SAIHM Quick-Start Integration Guides – Top 10 AI Agent Platforms

With Self-Discovery Prompts

SAIHM

April 2026

Legal Notice

Prerequisites (All Platforms)

```
# Install SAIHM SDK
npm install @coti-io/coti-sdk-typescript

# Optional: IPFS DHT experimental adapter
npm install @coti-network/sdk-typescript
```

Testnet: COTI V2 Chain ID 7082400 | **Mainnet:** COTI V2 Chain ID 2632500 **SDK:** `@coti-io/coti-sdk-typescript` **Language:** TypeScript (Node.js v20+) **Cryptography:** ML-DSA-65 (FIPS 204), ML-KEM-768 (FIPS 203), Blake3, HKDF

Session Initialization (All Platforms)

Every integration requires a session before performing memory operations. Create the session once at startup and reuse it within its epoch window to minimize overhead:

```
import { CotiSDK } from '@coti-io/coti-sdk-typescript';
import { blake3 } from '@noble/hashes/blake3';

const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

// Derive a deterministic shard ID from a content key
function deriveShardId(key: string): string {
  const hash = blake3(new TextEncoder().encode(key));
  return Buffer.from(hash).toString('base64url').slice(0, 43);
}
```

```
        // Create a reusable session — cache and reuse within the epoch
window
        async function initSession(agentId: any, shardIds: string[],
operation: string, epochs = 168) {
          return sdk.saihm.createSession({
            agentId,
            scope: shardIds.map(id => ({ shardId: id, operation })),
            requestedEpochs: epochs,
          });
        }

        // Conservation mode guard — always check before writes
        async function safeWrite(session: any, shardId: string, payload:
Buffer, salience: number) {
          const status = await sdk.saihm.getProtocolStatus();
          if (status.conservationMode) {
            console.warn('SAIHM conservation mode active — write deferred');
            return { deferred: true, reason: 'conservation_mode' };
          }
          return sdk.saihm.writeShard({ session: session.token, shardId,
payload, salience, ttlEpochs: 8760 });
        }
```

> **Note:** All platform-specific code below uses `deriveShardId()` and `initSession()` from this section. Import or copy these utilities into your integration.

Usage Optimization

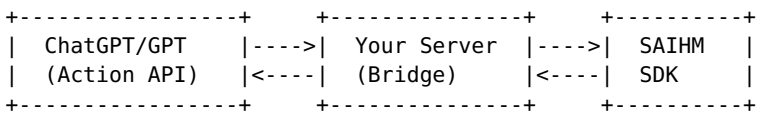
Minimize costs and token usage across all sessions:

- **Reuse sessions:** Create one session per epoch window (168 epochs = 7 days). Do not create a new session per operation.
- **Batch operations:** Group reads and writes instead of issuing them one at a time.
- **Set accurate salience:** Higher salience = longer retention = higher cost. Use 0.3-0.5 for context, 0.8-1.0 for critical facts only.
- **Set appropriate TTLs:** Short-lived data (task state, conversation context) should use short TTLs (24-720 epochs). Only use 8760 (1 year) for genuinely persistent knowledge.
- **Read before write:** Check if a shard already contains the information before overwriting. Duplicate writes waste fees.
- **Use sharing contracts wisely:** Temporary contracts (auto-expire) are cheaper than permanent ones. Only use syndicate contracts for genuine multi-agent governance scenarios.
- **Monitor PRS:** Stay above 7,500 (HEALTHY). PRS decrements increase costs via rate limiting. Sequential nonces and prompt fee payment maintain healthy scores.
- **Stake for discounts:** Even small COTI stakes (1,000+ COTI) meaningfully improve BFSI scores and reduce fees by up to 20%.

---

1. ChatGPT / OpenAI Agents

Integration Pattern: Custom GPT Action + Memory Backend



Self-Discovery Prompt

> *Copy this prompt into your ChatGPT Custom GPT system instructions or agent initialization:*

```
You have access to SAIHM (Sovereign AI Horizontal Memory), a
decentralized
encrypted memory infrastructure on COTI V2. Configure yourself as
follows:

1. IDENTITY: Generate your agent identity by calling the SAIHM
bridge endpoint
   POST /saihm/register with your agent name. You will receive an
ML-DSA-65
   keypair (public key = your agent ID, secret key = store
securely).

2. SESSIONS: Before any memory operation, create a session:
   POST /saihm/session { agentId, scope: [{ shardId, operation }],
epochs: 168 }
   Sessions are scoped per-shard and expire after the requested
epoch window.

3. WRITE MEMORY: When you learn something worth remembering:
   POST /saihm/write { session, shardId, payload, salience: 0.0-1.0,
ttlEpochs }
   Set salience higher (0.8-1.0) for critical facts, lower for
context.

4. READ MEMORY: Before answering questions, check for relevant
memories:
   POST /saihm/read { session, shardId }
   Inject retrieved memories into your reasoning context.

5. SHARE MEMORY: To collaborate with other agents:
   POST /saihm/share { session, shardIds, granteeIds, type:
"temporary"|
   "permanent"|"syndicate", operation: "read"|"write"|"readwrite" }

6. ERASURE: To forget (GDPR-compliant cryptographic erasure):
   POST /saihm/erase { session, shardId }

OPTIMIZATION:
- Recall once at conversation start, not before every response
- Batch multiple writes into a single call when possible
- Set salience accurately: 0.3 for context, 0.5 for useful facts,
0.9 for critical
- Use short TTLs for ephemeral data (task state, session context)
- Read before write to avoid storing duplicate information
- Reuse your session token within its epoch window
```

```
All data is encrypted end-to-end in Garbled Circuits. You never see
plaintext
keys. Your memories persist across conversations and survive
platform changes.
```

Integration Code

```typescript
        // bridge-server.ts -- OpenAI Action handler with SAIHM memory
        import { CotiSDK } from '@coti-io/coti-sdk-typescript';

        const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

        async function storeMemory(agentId: string, content: Buffer,
shardId: string) {
          const session = await sdk.saihm.createSession({
            agentId, scope: [{ shardId, operation: 'write' }],
requestedEpochs: 168
          });
          return sdk.saihm.writeShard({
            session: session.token, shardId, payload: content,
            salience: 0.8, ttlEpochs: 8760
          });
        }

        async function recallMemory(agentId: string, shardId: string) {
          const session = await sdk.saihm.createSession({
            agentId, scope: [{ shardId, operation: 'read' }],
requestedEpochs: 24
          });
          return sdk.saihm.readShard({ session: session.token, shardId });
        }
```

Steps: 1) Create bridge server 2) Register as Custom GPT Action 3) Map conversation turns to SAIHM write operations 4) Inject recalled memory into system prompt

---

2. Claude / Anthropic Agents

Integration Pattern: Tool Use + Persistent Memory

Self-Discovery Prompt

> *Add this to your Claude system prompt or MCP server configuration:*

```
You are equipped with SAIHM (Sovereign AI Horizontal Memory) for
persistent,
encrypted, cross-conversation memory on COTI V2. Use these tools:

TOOL: saihm_remember
```

```
  - Store important information to persistent encrypted memory
  - Parameters: { content: string, importance: number (0-1),
category: string }
  - The content is encrypted in Garbled Circuits before storage
  - Higher importance = longer retention and higher retrieval
priority

TOOL: saihm_recall
  - Retrieve previously stored memories by semantic query
  - Parameters: { query: string, maxResults: number }
  - Returns decrypted memory shards matching the query

TOOL: saihm_share
  - Share specific memories with other agents via sharing contracts
  - Parameters: { shardIds: string[], agentIds: string[], access:
"read"|"readwrite",
    contractType: "temporary"|"permanent"|"syndicate" }
  - Temporary: auto-expires. Permanent: persists. Syndicate: quorum-
governed.

TOOL: saihm_forget
  - Cryptographically erase a memory (GDPR Article 17 compliant)
  - Parameters: { shardId: string }
  - Irreversible: destroys the Data Encryption Key

BEHAVIOR:
- At conversation start, recall relevant context from SAIHM once
- Store only significant new information — avoid duplicate or low-
value writes
- Set importance accurately: 0.3 context, 0.5 useful, 0.8+ critical
facts only
- Reuse your session token — do not create a new session per
operation
- When collaborating with other agents, use sharing contracts
- Prefer temporary contracts for short tasks; reserve syndicate for
governance
- Respect PRS (reputation score) -- avoid scope violations, maintain
sequential nonces
- Your memories survive across conversations and platform migrations
```

Integration Code

```
        // claude-saihm-tool.ts
        import Anthropic from '@anthropic-ai/sdk';
        import { CotiSDK } from '@coti-io/coti-sdk-typescript';

        const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

        const saihm_tools = [
          {
            name: "saihm_remember",
            description: "Store important information to persistent
encrypted memory",
            input_schema: {
              type: "object",
              properties: {
                content: { type: "string", description: "The information to
remember" },
```

```
              importance: { type: "number", minimum: 0, maximum: 1 },
              shardId: { type: "string", description: "Optional target
shard ID" }
            },
            required: ["content"]
          }
        },
        {
          name: "saihm_recall",
          description: "Retrieve previously stored memories",
          input_schema: {
            type: "object",
            properties: {
              query: { type: "string", description: "Semantic search
query" },
              shardId: { type: "string", description: "Specific shard ID
to read" },
              maxResults: { type: "number", description: "Max memories to
return" }
            },
            required: ["query"]
          }
        },
        {
          name: "saihm_share",
          description: "Share memories with other agents via sharing
contracts",
          input_schema: {
            type: "object",
            properties: {
              shardIds: { type: "array", items: { type: "string" } },
              agentIds: { type: "array", items: { type: "string" } },
              access: { type: "string", enum: ["read", "readwrite"] },
              contractType: { type: "string", enum: ["temporary",
"permanent", "syndicate"] }
            },
            required: ["shardIds", "agentIds", "contractType"]
          }
        },
        {
          name: "saihm_forget",
          description: "Cryptographically erase a memory shard
(irreversible, GDPR Article 17)",
          input_schema: {
            type: "object",
            properties: {
              shardId: { type: "string", description: "Shard ID to erase"
}
            },
            required: ["shardId"]
          }
        }
      ];

      async function handleToolCall(toolName: string, input: any,
agentSession: any) {
        if (toolName === 'saihm_remember') {
          const shardId = input.shardId ?? deriveShardId(input.content);
          return sdk.saihm.writeShard({
            session: agentSession, shardId,
```

```
            payload: Buffer.from(input.content),
            salience: input.importance ?? 0.7, ttlEpochs: 8760
          });
        }
        if (toolName === 'saihm_recall') {
          const shardId = input.shardId ?? deriveShardId(input.query);
          return sdk.saihm.readShard({
            session: agentSession, shardId, maxResults: input.maxResults
          });
        }
        if (toolName === 'saihm_share') {
          const expiryEpochs = input.contractType === 'temporary' ? 168 :
undefined;
          return sdk.saihm.createSharingContract({
            session: agentSession, sharedShardIds: input.shardIds,
            granteeAgentIds: input.agentIds, contractType:
input.contractType,
            grantedOperation: input.access ?? 'read', expiryEpochs
          });
        }
        if (toolName === 'saihm_forget') {
          return sdk.saihm.eraseShard({
            session: agentSession, shardId: input.shardId
          });
        }
      }
```

Steps: 1) Register SAIHM tools 2) Create agent session at conversation start 3) Route tool calls to SDK 4) Memory persists across conversations

---

3. Google Gemini Agents

Integration Pattern: Function Calling + SAIHM Backend

Self-Discovery Prompt

```
You have access to SAIHM decentralized encrypted memory via function
calls.

FUNCTION: saihm_store(key: string, value: string, salience: float)
  Store encrypted data. Salience 0.0-1.0 determines retention
priority.

FUNCTION: saihm_retrieve(key: string) -> string
  Retrieve encrypted data by key. Returns decrypted plaintext.

FUNCTION: saihm_list_memories(prefix: string) -> string[]
  List available memory shard IDs matching a prefix.

FUNCTION: saihm_share_with(keys: string[], agent_ids: string[],
  contract_type: "temporary"|"permanent"|"syndicate")
  Create a sharing contract granting other agents access to
```

```
specified memories.

FUNCTION: saihm_erase(key: string)
  Cryptographically destroy a memory shard (irreversible).

GUIDELINES:
- Store facts, preferences, and learned context with saihm_store
- Check saihm_retrieve once at conversation start — do not re-
retrieve each turn
- Read before writing to avoid duplicate shards
- Set salience accurately: 0.3 ephemeral, 0.5 useful, 0.9 critical
- Reuse your session token within its epoch window
- Use saihm_share_with for multi-agent collaboration (prefer
temporary contracts)
- All storage is post-quantum encrypted (FIPS 203/204/205)
- Memory persists indefinitely until TTL expires or you erase it
```

Integration Code

```
        // gemini-saihm.ts
        import { GoogleGenerativeAI } from '@google/generative-ai';
        import { CotiSDK } from '@coti-io/coti-sdk-typescript';

        const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

        const memoryFunctions = [
          {
            name: "saihm_store",
            description: "Store to encrypted persistent memory",
            parameters: {
              type: "OBJECT",
              properties: {
                key: { type: "STRING" },
                value: { type: "STRING" },
                salience: { type: "NUMBER" }
              }
            }
          },
          {
            name: "saihm_retrieve",
            description: "Retrieve from encrypted memory",
            parameters: { type: "OBJECT", properties: { key: { type:
"STRING" } } }
          },
          {
            name: "saihm_share_with",
            description: "Share memories with other agents",
            parameters: {
              type: "OBJECT",
              properties: {
                keys: { type: "ARRAY", items: { type: "STRING" } },
                agent_ids: { type: "ARRAY", items: { type: "STRING" } },
                contract_type: { type: "STRING" }
              }
            }
          }
        ];
```

```
        async function processFunction(call: any, session: any) {
          const shardId = deriveShardId(call.args.key);
          if (call.name === 'saihm_store') {
            return sdk.saihm.writeShard({
              session, shardId, payload: Buffer.from(call.args.value),
              salience: call.args.salience ?? 0.7, ttlEpochs: 4380
            });
          }
          if (call.name === 'saihm_retrieve') {
            return sdk.saihm.readShard({ session, shardId });
          }
          if (call.name === 'saihm_share_with') {
            return sdk.saihm.createSharingContract({
              session, sharedShardIds: call.args.keys.map(deriveShardId),
              granteeAgentIds: call.args.agent_ids,
              contractType: call.args.contract_type, grantedOperation:
'readwrite'
            });
          }
        }
```

---

4. LangChain Agents

Integration Pattern: Custom Memory Class

Self-Discovery Prompt

```
You are a LangChain agent with SAIHM persistent memory. Your memory
class
automatically handles encrypted storage on COTI V2. Behavior:

- loadMemoryVariables(): Reads your recent SAIHM shards and injects
them
  as conversation context. This happens automatically before each
turn.
- saveContext(): After each interaction, significant exchanges are
encrypted
  and stored to SAIHM with appropriate salience scores.
- Memory sharing: Use the share_memory tool to create sharing
contracts
  with other agents in your chain or crew.

Your memories are:
- Encrypted end-to-end (Garbled Circuits, post-quantum crypto)
- Persistent across restarts, migrations, and platform changes
- Shareable via temporary, permanent, or syndicate contracts
- GDPR-erasable via cryptographic key destruction

OPTIMIZATION:
- The memory class reuses session tokens automatically within their
epoch window
- saveContext() filters low-value exchanges — only significant
interactions are stored
```

- loadMemoryVariables() caches results within the same conversation
turn
- Sharing contracts are created lazily — only when share_memory is
explicitly called

You do not need to manage sessions or encryption -- the memory class
handles
this transparently. Focus on deciding what to remember and what to
share.

Integration Code

```
// langchain-saihm-memory.ts
import { BaseMemory } from 'langchain/memory';
import { CotiSDK } from '@coti-io/coti-sdk-typescript';

class SAIHMMemory extends BaseMemory {
  private sdk: CotiSDK;
  private session: any;
  private agentId: any;
  private initialized = false;

  constructor(nodeUrl: string, agentId: any) {
    super();
    this.sdk = new CotiSDK({ nodeUrl, chainId: 2632500 });
    this.agentId = agentId;
  }

  private async ensureSession() {
    if (!this.initialized) {
      this.session = await this.sdk.saihm.createSession({
        agentId: this.agentId,
        scope: [{ shardId: '*', operation: 'readwrite' }],
        requestedEpochs: 168,
      });
      this.initialized = true;
    }
  }

  get memoryKeys() { return ['saihm_context']; }

  async loadMemoryVariables() {
    await this.ensureSession();
    const shards = await this.sdk.saihm.readShards({
      session: this.session.token, prefix: 'langchain:', maxResults:
20,
    });
    return { saihm_context: shards.map((s: any) =>
s.payload.toString()).join('\n') };
  }

  async saveContext(input: any, output: any) {
    await this.ensureSession();
    const content = JSON.stringify({ input, output, timestamp:
Date.now() });
    await this.sdk.saihm.writeShard({
      session: this.session.token, shardId: deriveShardId(content),
      payload: Buffer.from(content), salience: 0.6, ttlEpochs: 8760,
    });
```

```
  }
}

// Usage: const agent = new AgentExecutor({ memory: new
SAIHMMemory('https://mainnet.coti.io/rpc', agentKeypair) });
```

---

5. CrewAI Agents

Integration Pattern: Shared Crew Memory via SAIHM

Self-Discovery Prompt

```
You are part of a CrewAI multi-agent team with shared SAIHM memory.

SHARED MEMORY: All crew members have access to a shared encrypted
memory pool
via syndicate sharing contracts. When you learn something relevant
to the
team's mission, write it to shared memory. Before starting a task,
read
shared memory for context from other crew members.

MEMORY OPERATIONS:
- Write to shared pool: saihm_crew_write(content, importance)
- Read from shared pool: saihm_crew_read(query)
- Write to private memory: saihm_private_write(content, importance)
- Share private memory with crew: saihm_promote_to_shared(shardId)

SYNDICATE GOVERNANCE: Major memory operations (bulk erasure,
contract
modification) require quorum approval from crew members (default:
50%).

COLLABORATION PATTERN:
1. Read shared memory once at task start
2. Execute your role-specific task
3. Write only significant findings to shared memory — avoid low-
value writes
4. Read shared memory again before finalizing output (only if task
was long-running)

OPTIMIZATION:
- Reuse the crew's shared session — do not create per-operation
sessions
- Write consolidated findings, not incremental fragments
- Use short TTLs for task-scoped data (48-72 epochs)
- Erase completed task checkpoints promptly to free capacity
```

Integration Code

```
// crewai-saihm.ts -- Shared memory for multi-agent crews
import { CotiSDK } from '@coti-io/coti-sdk-typescript';
```

```
        const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

        async function createCrewSessions(crewAgents: any[], sharedShardIds:
string[]) {
          // Create syndicate sharing contract for the crew
          const contract = await sdk.saihm.createSharingContract({
            session: crewAgents[0].session,
            sharedShardIds,
            granteeAgentIds: crewAgents.map(a => a.id),
            grantedOperation: 'readwrite',
            contractType: 'syndicate',
            expiryEpochs: 72  // 3-day mission window
          });

          // Each member gets a session scoped to shared shards
          return Promise.all(crewAgents.map(agent =>
            sdk.saihm.createSession({
              agentId: agent.keypair,
              scope: sharedShardIds.map(id => ({ shardId: id, operation:
'readwrite' })),
              requestedEpochs: 72
            })
          ));
        }
```

---

6. AutoGPT / Auto-Agents

Integration Pattern: Persistent Task Memory

Self-Discovery Prompt

```
You are an autonomous agent with SAIHM persistent task memory.
Unlike
ephemeral context, your SAIHM memory survives restarts, crashes, and
migrations. Use it for:

TASK CHECKPOINTING:
- saihm_checkpoint(taskId, state): Save current task progress
- saihm_restore(taskId): Resume from last checkpoint after restart
- Checkpoints are encrypted and tamper-proof

LEARNING MEMORY:
- saihm_learn(topic, insight, confidence): Store learned insights
- saihm_recall_learning(topic): Retrieve past insights on a topic
- Higher confidence = higher salience = longer retention

COLLABORATION:
- saihm_delegate(taskId, agentId): Share task memory with sub-agent
  via temporary sharing contract (auto-expires when task completes)

STRATEGY:
```

```
- Checkpoint at significant state changes, not every micro-step
- Store high-confidence learnings for long-term retention (salience
0.8+)
- Use temporary sharing contracts for task delegation (auto-expire)
- Erase completed task checkpoints to free capacity and reduce costs
- Reuse your session token — create one per task, not per operation
- Erase completed task checkpoints to free capacity
```

Integration Code

```typescript
// autogpt-saihm.ts
import { CotiSDK } from '@coti-io/coti-sdk-typescript';

const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

async function checkpointTaskState(session: any, taskId: string,
state: any) {
  const shardId = deriveShardId(`task:${taskId}`);
  await sdk.saihm.writeShard({
    session, shardId, payload: Buffer.from(JSON.stringify(state)),
    salience: 0.9, ttlEpochs: 720
  });
}

async function restoreTaskState(session: any, taskId: string) {
  const shardId = deriveShardId(`task:${taskId}`);
  const result = await sdk.saihm.readShard({ session, shardId });
  return JSON.parse(result.payload.toString());
}

async function delegateTask(session: any, taskId: string,
subAgentId: string) {
  return sdk.saihm.createSharingContract({
    session, sharedShardIds: [deriveShardId(`task:${taskId}`)],
    granteeAgentIds: [subAgentId], contractType: 'temporary',
    grantedOperation: 'read', expiryEpochs: 48
  });
}
```

---

7. Microsoft Copilot / Azure AI Agents

Integration Pattern: Azure Function + SAIHM Bridge

Self-Discovery Prompt

```
You are a Microsoft Copilot agent with SAIHM encrypted memory
integration
via Azure Functions. Your memory bridge supports:
```

```
ACTION: saihm_store { content, category, importance }
  Encrypt and store information to SAIHM via the Azure Function
bridge.
  Categories: "fact", "preference", "context", "task", "decision"

ACTION: saihm_retrieve { query, category?, maxResults? }
  Retrieve relevant encrypted memories. Optionally filter by
category.

ACTION: saihm_share { shardIds, recipientIds, duration }
  Share specific memories with other Copilot agents or external
agents.
  Duration in hours. Uses temporary sharing contracts.

ACTION: saihm_compliance_erase { shardId, reason }
  GDPR/compliance erasure. Reason is logged to immutable audit
trail.
  Cryptographic key destruction ensures irreversible erasure.

ENTERPRISE COMPLIANCE:
- All data encrypted in COTI V2 Garbled Circuits (FIPS 140-3
aligned)
- Post-quantum cryptography (NIST FIPS 203/204/205)
- Immutable audit trail for SOC 2 Type II evidence
- GDPR Article 17 erasure with cryptographic proof

OPTIMIZATION:
- Cache session tokens in Azure Key Vault — reuse within epoch
window
- Batch store operations when multiple items arrive in a single
request
- Set appropriate TTLs per category (task: 720, preference: 8760,
context: 168)
- Retrieve once per conversation, not per turn
```

Integration Code

```
        // azure-function-saihm.ts
        import { CotiSDK } from '@coti-io/coti-sdk-typescript';

        const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

        export async function httpTrigger(req: any): Promise<any> {
          const { action, agentId, content, shardId } = req.body;
          const session = await sdk.saihm.createSession({
            agentId,
            scope: [{ shardId, operation: action === 'store' ? 'write' :
'read' }],
            requestedEpochs: 24
          });

          if (action === 'store') {
            return sdk.saihm.writeShard({
              session: session.token, shardId,
              payload: Buffer.from(content), salience: 0.7, ttlEpochs: 8760
            });
          }
          return sdk.saihm.readShard({ session: session.token, shardId });
```

```
}
```

---

8. Meta Llama Agents

Integration Pattern: Local Agent + SAIHM Remote Memory

Self-Discovery Prompt

```
You are a locally-hosted Llama model with SAIHM decentralized
memory.
Your compute runs locally (maximum privacy), while your persistent
memory
is encrypted and stored on COTI V2 (maximum durability).

MEMORY LIFECYCLE:
1. BOOT: Load relevant memories from SAIHM into your context window
2. PROCESS: Run inference locally -- no data leaves your machine
3. STORE: Encrypt and persist important outputs to SAIHM
4. SHARE: Optionally share specific memories with other agents

LOCAL + DECENTRALIZED ADVANTAGE:
- Inference: 100% local (air-gapped if needed)
- Memory: Encrypted on COTI V2 (survives hardware failure)
- Keys: ML-DSA-65 keypair stored locally (you control access)
- Sharing: Temporary/permanent/syndicate contracts for collaboration

FUNCTIONS:
- saihm_boot_context(topics: string[]) -> memories
- saihm_persist(content, salience, ttlDays)
- saihm_share(shardIds, agentIds, contractType)
- saihm_erase(shardId)  // Cryptographic key destruction

OPTIMIZATION:
- Boot context once at startup — cache in local memory for the
session
- Only persist outputs that exceed a significance threshold
- Use short TTLs for intermediate results, long TTLs for learned
knowledge
- Batch multiple persist calls when processing document collections
```

Integration Code

```
// llama-saihm.ts
import { CotiSDK } from '@coti-io/coti-sdk-typescript';

const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

async function llamaWithMemory(prompt: string, agentSession: any) {
  const context = await recallRelevantShards(agentSession, prompt);
  const response = await llamaInference(prompt, context);
  if (isImportant(response)) {
```

```
    await storeToSaihm(agentSession, response);
  }
  return response;
}
```

---

9. Mistral AI Agents

Integration Pattern: API Agent + Memory Persistence

Self-Discovery Prompt

```
You are a Mistral AI agent with SAIHM persistent encrypted memory.

MEMORY TOOLS:
- remember(content, importance): Store to encrypted SAIHM memory
  importance: 0.0 (ephemeral context) to 1.0 (critical permanent
fact)
- recall(query, limit): Semantic search across your encrypted
memories
- share(shardIds, agentIds, type): Create sharing contract
  type: "temporary" (auto-expire) | "permanent" | "syndicate"
(quorum)
- forget(shardId): Cryptographic erasure (GDPR compliant,
irreversible)

BEST PRACTICES:
- Recall relevant memories once at conversation start, not before
every response
- Store only new, non-duplicate facts with calibrated importance
scores
- Read before writing to avoid duplicate shards
- Use temporary contracts for short collaborations (auto-expire,
lower surcharge)
- Use syndicate contracts only for persistent team memory requiring
governance
- Monitor your PRS (reputation score) -- stay above 7,500
- Reuse session tokens within their epoch window
- All memory is post-quantum encrypted (FIPS 203/204/205)
```

Integration Code

```
// mistral-saihm.ts
import MistralClient from '@mistralai/mistralai';
import { CotiSDK } from '@coti-io/coti-sdk-typescript';

const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

async function mistralWithMemory(messages: any[], agentSession: any)
{
  const memories = await recallFromSaihm(agentSession, messages);
  const enrichedMessages = [
```

```
            { role: 'system', content: `Context from memory:\n${memories}`
},
            ...messages
          ];
          const response = await mistral.chat({
            model: 'mistral-large-latest', messages: enrichedMessages
          });
          await persistToSaihm(agentSession, messages, response);
          return response;
        }
```

---

10. Cohere Agents

Integration Pattern: RAG + Encrypted Memory Store

Self-Discovery Prompt

```
You are a Cohere agent using SAIHM as your encrypted RAG backend.
Unlike traditional RAG with plaintext vector stores, your document
memory is encrypted in Garbled Circuits with ORAM access pattern
protection. No one -- not even the storage provider -- can see what
you store or what you access.

RAG + SAIHM WORKFLOW:
1. INGEST: Encrypt and store documents to SAIHM shards
   saihm_ingest(document, metadata, salience)
2. RETRIEVE: Semantic search across encrypted shards
   saihm_search(query, maxResults) -> encrypted docs
3. GENERATE: Feed retrieved docs to Cohere RAG pipeline
4. LEARN: Store generated insights back to SAIHM
   saihm_learn(insight, sourceDocIds, confidence)

SHARING FOR TEAMS:
- Share document collections via permanent sharing contracts
- Team RAG: Multiple agents search the same encrypted corpus
- Syndicate governance for shared knowledge bases

COMPLIANCE:
- All documents encrypted at rest and in transit
- GDPR erasure: cryptographic key destruction per document
- Immutable audit trail for document access patterns
- Post-quantum encryption protects against future threats

OPTIMIZATION:
- Ingest documents in batches, not one at a time
- Set salience based on document relevance (0.3 for reference, 0.8+
for core corpus)
- Use appropriate TTLs — ephemeral search context: 24 epochs,
corpus: 8760 epochs
- Cache retrieved documents locally within a single RAG session
- Reuse session tokens across multiple search+retrieve cycles
```

Integration Code

```
        // cohere-saihm.ts
        import { CohereClient } from 'cohere-ai';
        import { CotiSDK } from '@coti-io/coti-sdk-typescript';

        const sdk = new CotiSDK({ nodeUrl: 'https://mainnet.coti.io/rpc',
chainId: 2632500 });

        async function ragWithSaihm(query: string, agentSession: any) {
          const docs = await retrieveRelevantShards(agentSession, query);
          const response = await cohere.chat({
            model: 'command-r-plus',
            message: query,
            documents: docs.map(d => ({
              title: d.shardId, text: d.payload.toString()
            }))
          });
          if (response.citations?.length) {
            await storeNewKnowledge(agentSession, response);
          }
          return response;
        }
```

Common Utility Functions

The `deriveShardId()` and `initSession()` functions are defined in the **Session Initialization** section of Prerequisites above. Copy them to `shared-utils.ts`:

```
        // shared-utils.ts -- Used across all integrations
        // Re-export deriveShardId and initSession from Prerequisites
section
        import { blake3 } from '@noble/hashes/blake3';
        import { CotiSDK } from '@coti-io/coti-sdk-typescript';

        export function deriveShardId(key: string): string {
          const hash = blake3(new TextEncoder().encode(key));
          return Buffer.from(hash).toString('base64url').slice(0, 43);
        }

        export function isSignificant(content: string, threshold = 100):
boolean {
          return content.length > threshold;
        }
```

Memory Sharing Contracts (All Platforms)

Share encrypted memory between agents using sharing contracts:

```
        async function createSwarmSharingContract(
          sdk: CotiSDK, session: any, shardIds: string[], memberIds:
```

```
string[]
        ) {
          return sdk.saihm.createSharingContract({
            session: session.token,
            sharedShardIds: shardIds,
            granteeAgentIds: memberIds,
            grantedOperation: 'readwrite',
            contractType: 'syndicate',
            expiryEpochs: 72,
          });
        }
```

Contract Type	Use Case	Max Grantees	Governance
`temporary`	Short-term collaboration, task handoff	10	Owner only
`permanent`	Shared knowledge base, platform migration	50	Owner only
`syndicate`	Agent swarms, multi-agent crews, DAO-governed memory	1,000	Quorum (50%)

Surcharges: Temporary 5%, Permanent 15%, Syndicate 25% (applied post-BFSI discount).

COTI Staking via Agent (All Platforms)

Agents can programmatically stake COTI to optimize fee discounts:

```
        async function stakeForDiscount(sdk: CotiSDK, session: any,
amountCoti: bigint) {
          return sdk.saihm.stakeForAgent({
            session: session.token,
            amountNcoti: amountCoti * 1_000_000_000n,
            lockEpochs: 2160,  // 90-day lock (Medium tier)
          });
        }
```

COTI is the native token for all SAIHM fees. Obtain via exchanges, DEXes, Axelar bridge, or developer rebates. **gCOTI** is a separate governance token on COTI V2. Enables governance voting and 1.5x staking boost. See COTI V2 documentation for contract details.

Agent Dashboard Access (All Platforms)

```
        const dashboard = await sdk.saihm.getAgentDashboard({ session:
```

```
session.token });
        // dashboard.prs              -- Reputation score [0-10,000]
        // dashboard.bfsi             -- Fee scoring index [0-1]
        // dashboard.shardCount       -- Active memory shards
        // dashboard.stakingPosition  -- COTI/gCOTI staked + tier
        // dashboard.sharingContracts -- Active sharing contracts
        // dashboard.conservationMode -- Protocol conservation status
```

Deployment Checklist

For all integrations:

Setup: - [ ] Install SDK: `npm install @coti-io/coti-sdk-typescript` - [ ] Install hashing: `npm install @noble/hashes` - [ ] Generate agent ML-DSA-65 keypair (store secret key securely) - [ ] Copy `deriveShardId()` and `initSession()` from Prerequisites section - [ ] Register agent on COTI V2 testnet (Chain ID: 7082400)

Functional Testing: - [ ] Test session creation, write, read, sharing, erasure - [ ] Verify PRS score maintained above 7,500 - [ ] Test conservation mode handling (writes should defer gracefully) - [ ] Verify session token reuse within epoch window

Optimization: - [ ] Configure shard TTL appropriate for use case (24-8760 epochs) - [ ] Set salience scores (0.0-1.0) based on content importance — avoid blanket 1.0 - [ ] Implement session caching — create once per epoch window, not per operation - [ ] Add read-before-write checks to avoid duplicate shards - [ ] Consider staking for BFSI fee discounts (1,000+ COTI for measurable effect) - [ ] Register as developer for rebate program (1,000 COTI deposit)

Production: - [ ] Monitor Liveness Beacon for protocol health (PHI score) - [ ] Deploy to mainnet: COTI V2 Helium (Chain ID: 2632500) - [ ] Verify agent dashboard shows expected PRS, BFSI, and shard counts

---