The City College of New York

# CSC 59866-E: Senior Project I

## *AI Agents for Decision Making in the Real World*

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu, sbandyopadhyay@gc.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at the City University of New York



May 11, 2026 CSC 59866

# Advanced Topics: Dynamic Memory and Standardized Protocols for Real-World Agents

# Logistics & The State of the Class

**Recall Lecture 24:** We examined multi-agent coordination, mixed motives, and human-agent alignment using generative partner modeling.

Two of the biggest problems faced by AI Agent researchers and engineers are **LLM context windows** that are too small for the task the Agent is trying to accomplish, and the overwhelming difficulty of **creating Python tools** for every single application your Agent needs to interact with to reach its goal!

Many of you have likely run into these exact problems while working on your project.

# Today's Agenda

1. **Using Dynamic Memory to overcome context window limitations**
2. **A-Mem & Agentic Adaptation.**

# Dynamic Memory in Real-World Agents

# Architectures of Multi-Agent Systems

Modern models have 1M to 2M context window on paper, but in reality it isn't practical to shove everything into one context window for two main reasons:

- Self-attention (in each Transformer block) **grows quadratically** with context: $O(L^2 \cdot d)$
- When important information is buried within larger and larger amounts of text, **accuracy can drop dramatically**

Instead of storing all the information in one place, what if we train our Agent to *dynamically retrieve memories?*

# Episodic vs. Semantic Memory

**Episodic Memory:** Log of specific past interactions, states, and observations.

- *Example:* "At 10:04 AM, the user rejected my API call because the payload was missing the 'date' parameter."

**Semantic Memory:** Generalized knowledge, facts, and rules extracted and synthesized from episodic memory over time.

- *Example:* "This user's API strictly requires a 'date' parameter formatted as ISO 8601."

**The Transition:** True agentic learning requires a mathematical pipeline to continuously distill raw episodes into semantic rules.

# Memory as Managed by a Sub-Agent

What if we have *another* Agent help our main Agent manage it's growing context?

Some LLM Agents have a dedicated lightweight LLM Agent solely for managing the context of the *main* Agent.

**Responsibilities of the memory Sub-Agent LLM:**

1. Keeping track of the information worth remembering (indexing).
2. Deleting obsolete, wrong, or irrelevant facts from the context (pruning).
3. Adding necessary context to the main  LLM Agent before it responds (injecting).

# Vector Databases + Knowledge Graphs

The Memory Sub-Agent uses a dual-engine approach for data storage.

- **Vector Embeddings:** Facilitate fuzzy semantic search using Cosine Similarity for tasks like finding similar conversations.

$$S_C(A, B) = \frac{A \cdot B}{||A||||B||}$$

- **Knowledge Graphs (KGs):** Store strict, factual relationships (Subject -> Predicate -> Object) for deterministic queries, such as identifying a manager.
- **Integration:** Modern frameworks like GraphRAG combine them, using vector similarity to rank starting nodes for KG traversal.

# Recent Academic Breakthroughs

# A-Mem: Agentic Memory for LLM Agents

Let's look at the cutting-edge of memory architecture: **A-Mem** (Xu et al., 2025).

Current memory systems (like standard RAG) enable basic storage and retrieval but lack sophisticated memory organization and adaptability.

A-Mem introduces an agentic memory system based on the **Zettelkasten method**; creating interconnected knowledge networks through dynamic indexing, linking, and note-taking.

# Note Construction

When an agent experiences something new, A-Mem doesn’t just dump the text into a database.

**Note Construction:** It uses an LLM to generate a comprehensive "note" containing structured attributes:

- Contextual Description
- Keywords
- Tags
- Abstract Summaries

This heavily enriches the embedding vector, improving retrieval accuracy by ensuring the metadata matches the semantic intent of future queries.

# Autonomous Linking and Evolution

**Memory Linking:** A-Mem analyzes historical memories to identify relevant connections. An edge is created between memory $i$ and memory $j$ if their similarity exceeds a dynamic threshold $\tau$:

$$Edge(i,j) = \begin{cases} 1 & \text{if } sim(h_i, h_j) > \tau \\ 0 & \text{otherwise} \end{cases}$$

**Memory Evolution:** As new memories are integrated, the system autonomously triggers updates to the contextual representations of *existing* historical memories. Old notes are re-written by the sub-agent to reflect new paradigms, entirely avoiding static, outdated context.

# Adaptation of Agentic AI

The 2025 survey "Adaptation of Agentic AI" (Jiang et al.) highlights that agents must accumulate persistent memory and reusable skills without breaking their foundational training.

**The Co-Adaptation Problem:** If an agent continuously updates its skill library (e.g., adding a new Python script to access a changing API), it risks destabilizing its existing policies.

How do we measure and maintain stability when an agent modifies its own underlying logic?

# Preventing Catastrophic Forgetting

As agents accumulate new semantic memories and adjust their behavioral weights, they suffer from **Catastrophic Forgetting** (erasing older, vital skills).

**Elastic Weight Consolidation (EWC):** We constrain the agent's policy updates. When learning a new skill $B$, we add a penalty to the loss function $\mathcal{L}$ that prevents changing the neural parameters $\theta$ that were highly important for old skill $A$:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

$F_i$ is the Fisher Information Matrix, which mathematically quantifies how critical parameter $\theta_i$ is to remembering past memories.

# Standards for Agent-to-Agent Communication

# Standardized Protocols for Agent Communication

**The 2024 Ecosystem:** Developers wrote custom Python wrappers for every single tool. An agent needed bespoke code to talk to Slack, different code for Jira, and different code for a SQL database.

**The 2025/2026 Ecosystem:** The industry has shifted to standardized interoperability.

Agents no longer learn specific APIs; they learn universal *protocols*.

# Model Context Protocol (MCP)

**The Concept:** Introduced by Anthropic and open-sourced, MCP is the industry standard for Agent-to-Tool communication.

It solves the context problem by providing a universal connection pattern for external resources.

**The Architecture:**

- **MCP Client:** The Agent.
- **MCP Server:** A lightweight wrapper around *any* data source (PostgreSQL, Notion, GitHub, local file system).

# How MCP Mechanics Work

**Standardized Schemas:** The MCP Server advertises its capabilities via standardized JSON schemas.

The Agent dynamically reads this schema and instantly knows how to use the tool, without the developer writing specific prompt instructions.

**JSON-RPC 2.0:** All communication (queries, tool executions, resource fetching) happens over JSON-RPC.

**The Result:** You can plug a completely new, proprietary corporate database into an LLM, and via MCP, the agent can immediately query it with zero-shot accuracy.

# Agent-to-Agent Protocol (A2A)

While MCP connects agents to *tools*, the **A2A Protocol** connects agents to *other agents*.

Developed for horizontal, peer-to-peer multi-agent coordination (e.g., an Inventory Agent talking to a Logistics Agent).

**The 3-Step Sequence:**

1. **Discovery:** Agents read "Agent Cards" (manifests) to find the right peer for a sub-task.
2. **Authorization:** Agents verify cryptographic identity and grant scope.
3. **Communication:** Dispatching tasks and streaming state via Server-Sent Events (SSE).

# A2A Task Bidding

- When a primary agent needs a sub-task completed, it doesn't just pick randomly. It broadcasts a Call for Proposals (CFP) to the A2A network.
- Peer agents calculate their estimated cost $C_i$ and expected success probability $P_i$ .
- They submit a bid based on Expected Utility:

$$U_i = R_{task} \cdot P_i - C_i$$

- The primary agent selects the peer that mathematically maximizes the global reward. This converts chaotic LLM chatting into rigorous, market-based distributed computing.

# Industrial Control for AI Agents

# Industrial Governance & Deployment

Moving from theoretical software to real-world deployment requires strict governance.

**"The Industrial AI Agent Manifesto"** (Digital Twin Consortium, 2026) outlines the "Ten Laws for Trustworthy Autonomous Operations."

If your agent controls a physical supply chain, a power grid, or financial assets, it is no longer just a chatbot—it is an industrial controller subject to regulatory auditing.

# Law 1: Deterministic Validation and Execution

LLMs are inherently probabilistic. Industrial systems require deterministic outcomes.

**The Law:** An AI agent must never directly execute a critical action. Its output must be parsed and validated by a deterministic, rules-based safety envelope.

If the agent hallucinates a command to "Set furnace temperature to 10,000 degrees," the deterministic boundary catches the physical violation and rejects the payload.

# Law 2: Physics-Aware and Process-Aware Intelligence

Industrial agents cannot operate purely on semantic language; they must respect the physical constraints of their environment.

**The Manifest Requirement:** Agents must be grounded in physics-based Digital Twins.

Before executing an A2A logistics route, the agent must query a digital twin simulator (via MCP) to verify that the physical truck actually has enough fuel and time to complete the route.

# Law 3: Reproducible Context and Audit Trails

In traditional software, if a bug occurs, you check the stack trace. How do you debug an LLM hallucination from three weeks ago?

**Reproducible Context:** The system must freeze and log the exact memory vectors, the exact Zettelkasten notes, and the exact system prompt present at the moment of decision.

**Auditability:** Without reproducible context, when an industrial agent makes a costly error, you cannot prove *why* it made that decision to regulators.

# Summary of Modern Agent Stacks

**Memory:** Agents use Zettelkasten-style frameworks (A-Mem) to dynamically link, evolve, and prune context, bypassing the quadratic compute cost of massive context windows.

**Tools:** The Model Context Protocol (MCP) standardizes how agents discover and execute external APIs.

**Coordination:** The A2A protocol allows agents to dynamically discover and negotiate with peers using market-bidding mathematics.

**Governance:** The Industrial Agent Manifesto ensures these probabilistic systems are bounded by deterministic safety constraints and rigorous audit trails.

# Questions?