Cross-Provider Conversation Threading with Governed Memory: An Interoperability Layer for Enterprise LLM Workflows

Abstract

We present an MVP design for a governed, cross-provider conversation hub that lets users start a thread with one model (e.g., Perplexity), "forward" it to another (e.g., OpenAI's GPT via the Responses API), and continue in a third (e.g., Google Gemini) **without losing context**, while enforcing per-turn read/write policies and creating an auditable provenance trail. Our methodology operationalizes the dynamic access graphs, policy-conditioned reads/writes, and coordinator/aggregator pattern described in *Multi-User Memory Sharing in LLM Agents with Dynamic Access Control* and adapts them to a real product with provider adapters for Perplexity, OpenAI Responses, Gemini, and a router/gateway like OpenRouter. (arhttps://docs.google.com/document/d/1HvF7dVPjuocf_EvS-m2lkXMjTvnLRqKXPOm14HiB_QU/edit?usp=sharingXiv)

---

1. Problem & Product Overview

Problem. Teams use multiple LLMs for complementary strengths—search-grounded answers (Perplexity), tool/structured outputs (OpenAI Responses), and long-context digestion (Gemini). Today, context and memory are siloed per vendor; hand-offs require manual copy/paste and risk over-sharing sensitive data. (Perplexity)

Product. A **Cross-LLM Thread Hub** that:

1. stores a canonical, vendor-neutral thread and memory,
2. on each user turn or "Forward to …" action, **builds a scoped memory view** (based on policy) and **rehydrates** the context for the target provider, and
3. records an **explainable audit** of what was shared and why. (arXiv)

What it achieves.

- **Continuity:** seamless hand-off between Perplexity ↔ OpenAI ↔ Gemini within one thread. (Perplexity)

- **Governance:** dynamic permissions and policy-conditioned memory sharing; immutable provenance per turn. (arXiv)

- **Interoperability / portability:** pluggable adapters (Perplexity/Responses/Gemini/OpenRouter) and a domain-aware router. (Perplexity)

---

2. Methodologies Adopted from the Paper (and How We Use Them)

1. **Dynamic Access Graphs.** The paper formalizes **time-varying, asymmetric permissions** using two bipartite graphs—user→agent and agent→resource—evaluated at each step. We implement these as org-scoped policy tables; a check runs **per provider call** to determine what memory/resources the target agent may read. (arXiv)

2. **Two-Tier Memory with Immutable Provenance.** Memory is split into **private** (user) and **shared** (project/org) fragments, each appended with provenance (who/when/which agent/resources). Our hub never mutates fragments; it appends new ones and stores provenance hashes for audits. (arXiv)

3. **Policy-Conditioned Reads/Writes.** Before calling a provider, a **read policy** constructs a **scoped memory view** (top-K fragments allowed for this user/agent/time); after the response, a **write policy** decides what to persist and at which tier (often after redaction/anonymization). We adopt the paper's "simple reads + transformation writes" as the safe default. (arXiv)

4. **Coordinator → Agents → Aggregator.** A central **coordinator** (our domain-aware router) selects eligible agents (providers), each produces an intermediate answer with its scoped memory; an **aggregator** optionally fuses outputs to the final reply. This matches our "Forward to Provider X" UX while remaining compliant. (arXiv)

---

3. Our Product Architecture

3.1 Provider Adapters

- **Perplexity Adapter.** Uses Perplexity's OpenAI-compatible **Chat Completions** for search-grounded Q&A with citations; ideal for fresh, web-sourced facts. (Perplexity)

- **OpenAI Adapter.** Uses the **Responses API** (unified chat + tools) for structured outputs, function/tool use, and downstream automations. (OpenAI Platform)

- **Gemini Adapter.** Uses **Gemini API** for very large context windows (API/Vertex docs detail ~1M-token support across 2.x models), enabling single-pass digestion of long PDFs/codebases. (Google AI for Developers)

- **OpenRouter (optional).** Unified gateway to hundreds of OSS/hosted models with fallbacks and cost/availability routing. (OpenRouter)

3.2 Router (Coordinator)

A **rule-plus-signals router** chooses the provider per turn:

- "Live, cited research" → Perplexity,

- "Tool/structured output" → OpenAI Responses,

- "Very long context" → Gemini,
  with manual override and OpenRouter fallback. The router always **rehydrates** the prompt from our canonical thread + scoped memory view to fit the target model's context/tokenizer. (Perplexity)

3.3 Memory & Policy Engine

- **Read policy**: `last-N dialogue turns` + `top-K allowed fragments` (vector search) + trimming/summarization to meet the **target** model's context budget. (OpenAI and Gemini publish long-context model families; we budget tokens per target.) (Reuters)

- **Write policy**: store distilled summary, entities, and task state; default **private**, promote to **shared** only when PII scrub passes; always attach provenance. (arXiv)

3.4 Audit & Explainability

Per turn, we store: chosen provider/model, policy decisions (why fragments were included/excluded), fragment IDs, and cryptographic hashes for the outgoing package and incoming response—to support SOC2-style review. (arXiv)

---

4. Relation to Existing Systems

Gateways/routers (OpenRouter) unify APIs and provide fallbacks/cost controls; **they don't implement cross-provider conversational memory semantics** or auditable scoped hand-offs. **Our contribution** is a governed **thread-forwarding** UX with memory views and per-turn audits. (OpenRouter)

Multi-agent workspaces (Dust, Vellum) let you build agents/workflows inside one platform; our focus is **interoperability across external providers** plus the policy engine guided by the paper's access-graph methodology. (Dust - User Guides - Developer Platform)

---

5. Implementation Details (MVP)

5.1 Data Model (Postgres)

- `orgs, users` with roles;
- `provider_accounts` (encrypted API keys per org);
- `threads, messages` (canonical transcript);
- `memory_fragments` (private/shared, immutable provenance);
- `user_agent_perm`, `agent_resource_perm` (access graphs with `granted_at/revoked_at`). (arXiv)

5.2 Call Flow (per turn or "Forward")

1. **Authorize** via dynamic access graphs → yield permitted agents/resources.
2. **Read policy** constructs scoped view (vector top-K + last-N), budgets tokens for the **target** provider.
3. **Adapter call** (Perplexity/OpenAI/Gemini/OpenRouter).

4. **Write policy** transforms & persists memory with provenance; **Audit log** records all decisions. (arXiv)

5.3 Provider-Specific Capabilities

- Perplexity: search-grounded answers, citations, OpenAI-compatible schema → easy drop-in. (Perplexity)
- OpenAI Responses: unified tools/agents orchestration for structured plans and function calling. (OpenAI Platform)
- Gemini: long-context ingestion (≈1M tokens in API/Vertex), ideal for big PDFs/code bundles. (Google AI for Developers)

---

6. Evaluation Plan

6.1 Scenarios (from paper → product)

- **Fully collaborative:** broad shared memory; expect fewer duplicate calls and faster completion.
- **Asymmetric:** one team member has private fragments; ensure policy excludes them when routing to external providers.
- **Dynamic/rolling access:** simulate join/leave/permission changes mid-thread; verify audits reflect changes. (arXiv)

6.2 Metrics

- **Task latency** across provider hand-offs (Perplexity→OpenAI→Gemini). (Perplexity)
- **Context fidelity** (hallucination rate vs. ground truth on Perplexity citations; structure correctness on OpenAI tool outputs). (Perplexity)
- **Long-doc accuracy** (checklist correctness after Gemini digest). (Google AI for Developers)

- **Policy correctness** (no unauthorized fragments revealed; pass/fail per turn) and **audit completeness** (verifiable hashes per package). (arXiv)

---

7. Security & Compliance

- **RBAC + ABAC:** org/project/thread scoping; evaluate access graphs every turn. (arXiv)
- **Provenance & immutability:** append-only fragments; signed hashes for payloads/responses. (arXiv)
- **Key management:** server-side encrypted provider keys; per-org budgets and throttles; optional routing via OpenRouter for fallbacks. (OpenRouter)
- **PII guardrails:** transformation writes (redaction/anonymization) before promoting to shared. (arXiv)

---

8. Limitations & Risks

- **No vendor UI session import:** we rehydrate from our canonical transcript and memory; we don't access opaque, consumer-UI session state. (APIs suffice for enterprise flows.) (Perplexity)
- **Tokenizer/context variance:** router must estimate tokens per target provider and compress accordingly; long-context claims are model/version-specific (check docs). (Reuters)
- **Provider policy/ToS drift:** keep adapters thin and swappable (OpenRouter helps reduce churn). (OpenRouter)

---

9. Roadmap

- **Phase 1 (MVP):** Perplexity + OpenAI Responses + Gemini adapters; rule-based router; read/write policies; audit UI. (Perplexity)

- **Phase 2:** auto-router and fallbacks via OpenRouter; richer aggregation; per-fragment differential privacy options. (OpenRouter)

- **Phase 3:** enterprise connectors (Drive/Box/Confluence), evaluations harness, formal policy DSL.

---

10. Conclusion

By **combining** the paper's access-graph and policy-conditioned memory methods with **practical provider adapters** (Perplexity/OpenAI/Gemini) and a **router that rehydrates context per hand-off**, our product delivers governed cross-LLM continuity that current gateways and single-workspace tools don't provide. This fills a clear interoperability gap for enterprises that need **explainable, auditable** collaboration across the best model for each step. (arXiv)

References

- *Multi-User Memory Sharing in LLM Agents with Dynamic Access Control* (arXiv:2505.18279). Methodology: dynamic access graphs, policy-conditioned read/write, coordinator/aggregator. (arXiv)

- Perplexity API (Chat Completions & compatibility). (Perplexity)

- OpenAI **Responses API** (agents/tools). (OpenAI Platform)

- Google Gemini API (long context; API/Vertex docs). (Google AI for Developers)

- OpenRouter (unified API & models). (OpenRouter)

- Comparative workspaces/orchestration (Dust, Vellum). (Dust - User Guides - Developer Platform)

If you want this as a formatted PDF/whitepaper with a one-page executive summary up front, say the word and I'll package it.