Human Memory Layer (HML) - Technical Documentation

Version 1.0.0 | January 2025

Table of Contents

Executive Summary

The Human Memory Layer (HML) is a foundational infrastructure protocol that provides persistent, searchable, and transferable memory capabilities for AI agents and human users. It enables AI systems to maintain context across conversations, platforms, and time, while ensuring users retain complete ownership and control of their cognitive data.

Key Principles

1. **User Sovereignty**: Users own their memories completely
2. **Protocol, Not Platform**: Open standards enable interoperability
3. **Privacy by Design**: Local-first architecture with optional federation
4. **AI-Native**: Designed for seamless AI agent integration
5. **Decentralized**: No single point of failure or control

Core Components

- **MTAP (Memory Transfer & Access Protocol)**: Standardized memory format
- **MCP (Model Context Protocol)**: AI agent communication bridge

- **Emma**: Reference implementation and user-facing application
- **Federation Network**: Distributed memory sharing and replication

---

Architecture Overview

System Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    USER LAYER                               │
├─────────────────────────────────────────────────────────────┤
│  Emma Apps │ Chrome Ext │ Mobile │ Desktop │ Web │ API      │
├─────────────────────────────────────────────────────────────┤
│                  PROTOCOL LAYER                             │
├─────────────────────────────────────────────────────────────┤
│         MTAP             │            MCP                   │
│  (Memory Format & Transfer) │  (AI Agent Communication)     │
├─────────────────────────────────────────────────────────────┤
│                  STORAGE LAYER                              │
├─────────────────────────────────────────────────────────────┤
│  Local Storage │ Edge Nodes │ Federation │ Cloud (Optional) │
├─────────────────────────────────────────────────────────────┤
│                    AI LAYER                                 │
├─────────────────────────────────────────────────────────────┤
│  ChatGPT │ Claude │ Gemini │ Local LLMs │ Custom Agents     │
└─────────────────────────────────────────────────────────────┘
```

Data Flow Architecture

```
User Input → Capture → Process → Store → Index → Retrieve → Present
     ↓          ↓         ↓        ↓       ↓        ↓          ↓
   [Raw]    [Extract] [Enhance] [MTAP] [Search] [Query]   [Format]
            Metadata  Semantic  Format  Index   Match     Display
```

Memory Lifecycle

```
1. CREATION
   Input → Validation → Enrichment → Encoding → Storage

2. RETRIEVAL
   Query → Search → Rank → Filter → Format → Deliver

3. FEDERATION
   Local → Sync → Replicate → Verify → Distribute

4. EVOLUTION
   Access → Learn → Update → Version → Archive
```

Core Protocols

MTAP (Memory Transfer & Access Protocol)

Protocol Specification v1.0

```json
```

```
{
  "mtap_version": "1.0.0",
  "memory": {
    "header": {
      "id": "uuid-v4",
      "version": "1.0.0",
      "created": "ISO-8601",
      "modified": "ISO-8601",
      "creator": "did:method:identifier",
      "signature": "base64-signature",
      "contentHash": "sha256-hash",
      "protocol": "MTAP/1.0"
    },
    "core": {
      "type": "text|image|audio|video|composite",
      "content": "actual-content-or-reference",
      "encoding": "UTF-8|base64|custom",
      "encrypted": false,
      "compression": null,
      "size": 1024
    },
    "semantic": {
      "summary": "AI-generated-summary",
      "keywords": ["keyword1", "keyword2"],
      "entities": [
        {
          "type": "person|place|org|date|url",
          "value": "entity-value",
          "confidence": 0.95
        }
      ],
      "emotions": ["joy", "curiosity"],
      "topics": ["AI", "memory"],
      "embeddings": {
        "model": "embedding-model-id",
        "vector": [0.1, 0.2, ...],
        "dimensions": 1536
      }
    },
    "relations": {
      "previous": "memory-id",
      "next": "memory-id",
      "parent": "memory-id",
    "children": ["memory-id-1", "memory-id-2"],
    "related": ["memory-id-3", "memory-id-4"],
    "references": ["url1", "url2"],
    "threads": ["thread-id-1"]
  },
  "permissions": {
    "owner": "did:method:identifier",
    "public": false,
    "shared": [
      {
        "did": "did:method:identifier",
        "permissions": ["read", "reference"],
        "expiry": "ISO-8601"
      }
    ],
    "agents": [
      {
        "agentId": "agent-identifier",
        "permissions": ["read", "write", "share"],
        "granted": "ISO-8601",
        "expiry": null
      }
    ],
    "encryption": {
      "method": "AES-256-GCM",
      "keyId": "key-identifier"
    }
  },
  "metadata": {
    "source": "chatgpt|claude|manual|api",
    "application": "emma-lite",
    "deviceId": "device-uuid",
    "location": {
      "lat": 0.0,
      "lon": 0.0,
      "accuracy": 10,
      "name": "location-name"
    },
    "context": {
      "activity": "conversation|research|note",
      "project": "project-id",
      "tags": ["tag1", "tag2"]
    },
    "custom": {}
    },
    "verification": {
      "witnesses": [
        {
          "did": "did:method:identifier",
          "signature": "base64-signature",
          "timestamp": "ISO-8601"
        }
      ],
      "blockchain": {
        "network": "ethereum|bitcoin|custom",
        "transaction": "tx-hash",
        "block": 12345678
      }
    }
  }
}
```

MTAP Operations

1. Create Memory

javascript

```
POST /mtap/memory
{
  "content": "Memory content",
  "type": "text",
  "metadata": {...}
}

Response:
{
  "id": "mem_abc123",
  "address": "mtap://memory/sha256hash",
  "created": "2025-01-20T10:00:00Z"
}
```

2. Retrieve Memory

javascript

```
GET /mtap/memory/{id}
GET mtap://memory/{contentHash}

Response: Full MTAP memory object
```

3. Update Memory

```javascript
PATCH /mtap/memory/{id}
{
  "operations": [
    {"op": "add", "path": "/metadata/tags", "value": ["new-tag"]},
    {"op": "replace", "path": "/core/content", "value": "updated"}
  ]
}
```

4. Search Memories

```javascript
POST /mtap/search
{
  "query": "search terms",
  "filters": {
    "type": "text",
    "dateRange": {
      "start": "2024-01-01",
      "end": "2024-12-31"
    },
    "source": ["chatgpt", "claude"]
  },
  "limit": 20,
  "offset": 0
}
```

MCP (Model Context Protocol)

Protocol Bridge Specification

```javascript
```

```
// MCP Request Format
{
  "protocol": "MCP/1.0",
  "operation": "getContext",
  "parameters": {
    "query": "User intent or question",
    "maxTokens": 4000,
    "relevanceThreshold": 0.7,
    "timeRange": {
      "start": "ISO-8601",
      "end": "ISO-8601"
    },
    "sources": ["chatgpt", "claude", "manual"],
    "format": "markdown|json|xml"
  },
  "authentication": {
    "method": "bearer",
    "token": "jwt-token"
  }
}

// MCP Response Format
{
  "protocol": "MCP/1.0",
  "context": {
    "memories": [
      {
        "id": "memory-id",
        "content": "Memory content",
        "relevance": 0.95,
        "timestamp": "ISO-8601",
        "source": "chatgpt",
        "metadata": {...}
      }
    ],
    "summary": "Contextual summary",
    "tokens": 3500,
    "truncated": false
  },
  "performance": {
    "searchTime": 45,
    "processingTime": 120,
    "memoryCount": 15
  }
}
```

MCP Integration Points

1. **Direct API Access**

```python
# Python SDK Example
from hml import MCP

mcp = MCP(api_key="your-api-key")
context = mcp.get_context(
    query="What did we discuss about quantum computing?",
    max_tokens=2000
)
```

2. **LangChain Integration**

```python
from langchain.memory import HMLMemory

memory = HMLMemory(
    api_key="your-api-key",
    user_id="user-did"
)

chain = ConversationChain(
    llm=llm,
    memory=memory
)
```

3. **OpenAI Function Calling**

```javascript
```

```
{
  "name": "query_hml_memory",
  "description": "Search user's memory layer",
  "parameters": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "Search query"
      }
    }
  }
}
```

Implementation Specifications

Storage Implementations

1. Local Storage (Browser)

javascript

```
// IndexedDB Schema
{
  databases: {
    EmmaLiteDB: {
      version: 2,
      stores: {
        memories: {
          keyPath: "id",
          indexes: ["timestamp", "source", "type"]
        },
        mtap_memories: {
          keyPath: "header.id",
          indexes: ["header.created", "header.contentHash"]
        },
        embeddings: {
          keyPath: "memoryId",
          indexes: ["timestamp"]
        }
      }
    }
  }
}
```

2. Edge Storage (Cloudflare Workers)

javascript

```
// Durable Objects Schema
class MemoryStore {
  constructor(state, env) {
    this.state = state;
    this.storage = state.storage;
  }

  async store(memory) {
    const key = `mem:${memory.header.id}`;
    await this.storage.put(key, memory);
    await this.updateIndex(memory);
  }

  async retrieve(id) {
    return await this.storage.get(`mem:${id}`);
  }

  async search(query, options) {
    // Vector search implementation
    const embeddings = await this.storage.list({
      prefix: "emb:",
      limit: 1000
    });

    return this.rankByRelevance(embeddings, query);
  }
}
```

3. Federation Storage (IPFS)

javascript

```
// IPFS Integration
import { create } from 'ipfs-http-client';

class IPFSMemoryStore {
  constructor() {
    this.ipfs = create({
      host: 'ipfs.infura.io',
      port: 5001,
      protocol: 'https'
    });
  }

  async store(memory) {
    const { cid } = await this.ipfs.add(
      JSON.stringify(memory)
    );

    await this.ipfs.pin.add(cid);
    return `ipfs://${cid}`;
  }

  async retrieve(cid) {
    const stream = this.ipfs.cat(cid);
    let data = '';

    for await (const chunk of stream) {
      data += chunk.toString();
    }

    return JSON.parse(data);
  }
}
```

Search & Retrieval

Vector Search Implementation

python

```
import numpy as np
from sentence_transformers import SentenceTransformer

class VectorSearch:
    def __init__(self):
        self.model = SentenceTransformer('all-MiniLM-L6-v2')
        self.index = None

    def create_embeddings(self, texts):
        return self.model.encode(texts)

    def build_index(self, memories):
        texts = [m['content'] for m in memories]
        embeddings = self.create_embeddings(texts)

        # Using FAISS for efficient similarity search
        import faiss
        dimension = embeddings.shape[1]
        self.index = faiss.IndexFlatL2(dimension)
        self.index.add(embeddings)

    def search(self, query, k=10):
        query_embedding = self.create_embeddings([query])
        distances, indices = self.index.search(query_embedding, k)
        return indices[0], distances[0]
```

Hybrid Search (Text + Vector)

javascript

```
class HybridSearch {
  async search(query, options = {}) {
    // Parallel search strategies
    const [textResults, vectorResults, graphResults] = await Promise.all([
      this.textSearch(query, options),
      this.vectorSearch(query, options),
      this.graphSearch(query, options)
    ]);

    // Fusion ranking
    return this.fuseResults([
      { results: textResults, weight: 0.3 },
      { results: vectorResults, weight: 0.5 },
      { results: graphResults, weight: 0.2 }
    ], options.limit);
  }

  fuseResults(resultSets, limit) {
    const scores = new Map();

    for (const { results, weight } of resultSets) {
      results.forEach((result, index) => {
        const score = (1 / (index + 1)) * weight;
        const existing = scores.get(result.id) || 0;
        scores.set(result.id, existing + score);
      });
    }

    return Array.from(scores.entries())
      .sort((a, b) => b[1] - a[1])
      .slice(0, limit)
      .map(([id, score]) => ({ id, score }));
  }
}
```

API Reference

REST API Endpoints

Authentication

```
http
```

```
POST /api/v1/auth/token
Content-Type: application/json

{
  "did": "did:web:user.example.com",
  "signature": "base64-signature",
  "timestamp": "ISO-8601"
}

Response: 200 OK
{
  "token": "jwt-token",
  "expires": "ISO-8601"
}
```

Memory Operations

Create Memory

```http
POST /api/v1/memories
Authorization: Bearer {token}
Content-Type: application/json

{
  "content": "Memory content",
  "type": "text",
  "source": "api",
  "metadata": {
    "tags": ["important"],
    "project": "project-id"
  }
}

Response: 201 Created
{
  "id": "mem_abc123",
  "address": "mtap://memory/hash",
  "created": "ISO-8601"
}
```

Get Memory

```http
GET /api/v1/memories/{id}
Authorization: Bearer {token}


Response: 200 OK
{
  Full MTAP memory object
}
```

Search Memories

```http
```

```
POST /api/v1/memories/search
Authorization: Bearer {token}
Content-Type: application/json

{
  "query": "search terms",
  "filters": {
    "type": ["text", "image"],
    "source": ["chatgpt"],
    "dateRange": {
      "start": "2024-01-01",
      "end": "2024-12-31"
    }
  },
  "sort": {
    "field": "relevance|date",
    "order": "desc"
  },
  "pagination": {
    "limit": 20,
    "offset": 0
  }
}

Response: 200 OK
{
  "results": [...],
  "total": 145,
  "hasMore": true
}
```

WebSocket API

Real-time Memory Stream

javascript

```
const ws = new WebSocket('wss://api.emma-hml.com/v1/stream');

ws.on('open', () => {
  ws.send(JSON.stringify({
    type: 'authenticate',
    token: 'jwt-token'
  }));

  ws.send(JSON.stringify({
    type: 'subscribe',
    channels: ['memories', 'updates']
  }));
});

ws.on('message', (data) => {
  const event = JSON.parse(data);
  switch(event.type) {
    case 'memory.created':
      handleNewMemory(event.data);
      break;
    case 'memory.updated':
      handleUpdatedMemory(event.data);
      break;
  }
});
```

GraphQL API

```
graphql
```

```
# Schema Definition
type Memory {
  id: ID!
  content: String!
  type: MemoryType!
  source: String!
  created: DateTime!
  modified: DateTime!
  creator: DID!
  metadata: JSON
  relations: Relations
  permissions: Permissions
}

type Query {
  memory(id: ID!): Memory
  memories(
    filter: MemoryFilter
    sort: MemorySort
    limit: Int = 20
    offset: Int = 0
  ): MemoryConnection!

  search(
    query: String!
    filters: SearchFilters
    limit: Int = 20
  ): [SearchResult!]!
}

type Mutation {
  createMemory(input: CreateMemoryInput!): Memory!
  updateMemory(id: ID!, input: UpdateMemoryInput!): Memory!
  deleteMemory(id: ID!): Boolean!

  shareMemory(id: ID!, with: DID!, permissions: [Permission!]!): Memory!
  revokeAccess(id: ID!, from: DID!): Memory!
}

type Subscription {
  memoryCreated(userId: DID!): Memory!
  memoryUpdated(userId: DID!): Memory!
  memoryShared(userId: DID!): SharedMemoryEvent!
}
```

Security & Privacy

Encryption Architecture

At-Rest Encryption

```
javascript
```

```
class MemoryEncryption {
  async encryptMemory(memory, userKey) {
    // Generate memory-specific key
    const memoryKey = await this.deriveKey(userKey, memory.id);

    // Encrypt content
    const encryptedContent = await this.encrypt(
      memory.core.content,
      memoryKey
    );

    // Encrypt metadata selectively
    const encryptedMetadata = await this.encryptMetadata(
      memory.metadata,
      memoryKey
    );

    return {
      ...memory,
      core: {
        ...memory.core,
        content: encryptedContent,
        encrypted: true
      },
      metadata: encryptedMetadata
    };
  }

  async encrypt(data, key) {
    const iv = crypto.getRandomValues(new Uint8Array(12));
    const encrypted = await crypto.subtle.encrypt(
      {
        name: 'AES-GCM',
        iv: iv
      },
      key,
      new TextEncoder().encode(data)
    );

    return {
      ciphertext: btoa(String.fromCharCode(...new Uint8Array(encrypted))),
      iv: btoa(String.fromCharCode(...iv))
    };
  }
}
```

Zero-Knowledge Architecture

```javascript
class ZeroKnowledgeStore {
  async store(memory, userPassword) {
    // Client-side encryption
    const key = await this.deriveKeyFromPassword(userPassword);
    const encrypted = await this.encrypt(memory, key);

    // Server never sees plaintext
    const response = await fetch('/api/store', {
      method: 'POST',
      body: JSON.stringify({
        encrypted: encrypted,
        proof: this.generateProof(encrypted)
      })
    });

    return response.json();
  }

  generateProof(encrypted) {
    // Zero-knowledge proof that data is valid
    // without revealing content
    return zkSnark.prove({
      public: [encrypted.hash],
      private: [encrypted.content],
      circuit: this.validationCircuit
    });
  }
}
```

Privacy Controls

Data Minimization

```javascript
```

```
class PrivacyFilter {
  sanitizeMemory(memory, level = 'standard') {
    const filters = {
      minimal: ['content', 'type', 'created'],
      standard: ['content', 'type', 'created', 'source', 'basic_metadata'],
      full: null // No filtering
    };

    if (level === 'full') return memory;

    const allowedFields = filters[level];
    return this.pickFields(memory, allowedFields);
  }

  anonymizeMemory(memory) {
    return {
      ...memory,
      header: {
        ...memory.header,
        creator: this.hashDID(memory.header.creator)
      },
      metadata: this.stripPII(memory.metadata)
    };
  }
}
```

Federation Network

Peer-to-Peer Protocol

Node Discovery

```javascript
```

```
class NodeDiscovery {
  async discoverPeers() {
    const methods = [
      this.dnsDiscovery(),      // DNS TXT records
      this.dhtDiscovery(),      // Distributed Hash Table
      this.mdnsDiscovery(),     // Local network
      this.bootstrapNodes()     // Known bootstrap nodes
    ];

    const peers = await Promise.all(methods);
    return this.deduplicate(peers.flat());
  }

  async dnsDiscovery() {
    const txtRecords = await dns.resolveTxt('_hml._tcp.example.com');
    return txtRecords.map(record => this.parsePeerInfo(record));
  }

  async dhtDiscovery() {
    const dht = new DHT({ bootstrap: this.bootstrapNodes });
    return await dht.findPeers('hml:network');
  }
}
```

Replication Protocol

javascript

```
class ReplicationProtocol {
  async replicate(memory, targetNodes = 3) {
    // Select nodes based on criteria
    const nodes = await this.selectNodes({
      geography: 'distributed',
      reputation: 'high',
      capacity: 'available',
      count: targetNodes
    });

    // Shard memory for redundancy
    const shards = this.createShards(memory, nodes.length);

    // Distribute shards
    const results = await Promise.all(
      shards.map((shard, i) =>
        this.sendShard(shard, nodes[i])
      )
    );

    // Store replication map
    await this.storeReplicationMap(memory.id, results);

    return results;
  }

  createShards(memory, n) {
    // Reed-Solomon erasure coding
    const encoder = new ReedSolomon(n, Math.floor(n * 0.6));
    return encoder.encode(memory);
  }
}
```

Identity & Trust

Decentralized Identity (DID)

javascript

```
class DIDManager {
  async createDID(method = 'web') {
    const methods = {
      web: this.createWebDID,
      key: this.createKeyDID,
      ethr: this.createEthereumDID,
      ion: this.createIONDID
    };

    const did = await methods[method]();
    const document = await this.createDIDDocument(did);

    return {
      did: did,
      document: document,
      keys: await this.generateKeys(did)
    };
  }

  async createDIDDocument(did) {
    return {
      '@context': 'https://www.w3.org/ns/did/v1',
      id: did,
      authentication: [{
        id: `${did}#keys-1`,
        type: 'Ed25519VerificationKey2020',
        controller: did,
        publicKeyMultibase: await this.getPublicKey()
      }],
      service: [{
        type: 'HMLService',
        serviceEndpoint: 'https://hml.example.com/api'
      }]
    };
  }
}
```

Development Guide

Getting Started

Installation

bash

```
# Install Emma CLI
npm install -g @emma-hml/cli

# Initialize new project
emma init my-memory-app

# Install SDK
npm install @emma-hml/sdk
```

Basic Implementation

javascript

```
// JavaScript/TypeScript SDK
import { HML, MTAP, MCP } from '@emma-hml/sdk';

// Initialize HML
const hml = new HML({
  storage: 'local', // local|edge|federation
  encryption: true,
  did: 'did:web:user.example.com'
});

// Create memory
const memory = await hml.createMemory({
  content: 'Important information to remember',
  type: 'text',
  metadata: {
    tags: ['important'],
    source: 'manual'
  }
});

// Search memories
const results = await hml.search('important information', {
  limit: 10,
  timeRange: 'last-week'
});

// Get context for AI
const context = await hml.getContext({
  query: 'What did I learn about quantum computing?',
  maxTokens: 2000
});
```

Python SDK

python

```
from emma_hml import HML, MemoryStore
import asyncio

class MyMemoryApp:
    def __init__(self):
        self.hml = HML(
            storage_type='local',
            user_did='did:web:user.example.com'
        )

    async def save_conversation(self, messages):
        """Save a conversation to memory layer"""
        for message in messages:
            await self.hml.create_memory(
                content=message['content'],
                role=message['role'],
                metadata={
                    'conversation_id': message['conversation_id'],
                    'timestamp': message['timestamp']
                }
            )

    async def get_relevant_context(self, query):
        """Retrieve relevant memories for AI context"""
        memories = await self.hml.search(
            query=query,
            limit=20,
            relevance_threshold=0.7
        )

        return self.format_for_ai(memories)
```

AI Agent Integration

LangChain Integration

```
python
```

```
from langchain.memory import BaseMemory
from emma_hml import HML

class HMLMemory(BaseMemory):
    """LangChain memory implementation using HML"""

    def __init__(self, hml_config):
        self.hml = HML(**hml_config)
        self.memory_key = "hml_context"

    @property
    def memory_variables(self):
        return [self.memory_key]

    def load_memory_variables(self, inputs):
        query = inputs.get("input", "")
        memories = self.hml.search(
            query=query,
            limit=10
        )

        context = self._format_memories(memories)
        return {self.memory_key: context}

    def save_context(self, inputs, outputs):
        self.hml.create_memory(
            content=f"User: {inputs['input']}\nAI: {outputs['output']}",
            type="conversation",
            metadata={
                "timestamp": datetime.now().isoformat(),
                "model": "gpt-4"
            }
        )
```

Deployment Strategies

Deployment Architectures

1. Standalone (Local-First)

```
yaml
```

```
# docker-compose.yml
version: '3.8'
services:
  emma-local:
    image: emma-hml/local:latest
    volumes:
      - ./data:/app/data
      - ./config:/app/config
    ports:
      - "8080:8080"
    environment:
      - STORAGE_MODE=local
      - ENCRYPTION=true
      - FEDERATION=disabled
```

2. Edge Deployment (Cloudflare Workers)

```javascript
```

```
// wrangler.toml
name = "emma-edge"
type = "javascript"
account_id = "your-account-id"
workers_dev = true

[durable_objects]
bindings = [
  { name = "MEMORY_STORE", class_name = "MemoryStore" }
]

[[kv_namespaces]]
binding = "MEMORY_INDEX"
id = "your-kv-namespace-id"

// worker.js
export default {
  async fetch(request, env) {
    const url = new URL(request.url);

    if (url.pathname.startsWith('/api/memory')) {
      const id = env.MEMORY_STORE.idFromName(userId);
      const stub = env.MEMORY_STORE.get(id);
      return stub.fetch(request);
    }

    return new Response('Not Found', { status: 404 });
  }
}
```

3. Federation Node

```
yaml
```

```yaml
# kubernetes deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hml-federation-node
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hml-node
  template:
    metadata:
      labels:
        app: hml-node
    spec:
      containers:
      - name: hml-node
        image: emma-hml/federation:latest
        ports:
        - containerPort: 8080  # API
        - containerPort: 9090  # P2P
        - containerPort: 4001  # IPFS
        env:
        - name: NODE_TYPE
          value: "federation"
        - name: ENABLE_IPFS
          value: "true"
        - name: ENABLE_DHT
          value: "true"
```

Performance Optimization

Caching Strategy

```
javascript
```

```
class MemoryCache {
  constructor() {
    this.l1 = new Map();        // In-memory cache
    this.l2 = new Redis();      // Redis cache
    this.l3 = new CDN();        // CDN cache
  }

  async get(key) {
    // L1 Cache (Memory)
    if (this.l1.has(key)) {
      return this.l1.get(key);
    }

    // L2 Cache (Redis)
    const l2Result = await this.l2.get(key);
    if (l2Result) {
      this.l1.set(key, l2Result);
      return l2Result;
    }

    // L3 Cache (CDN)
    const l3Result = await this.l3.get(key);
    if (l3Result) {
      await this.l2.set(key, l3Result);
      this.l1.set(key, l3Result);
      return l3Result;
    }

    // Origin
    const result = await this.fetchFromOrigin(key);
    await this.setCaches(key, result);
    return result;
  }
}
```

Appendices

A. Protocol Version History

Version	Date	Changes
0.1.0	2024-01	Initial protocol draft
0.5.0	2024-06	Added federation support
0.8.0	2024-10	MCP integration
1.0.0	2025-01	Production release

B. Reference Implementations

1. **Emma Lite** - Chrome Extension (14-day MVP)
   - Repository: github.com/emma-hml/emma-lite
   - Language: JavaScript
   - Storage: IndexedDB
2. **Emma Core** - Full Implementation
   - Repository: github.com/emma-hml/emma-core
   - Language: TypeScript/Rust
   - Storage: Multi-tier
3. **HML SDK** - Developer Kit
   - Repository: github.com/emma-hml/sdk
   - Languages: JS, Python, Go, Rust
   - Documentation: docs.emma-hml.com

C. Standards Compliance

- **W3C DID**: Decentralized Identifiers v1.0
- **IPFS**: InterPlanetary File System
- **OAuth 2.0**: Authorization Framework
- **JWT**: JSON Web Tokens (RFC 7519)
- **OpenAPI**: 3.0 Specification

D. Performance Benchmarks

Operation	Target	Actual	Notes
Memory Creation	< 100ms	45ms	Local storage
Search (1000 memories)	< 200ms	120ms	Text search
Vector Search (10k)	< 500ms	380ms	With embeddings
Federation Sync	< 2s	1.5s	3 nodes
Encryption/Decryption	< 50ms	35ms	AES-256

E. Glossary

- **DID**: Decentralized Identifier
- **DHT**: Distributed Hash Table
- **HML**: Human Memory Layer
- **MCP**: Model Context Protocol
- **MTAP**: Memory Transfer & Access Protocol
- **PII**: Personally Identifiable Information
- **P2P**: Peer-to-Peer
- **ZKP**: Zero-Knowledge Proof

F. Contact & Support

- **Website**: https://emma-hml.com
- **Documentation**: https://docs.emma-hml.com
- **GitHub**: https://github.com/emma-hml
- **Discord**: https://discord.gg/emma-hml
- **Email**: support@emma-hml.com

License

This specification is released under the MIT License.

End of Technical Documentation v1.0.0