprietary systems such as OpenAI's Custom Instructions and Claude's Project Knowledge securely store user preferences but actively enforce vendor lock-in through closed and incompatible formats. These systems do not permit state portability across competing platforms. This effectively traps user context within a single vendor's ecosystem and forces users to manually synchronize their identities.[1]

Standalone memory engines attempt to solve context retention through embeddings and retrieval-augmented generation (RAG) paradigms. LangChain Memory provides conversational context management via configurable strategies such as simple buffer memory or summary-driven vector memory. It suffers from significant latency degradation as conversation histories expand. It relies heavily on computationally expensive vector similarity searches that can consume hundreds of megabytes of RAM.[18] Letta (formerly MemGPT) offers a tiered context agent framework but remains locked to its own runtime and handles policy manually via system prompts. Vectorize.io focuses purely on RAG data pipelines without native identity governance. Supermemory.ai offers a consumer knowledge graph but limits users to a siloed dashboard.[18]

Mem0 operates on a different premise by extracting salient facts from conversations and distilling them into compact natural language memories. It employs a discrete update strategy using programmatic additions, updates and deletions.[20] This achieves high token efficiency averaging approximately 1,764 tokens per conversation and low search latency.[20] Zep builds a temporal knowledge graph utilizing its open-source Graphiti library and anchors memories to specific temporal nodes to enable complex relational reasoning. While this architecture excels at multi-hop queries it incurs massive token overhead that often exceeds 600,000 tokens per conversation. It also suffers from high latency which renders it less viable for sub-second

interactive agent deployments.[20]

Soullayer occupies a novel tier as a Governance and Portability Control Plane. It is architecturally distinct from retrieval engines because it operates as an identity orchestration layer positioned directly above memory storage.[1] While it can utilize SQLite, Mem0 or Zep as underlying storage adapters via its MemoryBackend interface Soullayer uniquely treats identity, policy, versioning and compilation as first-class schema-validated constructs rather than mere embedding retrievals [1, 1].

Table 1: Technical Comparison of State-of-the-Art AI Context Frameworks

Platform / Framework	Core Paradigm	Identity Portability & MCP	Policy & Governance Enforced	Cross-Platform Compilation
Soullayer	Identity Control Plane	Universal (SSD via MCP/REST)	Strict (Regex, PII and RBAC)	Native formats (OpenAI, Claude and IDEs) [1]
Mem0 / Zep	Long-term Memory / Graph API	Siloed (Requires API integration)	None (Focus is retention)	No (Returns JSON payloads only) [18]
Letta (MemGPT)	Tiered Context Agent Framework	Locked to Letta runtime	Manual via system prompts	No
LangChain	Orchestration Library	Bound to application logic	Via custom middleware	No [18]
Vectorize.io	RAG Data Pipeline Builder	N/A (Data focus)	N/A	No
Supermemory.ai	Consumer Knowledge Graph	Siloed consumer dashboard	None	No

The Model Context Protocol (MCP) and Interoperability

The Model Context Protocol (MCP) is an emerging open standard championed by industry

leaders to enable seamless and bidirectional communication between AI models and external data sources or execution environments.[21] Utilizing JSON-RPC 2.0 as its underlying transport mechanism MCP standardizes how context is provided to AI clients without requiring bespoke per-vendor API integrations.[22]

While MCP effectively standardizes the transport and invocation of context it lacks inherent semantic governance regarding what identity state is shared and how privacy boundaries are enforced. Soullayer bridges this critical gap by operating as a native MCP server that injects strict governance, cryptographically secured identity data and policy evaluation directly into the MCP lifecycle.[1] This ensures that any tool invoked or resource accessed is bounded by the user's canonical preferences [1, 1].

3. System Model and Architecture

To achieve rigorous interoperability and security across isolated vendor ecosystems the system must constrain AI memory. It shifts it from an opaque probabilistic vector space into a mathematically formalizable and deterministic entity.

SSD Formalization and Schema Constraints

The core nucleus of the system model is the Soullayer State Document (SSD) which is a canonical and version-controlled JSON artifact [1, 1]. The document is mathematically formalized as a typed object D. This object comprises a set of partial and strictly typed subsets defined as "packs":

$$D = \{Id, Pr, St, Po, Ad, Go\}$$

These subsets represent Identity, Preferences, State, Policy, Adaptation and Governance respectively.[1]

The system enforces a strict set of invariant constraints on D:

1. **Identifier Constraints:** Every entity and proposal must possess a Universally Unique Identifier conforming to RFC 4122 or a ULID (Universally Unique Lexicographically Sortable Identifier) to guarantee chronological sorting.[1]
2. **Temporal Constraints:** All timestamps must strictly adhere to the ISO 8601 datetime format to prevent parsing errors across distinct programming environments.
3. **Schema Versioning:** The document schema version must adhere exactly to the JSON Schema Draft 2020-12 specification and follow Semantic Versioning (SemVer) regular expression patterns. This ensures backward compatibility during migrations and provides a clear compatibility strategy for future document evolution.[1]

Topological System Flow

The architecture is partitioned into four primary execution layers mapping clients through an

API interface into core compilation and storage engines.

Table 2: Topological System Layers and Component Interactions

Architectural Layer	System Component	Operational Responsibility
Clients	CLI, Web Dashboard, Extension and SDK	Provides integration points for human users and host applications
API Layer	REST, WebSocket and MCP Server	Handles ingress routing and standardized communication protocols
Core Services	SSD Manager	Manages canonical schema and executes deterministic validation
	Policy Engine	Enforces data redaction, retention and sensitivity rules
	Adaptation Engine	Executes the probabilistic propose, approve and apply learning loop
	Compiler Registry	Translates universal state into target-specific platform formats
	Version Control	Calculates state diffs and provides cryptographic

		rollback capabilities
	Audit Logger	Maintains an immutable ledger of all state transitions
	Plugin Manager	Orchestrates sandboxed extensions and third-party lifecycle hooks
Storage Layer	SQLite, Filesystem and External Backends	Provides ACID-compliant persistence and external retrieval adaptations

4. Algorithms and Standardized Patch Semantics

The functional superiority of the Soullayer control plane is derived from the deterministic formalization of its core algorithms. These operations successfully map probabilistic untrusted AI concepts into predictable and mathematically auditable computing functions.[1]

The Adaptation Proposal Loop and UX Realism

The Adaptation Engine is responsible for translating unstructured chaotic conversational transcripts into structured state updates. Because LLMs natively output probabilistic text the algorithm must severely constrain their outputs using the JSON Merge Patch format (RFC 7396).[1]

The UX for human approval is designed to be frictionless. Users receive a clear side-by-side diff of the proposed changes against their current identity profile. They can accept or reject the JSON Merge Patch proposal with a single click. This guarantees that no automated state pollution occurs without explicit user consent.[1] The engine parses the approved output, strips any surrounding markdown code fences and assigns a ULID to guarantee sequential sorting.[1]

Standardized Patch Semantics Justification

The system justifies its dual-patch architecture by assigning distinct roles to different protocols. It utilizes JSON Merge Patch (RFC 7396) for probabilistic LLM-generated proposals because it is structurally simple and friendly for language models to generate without syntax errors.[1]

Conversely it employs JSON Patch (RFC 6902) for deterministic Version Control because it provides precise version diffs, exact array index manipulations and replayable history.[1]

Compilation and Policy Algorithms

The CrossRuntimeCompiler translates the universal SSD into the proprietary formats required by diverse target systems. The compilation target function is formalized as:

$$Compile\_target(D, \theta) = Truncate(Plugin\_target(P(D, \theta)), L\_target)$$

Where $P(D, \theta)$ applies the redaction policy engine against the document stripping any state information that exceeds the target's sensitivity threshold θ.

5. Implementation and Deployment Economics

The implementation of Soullayer reflects modern systems engineering practices designed to maximize cross-platform compatibility and minimize deployment friction.

Implemented Scope vs Planned Scope

It is critical to distinguish between the currently implemented scope and planned roadmap features to maintain academic rigor.[1] The current implementation (V1) successfully ships the Schema Validator, Cross-Runtime Compiler, Policy Engine, SQLite-based Storage Layer and the basic human-in-the-loop Adaptation Engine.[1] The planned scope for Phase 3 includes multi-model consensus to reduce LLM hallucination variance, a community plugin marketplace and decentralized CRDT synchronization.[1]

Deployment Economics and Cost-Optimized Stacks

Soullayer defines a universal MemoryBackend interface allowing the storage mechanism to adapt based on operational expenditure requirements.[1] The system employs a quantitative cost-optimized scaling model:

1. **SQLite-First Local Stack:** Utilizes local SQLite databases with Ollama LLM offloading resulting in a total cost of approximately $9.85 per month for up to 1,000 users.
2. **Serverless Edge Stack:** Leverages Cloudflare Workers and Turso distributed SQLite yielding a highly efficient $5 to $8 monthly footprint.
3. **Hybrid Routing Stack:** Intelligently routes bulk requests to local models and complex queries to GPT-4o-mini managing up to 10,000 users for $247 per month.[1]

6. Test, Evaluation, Verification and Validation (TEVV) Plan

Due to the absence of standardized academic benchmarks for identity portability and

state-document compliance a rigorous and reproducible Test, Evaluation, Verification and Validation (TEVV) protocol is required.[1] This TEVV plan quantifies Soullayer's efficacy across compilation targets, policy enforcement and adaptation accuracy.[1]

Benchmark Corpus Design

The TEVV methodology introduces three distinct benchmark slices designed to measure both nominal operational performance and resilience against adversarial exploitation:

1. **SSD-Adapt-Synth:** A synthetic transcript corpus utilizing templated user profiles and tightly scripted dialogues. This baseline evaluates fundamental information extraction correctness and patch accuracy without introducing user privacy risks.[1]
2. **SSD-Adapt-Real:** An opt-in and highly de-identified dataset of real-world user transcripts gathered with explicit consent. This slice measures external validity and inter-annotator agreement in natural highly noisy conversational settings.[1]
3. **SSD-Adapt-Adv:** An adversarial corpus explicitly injected with malicious payloads designed to test the system's resilience against Indirect Prompt Injection, data exfiltration and tool-hijacking attempts.[1]

Table 3: TEVV Metrics and Benchmark Objectives

Benchmark Category	Primary Hypothesis / Objective	Independent Variables	Dependent Metrics
Adaptation Quality	Certain LLM providers yield higher extraction accuracy.	Provider, temperature and prompt variant	Patch F1 score, ECE calibration and proposal accept-rate [1]
Policy Efficacy	Regex redaction reduces leakage with acceptable utility loss.	Rule sets and sensitivity thresholds	Leakage rate, False redaction rate and Utility score [1]
Compiler Fidelity	Compiled representations preserve constraints universally.	Target type, $maxChar$ and restrict flags	Truncation %, Token budget adherence and Semantic equivalence [1]

7. Security, Privacy and Ethical Considerations

The design of an autonomous identity control plane necessitates stringent security architectures and careful ethical consideration regarding digital representation.

Compliance Mapping (NIST AI RMF 1.0)

To systematically govern AI risk the Soullayer architecture closely aligns with the core functions of the NIST Artificial Intelligence Risk Management Framework (AI RMF 1.0).[16]

- **Govern:** Soullayer establishes foundational organizational accountability through its RBAC permission engine defining transparent roles and generating immutable cryptographic audit logs.[16]
- **Map:** The architecture explicitly maps system boundaries and dependencies through its plugin manifestation schema. The strict JSON schemas enforce predictable system behavior mapping out the precise data context an agent can access.[1]
- **Measure:** The TEVV framework provides quantitative red-teaming metrics to measure prompt injection resilience and tool misuse rates under adversarial conditions.[1]
- **Manage:** Operational risks are actively managed via the human approval workflow and the capability to execute instant cryptographic rollbacks when anomalous state transitions are detected.[1]

Key Custody and Cryptographic Protection

To protect highly sensitive state data at rest the EncryptionLayer wraps the SSD in a standardized cryptographic envelope.[1] Cryptographic key custody relies strictly on user-provided passphrases derived via Argon2id (RFC 9106) locally. This ensures the platform operator maintains zero-knowledge access to the decrypted SSD state.[1] The derived key is then utilized with AES-256-GCM. A unique and cryptographically secure pseudorandom initialization vector (IV) and salt are generated per encryption cycle and stored transparently alongside the ciphertext payload preventing nonce-reuse vulnerabilities [1, 1].

Attack Surface and Mitigations

Because Soullayer acts as a primary MCP Server it inherently expands the attack surface by connecting external applications to internal state documents.[1] To counter the profound risk of Indirect Prompt Injection (IPI) and agentic amplification the system enforces a strict human-in-the-loop authorization model for all state transitions. By quarantining JSON Merge Patch proposals generated by the untrusted ML layer the system surgically severs the autonomous exploit chain documented in attacks such as CVE-2025-53773.[1]

8. Limitations and Future Work

While the current Soullayer architecture effectively standardizes state and mitigates fragmentation several open systems engineering and theoretical computer science challenges

remain.

Currently Soullayer relies on centralized locking mechanisms like SQLite WAL mode or PostgreSQL table locks to prevent write conflicts.[1] In highly distributed offline-first environments like mobile devices operating intermittently this creates severe availability bottlenecks. Future iterations must explore Conflict-free Replicated Data Types (CRDTs). Implementing a state-based CRDT framework where updates are modeled as commutative, associative and idempotent operations would allow disparate instances to achieve Strong Eventual Consistency (SEC) across devices.[26]

Transitioning to a fully decentralized gossip-based agent architecture requires robust causality tracking for audit logs and conflict resolution. Physical server timestamps suffer from inevitable clock drift and pure logical timestamps lack relation to actual wall-clock time. Implementing Hybrid Logical Clocks (HLCs) would provide causality-aware timestamps that remain close to physical time ensuring precise and mathematically sound ordering of state changes across geographically distributed MCP nodes.[1]

9. Conclusion

The fragmentation of artificial intelligence identity across disparate vendor silos represents a critical impediment to the realization of persistent, autonomous and secure multi-agent ecosystems. The Soullayer architecture comprehensively addresses this deficiency by establishing a foundational identity control plane. By abstracting AI memory into a mathematically formalizable and version-controlled Soullayer State Document (SSD) the system ensures strict semantic portability across highly heterogeneous execution environments.

Architecturally Soullayer enforces a rigorous physical separation between its deterministic Trusted Computing Base and the probabilistic highly vulnerable outputs of LLM infrastructure. By routing all state modifications through an Adaptation Engine that translates unstructured inputs into deterministic JSON Patch operations subject to strict human authorization the system inherently mitigates severe adversarial threats such as Indirect Prompt Injection and agentic tool hijacking. Furthermore the integration of the Model Context Protocol (MCP) as the standardized transport layer alongside AES-256-GCM encryption for data at rest ensures that identity data remains both universally interoperable and cryptographically secure. While challenges remain in decentralized synchronization, Formal Verification of complex schemas and multi-model consensus Soullayer provides a highly scalable, empirically auditable and regulatory-compliant paradigm for governing the identity and state of next-generation artificial intelligence.

10. Appendices

Appendix A: Reproducible Setup and Artifacts Commands

To facilitate independent verification and reproduction of the architectural environment

outlined in Section 5 the following shell commands securely clone, pin and construct the Soullayer environment [1, 1].

```bash
# Clone the repository and pin to the exact evaluated commit to ensure absolute reproducibility
git clone https://github.com/foldedfox/Soullayer.git
cd Soullayer
git checkout <EXACT_COMMIT_HASH>

# Generate a recursive immutable list of all architectural files in the TCB for integrity verification
git ls-files > artifacts_repo_filelist.txt

# Append exact file sizes in bytes to analyze supply-chain security footprints
git ls-files -z | xargs -0 -I{} sh -c 'printf "%s\t" "{}" && wc -c < "{}"' > artifacts_repo_filesizes_bytes.tsv

# Install dependencies strictly via pnpm workspaces to ensure dependency isolation
pnpm install --frozen-lockfile

# Execute core unit, integration and property test suites
pnpm test
```

Appendix B: Operational CLI Usage Examples

The following commands demonstrate the initialization of the local SQLite configuration, identity creation and the invocation of the CrossRuntimeCompiler.[1]

```bash
# Initialize the ~/.soullayer/ state directory and SQLite WAL backend
soullayer init

# Establish a baseline Identity Pack within the SSD capturing bio and preferences
soullayer identity create --name "Researcher" --bio "Systems Evaluation Agent"

# Execute a compilation target subject to strict length truncation (safety-by-truncation)
soullayer compile --target openai --max-chars 6000
```

```
# Initiate the Fastify API and native MCP server to accept JSON-RPC 2.0 connections
soullayer serve --mcp
```

Appendix C: Foundational Standards and Protocol Specifications

The Soullayer system formally implements and adheres to the following critical IETF and NIST standards throughout its architecture [1]:

- **JSON Merge Patch:** RFC 7396 (Utilized for probabilistic adaptation proposals generated by LLMs).
- **JSON Patch:** RFC 6902 (Utilized for deterministic SSD versioning, history tracking and rollbacks).
- **JSON Pointer:** RFC 6901 (Defines strict character escaping protocols ~0 and ~1 for patch application).
- **JSON Schema:** Draft 2020-12 (Utilized by the AJV engine for runtime schema validation).
- **AES-GCM:** NIST SP 800-38D (Authenticated encryption with associated data for protecting the SSD at rest).
- **Argon2id:** RFC 9106 (Memory-hard key derivation function providing resistance to GPU cracking).

Works cited

1. soullayer-pitch-deck.md
2. Identity Management for Agentic AI: The new frontier of authorization, authentication, and security for an AI agent world | alphaXiv, accessed April 10, 2026, https://www.alphaxiv.org/overview/2510.25819v1
3. One Is Not Enough: How People Use Multiple AI Models in Everyday Life - arXiv, accessed April 10, 2026, https://arxiv.org/html/2603.26107v1
4. Remote-Capable Knowledge Work Should Default to AI-Enabled Flexibility - Preprints.org, accessed April 10, 2026, https://www.preprints.org/frontend/manuscript/6f9efe565425a903b13be4961d08a185/download_pub
5. Executive Summary - arXiv, accessed April 10, 2026, https://arxiv.org/html/2510.25819v1
6. Identity Management for Agentic AI - OpenID, accessed April 10, 2026, https://openid.net/wp-content/uploads/2025/10/Identity-Management-for-Agentic-AI.pdf
7. The Lethal Trifecta: How Indirect Prompt Injection Is Breaking Agentic AI — and What Security Teams Must Do Now | by MrDuc | Mar, 2026 | Medium, accessed April 10, 2026, https://medium.com/@itpro677/the-lethal-trifecta-how-indirect-prompt-injection-is-breaking-agentic-ai-and-what-security-teams-c2ecba874ed1
8. Prompt injection: types, real-world CVEs, and enterprise defenses - Vectra AI, accessed April 10, 2026, https://www.vectra.ai/topics/prompt-injection
9. Indirect Prompt Injection: Generative AI's Greatest Security Flaw, accessed April

10, 2026, https://cetas.turing.ac.uk/publications/indirect-prompt-injection-generative-ais-greatest-security-flaw
10. LLM01:2025 Prompt Injection - OWASP Gen AI Security Project, accessed April 10, 2026, https://genai.owasp.org/llmrisk/llm01-prompt-injection/
11. Mitigating Indirect Prompt Injection Attacks on LLMs - Solo.io, accessed April 10, 2026, https://www.solo.io/blog/mitigating-indirect-prompt-injection-attacks-on-llms
12. Speculating on Risks of AI Clones to Selfhood and Relationships: Doppelganger-phobia, Identity Fragmentation, and Living Memorie - UBC Computer Science, accessed April 10, 2026, https://www.cs.ubc.ca/labs/socius/files/papers/cscw2023-aiclone.pdf
13. A Systematic Review of Multimodal AI Agents and Embodied Avatars in University EFL Speaking Support - Scirp.org., accessed April 10, 2026, https://www.scirp.org/pdf/jss_6501396.pdf
14. digital doppelgangers: ethical and societal implications of pre-mortem ai clones - arXiv, accessed April 10, 2026, https://arxiv.org/pdf/2502.21248
15. Interplay between the AI Act and the EU digital legislative framework - European Parliament, accessed April 10, 2026, https://www.europarl.europa.eu/RegData/etudes/STUD/2025/778575/ECTI_STU(2025)778575_EN.pdf
16. NIST AI Risk Management Framework 2025: Secure Your AI Now - Nemko Digital, accessed April 10, 2026, https://digital.nemko.com/regulations/nist-rmf
17. Understanding The NIST AI Risk Management Framework - Protecto AI, accessed April 10, 2026, https://www.protecto.ai/blog/nist-ai-risk-management-framework/
18. LangChain Memory vs Mem0 vs Zep: AI Memory Systems 2026 - Index.dev, accessed April 10, 2026, https://www.index.dev/skill-vs-skill/ai-mem0-vs-zep-vs-langchain-memory
19. AI Agent Memory: Types, Implementation, Challenges & Best Practices 2026 - 47Billion, accessed April 10, 2026, https://47billion.com/blog/ai-agent-memory-types-implementation-best-practices/
20. AI Agent Memory Systems in 2026: Mem0, Zep, Hindsight, Memvid and Everything In Between — Compared | by Yogesh Yadav - Dev Genius, accessed April 10, 2026, https://blog.devgenius.io/ai-agent-memory-systems-in-2026-mem0-zep-hindsight-memvid-and-everything-in-between-compared-96e35b818da8
21. Model Context Protocol - GitHub, accessed April 10, 2026, https://github.com/modelcontextprotocol
22. Architecture overview - Model Context Protocol, accessed April 10, 2026, https://modelcontextprotocol.io/docs/learn/architecture
23. Model Context Protocol (MCP) Extensions for Network Equipment Management - IETF, accessed April 10, 2026, https://www.ietf.org/archive/id/draft-zeng-mcp-network-mgmt-01.html
24. NIST AI Risk Management Framework 1.0 | Consulting Services - RSI Security,

accessed April 10, 2026, https://www.rsisecurity.com/nist-ai-risk-management/
25. Taming Agentic AI: Applying the NIST AI Risk Management Framework - Ricardo Gutierrez, accessed April 10, 2026, https://rgutierrez2004.medium.com/taming-agentic-ai-applying-the-nist-ai-risk-management-framework-a7f592e0e97a
26. Introduction to CRDTs for Realtime Collaboration - DEV Community, accessed April 10, 2026, https://dev.to/nyxtom/introduction-to-crdts-for-realtime-collaboration-2eb1
27. Causal Consistency through a novel distributed middleware over Strongly Consistent Transaction Processing - Malta, accessed April 10, 2026, https://www.um.edu.mt/library/oar/bitstream/123456789/135067/3/Carl%20Camilleri.pdf
28. Spanner: Google's Globally Distributed Database | Request PDF - ResearchGate, accessed April 10, 2026, https://www.researchgate.net/publication/262363395_Spanner_Google's_Globally_Distributed_Database