Advances in Memory Architectures for Conversational History in LLM Agents

Ngô Văn Linh

Trường Công nghệ thông tin và Truyền thông,
Đại học Bách khoa Hà Nội

Nội dung

1. Giới thiệu

LLM (Large Language Model)

- Là các mô hình học sâu huấn luyện trên tập dữ liệu khổng lồ.
- Khả năng sinh văn bản, trả lời câu hỏi, lập luận và sáng tạo.
- Hạn chế: phụ thuộc vào cửa sổ ngữ cảnh cố định, không có bộ nhớ (quản lý lịch sử hội thoại) dài hạn tự nhiên.

Agentic AI

- Là tác nhân thông minh dựa trên LLM có khả năng hành động, lập kế hoạch và phối hợp.
- Không chỉ đối thoại, mà còn thực hiện các tác vụ, duy trì trạng thái, và tương tác với môi trường.
- Yêu cầu bộ nhớ dài hạn để ghi nhớ lịch sử, học từ kinh nghiệm và phối hợp với các tác nhân khác.

Bài toán xây dựng tác tử hội thoại cá nhân hóa

Bối cảnh: hội thoại đa phiên

- Tác tử tương tác với người dùng qua nhiều **phiên**, mỗi phiên là một khoảng thời gian hội thoại riêng biệt
  - Kết thúc bởi sự gián đoạn, xác nhận của người dùng, hoặc khi bắt đầu một luồng mới
- Trong mỗi phiên, hội thoại diễn ra theo **các lượt** (câu hỏi + phản hồi)
- Tác tử có **bộ nhớ ngoài** lưu trữ thông tin từ các phiên trước
- Mục tiêu: Sinh phản hồi **cá nhân hóa**, tận dụng ngữ cảnh hiện tại và bộ nhớ

Thách thức chính:

1. Xác định và lưu trữ thông tin quan trọng từ mỗi phiên, dự đoán nhu cầu truy xuất tương lai.
2. Truy xuất chính xác thông tin liên quan; tránh ngữ cảnh nhiễu làm giảm chất lượng phản hồi.

Quản lý bộ nhớ hội thoại

Cần tạo ra bộ nhớ ngoài cho LLM vì:

- Duy trì tính nhất quán hội thoại: tránh mâu thuẫn, quên chi tiết
- Cá nhân hóa trải nghiệm: ghi nhớ sở thích, đặc điểm người dùng
- Theo dõi tiến trình tác vụ: tiếp tục công việc chưa hoàn thành
- Hỗ trợ tác vụ phức tạp, đa bước và đa tác nhân
- Tối ưu tài nguyên và chi phí (token, độ trễ)

Quản lý bộ nhớ <=> RAG

- Phân đoạn lịch sử hội thoại: session, chunk, turn
- Phân loại các loại bộ nhớ: Core, episodic, semantic, etc.
- Tổ chức, biểu diễn dữ liệu lịch sử: Tree, graph
- Cập nhật bộ nhớ: Add, delete, update
- Truy xuất bộ nhớ: Embedding, retrieval

Các hướng tiếp cận quản lý bộ nhớ dài hạn

Các nghiên cứu gần đây đề xuất nhiều nhóm phương pháp:

- **Segment and Compression**: Chia hội thoại theo chủ đề và nén thông tin để loại bỏ nhiễu, tăng độ chính xác truy xuất.
- **Multi-Granularity Memory**: Kết hợp nhiều cấp độ bộ nhớ, chọn granularity tối ưu cho mỗi truy vấn.
- **Hierarchical Memory**: Lưu trữ dạng cây phân cấp, mỗi nút chứa nội dung đã tổng hợp và embedding ngữ nghĩa, hỗ trợ suy luận đa tầng.
- **Graph Memory**: Biểu diễn thông tin dưới dạng đồ thị, mô tả quan hệ phức tạp (nguyên nhân–kết quả, sự kiện liên tục), hỗ trợ suy luận đa bước.
- **Multi-Agent Memory**: Hệ thống đa tác nhân với nhiều loại bộ nhớ (Core, Episodic, Semantic, Procedural, Resource, Knowledge Vault), quản lý thông tin phong phú và đa phương tiện.

2. Segmentation and Compression

- Quản lý bộ nhớ được nghiên cứu trong các công việc trước gặp phải một số hạn chế:
  - Mức lượt (turn-level): Đơn vị nhỏ, giàu chi tiết, nhưng phân mảnh ngữ nghĩa, làm tăng số mảnh phải truy hồi và dễ bỏ sót bối cảnh liên quan trải dài nhiều lượt.
  - Mức phiên (session-level): Đơn vị quá lớn, chứa nhiều phần không liên quan; dù có bộ nhớ dài (32k+) có nhiễu và độ đặc thù ngữ cảnh thấp.
  - Tóm tắt/summarization: Tạo bản rút gọn nhưng có nguy cơ bỏ qua chi tiết quan trọng và khó tổng quát hoá qua tác vụ/miền

- Tổ chức lịch sử hội thoại theo các chủ đề dựa trên LLM
  - SeCom: On Memory Construction and Retrieval for Personalized Conversational Agents, ICLR 2025 [1]
  - In Prospect and Retrospect: Reflective Memory Management for Long-term Personalized Dialogue Agents, ACL 2025 [2]

SeCom: On Memory Construction and Retrieval for Personalized Conversational Agents, ICLR 2025

- SeCom gồm hai khối chính: SE (Segmentation) và COM (Compression as Denoising)

- ScCom đề xuất:
  - (i) thay vì lưu ở mức lượt (turn-level) hay phiên (session-level), phân đoạn theo chủ đề để có đơn vị nhớ vừa đủ;
  - (ii) nén mỗi đơn vị nhớ nhằm loại bỏ phần dư thừa—coi đó là nhiễu đối với truy hồi—trước khi đánh chỉ mục và truy hồi.

SeCom

- **Lịch sử hội thoại**
  - Gọi $H = \{c_i\}_{i=1}^{C}$ là lịch sử hội thoại giữa người dùng và tác tử, gồm C phiên.
  - Mỗi phiên $c_i = \{t_j\}_{j=1}^{T_i}$ gồm T_i lượt tương tác tuần tự.
  - Mỗi lượt $t_j = (u_j, r_j)$ với u_j: yêu cầu của người dùng, r_j: phản hồi của tác tử
- **Thành phần nền**: Bộ truy hồi: f_R và mô hình sinh phản hồi: f_{LLM}
- **Xây dựng bộ nhớ**: Bộ nhớ mức lượt (turn-level): mỗi $m \in M$ tương ứng với một lượt t, $|M| = \sum_{i=1}^{C} T_i$; Bộ nhớ mức phiên (session-level): mỗi m tương ứng với một phiên c, $|M| = C$
- **Truy hồi bộ nhớ**
  - Với yêu cầu mục tiêu u^* và số lượng đơn vị bộ nhớ N, truy hồi N đơn vị nhớ liên quan: $\{m_n \in M\}_{n=1}^{N} \leftarrow f_R(u^*, M, N)$
- **Sinh phản hồi**
  - Dùng N đơn vị nhớ đã truy hồi (theo thứ tự thời gian) làm ngữ cảnh.
  - Sinh phản hồi: $r^* = f_{LLM}\left(u^*, \{m_n\}_{n=1}^{N}\right)$

CONVERSATION SEGMENTATION

- Mục tiêu mô hình **conversation segmentation**:
  - Mỗi phiên c_i được chia thành K_i phân đoạn chủ đề: $\{s_k\}_{k=1}^{K_i}$
  - Xây dựng **segment-level memory bank**, trong đó: Mỗi đơn vị nhớ m tương ứng với một phân đoạn s. Kích thước bộ nhớ: $|M| = \sum_{i=1}^{C} K_i$
- Cho một phiên c, mô hình phân đoạn f_I trả về tập chỉ số đoạn:

$$I = \{(p_k, q_k)\}_{k=1}^{K}$$

- Trong đó:
  - (p_k, q_k): chỉ số lượt **đầu/cuối** của đoạn s_k
  - Điều kiện: $p_k \leq q_k$ và $p_{k+1} = q_k + 1$
- Kết quả phân đoạn:

$$f_I(c) = \{s_k\}_{k=1}^{K}$$

với

$$s_k = \{t_{p_k}, \ldots, t_{q_k}\}$$

CONVERSATION SEGMENTATION

- **Thách thức:**
  - Khó thu thập dữ liệu gán nhãn quy mô lớn cho hội thoại mở.
  - Điểm phân đoạn thường **mơ hồ**, gây khó khăn ngay cả với người gán nhãn thủ công.
- **Giải pháp:**
  - Sử dụng GPT-4 làm mô hình phân đoạn f_l, nhờ khả năng hiểu ngôn ngữ mạnh mẽ trên nhiều miền.
  - Tăng cường dữ liệu phiên c bằng cách thêm chỉ số lượt và định danh vai trò để tăng khả năng suy luận: `Turn j: [user] / [agent]`
- **Kết quả thực nghiệm:**
  - Các mô hình gọn nhẹ hơn (Mistral-7B, RoBERTa) cũng có thể thực hiện phân đoạn.
  - Phương pháp áp dụng được trong môi trường **hạn chế tài nguyên**.

COM — Nén như Khử nhiễu (LLMLingua-2)

Giả thuyết cốt lõi:

- Ngôn ngữ tự nhiên chứa nhiều **dư thừa**, đóng vai trò như **nhiễu** đối với bộ truy hồi.
- Loại bỏ dư thừa:
  - Tăng **recall** với cùng một lượng token bộ nhớ.
  - Tăng **tương đồng truy vấn–đoạn liên quan**.
  - Giảm **tương đồng với đoạn không liên quan**.

Mục tiêu: Với mỗi segment s_k, sinh phiên bản nén $\tilde{s}_k$ (ngắn hơn nhưng giữ ý nghĩa) để đánh chỉ mục và truy hồi hiệu quả hơn.
Công cụ: LLMLingua-2 [3] — bộ nén dựa trên phân loại nhị phân theo token (preserve / discard).

Kí hiệu và Ràng buộc

- Văn bản gốc: $x = (x_1, \ldots, x_N)$
- Sau nén: $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_M)$
- **Ràng buộc trích xuất:** $\hat{x}$ là dãy con theo thứ tự của x.

$$\exists\, \pi : \{1, \ldots, M\} \to \{1, \ldots, N\}, \quad \pi(1) < \cdots < \pi(M),\ \hat{x}_m = x_{\pi(m)}$$

- **Tỉ lệ giữ (budget):** $\tau = \frac{M}{N} \in (0, 1]$
- **Tỉ lệ nén (CR):** $CR = 1 - \tau$

Data Distillation và gán nhãn

- Dùng LLM mạnh sinh $\hat{x}$ từ x với ba ràng buộc:
  1. Rút ngắn ($M < N$)
  2. Giữ thông tin cốt lõi
  3. Trung thực (không bịa)
- Ràng buộc **extractive** giúp gán nhãn tự động ổn định.
- **Chỉ số phát hiện lỗi / Hallucination:**

$$VR = \frac{1}{|V(\hat{x})|} \sum_{w \in V(\hat{x})} 1[w \notin V(x)]$$

- Loại bỏ các mẫu có VR cao. Lý tưởng: $VR = 0$.

Huấn luyện LLMLingua-2

- Bộ phân loại token:

$$p_i = \Pr(y_i = 1 \mid x; \Theta) = \sigma(Wh_i + b), \quad h = f_\theta(x)$$

- f_θ: encoder hai chiều (RoBERTa / XLM-R).
- **Loss function (Binary Cross-Entropy, có trọng số):**

$$L(\Theta) = -\frac{1}{N} \sum_{i=1}^{N} [\alpha y_i \log p_i + (1 - \alpha)(1 - y_i) \log(1 - p_i)]$$

- $\alpha \in (0, 1]$: tăng phạt cho lớp "giữ" khi lớp này hiếm.

Truy hồi và Sinh phản hồi

- Đánh chỉ mục các $\{\tilde{s}_k\}$ bằng:
  - **Lexical**: BM25
  - **Dense**: MPNet
- Chọn mô hình tuỳ **tài nguyên** và để đảm bảo **công bằng so sánh** với các nghiên cứu trước.
- **Truy hồi**: lấy top-N đoạn theo độ liên quan.
- **Ghép ngữ cảnh**: sắp xếp top-N đoạn theo thứ tự thời gian $\Rightarrow$ tránh **mâu thuẫn ngữ cảnh**.
- **Sinh phản hồi**: nối các đoạn và đưa vào prompt của f_{LLM}.

Pipeline tổng thể — Ý tưởng

SECOM hợp nhất hai thao tác bổ trợ:

1. **Cắt theo chủ đề**: chia phiên dài thành các segment mạch lạc ngữ nghĩa.
2. **Nén như khử nhiễu**: làm "đặc" tín hiệu của từng segment trước khi chỉ mục.

Tại thời điểm truy vấn: Chỉ cần gọi lại một số rất ít segment đúng chủ đề và ít nhiễu để đưa vào cửa sổ ngữ cảnh của LLM.

Pipeline tổng thể — Dòng dữ liệu

1. **Offline (Xây bộ nhớ):**
   - Với mỗi phiên: phân đoạn $\rightarrow$ các s_k.
   - Mỗi s_k được nén $\rightarrow \tilde{s}_k$.
   - Đánh chỉ mục $\tilde{s}_k$ (BM25/dense) + lưu metadata (mốc thời gian, id phiên, phạm vi lượt).
2. **Online (Truy hồi):**
   - Nhận truy vấn u^*, retriever tìm top-N đoạn.
   - Nếu điểm số tương đương: ưu tiên đoạn gần về **thời gian** hoặc **liên tục** để giảm đứt gãy.

Pipeline tổng thể — Dòng dữ liệu (tiếp)

3. Ghép ngữ cảnh:

- Sắp xếp theo thời gian, loại trùng lặp.
- Đảm bảo tổng token $\leq$ ngân sách.
- Nếu vượt: giảm bằng nén (hạ τ) hoặc loại bỏ phần ít liên quan.

4. Sinh phản hồi:

- Đưa truy vấn u^* và chuỗi đoạn đã chọn vào LLM.
- Dùng **system prompt chuẩn** để:
  - ép mô hình chỉ dựa trên **bằng chứng trong đoạn**,
  - hạn chế hallucination.

In Prospect and Retrospect: Reflective Memory Management for Long-term Personalized Dialogue Agents, ACL 2025 [2]

Hạn chế của cơ chế bộ nhớ ngoài hiện nay:

- **Phân đoạn cố định:** xử lý theo lượt, phiên, mốc thời gian → không phản ánh cấu trúc ngữ nghĩa tự nhiên (ví dụ: khi đổi chủ đề)
- **Bộ retrieval cố định:** khó thích ứng với nhu cầu truy xuất đa dạng giữa các miền hội thoại và kiểu tương tác cá nhân. ⇒ Việc thu thập dữ liệu gán nhãn để huấn luyện rất tốn kém.

Quản lý Bộ nhớ Phản chiếu (Reflective Memory Management - RMM)

- **Prospective Reflection:** Tóm tắt hội thoại thành các chủ đề, hợp nhất thông tin rời rạc thành bộ nhớ mạch lạc → tối ưu truy xuất tương lai.
- **Retrospective Reflection:** Sử dụng LLM để gán nhán cho retrieval để tinh chỉnh truy xuất trực tuyến → thích ứng theo miền.

Tổ chức Bộ nhớ Dựa trên Chủ đề

- Hệ thống quản lý bộ nhớ truyền thống dựa vào ranh giới cố định (phiên, lượt) để cấu trúc lịch sử hội thoại.
- Vấn đề: ranh giới này không trùng khớp với các **đơn vị ngữ nghĩa** thực tế $\rightarrow$ thông tin quan trọng bị phân tán, gây khó khăn cho truy xuất.

Giải pháp: Prospective Reflection

- Tổ chức bộ nhớ dựa trên **các chủ đề ngữ nghĩa mạch lạc**.
- "Chủ đề" = một đơn vị thảo luận có thể kéo dài qua nhiều lượt trong một phiên.
- Mỗi chủ đề gắn với (các) đoạn hội thoại gốc liên quan.
- Chủ đề có thể từ ý định cụ thể (vd: hỏi công thức ăn chay) đến khái quát (vd: lên kế hoạch du lịch).

Prospective Reflection

Diễn ra vào cuối mỗi phiên, gồm 2 bước chính:

1. **Trích xuất bộ nhớ**
   - Sử dụng LLM (prompt) để trích các đoạn hội thoại trong phiên.
   - Sinh bản tóm tắt tương ứng, theo các **chủ đề** được đề cập.
2. **Cập nhật bộ nhớ**
   - Với mỗi bộ nhớ được trích xuất $\rightarrow$ truy xuất *Top-K* bộ nhớ tương đồng cao nhất trong ngân hàng.
   - LLM quyết định:
     - Thêm trực tiếp nếu là **chủ đề mới**.
     - Gộp với bộ nhớ hiện có nếu là **mở rộng thông tin cũ**.

Kết quả: Ngân hàng bộ nhớ duy trì biểu diễn **mạch lạc, hợp nhất**, được tổ chức quanh **các chủ đề ngữ nghĩa**.

Retrospective Reflection: Reranker Design

Vấn đề: Bộ truy xuất sẵn có có thể xác định bộ nhớ liên quan, nhưng hiệu năng suy giảm khi áp dụng cho nhiều miền hội thoại và người dùng khác nhau.

Giải pháp: Giới thiệu **Bộ xếp hạng lại (Reranker)** để tinh chỉnh danh sách bộ nhớ truy xuất:

- Thích ứng động với đặc thù miền và sở thích người dùng.
- Không cần tinh chỉnh trực tiếp bộ truy xuất.

Quy trình:

1. Xử lý Top-K bộ nhớ được retriever trả về.
2. Thích ứng vector:

$$q' = q + W_q q, \quad m_i' = m_i + W_m m_i$$

3. Tính điểm liên quan: $s_i = q'^{\top} m_i'$.
4. Lựa chọn Top-M ứng viên phù hợp nhất.

Retrospective Reflection: LLM Attribution as Rewards

Vấn đề: Thu thập dữ liệu gán nhãn chất lượng cao cho từng người dùng rất tốn kém.

Giải pháp: Tận dụng chính **LLM** để sinh phản hồi kèm trích dẫn:

- Prompt LLM để sinh cả phản hồi và citation trong một lần gọi.
- Citation có điều kiện theo phản hồi $\rightarrow$ hiệu quả cao hơn so với sinh riêng lẻ.

Phần thưởng:

- $+1$ nếu bộ nhớ được trích dẫn trong phản hồi cuối cùng.
- -1 nếu không được trích dẫn.

$\rightarrow$ *Reranker học được chiến lược truy xuất tốt hơn, đồng bộ với bằng chứng mà LLM thực sự sử dụng.*

Cập nhật Reranker bằng học tăng cường RL

3. Memory Representation Architectures

Hạn chế của biểu diễn phẳng (thuần text)

- Phân mảnh, không có liên kết ngữ nghĩa.
- Khó thực hiện reasoning đa-bước (multi-hop).

Lợi ích của tổ chức bộ nhớ có cấu trúc

- Giữ rõ ràng quan hệ giữa thực thể, sự kiện, trạng thái.
- Truy xuất chính xác hơn (node/edge, nhánh tree).
- Hỗ trợ abstraction, aggregation và multi-hop reasoning.

Tree-structured Memory (MemTree)

- Tổ chức phân cấp: lá (chi tiết) $\rightarrow$ nút trung gian (topic summaries) $\rightarrow$ gốc (schemas).

Graph-based Memory

- Biểu diễn thực thể/sự kiện là nodes, quan hệ là edges.

From Isolated Conversations to Hierarchical Schemas: Dynamic Tree Memory Representation for LLMs, ICLR 2025 [7]

Vấn đề:

- LLMs xử lý tốt lượng văn bản lớn.
- Nhưng **khó ghi nhớ và suy luận** dựa trên thông tin **dài hạn**.

1. Phương pháp sử dụng bảng tra cứu phẳng (Flat Lookup Tables)

- Ví dụ: SeCom [1], RMM [2], MemGPT [4].
- **Vấn đề:** Mỗi mảnh nhớ được lưu tách biệt trong DB.
- **Hạn chế:** Thiếu cấu trúc $\rightarrow$ khó kết nối và kết hợp thông tin khi bộ nhớ lớn.

2. Phương pháp sử dụng cấu trúc tĩnh (Offline)

- Ví dụ: RAPTOR [5], GraphRAG [6].
- **Vấn đề:** Dùng cấu trúc để cải thiện truy xuất.
- **Hạn chế:** Không hỗ trợ cập nhật thời gian thực.
- Khi có thông tin mới $\rightarrow$ phải xây lại toàn bộ cấu trúc (chậm, tốn kém).

Động lực và Giải pháp: MemTree

Động lực:

- Con người tổ chức ký ức bằng **schema** — cấu trúc động để liên kết và sắp xếp thông tin.
- Truyền cảm hứng: Xây dựng cơ chế bộ nhớ động cho LLMs.

Giải pháp: MemTree

- Thuật toán mới tạo ra **cấu trúc bộ nhớ dạng cây động**.
- Tích hợp trực tiếp vào quá trình trò chuyện.
- Quản lý tốt hơn:
  - Các cuộc hội thoại dài.
  - Nhiệm vụ phức tạp đòi hỏi suy luận nhiều bước.
- Khắc phục hạn chế của lookup phẳng và cấu trúc tĩnh.

Minh họa MemTree

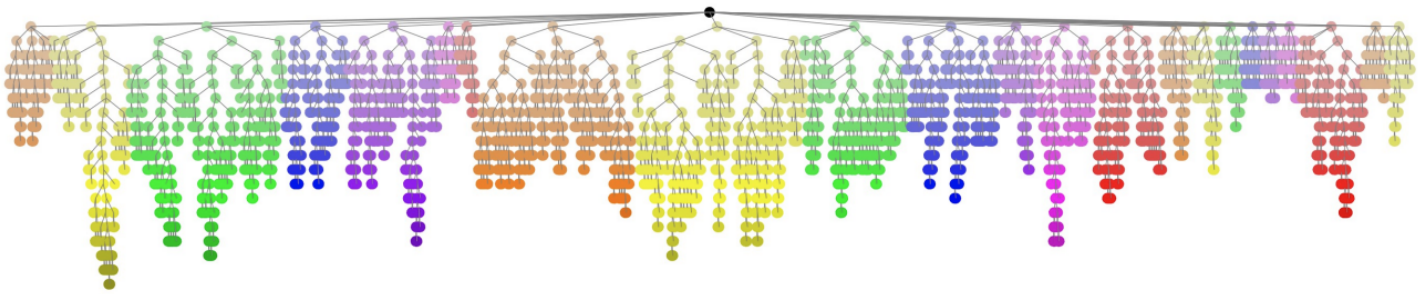


Figure 1: **MemTree** (subset) developed on the MultiHop dataset (Tang & Yang, 2024). MemTree updates its structured knowledge when new information arrives, enhancing inference-time reasoning capabilities of LLMs.

Hình 1: Minh họa cây MemTree

Cấu trúc của MemTree: Cây gia phả tri thức

- **MemTree không phải danh sách phẳng**, mà là một **cấu trúc dạng cây**.
- Tương tự:
  - Hệ thống thư mục trên máy tính.
  - Hoặc một cây gia phả.
- **Nút gốc (Root Node)**:
  - Điểm khởi đầu của cây.
  - Mang tính cấu trúc, **không chứa nội dung**.
- **Các nhánh (Branches)**:
  - Mỗi nhánh đại diện cho một chủ đề hoặc một mảnh thông tin.

Thành phần của một nút trong MemTree

Mỗi nút trong cây MemTree chứa **4 yếu tố quan trọng**:

1. **Nội dung (c_v)**:
   - Văn bản, có thể là tóm tắt hoặc chi tiết cụ thể.
2. **Embedding (e_v)**:
   - Biểu diễn ý nghĩa nội dung.
3. **Liên kết Cha–Con (Parent–Child Links)**:
   - Mỗi nút biết ai là "cha" và ai là các "con" của mình.
4. **Độ sâu (d_v)**:
   - Xác định mức bậc (level) của nút trong cây.

Ý tưởng chính:

- Các nút gần gốc $\rightarrow$ thông tin **tổng quát**.
- Càng xuống sâu $\rightarrow$ thông tin **chi tiết, cụ thể**.

Cách MemTree học hỏi và phát triển

Bước 1: Hành trình bắt đầu từ gốc

- Thông tin mới $\rightarrow$ tạo thành một nút mới (v_{new}).
- Biến thông tin thành embedding vector.
- Hành trình tìm vị trí phù hợp bắt đầu từ **nút gốc**.

Bước 2: Quyết định tại mỗi ngã rẽ

- So sánh embedding của v_{new} với embedding của **các nút con trực tiếp** (dùng cosine similarity).
- 3 tình huống có thể xảy ra:
  1. **Đi sâu hơn (Traverse Deeper)**: Nếu rất giống một nút con (> ngưỡng).
  2. **Tạo nhánh mới (New Leaf)**: Nếu không đủ giống bất kỳ nút con nào.
  3. **Mở rộng nút lá (Expand Leaf)**: Nếu dừng ở nút lá $\rightarrow$ "nâng cấp" thành nút cha mới.

Ngưỡng thích ứng (Adaptive Threshold)

- **Ngưỡng độ tương đồng không cố định**.

$$\theta(d) = \theta_0 e^{\lambda d}$$

  - d: độ sâu của nút.
  - θ_0: ngưỡng cơ bản tại gốc.
  - λ: hệ số điều chỉnh tốc độ tăng.
- Càng **đi sâu vào cây**, ngưỡng càng **cao hơn**.
- Ý nghĩa:
  - Mức gốc $\rightarrow$ chấp nhận thông tin tổng quát.
  - Mức sâu hơn $\rightarrow$ chỉ nhận thông tin cực kỳ liên quan.

Lợi ích:

- Giúp cây duy trì cấu trúc chặt chẽ.
- Thông tin ở sâu $\rightarrow$ chi tiết, cụ thể.
- Thông tin gần gốc $\rightarrow$ khái quát, tóm lược.

Cập nhật ngược (Update Backwards) và bảo toàn dữ liệu

Bước 3: Cập nhật ngược

- Sau khi chèn nút mới, thuật toán đi ngược lên cây.
- Các nút cha/ông được **LLM tổng hợp** nội dung cũ + mới.
- Kết quả: tạo bản tóm tắt toàn diện hơn.

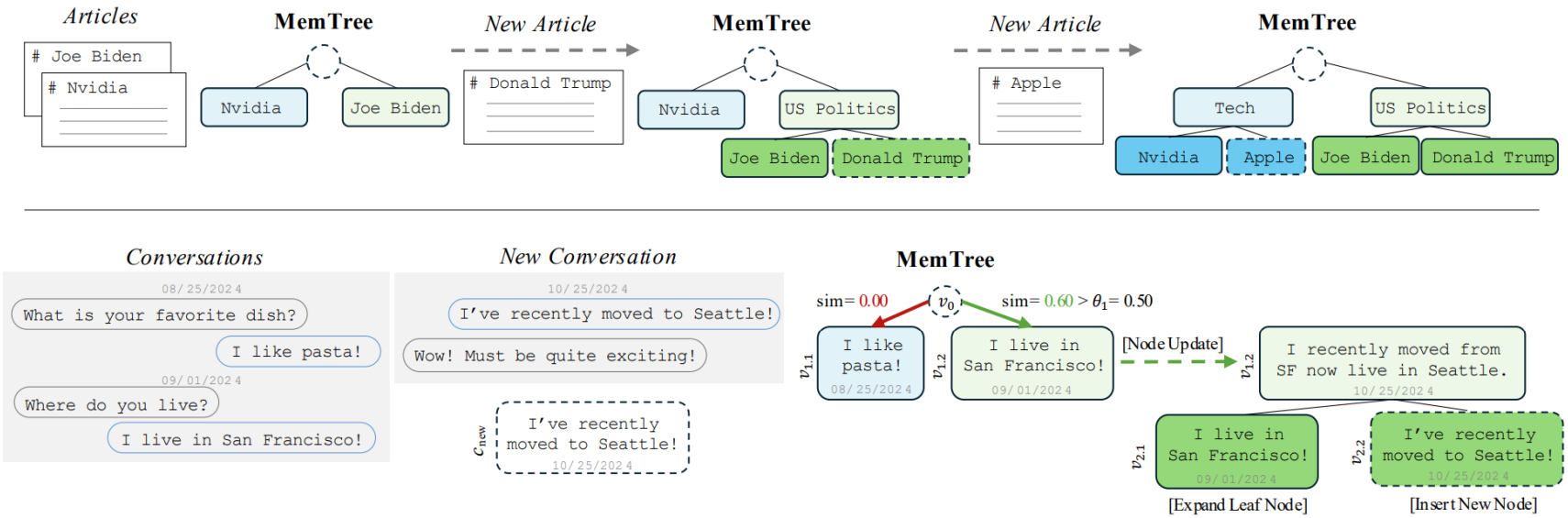
Nguyên tắc bảo toàn dữ liệu:

- **Dữ liệu gốc không bao giờ bị mất**.
- Khi một nút lá được nâng cấp:
  - Nội dung cũ trở thành nút con.
  - Thông tin mới cũng trở thành nút con khác.
  - Nút cha mới chỉ chứa **tóm tắt AI-generated**.

Kiểm soát LLM:

- Chỉ được tóm tắt dựa trên dữ liệu trong cây (không dùng kiến thức riêng).
- Đảm bảo MemTree được đánh giá đúng về **khả năng truy xuất**.

Minh họa cập nhật MemTree



Hình 2: Cập nhật MemTree

Truy xuất thông tin trong MemTree

Ý tưởng chính: Collapsed Tree Retrieval

- "Làm phẳng" cây bộ nhớ $\rightarrow$ coi tất cả nút (cả tóm tắt lẫn chi tiết) như một tập hợp duy nhất.
- Tìm kiếm không cần duyệt tuần tự qua từng nhánh.

Quy trình:

1. **Mã hoá truy vấn:** Câu hỏi $\rightarrow$ vector embedding.
2. **So sánh toàn diện:** So sánh embedding truy vấn với embedding của tất cả các nút.
3. **Lọc và chọn:** Loại bỏ nút kém liên quan, giữ lại top-k nút giống nhất.
4. **Tổng hợp:** Nội dung từ top-k nút $\rightarrow$ LLM để tạo câu trả lời cuối cùng.

Tại sao Collapsed Retrieval hiệu quả?

- Kết hợp thông tin từ **nhiều cấp độ trừu tượng**:
  - Nút nông (cha) $\rightarrow$ **tóm tắt khái quát**.
  - Nút sâu (lá) $\rightarrow$ **chi tiết cụ thể**.
- MemTree giống như một **cuốn sách thông minh**:
  - **Mục lục / tiêu đề chương** = nút cha.
  - **Các trang chi tiết** = nút lá.
- Đảm bảo câu trả lời vừa **tổng quát**, vừa **chính xác chi tiết**.

Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory, 2025 [8]

Mục tiêu của Mem0

- **Bền vững:** Lưu trữ tri thức quan trọng trong bộ nhớ dài hạn.
- **Tối giản:** Chỉ giữ lại fact cốt lõi thay vì các chunk văn bản lớn.
- **Truy hồi có cấu trúc:**
  - Tận dụng quan hệ giữa thực thể.
  - Hỗ trợ suy luận đa bước và bài toán temporal.

Kiến trúc tổng thể Mem0

- Pipeline gồm **hai giai đoạn chính:**
  1. **Trích xuất (Extraction)**: nhận diện và tạo candidate memory từ tương tác hội thoại.
  2. **Cập nhật (Update)**: hợp nhất, chỉnh sửa hoặc bổ sung vào bộ nhớ lâu dài.
- Đầu vào: cặp thông điệp (m_{t-1}, m_t)
  - m_t: thông điệp hiện tại (thường là phản hồi trợ lý).
  - m_{t-1}: thông điệp liền trước (thường là tin nhắn người dùng).
  - $\Rightarrow$ Tạo thành một đơn vị tương tác hoàn chỉnh.

Giai đoạn Trích xuất

Ngữ cảnh hỗ trợ trích xuất:

- **Bản tóm tắt hội thoại S:**
  - Được lấy từ cơ sở dữ liệu.
  - Bao quát ngữ nghĩa xuyên suốt toàn bộ lịch sử hội thoại.
- **Chuỗi thông điệp gần $\{m_{t-m}, \ldots, m_{t-2}\}$:**
  - Cung cấp chi tiết thời gian gần.
  - Có thể chứa thông tin mới chưa được gộp vào tóm tắt.
  - m: siêu tham số quy định cửa sổ ngữ cảnh.

Mô-đun tóm tắt bất đồng bộ:

- Định kỳ làm mới bản tóm tắt S.
- Hoạt động độc lập với pipeline chính.
- Đảm bảo ngữ cảnh luôn cập nhật mà không gây trễ xử lý.

Hàm trích xuất bộ nhớ

Prompt trích xuất:

$$P = (S, \{m_{t-m}, \ldots, m_{t-2}\}, m_{t-1}, m_t)$$

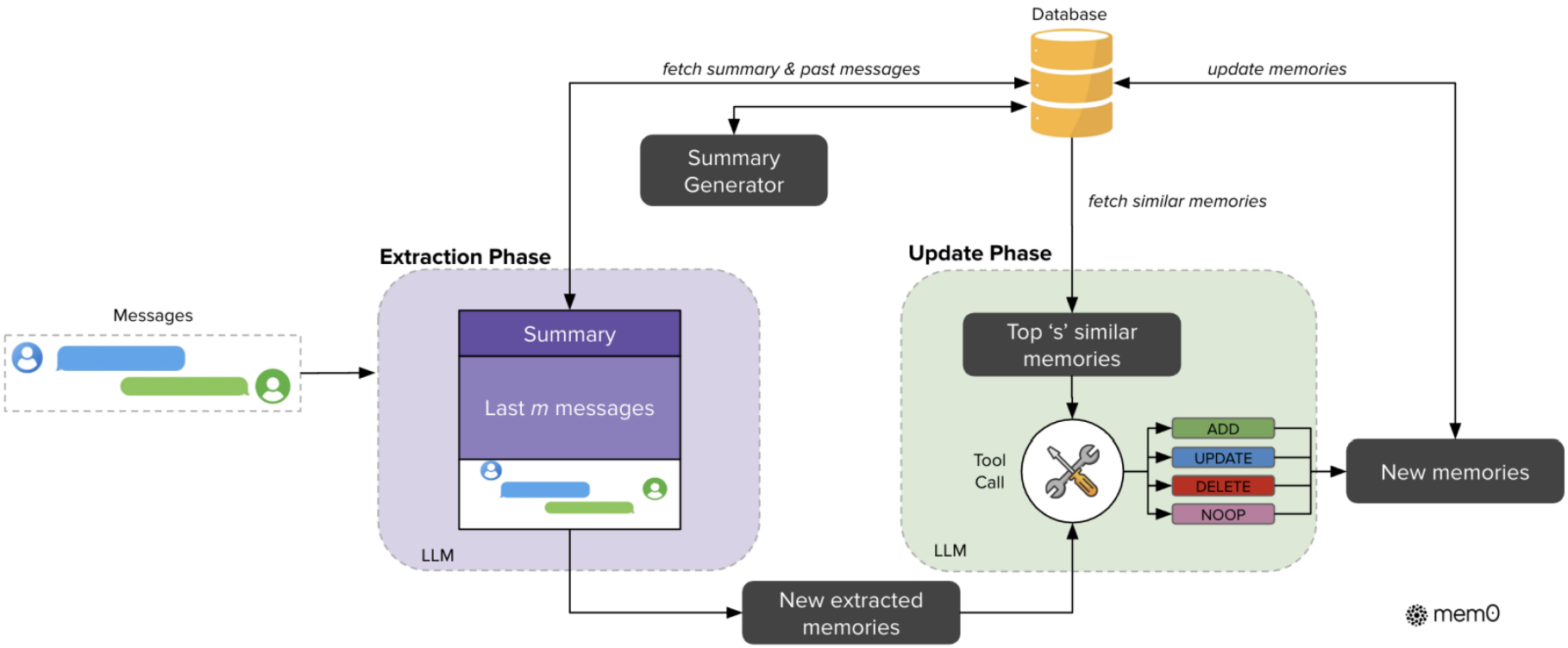
Hàm trích xuất:

$$\phi(P) \rightarrow \Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$$

Giải thích

- S: bản tóm tắt hội thoại (tổng thể, xuyên suốt).
- $\{m_{t-m}, \ldots, m_{t-2}\}$: chuỗi thông điệp gần (ngữ cảnh chi tiết).
- (m_{t-1}, m_t): cặp tương tác mới nhất.
- ϕ: LLM-based extraction function.
- Ω: tập bộ nhớ nổi bật, ngắn gọn, phản ánh thông tin quan trọng từ trao đổi mới.

Minh họa kiến trúc Mem0



Hình 3: Kiến trúc Mem0

Giai đoạn Cập nhật Bộ nhớ

Quy trình cập nhật

1. Với mỗi dữ kiện ứng viên $\omega_i \in \Omega$, truy xuất **top-s bộ nhớ tương đồng** từ CSDL (dựa trên dense embeddings).
2. Đưa ω_i và các bộ nhớ tương đồng vào LLM thông qua *tool call*.
3. LLM quyết định thao tác quản lý bộ nhớ phù hợp.

Bốn thao tác khả dĩ

- **ADD** – Tạo bộ nhớ mới (chưa có tương đương).
- **UPDATE** – Bổ sung/ghi đè thông tin cho bộ nhớ hiện có.
- **DELETE** – Loại bỏ bộ nhớ khi thông tin mới phủ định.
- **NOOP** – Không thay đổi (giữ nguyên).

Tham số: Tham số ngữ cảnh: $m = 10$ (thông điệp gần), số bộ nhớ so sánh: $s = 10$, LLM dùng để suy luận: GPT-4o-mini và Vector DB: dense embeddings (tìm kiếm tương đồng).

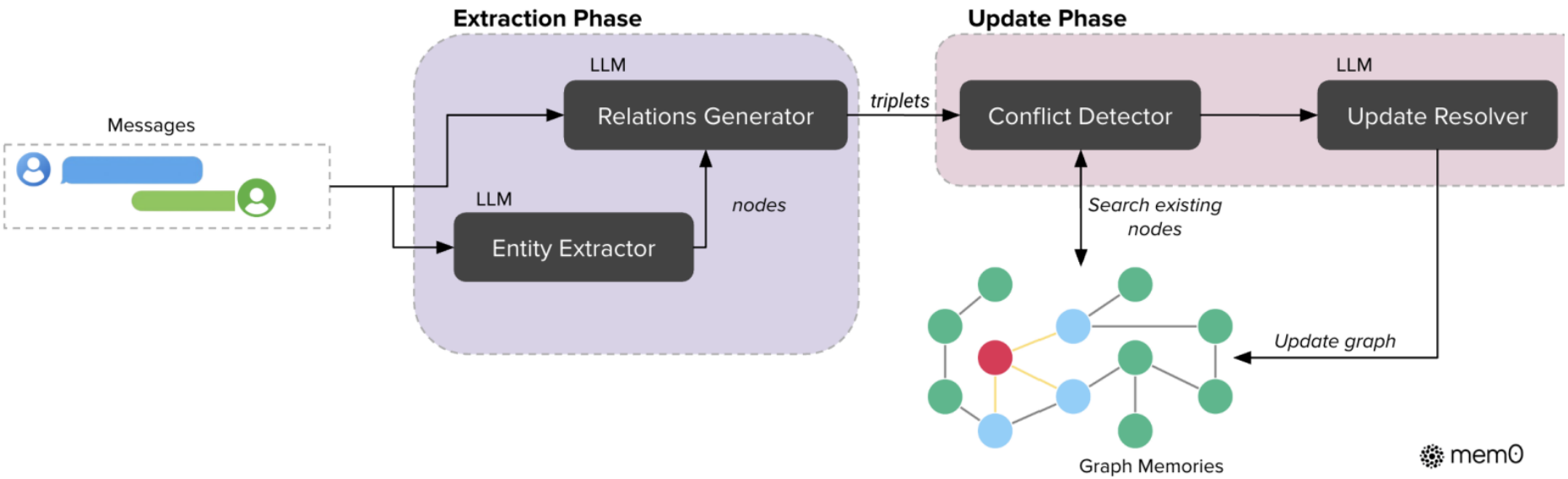
Mem0g: Biểu diễn đồ thị bộ nhớ

- Đồ thị có hướng gán nhãn $G = (V, E, L)$.
- **Nút** V: thực thể (Alice, San Francisco).
- **Cạnh** E: quan hệ (lives_in).
- **Nhãn** L: kiểu ngữ nghĩa (Person, City).
- Mỗi nút $v \in V$ gồm:
  1. Phân loại kiểu thực thể (Person, Location, Event,...).
  2. Vector embedding e_v (ngữ nghĩa).
  3. Metadata: timestamp t_v.

Mem0g: Trích xuất quan hệ

- Bộ sinh quan hệ dựa trên LLM.
- Phân tích ngữ cảnh hội thoại + thực thể.
- Sinh bộ ba (v_s, r, v_d):
  - v_s: nút nguồn, v_d: nút đích, r: quan hệ.
  - Ví dụ: (Alice, lives_in, San Francisco).
- Prompt engineering hướng LLM:
  - Khai thác thông tin hiển ngôn và hàm ẩn.
  - Sinh cạnh có nhãn phản ánh quan hệ ngữ nghĩa.

Minh họa kiến trúc Mem0g



Hình 4: Kiến trúc Mem0g

Mem0g: Lưu trữ và Cập nhật

- Khi có bộ ba quan hệ mới:
  1. Tính embedding cho các thực thể.
  2. Tìm nút hiện có với độ tương đồng $> \theta$.
- Chiến lược lưu trữ:
  - Tạo 2 nút mới.
  - Tạo 1 nút mới + tái sử dụng nút cũ.
  - Tái sử dụng toàn bộ nút cũ.
- Duy trì nhất quán:
  - Phát hiện xung đột quan hệ.
  - LLM đánh dấu quan hệ lỗi thời *(invalid)* thay vì xóa.

Mem0g: Truy xuất và Thực thi hệ thống

Truy xuất hai hướng:

1. **Entity-centric**:
   - Xác định thực thể chính trong truy vấn.
   - Tìm nút tương ứng $\rightarrow$ mở rộng bằng cách duyệt lân cận.
   - Xây dựng tiểu đồ thị ngữ cảnh.
2. **Semantic triplet**:
   - Mã hóa truy vấn $\rightarrow$ embedding.
   - So khớp với embedding các bộ ba.
   - Trả về bộ ba vượt ngưỡng liên quan.

Thực thi hệ thống:

- DB: Neo4j (graph store).
- LLM: GPT-4o-mini + function calling.

MemGAS: Towards Multi-Granularity Memory Association and Selection for Long-Term Conversational Agents, 2025 [9]

Thách thức của tác tử hội thoại dựa trên LLM:

- Cần duy trì **ký ức** qua nhiều phiên / tương tác kéo dài.
- Khi chỉ "nhét" lịch sử vào ngữ cảnh, nảy sinh 2 vấn đề:
  1. **Thiếu liên kết xuyên phiên**: tri thức phân tán ở nhiều đoạn, khó kết nối.
  2. **Nhiễu truy hồi**: mảnh văn bản quá thô hoặc quá chi tiết, không khớp truy vấn.

Hạn chế của RAG truyền thống và hệ trí nhớ hiện có:

- Thường **cố định mức bộ nhớ** (theo phiên hoặc lượt).
- Có cấu trúc (cây/đồ thị) nhưng vẫn thiên về một mức.
- Hệ quả: bỏ sót **tương quan chéo** giữa các mức và thiếu cơ chế **chọn cấp bộ nhớ phù hợp** theo truy vấn.

Mục tiêu và Khung MemGAS

Mục tiêu của MemGAS:

1. **Bảo toàn đa góc nhìn ký ức** (multi-granularity).
2. **Liên kết thích nghi** giữa ký ức mới và cũ.
3. **Chọn cấp bộ nhớ phù hợp** theo từng truy vấn.

Khung tổng thể gồm 2 pha:

- **Xây Liên kết**:
  - Tạo tóm tắt, từ khoá, đoạn lượt.
  - Mã hoá và liên kết bằng **GMM**.
- **Truy hồi và Lựa chọn**:
  - Định tuyến theo **entropy**.
  - **PPR** trên đồ thị liên kết.
  - Lọc bằng **LLM**.

Đơn vị nhớ đa mức $M_i = \{S_i, T_i, U_i, K_i\}$

Định nghĩa (với mỗi phiên S_i):

- S_i (**session view**): Phiên i ghép thành một chuỗi, giữ thứ tự.
- T_i (**turn view**): list các lượt (cặp User–Assistant) trong phiên i.
- U_i (**summary**): tóm tắt phiên i (sinh bởi LLM).
- K_i (**keyword**): từ khoá của phiên i (sinh bởi LLM).

Cách mã hoá (embedding):

- Encoder dense (Contriever / MPNet / MiniLM) $\rightarrow$ thu embedding: $e(S_i),\ e(U_i),\ e(K_i),\ e(t)$ cho từng lượt $t \in T_i$
- Với T_i: $e(T_i) = \frac{1}{|T_i|} \sum_{t \in T_i} e(t)$ (mean-pool theo lượt)

Quan trọng: $e(S_i) \neq e(T_i)$ do cách mã hoá khác nhau: S_i: encode toàn văn một lần và T_i: trung bình embedding từng lượt.

Đồ thị ký ức: Mỗi phiên i sinh **4 nút**: $\{S_i, T_i, U_i, K_i\}$.

Liên kết Ký ức Động (Dynamical Memory Association)

Cập nhật bộ nhớ:

- Khi thêm một ký ức mới M_{new}, bộ nhớ hiện tại được cập nhật:

$$M_{\text{cur}} \leftarrow M_{\text{cur}} \cup M_{\text{new}}$$

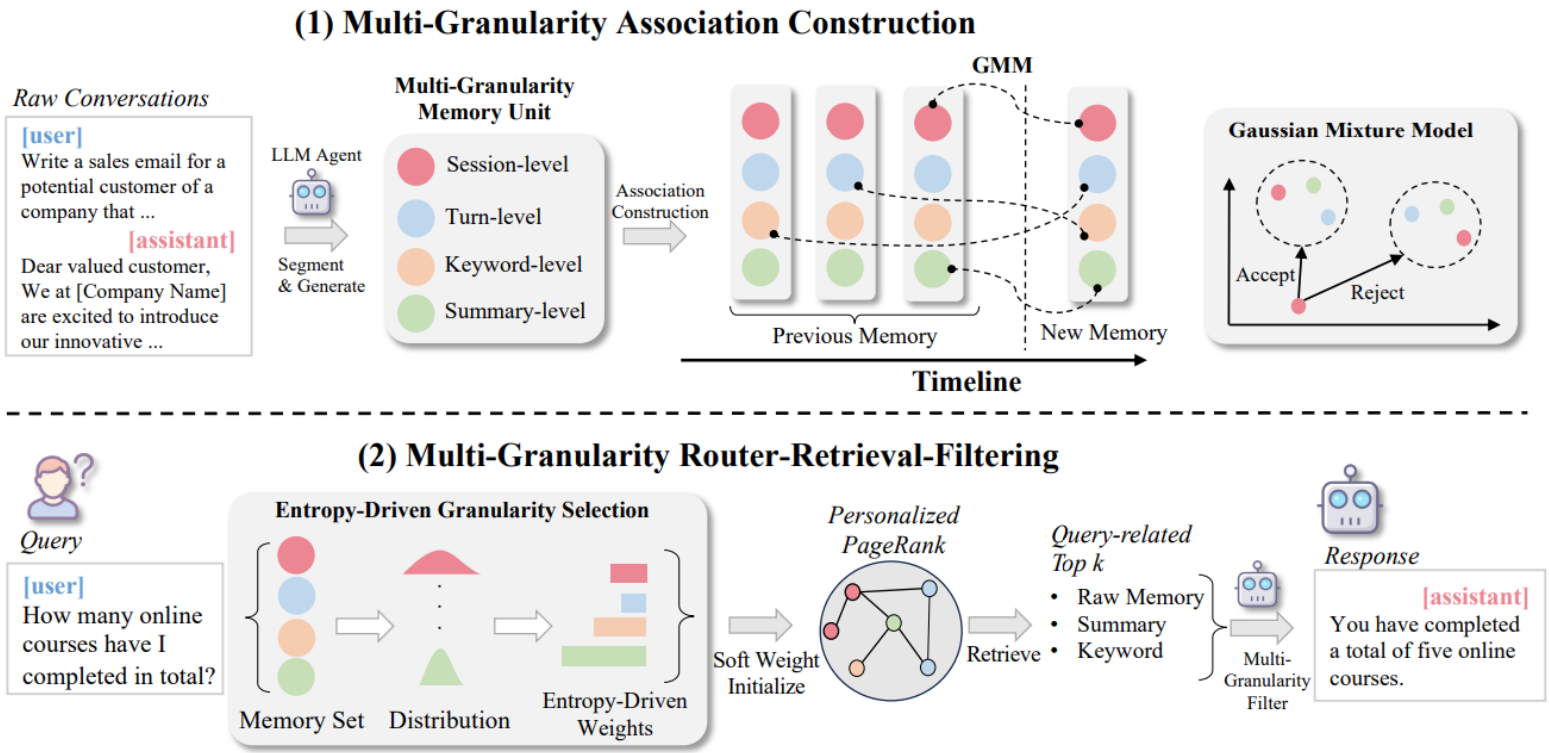
Liên kết với ký ức cũ:

- Tất cả ký ức được mã hoá thành vector đặc (dense) bằng **Contriever**.
- Tính độ tương đồng giữa $e(M_{\text{new}})$ và mỗi $e(M_j) \in e(M_{\text{cur}})$.
- Tập vector tương đồng s_{sim} được gom cụm bằng **Gaussian Mixture Model (GMM)** thành 2 nhóm:
  - **Tập chấp nhận (Accept Set):** ký ức tương đồng cao, liên kết trực tiếp với M_{new}.
  - **Tập loại bỏ (Reject Set):** ký ức không liên quan, không tạo kết nối ngay.

Đặc điểm:

- Tương đồng được tính theo từng **cấp độ chi tiết** (granularity).
- Mỗi cấp của M_{new} là một nút, kết nối tới các nút trong M_{cur}.
- Đồ thị liên kết A_{cur} được cập nhật: $A_{\text{cur}} \leftarrow A_{\text{cur}} \cup A_{\text{new}}$

Minh họa luồng hoạt động của MemGAS



Hình 5: MemGAS

Bộ định tuyến đa mức (Multi-Granularity Router)

Vấn đề:

- Các phương pháp hiện tại dùng *một mức nhớ cố định* (granularity) để truy hồi.
- Thiếu linh hoạt khi cần ưu tiên thông tin **tổng quát** (coarse) hoặc **chi tiết** (fine).

Ý tưởng chính: Entropy-based Router

- Với một truy vấn q, tính độ tương đồng với tất cả mảnh nhớ ở mỗi mức: $g \in \{\text{session}, \text{turn}, \text{summary}, \text{keyword}\}$
- Vector điểm tương đồng tại mức g (n là memories chunks ở granularity g): $s_g = [\text{sim}(q, M_1^g), \ldots, \text{sim}(q, M_n^g)]$
- Chuẩn hoá thành phân phối xác suất:

$$p_i^g(s_g) = \frac{\exp(\text{sim}(q, M_i^g)/\lambda)}{\sum_{j=1}^{n} \exp(\text{sim}(q, M_j^g)/\lambda)}$$

- Tính entropy Shannon để đo độ bất định: $H_g = -\sum_{i=1}^{n} p_i^g(s_g) \log p_i^g(s_g)$

Soft Router Weights và Ý nghĩa

Nguyên lý:

- H_g thấp $\Rightarrow$ phân phối tập trung $\Rightarrow$ độ chắc chắn cao $\Rightarrow$ tin cậy.
- H_g cao $\Rightarrow$ phân phối dàn trải $\Rightarrow$ bất định lớn $\Rightarrow$ ít tin cậy.

Tính trọng số mềm:

$$w_g = \frac{1/H_g}{\sum_{g'=1}^{G} 1/H_{g'}}$$

trong đó G là số mức nhớ.

Ý nghĩa:

- Mỗi mức nhớ được gán trọng số **tỷ lệ nghịch** với mức độ bất định.
- Làm nổi bật các mức có độ chắc chắn cao nhất khi truy hồi.
- Giúp hệ thống **tự động chọn mức nhớ phù hợp** cho từng truy vấn, không cần can thiệp thủ công.

Trực quan:

- Query $q \rightarrow$ so khớp với 4 mức $\rightarrow$ tính entropy $H_g \rightarrow$ Router $\rightarrow$ trọng số $w_g \rightarrow$ chọn lọc ký ức.

Memory Retrieval: Personalized PageRank (PPR)

Mục tiêu: Truy hồi các ký ức đa mức liên quan nhất với truy vấn q.

Bước 1: Khởi tạo điểm liên quan

$$\text{score}_i = w_g \cdot \text{sim}(q, M_i^g)$$

- w_g: trọng số mức nhớ từ Entropy-based Router.
- $\text{sim}(q, M_i^g)$: cosine similarity giữa embedding truy vấn $e(q)$ và embedding của mảnh nhớ $e(M_i^g)$.

Bước 2: Personalized PageRank (PPR)

- Chọn Top-α nodes làm **seed**.
- Chạy PPR trên đồ thị liên kết ký ức (M, A).
- Tín hiệu liên quan được lan truyền qua các cạnh $\rightarrow$ nhấn mạnh các ký ức vừa **liên quan trực tiếp** vừa **kết nối mạnh**.

Bước 3: Chọn ứng viên

- Sau hội tụ, chọn Top-K nodes theo PPR score làm **candidate context**.
- K được phân tích thực nghiệm.

LLM-Based Redundancy Filtering

Vấn đề:

- Truy hồi đa mức dễ gây **nhiễu** và **trùng lặp**.
- Nếu giữ nguyên, sẽ làm phản hồi dài và kém tập trung.

Giải pháp: Lọc bằng LLM

- Input: Top-K memories + truy vấn q.
- Prompt đặc thiết kế hướng dẫn LLM:
  - Loại bỏ nội dung **không liên quan**.
  - Gộp bỏ nội dung **lặp lại**.

Kết quả:

- Context cuối cùng: **ngắn gọn, tập trung, cá nhân hoá**.
- Đảm bảo chỉ giữ lại thông tin **thiết yếu nhất** cho việc sinh phản hồi.

4. Agent Memory Manager

Tại sao cần Agent quản lý bộ nhớ?

- LLM có giới hạn context window $\rightarrow$ cần cơ chế thêm/xóa thông tin có chọn lọc.
- Tránh nhiễu, trùng lặp, và đảm bảo thông tin nhất quán trong dài hạn.
- Cho phép tác vụ phức tạp (lập kế hoạch, reasoning) dựa trên bộ nhớ được tổ chức.

Các chức năng chính của Agent Memory Manager

- **Thêm (Insert):** lưu hội thoại, sự kiện, tri thức mới vào bộ nhớ (theo layer hoặc schema).
- **Xóa (Delete):** loại bỏ thông tin dư thừa, hết hạn hoặc mâu thuẫn.
- **Sửa/Update:** cập nhật thông tin khi có phiên bản mới hoặc điều chỉnh tri thức.
- **Truy xuất (Retrieve):** tìm thông tin liên quan theo ngữ cảnh hoặc truy vấn.
- **Tóm tắt (Summarize):** nén ngữ cảnh cũ thành phiên bản ngắn gọn để lưu lâu dài.

Agent Memory Manager

Ý tưởng triển khai

- Xây dựng **Memory Manager Agent** làm "API" giữa LLM và bộ nhớ.
- Tách biệt các lớp bộ nhớ: hot (ngắn hạn), warm (tóm tắt), cold (lưu trữ lâu dài).
- Dùng policy/heuristics hoặc LLM để quyết định khi nào thêm, xóa, sửa.

Ví dụ: MemGPT [4], Mem0 [8] và MIRIX [10] đều triển khai các chức năng này với kiến trúc khác nhau.

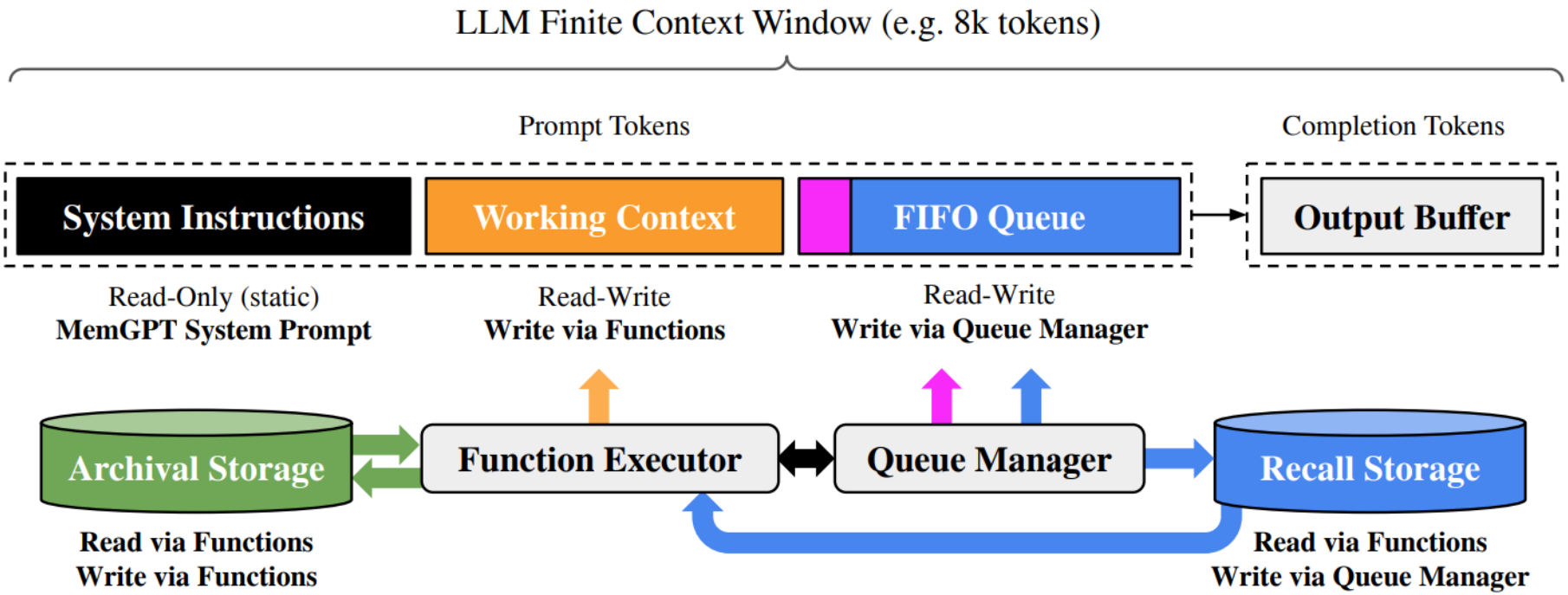
MemGPT: Towards LLMs as Operating Systems, 2024 [4]

- **Hạn chế ứng dụng:**
  - Giới hạn cửa sổ ngữ cảnh làm suy giảm hiệu quả trong:
    - Hội thoại kéo dài.
    - Phân tích văn bản/tài liệu lớn.
- **Mở rộng ngữ cảnh không phải giải pháp tối ưu:**
  - **Chi phí tính toán cao:** Độ dài ngữ cảnh $\uparrow \Rightarrow$ chi phí bộ nhớ và tính toán tăng theo **bậc hai** do self-attention.
  - **Hiệu quả giảm dần ("lost in the middle"):** LLM thường nắm bắt thông tin ở **đầu/cuối** tốt hơn, nhưng thông tin ở **giữa** lại dễ bị bỏ sót.

MemGPT

- **Ảo giác ngữ cảnh vô hạn:** Mục tiêu: tạo cảm giác rằng LLM có thể làm việc với **context vô hạn**.
- **Quản lý bộ nhớ ảo (Virtual Context Management):** Giống như OS sử dụng *paging*, MemGPT cho phép **di chuyển dữ liệu** giữa:
  - Bộ nhớ chính (cửa sổ ngữ cảnh của LLM).
  - Bộ nhớ ngoài (external storage).
- **Hệ thống bộ nhớ phân cấp:** Tương tự OS quản lý **RAM tương đương ổ cứng**, MemGPT thiết lập **nhiều tầng bộ nhớ**.
- **LLM tự quản lý bộ nhớ:** LLM quyết định khi nào cần **đọc/ghi dữ liệu** qua các **function calls**, hoạt động như *system calls* trong OS.

Minh họa kiến trúc MemGPT



Hình 6: Kiến trúc MemGPT

MemGPT: Kiến trúc bộ nhớ phân cấp

1. **Main Context (Bộ nhớ chính ≈ RAM):**
   - Cửa sổ ngữ cảnh hữu hạn mà LLM trực tiếp xử lý.
   - Gồm 3 phần:
     - **System Instructions:** Chỉ đọc, chứa hướng dẫn hệ thống và cách gọi hàm.
     - **Working Context:** Vùng đọc/ghi, lưu thông tin quan trọng (sự thật, sở thích, mục tiêu).
     - **FIFO Queue:** Hàng đợi tin nhắn gần đây nhất giữa user và agent.
2. **External Context (Bộ nhớ ngoài ≈ Disk):**
   - Lưu trữ ngoài cửa sổ ngữ cảnh của LLM.
   - Gồm 2 phần:
     - **Recall Storage:** Lưu toàn bộ lịch sử hội thoại, có thể tìm kiếm.
     - **Archival Storage:** Lưu tài liệu, file lớn, hoặc khối tri thức người dùng cung cấp.

MemGPT: Cơ chế hoạt động như một Hệ điều hành

- **Luồng điều khiển theo sự kiện (Event-driven):** MemGPT được kích hoạt bởi các sự kiện như:
  - Tin nhắn từ người dùng.
  - Cảnh báo hệ thống (ví dụ: bộ nhớ gần đầy).
  - Các sự kiện được hẹn giờ.
- **Cảnh báo áp lực bộ nhớ (Memory Pressure):** Khi hàng đợi FIFO Queue gần đầy, một *Queue Manager* sẽ gửi cảnh báo vào ngữ cảnh.

MemGPT: LLM như CPU điều phối bộ nhớ

LLM tự ra quyết định khi nhận cảnh báo:

- **Lưu trữ thông tin quan trọng:**
  - Di chuyển dữ liệu từ FIFO Queue $\rightarrow$ Working Context (RAM).
  - Hoặc lưu dài hạn vào Archival Storage (Disk).
- **Truy xuất thông tin:**
  - Tìm trong Recall Storage (lịch sử hội thoại).
  - Hoặc tìm trong Archival Storage (tài liệu, file).

Tương tự OS

LLM đóng vai trò như **CPU**, chủ động quản lý dữ liệu giữa RAM (Main Context) và Disk (External Context).

MemGPT: Cơ chế tự chỉnh sửa và truy xuất

- **Điều phối dữ liệu tự động:** LLM chủ động quyết định khi nào cần di chuyển các mục giữa Main Context và External Context.
  - Ví dụ: khi lịch sử hội thoại trở nên quá dài.
  - Chỉnh sửa ngữ cảnh chính để phản ánh mục tiêu, trách nhiệm hiện tại.
- **System Instructions:** Định hướng cho LLM cách sử dụng hệ thống bộ nhớ:
  - Mô tả chi tiết hệ thống bộ nhớ phân cấp và chức năng.
  - Lược đồ hàm (function schema) + mô tả tự nhiên cho từng thao tác.

MemGPT: Vòng lặp phản hồi tự động

- **Chu kỳ suy luận:**
  1. LLM lấy Main Context (RAM) nối chuỗi $\rightarrow$ sinh ra output.
  2. MemGPT phân tích output, kiểm tra lời gọi hàm.
  3. Nếu hợp lệ $\rightarrow$ thực thi hàm; nếu lỗi $\rightarrow$ trả runtime error về LLM.
- **Vòng lặp phản hồi:**
  - Hệ thống học từ hành động của chính nó.
  - Điều chỉnh hành vi theo giới hạn ngữ cảnh và token.
- **Cơ chế bổ trợ:**
  - Cảnh báo giới hạn token để định hướng quản lý bộ nhớ.
  - Truy xuất có phân trang (pagination) để tránh tràn cửa sổ ngữ cảnh.

Control Flow trong MemGPT

- **Events → LLM Inference:**
  - Sự kiện = đầu vào tổng quát của MemGPT.
  - Bao gồm:
    1. Tin nhắn người dùng (chat).
    2. Tin nhắn hệ thống (cảnh báo dung lượng context).
    3. Tương tác người dùng (vd: upload tài liệu xong).
    4. Timed events (chạy định kỳ, không cần prompt).
  - MemGPT parser chuyển sự kiện → văn bản thuần → đưa vào Main Context → LLM xử lý.
- **Ý nghĩa:** Cho phép MemGPT phản ứng đa dạng, không chỉ khi người dùng gửi prompt.

MIRIX: Multi-Agent Memory System for LLM-Based Agents, 2025 [10]

- **Thiếu cấu trúc và trừu tượng hóa**: Lưu trữ dạng flat vector store hoặc graph đơn giản, khó truy xuất hiệu quả.
- **Multi-modal kém**: Chủ yếu cho văn bản, ít hỗ trợ hình ảnh/giao diện/dữ liệu phi văn bản.
- **Khả năng mở rộng hạn chế**: Lưu dữ liệu thô (ảnh độ phân giải cao) gây tốn dung lượng, thiếu lớp trừu tượng.
- **Stateless**: Chỉ ghi nhớ trong một phiên hội thoại, không duy trì thông tin lâu dài của người dùng.

Giải pháp MIRIX

- **Kiến trúc bộ nhớ module hóa**: Sáu loại bộ nhớ chuyên biệt, lấy cảm hứng từ bộ nhớ con người.
- **Multi-Agent Framework**: Nhiều agent phối hợp để cập nhật, điều phối, và truy xuất thông tin hiệu quả.
- **Hỗ trợ đa phương thức từ gốc**: Xử lý đồng thời văn bản và hình ảnh $\rightarrow$ suy luận đa phương thức.
- **Cơ chế Truy xuất Chủ động (Active Retrieval)**: Agent tự động nhận diện chủ đề hội thoại và gọi tool để truy xuất bộ nhớ.

MIRIX: Sáu Module Bộ Nhớ Chức Năng

1. **Core Memory (Bộ nhớ Lõi)**: Thông tin bền vững về người dùng và agent (tên, sở thích, tính cách). Luôn nằm trong ngữ cảnh.
2. **Episodic Memory (Bộ nhớ Sự kiện)**: Nhật ký có cấu trúc về sự kiện/trải nghiệm theo dòng thời gian (vd: "Người dùng tham gia họp lúc 10h sáng").
3. **Semantic Memory (Bộ nhớ Ngữ nghĩa)**: Kiến thức trừu tượng, khái niệm, sự thật, mối quan hệ (vd: "Hà Nội là thủ đô VN").
4. **Procedural Memory (Bộ nhớ Quy trình)**: Các hướng dẫn từng bước để thực hiện tác vụ/quy trình (vd: "cách commit code lên Git").
5. **Resource Memory (Bộ nhớ Tài nguyên)**: Lưu trữ tài liệu, file, media liên quan công việc (vd: PDF, ghi âm họp).
6. **Knowledge Vault (Kho Tri Thức)**: Lưu thông tin nhạy cảm, chính xác tuyệt đối (vd: mật khẩu, API keys).

Kiến trúc Đa Tác tử (Multi-Agent) trong MIRIX

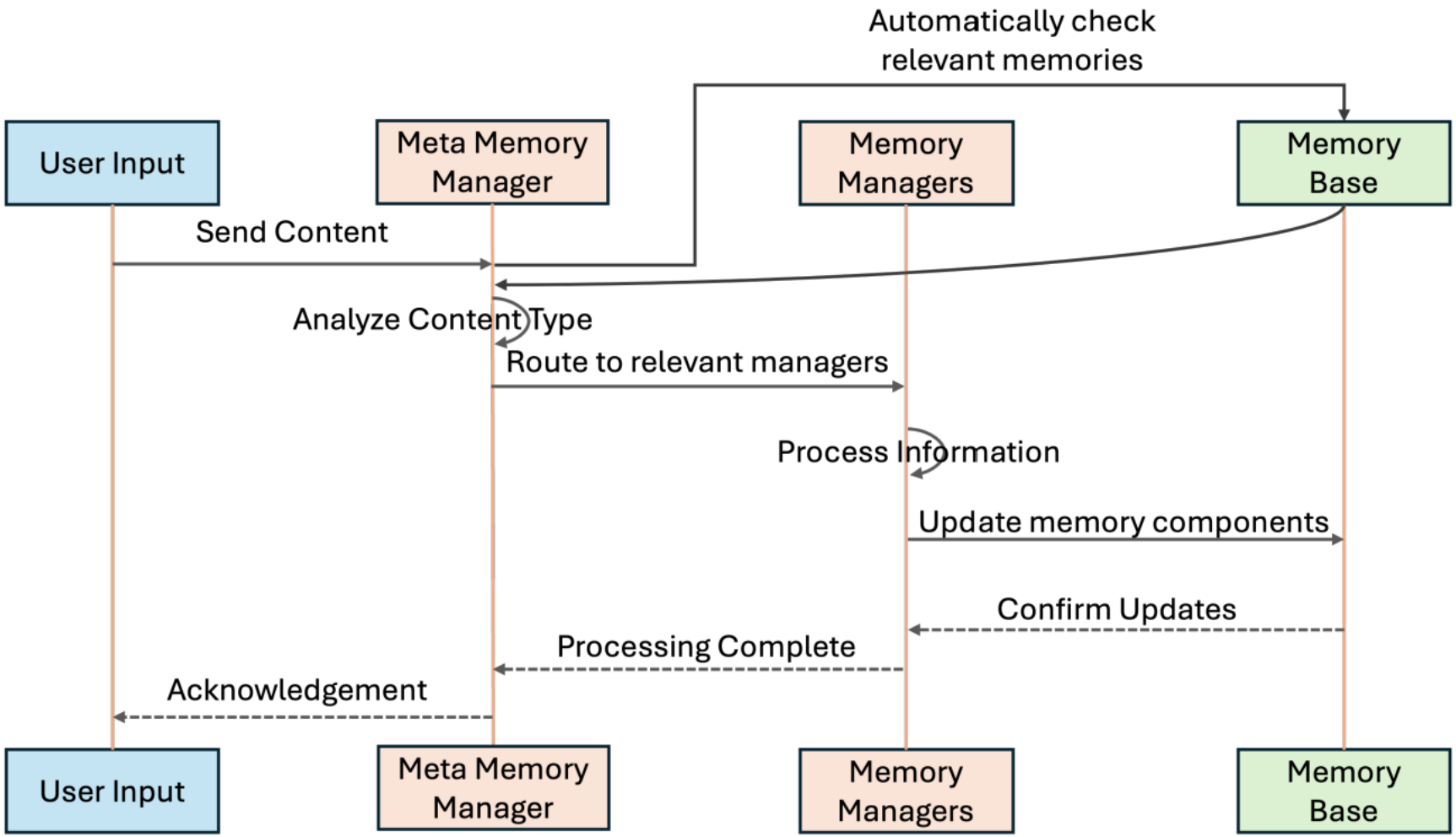
- **Meta Memory Manager** Agent trung tâm, phân tích đầu vào và định tuyến thông tin đến các Memory Manager phù hợp.
- **Memory Managers (6 agents)** Mỗi agent chịu trách nhiệm quản lý, cập nhật và truy xuất cho một trong sáu loại bộ nhớ.
- **Chat Agent** Quản lý giao tiếp với người dùng, phân tích câu hỏi và phối hợp với Memory Managers để tạo câu trả lời.

Ý tưởng chính: Hệ thống đa tác tử dạng mô-đun giúp xử lý đầu vào động và không đồng nhất, đảm bảo cập nhật và truy xuất thông tin hiệu quả trên toàn bộ sáu module bộ nhớ.

Quy trình Cập nhật Bộ nhớ

1. **Nhận đầu vào mới:** Hệ thống tiếp nhận thông tin từ người dùng.
2. **Truy xuất thông tin liên quan:** Tìm kiếm trong kho bộ nhớ để lấy các dữ kiện liên quan.
3. **Xử lý trung tâm:**
   - Bộ Quản lý Siêu Bộ nhớ (Meta Memory Manager) phân tích nội dung.
   - Xác định loại bộ nhớ cần cập nhật.
4. **Định tuyến và cập nhật:**
   - Đầu vào được chuyển đến các Bộ Quản lý Bộ nhớ chuyên biệt.
   - Các bộ này cập nhật song song, đảm bảo tránh trùng lặp thông tin.
5. **Phản hồi và xác nhận:**
   - Các Memory Manager báo cáo lại cho Meta Memory Manager.
   - Hệ thống gửi thông báo xác nhận cập nhật thành công.

Minh họa update bộ nhớ trong MIRIX

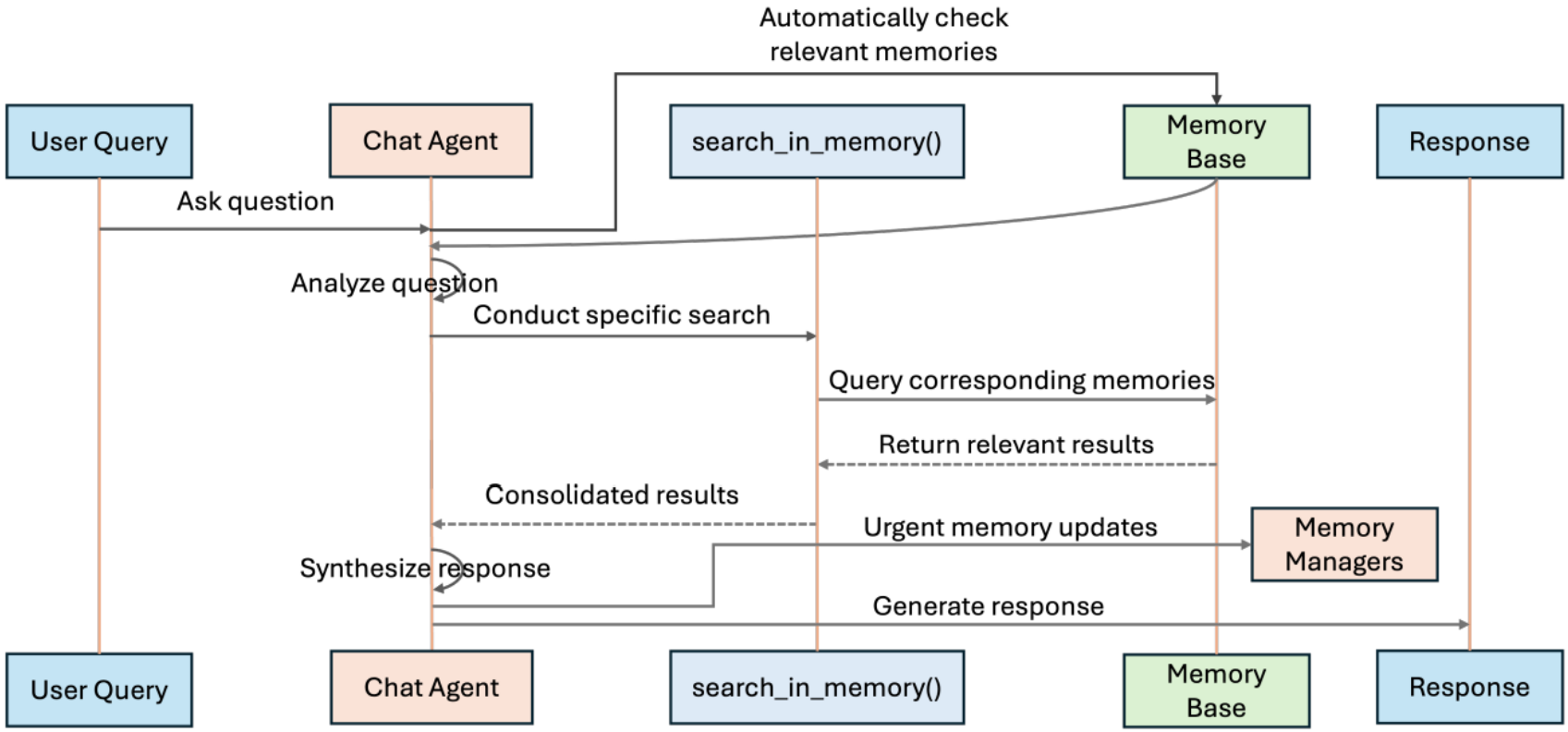


Hình 7: Update bộ nhớ

Quy trình Truy hồi trong Hội thoại

1. **Tiếp nhận truy vấn:** Chat Agent nhận câu hỏi từ người dùng.
2. **Truy hồi thô:**
   - Tìm kiếm trên **toàn bộ 6 loại bộ nhớ**.
   - Chỉ trả về **các tóm tắt cấp cao**, không chi tiết.
3. **Phân tích và truy hồi tinh:**
   - Chat Agent phân tích truy vấn để chọn **bộ nhớ liên quan**.
   - Thực hiện truy xuất sâu hơn với phương pháp phù hợp.
4. **Tổng hợp câu trả lời:** Ghép thông tin từ các bộ nhớ liên quan để sinh ra phản hồi cuối cùng.
5. **Cập nhật khi cần:** Nếu truy vấn chứa thông tin mới (sự kiện, chỉnh sửa), Chat Agent tương tác trực tiếp với Memory Managers để **cập nhật bộ nhớ chính xác**.

Minh họa sinh trả lời



Hình 8: Luồng truy xuất bộ nhớ

5. Datasets and Experiments

Các bộ dữ liệu nền tảng cho hội thoại dài hạn

- **LOCOMO [11] :** Benchmark tiêu chuẩn để đánh giá khả năng ghi nhớ các cuộc hội thoại rất dài, được tạo bởi hai agent có trải nghiệm sống khác nhau qua hàng tuần/ hàng tháng.
- **Long-MT-Bench+ [12]:** Bộ dữ liệu tập trung vào các câu hỏi yêu cầu suy luận trên ngữ cảnh dài, được xây dựng từ cuộc hội thoại giữa con người và chatbot.
- **LongMemEval (s/m) [13]:** Kiểm tra bộ nhớ ở hai quy mô khác nhau (nhỏ và trung bình) để đánh giá khả năng mở rộng. Được tạo bởi người nói chuyện với chatbot qua một thời gian dài.
- **MSC (Multi-Session Chat) & MSC-E [14]:** Các cuộc hội thoại đa phiên hai người nói chuyện với nhau, tập trung vào tính tự nhiên của hội thoại và sự cá nhân hóa.

Bộ dữ liệu LOCOMO: Mô tả và Cấu trúc

Mô tả chung

- **Mục tiêu chính:** Đánh giá chuyên sâu khả năng ghi nhớ dài hạn của các tác tử hội thoại (LLM Agents).
- **Quy mô:** Gồm 10 cuộc hội thoại rất dài, trung bình mỗi cuộc chứa khoảng 600 lượt thoại và 26,000 token.

Phân loại câu hỏi kiểm tra

- **Single-hop (Một bước):** Tìm thông tin cụ thể trong một lượt thoại.
- **Multi-hop (Nhiều bước):** Tổng hợp thông tin từ nhiều phiên khác nhau.
- **Temporal (Thời gian):** Suy luận về trình tự, thứ tự của các sự kiện.
- **Open-domain (Miền mở):** Kết hợp kiến thức trong hội thoại với kiến thức chung.

Bộ dữ liệu Long-MT-Bench+

Nguồn gốc và Mục tiêu

- Được tái cấu trúc từ bộ dữ liệu MT-Bench+ (một tập hội thoại giữa con người và chatbot) để khắc phục các hạn chế về độ dài hội thoại và số lượng câu hỏi yêu cầu ngữ cảnh dài.
- **Mục tiêu:** Tạo ra một benchmark thử thách hơn cho việc đánh giá bộ nhớ và khả năng suy luận trong các cuộc hội thoại rất dài.

Phương pháp Tái cấu trúc

- **Tăng độ dài:** Gộp 5 phiên (sessions) hội thoại liên tiếp thành một cuộc trò chuyện dài duy nhất.
- **Tăng số lượng câu hỏi:** Sử dụng các câu hỏi chất lượng cao do chuyên gia viết làm ví dụ few-shot để GPT-4 tự động sinh thêm các câu hỏi yêu cầu ngữ cảnh dài cho mỗi chủ đề.

Bộ dữ liệu LongMemEval (s/m)

Mục tiêu và Thiết kế

- **Mục tiêu:** Đánh giá khả năng của các trợ lý ảo trong việc xử lý bộ nhớ tương tác dài hạn, đặc biệt là khả năng ghi nhớ động và duy trì sự nhất quán lịch sử.
- **Thiết kế:** Được xây dựng để kiểm tra khả năng mở rộng (scalability) của các hệ thống bộ nhớ bằng cách cung cấp hai quy mô dữ liệu khác nhau.

Hai phiên bản chính

- **LongMemEval-s (small):** Quy mô nhỏ, bao gồm 50 phiên hội thoại cho mỗi câu hỏi, với độ dài trung bình khoảng 103k token.
- **LongMemEval-m (medium):** Quy mô trung bình, mở rộng lên 500 phiên cho mỗi câu hỏi, với độ dài trung bình hơn 1 triệu token.

Bộ dữ liệu Multi-Session Chat (MSC và MSC-E)

MSC: Nền tảng

- Chứa các cuộc hội thoại đa phiên do con người tạo ra, trong đó người tham gia phải duy trì một tính cách (persona) nhất quán, tập trung vào sự mạch lạc, tự nhiên và cá nhân hóa.

- MemGPT giới thiệu các tác vụ chuyên biệt trên MSC như **Deep Memory Retrieval (DMR)** để kiểm tra tính nhất quán và **Conversation Opener** để đánh giá mức độ tương tác.

MSC-E: Phiên bản Mở rộng

- Được tạo ra bởi nhóm tác giả MemTree để tăng độ khó và thử thách các mô hình xử lý ngữ cảnh dài.

- **Điểm khác biệt chính:** Kéo dài mỗi cuộc hội thoại lên đến khoảng 200 lượt (rounds), dài hơn nhiều so với MSC gốc (15 lượt).

Các độ đo đánh giá chính

Đo lường chất lượng câu trả lời (QA Metrics)

- **Độ đo truyền thống:** F1, BLEU, ROUGE, BERTScore (dựa trên so khớp từ vựng và ngữ nghĩa).
- **Độ đo hiện đại: LLM-as-a-Judge (GPT4Score, GPT4o-J)** dùng LLM mạnh để đánh giá toàn diện, được xem là đáng tin cậy nhất.

Đo lường hiệu suất truy xuất (Retrieval Metrics)

- **Recall@k:** Kiểm tra xem thông tin cần thiết có nằm trong top-k kết quả truy xuất không.
- **NDCG@k:** Đánh giá chất lượng xếp hạng, ưu tiên các kết quả liên quan ở vị trí cao hơn.

Bảng 1: So sánh hiệu suất trên LOCOMO

Bảng 1: So sánh các phương pháp trên LOCOMO. J Score là độ đo chính.

Mô hình	**J Score (%)** ↑	**F1** ↑	**BLEU-1** ↑	**ROUGE-L** ↑
Baselines không có bộ nhớ phức tạp				
Full-Context	72.9	12.2	–	11.7
RAG (k=2, 8k)	61.0	–	–	–
Contriever (RAG)	40.3	15.7	2.7	15.0
Mô hình có kiến trúc bộ nhớ				
SeCom	44.2	13.8	2.3	13.3
MemGAS	41.1	**17.7**	**3.6**	**17.0**
Mem0 (Graph)	68.4	–	–	–
MIRIX	**85.4**	–	–	–

Bảng 2: So sánh hiệu suất trên Long-MT-Bench+

Bảng 2: So sánh các phương pháp trên Long-MT-Bench+.

Mô hình	**GPT4Score ↑**	**F1 ↑**	**BLEU ↑**	**ROUGE-L ↑**
Baselines				
Full History	67.4	36.1	11.3	29.3
Contriever (RAG)	63.5	36.3	11.6	29.7
Mô hình có kiến trúc bộ nhớ				
SeCom	64.6	36.7	12.0	29.7
RAPTOR	59.7	37.7	13.5	30.9
HippoRAG 2	63.5	35.6	11.1	28.8
MemGAS	**69.4**	**41.5**	**15.6**	**34.7**

Bảng 3: So sánh hiệu suất trên Multi-Session Chat

Bảng 3: Hiệu suất trên MSC (tác vụ DMR) và MSC-E (hội thoại dài).

Mô hình	**Accuracy (%)** ↑	**ROUGE-L(R)** ↑
Trên MSC (15 lượt thoại, tác vụ DMR)		
GPT-4 (Baseline)	32.1	0.296
GPT-4 + MemGPT	**92.5**	**0.814**
Trên MSC-E (200 lượt thoại)		
Full History (Baseline)	78.0	–
MemoryStream	80.7	–
MemTree	**82.5**	–

Bảng 4: So sánh hiệu suất trên LongMemEval-s

Bảng 4: So sánh QA và Retrieval trên LongMemEval-s.

Mô hình	**J Score (%)** ↑	**F1** ↑	**Recall@10** ↑	**NDCG@10** ↑
Baselines				
Full History	50.6	11.5	–	–
Contriever (RAG)	55.4	13.8	90.0	84.3
Mô hình có kiến trúc bộ nhớ				
SeCom	56.0	13.0	89.2	85.1
HippoRAG 2	57.6	14.7	91.3	88.7
MemGAS	**60.2**	**20.4**	**94.5**	**90.0**

6. Conclusion

- Bộ nhớ dài hạn là yếu tố then chốt cho conversational agents bền vững.
- Phân đoạn và nén ngữ cảnh giúp vượt giới hạn context window và giảm dư thừa.
- Kiến trúc bộ nhớ có cấu trúc (tree, graph) hỗ trợ tổ chức và reasoning hiệu quả hơn.
- Agent Memory Manager cho phép thêm, xóa, cập nhật, tóm tắt và truy xuất bộ nhớ chủ động.
- Dataset và benchmark chuyên biệt là cần thiết để đánh giá khả năng nhớ và suy luận dài hạn.

Tương lai: Hướng tới hệ thống bộ nhớ thống nhất, có cấu trúc và được quản lý bởi agent.

[1] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H Vicky Zhao, Lili Qiu, et al. Secom: On memory construction and retrieval for personalized conversational agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

[2] Zhen Tan, Jun Yan, I Hsu, Rujun Han, Zifeng Wang, Long T Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. *arXiv preprint arXiv:2503.08026*, 2025.

[3] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*, 2024.

[4] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

[5] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024.

[6] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*, 2024.

[7] Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. *arXiv preprint arXiv:2410.14052*, 2024.

[8] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[9] Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, Tong Xu, et al. Towards multi-granularity memory association and selection for long-term conversational agents. *arXiv preprint arXiv:2505.19549*, 2025.

[10] Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

[11] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

[12] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[13] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

[14] Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. *arXiv preprint arXiv:2107.07567*, 2021.