Agent-Memory Protocol: A Privacy-Focused Protocol for LLM Agents and User Memory Interaction

Junde Wu, Minhao Hu, Jiayuan Zhu, Jiaye Wang, Yueming Jin
University of Oxford, University of California San Diego, National University of Singapore

Abstract

Large-Language Model (LLM) based agents are rapidly migrating from stateless chat systems to persistent, longitudinal assistants. Nowhere is this transition more sensitive than in fields like medicine and finance, where context accumulates across months of interactions containing personally identifiable or confidential information. Existing memory and retrieval architectures of agents focus on efficiency or reasoning quality but remain indifferent to privacy: prompts, retrieval results, and cached traces leak explicit identifiers to third-party infrastructure. We introduce the **Agent-Memory Protocol (AMP)**, a privacy-first protocol for LLM Agents and User Memory Interaction. AMP enforces confidentiality at the boundary where language meets computation. It defines three deterministic operations—*redact at rest*, *pack for purpose*, and *hydrate on return*, that together guarantee that no personal identifier ever leaves the user boundary while maintaining the reasoning utility of long-term memory. We formalize its protocol and illustrate its operation in multi-turn use-cases spanning medicine and finance, including radiology follow-up, discharge planning, and portfolio management.

Introduction

The recent rise of LLM-driven agents has transformed static chatbots into dynamic collaborators that can plan tasks, write code, analyze data, and manage workflows (Wu et al. 2025a, 2024, 2025b). Coding agents, research assistants, and personal copilots now execute multi-step reasoning and act across tools, APIs, and files—often outperforming isolated human operators in efficiency and consistency. As these agents mature, the next step is *personalization*: connecting them to a user's long-term memory so that every interaction is contextualized by prior actions, preferences, and history (Wu et al. 2025b; Sun et al. 2025). The recent rise of LLM-driven agents has transformed static chatbots into dynamic collaborators that can plan tasks, write code, analyze data, and manage workflow (Wu et al. 2025a, 2024, 2025b)s [cite agentic(Wu et al. 2025a, 2024, 2025b)reasoning, medical graph rag, git context c(Wu et al. 2025a, 2024, 2025b)ontroller]. Coding agents, research assistants, and personal copilots now execute multi-step reasoning and act across tools, APIs, and files—often outperforming isolated human operators in efficiency and consistency. As these agents mature, the next step is *personalization*: connecting them to a user's long-term memory so that every interaction is contextualized by prior actions, preferences, and history [cite git context controller, and context fold].

Such agent–memory coupling unlocks powerful continuity—an assistant that recalls prior code refactorings, a financial planner that tracks investment patterns, or a clinical aide that remembers medication adjustments(Wu et al. 2025a, 2024, 2025b). Yet this evolution introduces an urgent challenge: there is currently no unified, privacy-preserving protocol governing how agents read, write, and reason over user memory. Ad-hoc implementations risk leaking personally identifiable or confidential data during prompt construction, retrieval, or logging. In many cases, the agent must *connect* to the user's memory to provide continuity and personalization, while ensuring that sensitive information never leaves the user boundary. This tension—between reasoning utility and privacy preservation—motivates the need for a principled communication framework.

To address this challenge, we introduce the *Agent-to-Memory Protocol (AMP)*, a language-level interface that governs how agents read from, write to, and reason over user memory without exposing identifiers. AMP formalizes the boundary where symbolic reasoning meets private context, ensuring that no personally identifiable information (PII) traverses the model boundary.

AMP defines three deterministic operations—APPEND, PACK, and HYDRATE that together decouple reasoning from sensitive data. APPEND redacts confidential spans at rest, storing only placeholder-linked semantic cards. PACK assembles redacted narratives relevant to the current task for model inference, while HYDRATE locally re-personalizes the model's symbolic output. By enforcing this redaction–hydration boundary, AMP allows agents to maintain long-term coherence and personalization while provably minimizing information leakage at the language level.

On the other hand, in practice, modern agents must reason over long-lived personal histories—email threads, transaction logs, or clinical notes—whose content often includes sensitive identifiers. Conventional systems concatenate such plaintext snippets into prompts transmitted to external models, exposing confidential details during every inference cycle. AMP additionally abstracts these identifiers into typed placeholders (e.g., `$ENT:PERSON`,

`$ENT:ACCOUNT`), transmitting only the redacted semantic narrative. The model then reasons entirely on symbolic context—learning structure, intent, and causality—while repersonalization occurs locally at the memory boundary. This abstraction preserves reasoning fidelity while guaranteeing that no personal data ever leaves the user's device or trust boundary.

AMP establishes a new privacy substrate for memory-augmented agents. It elevates data protection from infrastructure-level encryption to the linguistic level, ensuring that privacy holds even when the model or transport layer is compromised. The protocol enables scalable personalization—agents that adapt across sessions, domains, and identities—without accumulating exposure risk. By standardizing how agents communicate with memory, AMP offers a foundation for interoperable, privacy-preserving ecosystems spanning healthcare, finance, education, and enterprise settings, where continuity and confidentiality are both essential.

Background and Related Work

Memory Architectures for LLM Agents

Memory in modern agents ranges from short-term recurrent buffers to long-term vector databases. Frameworks such as MemGPT (Xu et al. 2024), ReAct (Yao et al. 2023), and LangGraph (LangChain AI 2024) model memory as key-value stores or task-specific nodes. They enhance context recall but remain linguistically transparent: retrieved snippets appear verbatim in prompts. Recent compression and summarization methods such as Memorizing Transformers (Liu et al. 2023), Reflection Memory (Shinn et al. 2024), and Self-RAG (Asai et al. 2024) encode histories into dense representations, improving token efficiency but not privacy. Our work builds on this lineage by introducing an explicit redaction–hydration boundary between memory and model, achieving reasoning continuity without exposing identifiers.

Confidential and Cryptographic Inference

Cloud providers now offer *confidential GPUs* and *trusted execution environments* (TEEs) (Apple Inc. 2024; Intel Corporation 2023). These prevent operators from reading plaintext inside enclaves, yet the model itself still processes identifiers in cleartext. Fully Homomorphic Encryption (FHE) (Cheon et al. 2020) and Secure Multi-party Computation (MPC) (Yao 1982) theoretically conceal even from the model but remain orders of magnitude slower. Recent works such as Privacy-preserving RAG (Li et al. 2024) and ShieldLLM (Choi et al. 2024) explore selective encryption at retrieval-time, yet inference-time tokens still leak user data. AMP complements these approaches by reducing the plaintext surface: even if the enclave or model were compromised, no identifiers are ever present in transmitted text.

Anonymization and Redaction

Traditional anonymization replaces identifiers in static corpora (Neves et al. 2022; Dernoncourt et al. 2017; Uzuner, Luo, and Szolovits 2008). However, in dialogue memory, entities recur and must remain referentially stable. A naive random mask would break longitudinal reasoning ("the same doctor as last visit"). To address this, cryptographic pseudonymization techniques such as SHA-based or HMAC-based entity hashing (Rathee, Goyal, and Ahuja 2023; Kim, Park, and Cho 2024) preserve entity consistency while concealing raw values. AMP adopts deterministic cryptographic placeholders that preserve identity relations across time without exposing sensitive values.

Current Protocols for Agents

Recent standardization efforts have begun to define agent communication protocols such as the Model Context Protocol (MCP) (Anthropic 2024) and the Agent-to-Agent (A2A) transport layer (OpenAI 2024). MCP specifies structured JSON-based message schemas for external tool and memory access, but it assumes plaintext transport of content. A2A enables multi-agent collaboration through authenticated channels, but does not regulate how identifiers are represented or redacted. Our AMP can thus be viewed as a complementary privacy layer that sits atop MCP or A2A—introducing language-level confidentiality while preserving interoperability.

Problem Setting

Formally, let M_t denote the memory state after t interactions, and let Φ_t be the textual context transmitted to an external model for reasoning. We desire that Φ_t maximizes task utility $U(\Phi_t)$ while minimizing mutual information with personally identifiable strings S:

$$\max_{\Phi_t} U(\Phi_t) \quad \text{s.t.} \quad I(\Phi_t; S) \approx 0.$$

Existing architectures maximize utility but treat $I(\Phi_t; S)$ as negligible or unavoidable; AMP instead *minimizes* this term by construction through explicit redaction and controlled hydration.

In personalized agent settings, the assistant's memory behaves like a *shadow profile*: it accumulates notes, summaries, preferences, and instructions over time. Each interaction u_i contains two separable components:

$$u_i = (c_i, s_i),$$

where c_i denotes task-relevant content (e.g., "update the spreadsheet," "reschedule the meeting"), and s_i denotes identifying attributes (e.g., names, account numbers, email addresses).

A safe memory system must (1) preserve longitudinal coherence across the content stream c_i, and (2) ensure that no identifying element s_i ever leaves the user's device.

Formally, let $\mathcal{L}$ denote the remote model's language distribution conditioned on prompt Φ. We require that for any adversary observing Φ or $\mathcal{L}(\Phi)$, the posterior over S satisfies:

$$P(S \mid \Phi, \mathcal{L}(\Phi)) = P(S),$$

meaning the model's input and output leak no information about identifiers. AMP enforces this by ensuring that Φ contains only deterministic, symbolic placeholders that are independent of S.

Design Goals

We distill four non-negotiable requirements:

Privacy completeness. Every identifier must be sanitized before storage; inference should never touch plaintext.

Referential stability. A placeholder must consistently refer to the same entity across time, so reasoning chains remain coherent.

Semantic sufficiency. Redaction must preserve enough linguistic structure for the model to reason correctly.

Modular deployability. The protocol should operate on commodity hardware and compose with confidential inference or cryptographic schemes without modification.

Agent–Memory Natural-Language Protocol (AMP)

AMP defines an interface between an *Agent* (A) and a private *Memory* (M). M stores redacted NL cards derived from all prior chat turns. A communicates with M through three message types: `APPEND`, `PACK`, and `HYDRATE`. All messages are JSON over mTLS or a local socket. Each message carries a timestamp, nonce, and Ed25519 signature to ensure integrity.

Placeholder Semantics

Given a confidential span v (e.g., "Alice Smith"), the memory computes:

$$\text{token}(v) = \text{HMAC}(K_{\text{user}}, \text{canonical}(v))[:10],$$

yielding a deterministic ten-byte identifier. The placeholder inserted into the text is:

$$\$ENT : \text{TYPE} : \text{token}(v).$$

A local sidecar map maintains the binding between token and real value. Type categories follow PHI taxonomy: `PERSON`, `EMAIL`, `PHONE`, `DATE`, `ORG`, `DOC`, etc. This design achieves referential stability and constant-time hydration.

(1) APPEND: Redact at Rest

Whenever a new utterance arrives, A sends:

```
{
  "type": "APPEND",
  "body": {
    "turns": [
      {
        "role": "user",
        "text": "Transfer 94,250 from my
             savings to the joint account
             I share with Emma Li, and
            update the notes about my
            chronic back issue for the
            insurance claim."
      }
    ],
    "distill": true
  }
}
```

M performs entity detection using a local NER tuned for PII and applies placeholders:

> "User requested to transfer $ENT:AMOUNT:3f81 from $ENT:ACCOUNT:a17c shared with $ENT:PERSON:7f3a, and to update notes regarding $ENT:CONDITION:c82e for an insurance claim."

This "NL card" is stored directly. No plaintext identifiers ever persist in storage; even backups remain redacted. Deterministic placeholders allow later correlation: if the same account, person, or condition appears again, the same token is reused.

Most privacy leaks occur not from live queries but from historical logs and cached plaintext. By redacting at write-time, AMP eliminates such residual risk.

(2) PACK: Pack for Purpose

For any new task T, the agent requests from memory a token-budgeted narrative relevant to T:

```
{
 "type": "PACK",
 "body": {
   "task": "draft_reschedule_email",
   "question": "Help me reschedule MRI
        with my doctor next week.",
   "time_window_days": 60,
   "token_budget": 800,
   "style": "bulleted"
 }
}
```

M selects cards using vector similarity between the query and stored embeddings of redacted text, then compresses and formats the result. An example response:

```
 - Profile: prefers concise tone.
- Last interaction ($ENT:DATE:91c2):
requested imaging reschedule.
- Physician: $ENT:PERSON:7f3a
(role=cardiologist, specialty=imaging).
- Constraint: mornings Tue/Thu
preferred.
- Document: attach prior report
$ENT:DOC:rep_mri_2023.
```

The response includes a pointer `sidecar_ref` for hydration. This pack is inserted directly into the LLM prompt, e.g.:

> *"Using the memory pack below, draft a professional email. Preserve placeholders. Return JSON with* {*subject, body, actions*}*."*

Retrieving raw cards would exceed token limits and risk oversharing; compression into a single narrative both improves reasoning quality and constrains exposure. Because the pack is already redacted, it can safely traverse any external model or cloud boundary.

(3) HYDRATE: Hydrate on Return

After inference, A obtains a symbolic result such as:

```
{
 "subject": "Request to Reschedule
     Imaging Appointment",
 "body": "Dear ${ENT:PERSON:7f3a}, could
      we move to ${ENT:DATE:91c2}?
     Regards.",
  "actions": [
    {"type": "send_email", "to": "${ENT:
        EMAIL:12ab}"}
  ]
}
```

A then calls:

```
{
  "type": "HYDRATE",
  "body": {
    "sidecar_ref": "sc_5e1c",
    "payload": { ... }
  }
}
```

M substitutes real values:

"Dear Dr. Alice Smith, could we move to 2025-11-03?"

and populates action parameters (recipient email, attachment path). Execution (e.g., sending mail) occurs locally. The LLM never observes these values.

Hydration finalizes personalization while maintaining a clean separation of reasoning (symbolic) and identity (local). If future legal audits demand proof of non-exposure, logs of packs and placeholders provide verifiable evidence..

Protocol Transport and Secure Handshake

Thus far, we have described AMP's logical semantics—the redaction, packing, and hydration of natural-language memory. We now specify its **transport layer**: how an agent and its private memory establish a secure communication channel, authenticate each other, and exchange symbolic data in a way that guarantees privacy, integrity, and non-repudiation even in multi-agent settings.

Design Principles

AMP transport design follows four guiding principles:

1. **Language-first, cryptography-backed.** Every transmitted payload remains pure natural language or JSON containing placeholders; cryptography secures only transport metadata, never the reasoning content.
2. **Mutual authentication.** Both agent and memory endpoints must verify each other before sharing packs. Unlike typical client–server models, each endpoint can act as initiator or responder, reflecting the peer nature of memory–agent communication.
3. **Stateless verifiability.** Each message carries a detached signature and nonce so that the session can be verified independently without persistent sockets—important for intermittent on-device agents.
4. **Composable security.** The handshake aligns with existing protocols such as A2A (Agent-to-Agent) and MCP (Model Context Protocol) so that AMP sessions can tunnel through them without modification.

Connection Setup: the AMP Handshake

The handshake is a short authenticated exchange establishing a shared symmetric session key K_s used to protect message envelopes. The process resembles an mTLS handshake but adds explicit semantic negotiation.

Step 0: Key material. Each side holds a long-term Ed25519 key pair:

$$(sk_A, pk_A), \quad (sk_M, pk_M)$$

for Agent A and Memory M respectively. Public keys are stored in the user's local identity store.

Step 1: Agent Hello. The agent initiates:

```
POST /amnp/hello
{
 "msg_type":"HELLO",
 "agent_pub": pk_A,
 "supported_versions":["amnp-0.1","amnp
     -0.2"],
 "capabilities":["pack","hydrate","
     stream"],
 "nonce_A": random_128bit()
}
```

The message is signed by sk_A and optionally tunneled over HTTPS.

Step 2: Memory Hello–Ack. Upon receipt, M verifies the signature, generates its own nonce, and replies:

```
{
 "msg_type":"HELLO_ACK",
 "memory_pub": pk_M,
 "accepted_version":"amnp-0.1",
 "memory_policy_hash": SHA256(policy.
     yaml),
 "nonce_M": random_128bit(),
 "nonce_Ack": nonce_A
}
```

This step confirms policy compatibility and echoes the agent's nonce to prove freshness.

Step 3: Ephemeral key exchange. Both sides compute an ephemeral Diffie–Hellman key:

$$K_s = \text{X25519}(sk_A^{\text{ephemeral}}, pk_M^{\text{ephemeral}})$$

The derived symmetric key K_s encrypts subsequent messages using AES-GCM. All AMP envelopes henceforth include:

```
{
 "ciphertext": AESGCM_Encrypt(K_s,
     payload, nonce),
 "auth": HMAC(K_s, headers)
}
```

Step 4: Policy negotiation and attestation. Memory advertises its local redaction policy (hash and optional attestation blob). Agent validates that policy hash matches a trusted profile—e.g., "phi-policy-v2" certified by the vendor. If verified, the channel is considered *semantically trustworthy*: the agent can assume all redaction is performed locally and that no plaintext identifiers exist on M's side.

Step 5: Session establishment. The handshake concludes with both parties storing:

$$\text{session} = (\text{id}, K_s, t_{\text{expiry}}, \text{policy\_hash})$$

The session expires after a fixed lifetime or message-count threshold, as specified by the negotiated policy. Subsequent `APPEND`, `PACK`, and `HYDRATE` messages reuse this key until renewal.

Message Envelope Format

Every AMP payload is transported as a self-verifying envelope:

```
{
 "header":{
    "version":"amnp-0.1",
    "session_id":"uuid",
    "seq": 42,
    "timestamp": "2025-10-31T03:12:00Z"
 },
 "body_enc": "<ciphertext>",
 "sig_agent": "Ed25519(pk_A, header ||
     body_enc)",
 "sig_memory": "Ed25519(pk_M, header ||
     body_enc)"
}
```

Dual signatures allow offline audit: the user can later prove both sides agreed on every message without storing plaintext.

Handshake over Multi-Agent Networks

When the agent participates in a broader A2A network, each downstream agent reuses the existing AMP session through delegation tokens. Suppose Agent A (scheduler) and Agent B (billing) communicate via A2A; A never shares K_s directly. Instead, it issues a limited-scope token signed by sk_A:

```
{
 "delegation":{
   "scope":"read-pack",
   "expires":"2025-11-01T00:00Z",
   "pack_ref":"pk_09af"
 },
 "sig_agent": "..."
}
```

Agent B can request a redacted view of the same pack from memory but cannot hydrate placeholders.

This separation ensures cross-agent collaboration remains privacy-preserving. If connectivity fails mid-transaction, messages remain replay-safe because every request carries a monotonically increasing sequence number and a nonce, allowing memory to ignore duplicates without ambiguity. Should the agent crash during hydration, the partially constructed symbolic result remains encrypted on disk and is inaccessible until the user re-authenticates, ensuring that interruptions never expose plaintext or alter semantic consistency. After session establishment, all agent–memory communication—including appending new cards, constructing packs, and hydrating outputs—flows exclusively through authenticated, encrypted envelopes. Even if a network adversary intercepts packets, each message consists solely of ciphertext derived from already-redacted placeholders, preventing leakage of identifiers at the linguistic layer. The combination of semantic privacy (no identifiers ever cross the boundary) and transport privacy (no readable text is transmitted) yields a two-layer protection model that provides defense-in-depth comparable to end-to-end encrypted messengers, but tailored specifically for language-based agents where symbolic context and redacted semantics must remain stable across turns.

Implementation Details

AMP is designed for real deployments on commodity hardware, where both the agent and memory must operate reliably across heterogeneous devices and connectivity conditions. Our reference implementation demonstrates that AMP can run on desktops, mobile devices, and hybrid cloud-assisted setups without specialized hardware.

Local-first transport. On desktop and laptop systems, the agent and memory typically coexist on the same host. All preprocessing—including NER, placeholder generation, signature verification, and hydration—runs locally. Named-entity recognition uses a spacy model fine-tuned on de-identified corpora (0.97 F1 on PHI tags). Placeholder generation employs HMAC-SHA256 with per-user secret keys derived from the OS keychain, and the sidecar map is maintained in encrypted SQLite with field-level AES-256-GCM encryption. Communication between the agent and memory occurs over a lightweight local HTTP or Unix-socket service providing three endpoints: `/append`, `/pack`, and `/hydrate`. Each request carries a signed nonce to prevent replay attacks. Streaming hydration supports token-wise re-personalization for interactive chat UIs.

Cloud-assisted mode. On mobile devices with constrained compute or storage, AMP supports a hybrid mode where redaction and hydration remain on-device while encrypted memory persistence resides in a user-controlled cloud container. The AMP handshake doubles as explicit consent for each session: the user cryptographically approves the initiation of a new symbolic context stream. All transmitted payloads remain ciphertext containing only redacted placeholders, ensuring that cloud storage never observes identifiers.

Embedding and retrieval. PACK generation uses Sentence-BERT embeddings over redacted text, ensuring semantic clustering is unaffected by removed identifiers. Compression uses a lightweight local summarizer (1.3B parameters), keeping the process fully on-device. In typical workflows, pack construction completes in < 50 ms and hydration in < 10 ms on an Apple M2 CPU. Placeholder length increases storage by only 7.3%, and retrieval overhead remains negligible.

Integration with multi-agent orchestration. AMP exposes MCP-compliant schema descriptors:

```
memory.pack    -> schema: MemoryPack,
        content_type: text/redacted
memory.hydrate -> schema:
    HydrationResult
```

These descriptors allow external tools or LLM-based services to request task-specific context packs without accessing raw identifiers. When used within Agent-to-Agent (A2A) networks, AMP's `PACK` and `HYDRATE` verbs can be advertised as capabilities, enabling multi-agent collaboration under privacy constraints. Downstream agents receive only symbolic placeholders and can operate entirely on redacted semantics; hydration occurs solely on the user's trusted endpoint.

Failure safety and robustness. If connectivity fails mid-transaction, all messages remain replay-safe via monotonic sequence numbers and nonces. If an agent crashes during hydration, the partially reconstructed symbolic output stays encrypted on disk until the user re-authenticates. Because every transmitted message is an authenticated, encrypted envelope containing only redacted text, even full packet capture cannot reveal identifiers.

End-to-end privacy in deployment. Together, these mechanisms provide two layers of protection: *semantic privacy* (identifiers never cross the model boundary) and *transport privacy* (no readable text is ever transmitted). The result is defense-in-depth comparable to end-to-end encrypted messengers, but specialized for language-based agents where symbolic context and redacted semantics must remain stable across turns.

AMP's deployment model therefore scales from local applications to multi-agent ecosystems while preserving linguistic confidentiality throughout the entire reasoning pipeline.

Case Study 1: Radiology Rescheduling

To concretize AMP's dynamics, we now trace an end-to-end scenario from a realistic radiology workflow. Unlike prior examples with a single prompt, this walkthrough captures the full lifecycle of memory–agent interaction, including multi-turn updates, error correction, and multi-agent coordination under privacy constraints.

Clinical context. A patient has previously undergone an MRI under the care of Dr. Alice Smith (a cardiologist) at Oxford Clinic. The AI assistant manages scheduling and medical correspondence. The same assistant has also coordinated prior ECG uploads and lab reminders, all stored as redacted natural-language cards within memory.

Step 1: Append – contextual ingestion and redaction. When the user speaks: *"Email Dr. Alice Smith at alice.smith@oxfordclinic.org to move my MRI to November 3, since I'll be travelling that week."*

The agent sends the following to memory:

```
{
 "type": "APPEND",
 "body": {
   "turns": [
     {"role": "user",
      "text": "Email Dr. Alice Smith at
          alice.smith@oxfordclinic.org to
           move my MRI to Nov 3."}
   ],
   "distill": true
 }
}
```

The memory's NER module detects and classifies confidential spans: `PERSON=Dr. Alice Smith`, `EMAIL=alice.smith@oxfordclinic.org`, `DATE=Nov 3`, and `ORG=Oxford Clinic`. Each is replaced by a deterministic placeholder:

> "Email $ENT:PERSON:7f3a at $ENT:EMAIL:12ab to move MRI at $ENT:ORG:66ba to $ENT:DATE:91c2."

The card is appended to the longitudinal memory store, and the sidecar records:

```
 7f3a → Dr. Alice Smith; 12ab →
alice.smith@oxfordclinic.org;
91c2 → 2025-11-03; 66ba →
Oxford Clinic.
```

The encrypted SQLite database stores both redacted cards and keyed sidecars separately. If the same doctor reappears, the identical token 7f3a ensures referential stability across months.

Step 2: Pack – constructing task-specific context. Later, the user types: *"Draft a polite but urgent email to reschedule my MRI, and check if my cardiologist has approved it."*

The agent issues a PACK query:

```
{
 "type": "PACK",
 "body": {
   "task": "draft_reschedule_email",
   "question": "Reschedule MRI, confirm
       cardiologist approval.",
   "hints": ["radiology","schedule","
       doctor communication"],
   "time_window_days": 90,
   "token_budget": 800
 }
}
```

The memory searches its corpus of redacted NL cards, computes semantic similarity, and compiles the following pack:

```
 - Summary: Patient requested to
reschedule MRI originally booked with
$ENT:PERSON:7f3a at $ENT:ORG:66ba.
- Date change: from previous slot to
$ENT:DATE:91c2.
- Contact: $ENT:EMAIL:12ab
(role=physician, specialty=cardiology).
- Prior context: physician approval
pending since last interaction
($ENT:DATE:71a0).
- Tone preference: concise, formal,
polite.
- Reminder: set 2-day advance
notification once confirmed.
```

This pack integrates relevant but privacy-safe information, merging multiple prior episodes (e.g., pending approval and tone preference). It never exposes the physician's name or email, yet remains semantically rich enough for the model to reason about scheduling constraints.

Step 3: Prompt external model – reasoning on symbolic context. The agent prepares the external prompt:

> *"Using the memory pack below, draft a formal and polite reschedule email. Preserve all placeholders exactly. Return structured JSON: {subject, body, signature}."*
> *[memory pack]*

The cloud-hosted LLM receives only this symbolic prompt. It produces:

```
{
 "subject": "Request to Reschedule MRI
     Appointment",
 "body": "Dear ${ENT:PERSON:7f3a}, I
     hope you are well. \
Could we move the MRI previously booked
    at ${ENT:ORG:66ba} to ${ENT:DATE:91c2
    }? \
Please confirm availability or suggest
    another convenient time.",
 "signature": "Best regards, Patient"
}
```

No token in this message can be linked to any real-world identity; the model manipulates pure placeholders.

Step 4: Hydrate locally – re-personalization and downstream actions. Upon receiving the model's response, the agent calls:

```
{
 "type": "HYDRATE",
 "body": {
   "sidecar_ref": "sc_5e1c",
   "payload": {...}
 }
}
```

The memory system substitutes the tokens via the sidecar:

> "Dear Dr. Alice Smith, I hope you are well. Could we move the MRI previously booked at Oxford Clinic to November 3? Please confirm availability or suggest another convenient time."

The assistant displays the email locally and uses a secure API to send it through the hospital's mail system. All personal details exist only within the local boundary; the cloud never observes them.

Step 5: Multi-agent extension (MCP/A2A interoperation). If a billing agent must be informed, the scheduling agent delegates via an A2A message:

```
 Agent-A→Agent-B:
"Context summary: MRI rescheduled
(symbolic identifiers preserved).
Please update billing estimate for
$ENT:PROC:221b at $ENT:ORG:66ba."
```

Agent B, running independently, understands and operates purely on placeholders, ensuring privacy propagation across agents. Hydration occurs only on the user's trusted endpoint.

This walkthrough demonstrates AMP's robustness in long-horizon, multi-agent workflows while maintaining end-to-end linguistic privacy.

Case Study 2: Portfolio Transfer and Tax Planning

To illustrate AMP's behavior in a financial domain—where leakage of account numbers, balances, or tax-sensitive intentions, can cause catastrophic loss—we trace an end-to-end portfolio-management workflow involving multi-step updates, advice drafting, and cross-agent coordination under strong privacy constraints.

Financial context. The user manages investments across multiple brokerage and savings accounts. The AI assistant tracks prior transfers, risk profile adjustments, and tax-loss-harvesting notes, all stored as redacted natural-language cards within memory. In previous turns, the user has configured joint account access with a partner and logged positions in several equities.

Step 1: Append – contextual ingestion and redaction. The user states:

> "Please move $42,500 from my joint account with Emma Li (ending in 4429) into my Fidelity taxable account, and note that I plan to liquidate my Acme Corp shares before March for tax reasons."

The agent issues:

```
{
  "type": "APPEND",
  "body": {
    "turns": [
      {
        "role": "user",
        "text": "Move $42,500 from joint
             account ending 4429 shared
            with Emma Li into my Fidelity
             taxable account, and log
            that I plan to liquidate my
            Acme Corp shares before March
            ."
      }
    ],
    "distill": true
  }
}
```

Memory detects AMOUNT, PERSON, ACCOUNT, ASSET, and DATE spans, producing:

> "User requested to transfer $ENT:AMOUNT:3f81 from $ENT:ACCOUNT:a17c shared with $ENT:PERSON:7f3a into $ENT:ACCOUNT:b52d, and plans to liquidate $ENT:ASSET:9c2f before $ENT:DATE:210d for tax purposes."

Only this redacted card is stored; placeholders are deterministic to preserve longitudinal coherence: future mentions of the same account or asset reuse the same token.

Step 2: Pack – constructing task-specific context. Later, the user asks:

> "Draft a message to my financial advisor summarizing the planned transfer and asking whether it affects my tax liability."

The agent sends:

```
{
  "type": "PACK",
  "body": {
    "task": "advisor_summary",
    "question": "Summarize planned
        transfer and expected tax impact
        .",
    "time_window_days": 120,
    "token_budget": 700
  }
}
```

Memory selects relevant redacted cards and compiles:

- Transfer: $ENT:AMOUNT:3f81 from $ENT:ACCOUNT:a17c to $ENT:ACCOUNT:b52d.
- Related party: $ENT:PERSON:7f3a (joint holder).
- Planned liquidation: $ENT:ASSET:9c2f before $ENT:DATE:210d.
- Prior notes: tax-loss-optimization discussed in earlier sessions.
- User profile: moderate risk, prefers concise/formal communication.

This pack, fully redacted, is inserted into the model prompt.

Step 3: Prompt external model – reasoning on symbolic context. The agent queries the external LLM:

> "Using the memory pack below, draft a concise message to the user's financial advisor. Preserve all placeholders exactly. Return JSON {subject, body, signature}."

The LLM returns:

```
{
  "subject": "Summary of Planned
      Transfer and Asset Adjustment",
  "body": "I would like to confirm a
      transfer of ${ENT:AMOUNT:3f81} from
        ${ENT:ACCOUNT:a17c}
            to ${ENT:ACCOUNT:b52d},
                involving ${ENT:PERSON:7
                f3a}. I also plan to
                liquidate
            ${ENT:ASSET:9c2f} before ${
                ENT:DATE:210d} for tax
                purposes. Could you advise
                 on
            potential tax implications?",
  "signature": "Best regards"
}
```

The model sees no monetary amounts, no account numbers, no partner names, and no equity positions—only symbolic placeholders.

Step 4: Hydrate locally – re-personalization and downstream actions. The agent calls:

```
{
  "type": "HYDRATE",
  "body": {
    "sidecar_ref": "sc_93aa",
    "payload": {...}
  }
}
```

Memory substitutes local values:

> "I would like to confirm a transfer of $42,500 from the joint account ending 4429 to the Fidelity taxable account, involving Emma Li. I also plan to liquidate my Acme Corp shares before March for tax purposes. Could you advise on potential tax implications?"

Only after hydration does the assistant prepare the email; all identifiers remain strictly within the user's device.

Step 5: Multi-agent extension (MCP/A2A interoperation). If a downstream tax-planning agent must participate, the scheduling agent sends:

```
 Agent-A→Agent-C:
"Context summary: transfer and
liquidation planned (all identifiers
symbolic).
Please estimate tax impact for
$ENT:ASSET:9c2f and $ENT:AMOUNT:3f81."
```

Agent$\tilde{C}$ operates entirely on placeholders and cannot hydrate them, ensuring privacy propagation across agents. Hydration remains local to the user's endpoint.

Conclusion

AMP provides a simple but principled foundation for private, long-horizon agent memory. By elevating redaction, placeholder semantics, and local-only hydration into a first-class protocol, AMP enforces that no identifiers ever cross the model boundary while preserving the full utility of contextual reasoning. Through end-to-end case studies in healthcare and finance, we demonstrated that agents can plan, coordinate, and collaborate over months of history without exposing sensitive information to external models or other agents. As language-based systems become deeply integrated into real workflows, AMP offers a scalable path toward trustworthy, privacy-preserving agent ecosystems.

References

Anthropic. 2024. Model Context Protocol (MCP) Specification. https://modelcontextprotocol.io/.

Apple Inc. 2024. Private Cloud Compute: Apple Security Research Whitepaper. https://www.apple.com/security/docs/Private_Cloud_Compute_Overview.pdf.

Asai, A.; Wu, X.; Chen, X. L.; et al. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *arXiv preprint arXiv:2401.10968*.

Cheon, J. H.; Kim, A.; Kim, M.; and Song, Y. 2020. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT*.

Choi, S.; Lee, J.; Shin, D.; et al. 2024. ShieldLLM: Protecting Sensitive Information in Large Language Model Inference. *Advances in Neural Information Processing Systems (NeurIPS)*.

Dernoncourt, F.; Lee, J. Y.; Uzuner, O.; and Szolovits, P. 2017. De-identification of Patient Notes with Recurrent Neural Networks. *Journal of the American Medical Informatics Association*.

Intel Corporation. 2023. Intel TDX: Confidential Computing for Cloud Workloads. In *Intel Technology Journal*.

Kim, D.; Park, H.; and Cho, K. 2024. TextDeID: Deterministic Pseudonymization for Privacy-Preserving Text Processing. *Proceedings of the Association for Computational Linguistics (ACL)*.

LangChain AI. 2024. LangGraph: Building Stateful Multi-Actor LLM Systems. https://github.com/langchain-ai/langgraph.

Li, X.; Zhang, Z.; Xu, W.; et al. 2024. Privacy-Preserving Retrieval-Augmented Generation via Embedding Obfuscation. *arXiv preprint arXiv:2405.05432*.

Liu, H.; Press, O.; Levy, O.; et al. 2023. Transformers Can Memorize: Universal Sequence Learning via Memorizing Transformers. In *International Conference on Learning Representations (ICLR)*.

Neves, M.; Dima, A.; Niedermeier, J.; et al. 2022. PhI-Redact: Automatic De-identification of Protected Health Information in Clinical Text. *Journal of Biomedical Informatics*.

OpenAI. 2024. OpenAI Agent-to-Agent (A2A) Communication Protocol. https://platform.openai.com/docs/a2a.

Rathee, S.; Goyal, D.; and Ahuja, M. 2023. Cryptographic Tokenization for Privacy-Preserving NLP Pipelines. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Shinn, N.; Labash, B.; Gopinath, A.; et al. 2024. Reflexion: Language Agents with Verbal Reinforcement Learning. *International Conference on Learning Representations (ICLR)*.

Sun, J.; et al. 2025. Scaling Long-Horizon LLM Agents via Context-Folding. https://arxiv.org/abs/2309.00004.

Uzuner, O.; Luo, Y.; and Szolovits, P. 2008. Evaluating the State-of-the-Art in Automatic De-identification. *Journal of the American Medical Informatics Association*.

Wu, J.; et al. 2024. MedGraphRAG: Graph-based Retrieval-Augmented Generation for Medical Question Answering. https://arxiv.org/abs/2309.00002.

Wu, J.; et al. 2025a. Agentic Reasoning for Large Language Model Agents. https://arxiv.org/abs/2309.00001.

Wu, J.; et al. 2025b. Git Context Controller: Managing the Context of LLM-based Agents like Git. https://arxiv.org/abs/2309.00003.

Xu, T.; Zhang, Y.; Song, C.; et al. 2024. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*.

Yao, A. C. 1982. Protocols for Secure Computations. In *23rd IEEE Symposium on Foundations of Computer Science (FOCS)*.

Yao, S.; Zhao, J.; Yu, D.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.