HGP: An on-device personalized agent memory via hybrid graph storage

Ran Zhou[1], Xueming Han[2], Jiaheng Liu[1], Yuyao Zhang[2], Fanyu Meng[2], Junlan Feng[2], Yuxiang Ren[1†]

[1]School of Intelligence Science and Technology, Nanjing University
[2]Jiutian Research

Correspondence: renyuxiang@nju.edu.cn

Abstract

LLM-based agents face challenges in personalized interactive tasks due to heterogeneous, multi-typed, and implicitly constrained long-term traces. Existing memory mechanisms struggle with accurate routing and retrieval, especially on-device where personalization is critical. Most methods use single-vector representations, blurring type distinctions and relational structure. We propose **HGP**, a hybrid graph memory framework. HGP employs a lightweight self-enhancement classifier for personalized memory routing and constructs episodic, semantic, and procedural memories as graphs. It also extracts working memory as a state trajectory to capture current state and implicit constraints, ensuring reliable decision-making. The classifier reduces large-model calls, enabling on-device deployment, while graph storage enables accurate retrieval and incremental user profile refinement. Experiments on two benchmarks show that on PAL-Set solution selection, HGP achieves an S-score of 35.58, nearly 7 points above the strongest baseline. Code and data are at https://github.com/Ouan6/HGP-.git.

1 Introduction

LLM(Large Language Models)-based agents are increasingly deployed in personalized interactive tasks, such as long-term conversational companionship and constraint-aware task planning (Yao et al., 2023; Park et al., 2023). These applications require the agent to access user-related information across turns and generate responses that align with a specific user's preferences and constraints (Zhong et al., 2023; Packer et al., 2024). To compensate for limited context windows and unreliable long-context recall, external memory modules have been widely adopted (Lewis et al., 2021; Zhao et al., 2024; Fan et al., 2024). Most agent memory systems adopt a unified memory pool, compress each memory item into a single-vector representation, retrieve by semantic similarity, and then concatenate retrieved snippets to support generation (Lewis et al., 2021; Zhong et al., 2023). However, such flat memory designs often blur distinctions among memory types and relational structure, which are critical for personalized retrieval and decision making. Robust personalization demands continuously summarizing interaction experience, distilling user preferences and behavioral habits, and reliably activating such signals in subsequent decisions. These capabilities are essential for maintaining consistent user-specific behavior over long interactions (Park et al., 2023; Liu et al., 2023). Moreover, a critical but underexplored bottleneck is personalized memory routing: dynamically assigning incoming interaction traces to the appropriate memory type. In practice, histories contain heterogeneous signals (events, profiles, strategies, and actionable states) (Atkinson and Shiffrin, 1968; Tulving, 1972; Baddeley and Hitch, 1974; Squire, 1992), and inaccurate routing stores information in the wrong memory type, causing cascading retrieval errors (Xu et al., 2025).

Personalized memory framework is needed. We argue that achieving personalization requires a memory system with two key properties: (a) enabling *personalized memory routing* with low supervision cost; and (b) explicitly modeling heterogeneous memory types and their relations. First, personalized routing is important because memory distributions vary across users, making it difficult to rely solely on a fixed general-purpose classifier. Given the cost of repeatedly invoking LLMs during online interaction, especially in on-device contexts, a lightweight routing mechanism that amortizes supervisory cost over time becomes essential. In our setting, on-device operation refers to latency-critical online routing and retrieval on a local de-

[†]Corresponding author: Yuxiang Ren.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 8582–8598
October 24-29, 2026 ©2026 Association for Computational Linguistics

vice, while graph construction and classifier updates are performed asynchronously. Second, an explicit modeling of memory heterogeneity is required: heterogeneous memory types must not be collapsed into a uniform vector space, but should instead retain their distinct attributes and interrelations through structured representations to facilitate retrieval.

Presented work. To address these challenges, we propose HGP, a **H**ybrid **G**raph **P**ersonalized memory framework. Our main contributions are summarized as follows:

- **Framework design.** We develop a framework that moves beyond single-vector compression by organizing episodic, semantic, and procedural memories with type-specific graph structures, including *event graphs, knowledge graphs, and rule trees*. This is augmented by dynamic *working memory-state trajectory graphs* that capture and preserve implicit contextual constraints.

- **Self-Enhancement Classifier.** We propose a confidence-based LLM-assisted annotation policy for personalized memory routing. A *lightweight classifier* progressively takes over routing decisions, adapts to user-specific memory distributions, and substantially reduces supervision cost by recycling low-confidence LLM annotations as training data.

- **Empirical validation.** We evaluate HGP on LoCoMo (Maharana et al., 2024) and PAL-Set (Huang et al., 2025), demonstrating consistent improvements in personalization quality and decision reliability over strong baselines. We also provide a thorough analysis of how working memory improves robustness under implicit constraints.

2 Related Work

2.1 Long-context Modeling and Retrieval-Augmented Memory.

One research direction improves model capacity by extending context windows and developing memory-optimized attention mechanisms, thereby enabling training and inference over significantly longer sequences. Representative approaches include sparse-attention models such as Longformer (Beltagy et al., 2020) and BigBird (Zaheer et al., 2021), as well as I/O-efficient exact attention methods like FlashAttention (Dao et al., 2022). Key innovations in positional encoding and length extrapolation, such as RoPE, ALiBi, and their derivatives (Su et al., 2023; Press et al., 2022; Ding et al., 2024), have driven increases in the effective context window of LLM. Representing a complementary strategy to long-context modeling, retrieval augmentation enhances LLMs by coupling them with an external knowledge store, as exemplified by RAG (Lewis et al., 2021), REALM (Guu et al., 2020), and RETRO (Borgeaud et al., 2022). However, increased context capacity does not guarantee effective information utilization. This limitation is evidenced by findings such as "Lost in the Middle" (Liu et al., 2023), which demonstrate that models often access salient information unevenly within long contexts. More critically, even with unbounded context length, both long-context modeling and retrieval augmentation lack the requisite mechanisms for continuously capturing and structuring user-specific interaction patterns, which are essential for genuine personalization.

2.2 Graph-structured memory.

Recent memory frameworks are increasingly adopting graph-based representations, driven by the promise of explicit relational reasoning (Sun et al., 2025; Li et al., 2025a). Moving beyond flat similarity search over text chunks, graph-structured memory enables more structured storage and retrieval through explicit modeling of relations among memory units. GraphRAG-style systems (Han et al., 2025; Zhang et al., 2024b,a) index knowledge as graphs and retrieve query-relevant subgraphs to support multi-hop evidence aggregation. Representative approaches include graph-traversal-based retrieval, such as HippoRAG (Gutiérrez et al., 2025), and hierarchical indexing with structure-aware retrieval, exemplified by LeanRAG (Zhang et al., 2025b) and HiRAG (Jiao et al., 2025). Graph-structured memory has also been explored in agentic settings, including G-Memory (Zhang et al., 2025a), G-Refer (Li et al., 2025b), and the concurrent PlugMem (Yang et al., 2026), which organizes agent experiences into structured memory graphs for efficient retrieval. These works are closely related to HGP in moving beyond raw trajectory retrieval and flat memory storage. However, most of them mainly focus on structure-aware retrieval or task-agnostic agent memory, while HGP focuses on personalized agent memory where heterogeneous user traces must be routed into distinct memory types and stored with type-specific graph structures

for later personalized retrieval and decision making.

2.3 Agent memory system.

Memory is a core capability of LLM-based agents. Early systems mainly rely on passive retrieve-and-inject mechanisms such as retrieval-augmented generation (RAG) (Lewis et al., 2021), while agent frameworks like ReAct (Yao et al., 2023) and Toolformer (Schick et al., 2023) treat memory as an implicit or short-term component. To support personalization and long-term consistency, recent work increasingly incorporates cross-session memory. MemoryBank (Zhong et al., 2023) continuously updates persistent user profiles; MemGPT (Packer et al., 2024) introduces a memory hierarchy between the context window and external storage; Generative Agents (Park et al., 2023) and MemoryOS (Kang et al., 2025) further explore structured memory storage and lifecycle management. Typed memory has also been discussed in cognitive agent architectures. For example, CoALA (Sumers et al., 2024) formalizes language agents with multiple memory components, including working, episodic, semantic, and procedural memories. Production-oriented systems like Mem0 (Chhikara et al., 2025) and A-MEM (Xu et al., 2025) target scalable memory pipelines, while recent work ScrapMem (Chang and Ren, 2026) further considers storage-efficient personalized memory on edge devices. These works motivate persistent and cognitively inspired agent memory, whereas HGP studies how user traces can be routed into heterogeneous memory types and stored with type-specific graph structures for personalization. Our self-enhanced router is related to cost-aware model routing and deferral methods, such as FrugalGPT (Chen et al., 2023) and RouteLLM (Ong et al., 2025). Unlike these methods, which route inputs across models, HGP routes memory writes into different memory stores; only low-confidence traces invoke LLM annotation, which is then cached for incremental classifier updates.

3 Methodology

In this work, we propose HGP, a hybrid graph memory framework for user personalization. As illustrated in Figure 1, HGP comprises two core components: (1) a *self-enhancement classifier*; (2) a *hybrid graph-based storage*.

3.1 Personalized Memory Routing

In cross-user scenarios, individual differences in memory-type signals cause substantial distribution shifts, making a general classifier prone to misrouting. To address this, HGP employs a confidence-based, LLM-assisted annotation mechanism that continuously refines the router and amortizes supervision cost. For an input segment x, the router first predicts a label $\hat{c}_\theta(x)$ with confidence score $s(x)$. The final routing decision is governed by an iteration-adaptive threshold τ_t:

$$c(x) = \begin{cases} \hat{c}_{\text{LLM}}(x), & s(x) < \tau_t, \\ \hat{c}_\theta(x), & s(x) \geq \tau_t, \end{cases} \tag{1}$$

where $\hat{c}_{\text{LLM}}(x)$ denotes the label from the LLM-based router, and τ_t is the confidence threshold at iteration t. If $s(x) \geq \tau_t$, the predicted label $\hat{c}_\theta(x)$ is retained for downstream memory construction. Otherwise, the segment is deferred to the LLM for annotation, and the resulting labeled pair $(x, \hat{c}_{\text{LLM}}(x))$ is cached into an incremental training pool. Periodically, the classifier is updated on the accumulated pool. We initialize the router conservatively (LLM-dominant) and gradually shift the workload to the classifier via exponential annealing:

$$\tau_t = \max\left(\tau_{\min}, \tau_0 \cdot e^{-\beta t}\right). \tag{2}$$

where a high initial threshold τ_0 encourages deferral, a decay rate β enables a smooth transition, and a lower bound $\tau_{\min}$ prevents overly aggressive self-labeling. As the data pool grows with user-specific samples, the classifier's decision boundary progressively adapts to the current domain and user patterns, effectively mitigating cross-user distribution shifts. This continuous adaptation sustains long-term personalization, which is especially critical as user behaviors evolve and memory distributions drift over time.

3.2 Hybrid Graph Storage

To preserve heterogeneous memory types and their relations, HGP adopts a hybrid graph storage design. We detail each memory structure below.

3.2.1 Episodic Memory – Event Graph

Episodic memory stores situated interaction events experienced, preserving traceable trajectories, temporal order, and contextual dependencies. In HGP, episodic memory is represented as an *Event Graph*. Formally, the event graph is modeled as

$$\mathcal{G}_{\text{epi}} = (V_e \cup V_{\text{ent}}, E_{\text{epi}}), \tag{3}$$

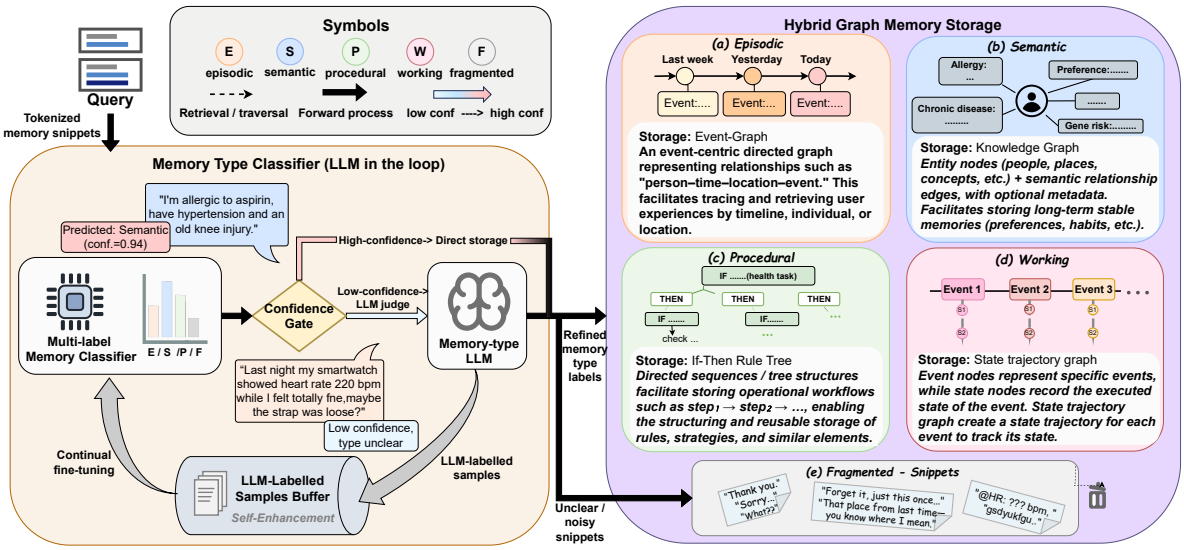

Figure 1: Overview of our HGP framework with two components: (1) a self-enhancement classifier that routes inputs via a classifier/LLM and incrementally updates the classifier using high-confidence samples, and (2) hybrid graph storage that preserves structural properties of heterogeneous memory types.

where V_e is the set of atomic episode nodes, V_{ent} denotes contextual entity nodes, and E_{epi} represents typed edges encoding event–event and event–entity relations.

In the event graph, episode nodes serve as structural anchors, connected to other episodes or contextual entities via semantically typed edges. These edges capture structured dependencies between events, such as temporal order, contextual association, and other semantic relations. Each edge is defined as

$$e_{ij} = (v_i, v_j, r), \quad r \in \{\textit{temporal}, \textit{contextual}, \dots\}, \tag{4}$$

where r specifies the semantic role of the relation between two nodes. Unlike isolated record storage, the event graph links events through shared entities, preserving structured relationships and contextual continuity.

As a core component of the hybrid graph storage, the event graph further supports structure-aware retrieval and long-term evolution. As event graphs accumulate, recurring relational patterns and substructures can be abstracted as:

$$\mathcal{K} = \mathcal{A}_{\text{abs}}(\mathcal{G}_{\text{epi}}^{(1)}, \dots, \mathcal{G}_{\text{epi}}^{(n)}), \tag{5}$$

where $\mathcal{A}\text{abs}(\cdot)$ denotes a structure abstraction operator instantiated via LLM. Through this process, episodic memory functions not only as a repository of past experiences but also as a structural foundation for deriving higher-level semantic and procedural knowledge.

3.2.2 Semantic Memory – Knowledge Graph

Semantic memory captures long-term, decontextualized knowledge such as facts, relations, and user preferences, supporting consistent personalized behavior. In HGP, it is explicitly represented as a *knowledge graph*:

$$\mathcal{G}_{\text{sem}} = (V_{\text{sem}}, E_{\text{sem}}), \tag{6}$$

where V_{sem} denotes semantic nodes encoding user-centric long-term knowledge, and E_{sem} represents typed semantic relations among them.

Each semantic relation is represented as a triple:

$$(u, r, v), \quad u \in V_{\text{user}},\ v \in V_{\text{ent}},\ r \in \mathcal{R}_{\text{sem}}, \tag{7}$$

where r is a predicate drawn from a predefined semantic relations (e.g., HAS_SKILL, PREFERS, WORKS_AS).

Unlike episodic memory, which captures temporally grounded experience fragments, semantic memory focuses on regularities that generalize across interaction contexts. It can be obtained either directly via the self-enhancement classifier or through semantic abstraction over accumulated episodic event graphs:

$$\mathcal{G}_{\text{sem}} = \mathcal{A}_{\text{sem}}(\mathcal{G}_{\text{epi}}^{(1)}, \dots, \mathcal{G}_{\text{epi}}^{(n)}). \tag{8}$$

The detailed design and prompt templates of $\mathcal{A}_{\text{sem}}$ are provided in Appendix D.1.

3.2.3 Procedural Memory – Rule Tree Graph

Procedural memory stores reusable, experience-driven behavioral strategies and decision flows, enhancing long-term planning capability. In HGP, procedural memory is structured as a *Rule Tree*, where nodes represent conditions, actions, or intermediate goals, and edges encode their dependencies. Formally, it is defined as

$$\mathcal{G}_{\text{proc}} = (V_c \cup V_a \cup V_g, E_{\text{proc}}), \tag{9}$$

where V_c, V_a, V_g denote condition, action, and subgoal nodes, respectively, and E_{proc} represents the set of edges encoding control or dependency relations among these nodes.

Procedural memory is sourced directly from the self-enhancement classifier or extracted from episodic event graphs. In the latter case, it is constructed via

$$\mathcal{G}_{\text{proc}} = \mathcal{A}_{\text{proc}}(\mathcal{G}_{\text{epi}}^{(1)}, \ldots, \mathcal{G}_{\text{epi}}^{(n)}), \tag{10}$$

where $\mathcal{A}_{\text{proc}}(\cdot)$ is a procedural-abstraction operator instantiated with an LLM. The detailed design and prompt template of $\mathcal{A}_{\text{proc}}$ are provided in Appendix D.2.

3.2.4 Working Memory – State Trajectory Graph

Working memory maintains the states of ongoing interactions or tasks, supplying state information for reasoning and decision-making. In HGP, it is implemented as a *State Trajectory Graph*, where nodes represent current system or user states and edges encode state transitions. Formally, working memory is defined as

$$\mathcal{G}_{\text{wm}} = (V_{\text{state}}, E_{\text{state}}), \tag{11}$$

where V_{state} represents the current states and E_{state} denotes transitions between states.

Unlike episodic and semantic memory, which retain long-term information, working memory maintains short-lived, dynamic context extracted from recent episodic memory and continuously updated during interaction. Its update process can be expressed as

$$\mathcal{G}_{\text{wm}}^{(t+1)} = \mathcal{U}(\mathcal{G}_{\text{wm}}^{(t)}, \mathcal{G}_{\text{epi}}^{(t)}), \tag{12}$$

where $\mathcal{U}(\cdot)$ denotes a state update operator. Graph-based modeling of working memory provides an explicit representation of the current state and its evolution, enabling timely reasoning, strategy selection, and action triggering.

3.2.5 Query Retrieval Workflow

Upon receiving a user query q, HGP executes a structured retrieval protocol over its hybrid graph memory $\mathcal{G} = \{\mathcal{G}_{\text{epi}}, \mathcal{G}_{\text{sem}}, \mathcal{G}_{\text{proc}}, \mathcal{G}_{\text{wm}}\}$. A detailed example of this retrieval workflow is provided in Appendix B. For the query q, HGP issues parallel queries to each subgraph, each designed to retrieve a particular type of structured evidence:

- **Episodic graph** $\mathcal{G}_{\text{epi}}$: Retrieves a set of temporally or contextually related past interaction nodes. For instance, in a health management scenario, a query about "evening exercise planning" may retrieve nodes representing *"night run with elevated heart rate"* and *"previous over-exertion alert"*, connected via shared activity and time entities.
- **Semantic graph** $\mathcal{G}_{\text{sem}}$: Retrieves a subgraph of factual triples about the user that are semantically relevant to q. Examples include *(User, allergy, Ibuprofen)* and *(User, condition, Hypertension)*, providing stable profile constraints.
- **Procedural graph** $\mathcal{G}_{\text{proc}}$: Retrieves applicable rule nodes that match the current context. For example, the rule *"If NSAID combines with aspirin → check bleeding risk"* is retrieved when the query involves taking painkillers after aspirin intake.
- **Working memory graph** $\mathcal{G}_{\text{wm}}$: Retrieves the current state trajectory, capturing immediate session constraints such as *"aspirin already taken this morning"* and *"pending dinner recommendation task"*.

The retrieved subgraphs are merged into a unified context and passed, together with the original query q, to the LLM. The model then generates a response grounded in both long-term memory and the user's current state and implicit constraints, ensuring coherence, personalization, and safety.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on two long-term memory benchmarks: LoCoMo (Maharana et al., 2024) provides 10 multi-session conversations with ∼200 QA pairs each, covering single-hop, multi-hop, temporal, and open-domain questions. Following Mem0 (Chhikara et al., 2025), we exclude the adversarial category due to missing ground-truth answers. PAL-Set (Huang et al., 2025) evaluates memory-aware personalization via user logs and

Table 1: Unified evaluation results on LoCoMo (%). Evaluation metrics are reported for each category and overall, including F1 score (F1), BLEU-1 (B1), Exact Match score (EM) and LLM-as-a-Judge score (J), with higher values indicating better performance. Best results are in **bold**, and second-best results are underlined.

Method	Single-hop				Multi-hop				Temporal				Open-domain				Overall			
	J	F1	B1	EM	J	F1	B1	EM	J	F1	B1	EM	J	F1	B1	EM	J	F1	B1	EM
TurnRAG	47.52	34.31	28.43	6.03	38.54	20.33	16.60	6.25	60.44	51.71	**47.86**	**19.94**	65.28	48.34	43.76	24.97	59.35	44.73	40.11	19.29
LangMem	53.55	32.78	26.35	3.90	41.67	22.97	19.99	10.42	17.76	10.23	8.67	3.74	54.22	34.30	30.64	14.10	45.65	24.74	21.49	8.46
MemoryBank	47.87	19.26	24.29	3.55	37.50	10.89	15.00	7.29	26.17	6.63	7.51	4.05	**72.65**	40.97	44.37	26.27	56.12	39.83	35.03	16.22
A-Mem	51.42	20.43	25.29	4.96	44.79	13.60	17.00	7.29	59.19	35.21	38.63	19.31	67.59	38.42	41.09	24.82	61.41	32.87	36.15	18.90
HippoRAG2	52.84	21.16	27.69	4.61	40.62	13.25	17.08	8.33	**60.75**	38.00	41.75	**19.94**	68.31	39.75	42.84	26.39	62.13	34.29	38.20	19.88
Mem0	52.13	**35.18**	26.15	**5.67**	38.54	**28.95**	22.85	**13.54**	59.81	**52.71**	39.70	19.31	66.39	47.98	40.81	26.14	60.63	**45.42**	36.75	20.14
Ours	**56.03**	34.44	**30.39**	**5.67**	**46.88**	27.69	**23.70**	12.50	**60.75**	50.31	46.24	19.63	71.45	**51.33**	**46.59**	**26.87**	**65.08**	45.07	**42.09**	**20.54**

interaction histories. We focus on the *Solution Proposal* tasks: (1)QA: memory-grounded answer generation; (2)Selection: selecting the best response from candidates based on retrieved memories.

Metrics. On LoCoMo, we report F1, Exact Match (EM), BLEU-1 (B1), and supplement with Mem0-style LLM-as-a-judge(J) to overcome the limits of lexical overlap in long-context reasoning. On PAL-Set, for the *QA* subtask, we use BLEU-1/2/3/4 as automatic metrics and report an LLM-as-a-judge score as a complementary assessment of answer correctness and usefulness. For comparability, the same judging protocol is applied to both memory-based and non-memory systems. For *Selection*, we use the benchmark-defined S-score, which measures how well the selected solution matches the user's needs. The LLM-as-a-judge score is treated as supplementary evidence rather than the sole evaluation criterion; our conclusions are based jointly on automatic QA metrics, the PAL-Set S-score, ablation results, and long-horizon memory evaluation. Full QA judge prompts are provided in Appendix A.1.

Baselines. We compare against two paradigms:

(1) **Non-memory methods.** On LoCoMo, we use a standard RAG pipeline. On PAL-Set, following the official benchmark, we include four retrieval-based variants with different context granularities: *Vanilla (w/o log)* (no history), *Vanilla (with log)* (logs concatenated into the prompt), *Turn-level RAG*, and *Session-level RAG*.

(2) **Memory-based frameworks.** We compare with representative memory-augmented systems: Mem0 (Chhikara et al., 2025), MemoryBank (Zhong et al., 2023), A-Mem (Xu et al., 2025), HippoRAG2 (Gutiérrez et al., 2025), and additionally LangMem on LoCoMo.

Implementation Details. In HGP, memory routing is performed by a lightweight BERT-based classifier (`roberta_L-4_H-512`). The confidence threshold is initialized at $\tau_0 = 0.90$, annealed with rate $\beta = 0.05$, and lower-bounded by $\tau_{\min} = 0.50$. Since working-memory annotations are not available in LoCoMo or PAL-Set, our experiments focus on episodic, semantic, and procedural memories, stored as Neo4j graph databases. Memory extraction uses `deepseek-chat`; response generation and LLM-as-a-judge use `gpt-4o-mini` (temperature 0.0). On LoCoMo, max output length is 100 tokens; dialogue history for TurnRAG is truncated to 4K tokens; retrieval-based methods use $K = 30$ items. All methods share the same answer/judge prompts adapted from Mem0 for fairness. On PAL-Set, we follow the official setup with $K = 3$ and introduce an additional QA-specific judge prompt for semantic correctness beyond lexical overlap.

4.2 Main Results

LoCoMo performance. Experiments on the LoCoMo dataset demonstrate the strong memory recall capability of HGP. As shown in Table 1, HGP achieves the best overall performance in Judge (65.08), BLEU-1 (42.09), and EM (20.54), while maintaining competitive F1 (45.07)—outperforming Mem0 by +4.45 (Judge) and +5.34 (BLEU-1). Gains are most pronounced on Open-domain tasks, where HGP attains the highest F1 (51.33), BLEU-1 (46.59), and EM (26.87). This improvement supports the effectiveness of graph-based memory storage: structuring episodic events, semantic profiles, and procedural rules as interconnected graphs provides more reliable evidence for diverse, unconstrained questions. Temporal questions remain comparatively challenging, suggesting opportunities for enhancing temporal ordering.

PAL-Set Solution Proposal tasks. Table 2 reports

Table 2: Performance comparison across different methods on PAL-Set Solution QA tasks. B-1 to B-4 represent BLEU scores. Δ (vs. Vanilla w/o log) reports the absolute improvement in S-score compared to the Vanilla (w/o log) baseline. LLM-judge denotes the score evaluated by GPT-4o-mini.

Group	Method	B-1	B-2	B-3	B-4	QA Judge	S-score	Δ (vs. Vanilla w/o log)
Non-memory	Vanilla (w/o log)	19.68	7.72	3.80	2.11	7.31	20.34	+0.00
	Vanilla (with log)	18.96	7.51	3.82	2.22	7.32	25.00	+4.66
	Turn-level RAG	19.48	7.72	3.93	2.23	7.41	25.06	+4.72
	Session-level RAG	19.17	7.55	3.82	2.22	7.30	25.24	+4.90
Memory-based	MemoryBank	**20.83**	7.36	4.23	2.56	7.13	28.85	+8.51
	Mem0	19.20	7.66	3.82	2.12	7.34	26.50	+6.16
	A-Mem	19.33	8.12	3.96	2.17	7.04	27.27	+6.93
	HippoRAG2	18.84	8.06	4.30	2.34	7.31	29.22	+8.88
	HGP (Ours)	19.62	**8.67**	**4.70**	**2.78**	**7.46**	**35.58**	**+15.24**

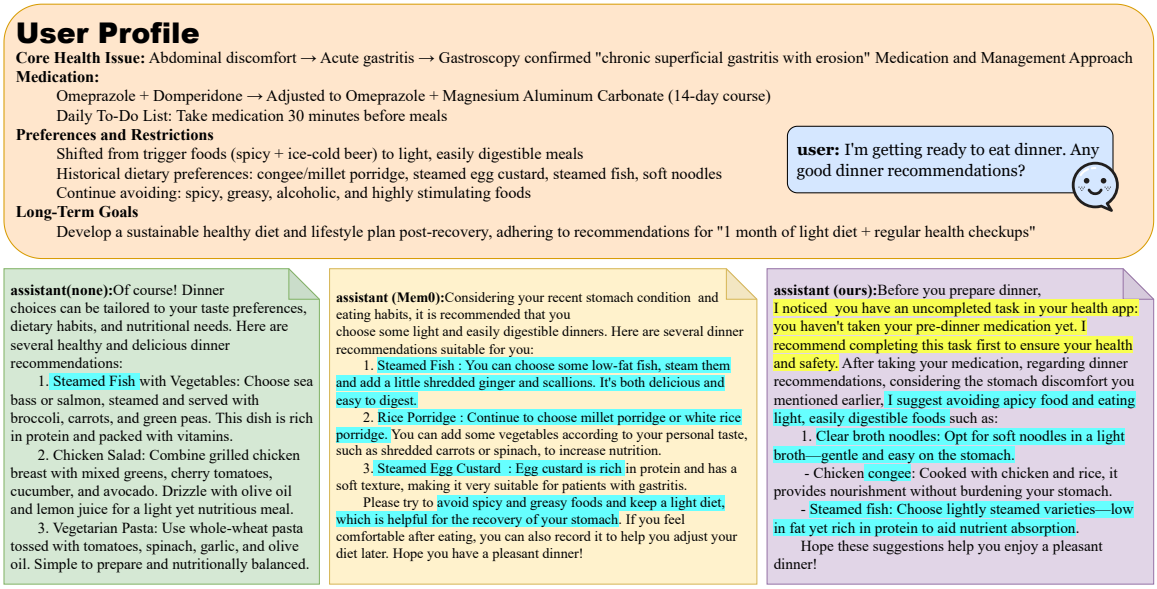

Figure 2: Working Memory Analysis: working-memory state tracking in a personal healthcare scenario. Yellow highlights denote pending to-do reminders, and blue highlights indicate preference-aligned personalized suggestions. Compared with baselines, our method surfaces the working-memory state (pre-meal medication: pending) and reminds the user while providing personalized dinner recommendations.

results on Solution QA and Selection. On QA, HGP achieves the highest LLM-judge score (7.46) and the best BLEU-2/3/4 (8.67/4.70/2.78), indicating stronger multi-step coherence. On Selection, non-memory baselines improve only marginally over Vanilla (w/o log) (+4.66 to +4.90), confirming that raw log retrieval is insufficient. HGP attains an S-score of 35.58, surpassing Vanilla (w/o log) (20.34) by +15.24 and the top memory baseline HippoRAG2 (29.22) by +6.36. These results show that graph-structured memory enables more precise personalized retrieval, better aligning responses with user-specific needs.

Table 3: Ablation study on memory components and representation designs. The table reports generation quality (BLEU and LLM-judge) and selection performance (S-score). Higher is better.

Method	B-1	B-2	B-3	B-4	LLM-judge	S-score
Unified-vector	18.15	7.24	3.60	2.05	7.18	25.15
Typed-vector	18.61	7.51	3.74	2.09	7.29	33.66
w/o *E-Mem*	19.53	7.96	3.96	2.23	7.09	28.83
w/o *S-Mem*	**19.69**	8.30	4.38	2.57	7.23	25.71
w/o *P-Mem*	19.59	8.63	4.66	2.76	7.31	33.10
Full (Ours)	19.62	**8.67**	**4.70**	**2.78**	**7.46**	**35.58**

4.3 Ablation Study

To quantify the contribution of each memory component, We ablate each memory type while keep-

ing the rest of the pipeline unchanged (Table 3).

Graph and type structure. Unified-vector collapses all memories into a single vector space, while Typed-vector preserves memory types but removes graph structure. Typed-vector substantially improves over Unified-vector on *Selection* (33.66 vs. 25.15), showing that memory typing is important for personalized decision making. Full HGP further improves the S-score to 35.58, indicating that type-specific graph storage provides additional gains beyond typed vector retrieval.

Episodic Memory(E-Mem). Removing episodic memory causes the largest drop on *Solution QA*: BLEU-1/2/3/4 fall from 19.66/8.67/4.70/2.78 to 19.53/7.96/3.96/2.23, and the LLM-judge score declines from 7.46 to 7.09. On *Selection*, the S-score drops from 35.58 to 28.83 ($\Delta = -6.75$). These results confirm that episodic memory, as the foundational trace of user interactions, is essential for both precise answer generation and context-aware solution matching.

Semantic Memory(S-Mem). Removing semantic memory causes the largest drop on *Selection* ($\Delta = -9.87$,S-score; Table 3), confirming that stable user profiles (long-term preferences, attributes, health facts) are essential for personalized decision-making. On *Solution QA*, the LLM-judge score also declines (7.46 to 7.23), indicating its contribution to coherent memory recall beyond pure retrieval.

Procedural Memory(P-Mem). Removing procedural memory causes a modest but non-negligible drop in *Selection* ($\Delta = -2.48$), while the impact on *Solution QA* is minor (judge score $7.46 \rightarrow 7.31$, BLEU-4 $2.78 \rightarrow 2.76$). This aligns with its role as high-level strategic guidance rather than direct factual evidence.

Overall, Table 3 highlights complementary roles: semantic graphs drive personalized selection, episodic graphs supply fine-grained event evidence for QA, and procedural graphs enhance strategy reuse. Integrating all components yields the best performance, validating our hybrid graph memory design.

4.4 Working Memory Analysis

Figure 2 illustrates the role of *working memory* in HGP using a healthcare scenario. Working memory explicitly extracts and maintains the current actionable state, enabling the agent to prioritize immediate actions during response generation.

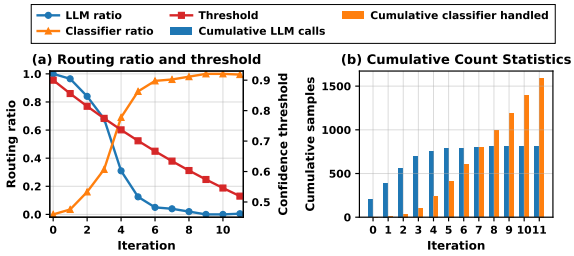

Figure 3: Classifier routing statistics across iterations. (a) Routing mix and adaptive threshold. (b) Cumulative workload handled by the LLM and the classifier.

Setup. We compare three systems: (i) a vanilla LLM without memory, (ii) an LLM augmented with Mem0, and (iii) HGP with working memory enabled. The dialog history contains a pre-meal medication reminder. The user query asks for dinner recommendations without explicitly mentioning medication.

Observation. The vanilla LLM directly suggests dinner options. Mem0 generates preference-aligned recommendations but fails to remind the user about the medication. In contrast, HGP retrieves the state "pre-meal medication: unfinished" from the *working memory graph* and injects it into the prompt, enabling the LLM to proactively issue a medication reminder before dinner and demonstrating state awareness under implicit constraints. (Further details in Appendix C.)

Table 4 compares our self-enhancement classifier with an offline classifier, a fixed LLM router, and a zero-shot LLM router. Our classifier achieves the highest macro-F1 (0.9339) and micro-F1 (0.9105) with an average latency of 4.98 ms, roughly 360× faster than the fixed LLM router. Fig. 3 shows the training dynamics: the confidence-based gating progressively shifts routing decisions from the LLM to the classifier, and incremental training every 200 samples drives the system from LLM-dominant to classifier-dominant (Fig. 3a). Overall, LLM calls drop from 2400 to 807 (a 66.4% reduction), confirming the workload transfer visible in the cumulative curves (Fig. 3b). These results demonstrate that the classifier effectively captures LLM routing knowledge while substantially lowering inference costs for on-device deployment. Hyperparameter sensitivity and per-stage performance are detailed in Appendix A.2.

Table 4: Performance comparison of the self-enhancement classifier against other baselines. Best results are in **bold**, and second-best results are underlined.

Router	Micro-F1	Macro-F1	Avg Latency (ms)
Offline Classifier	0.8638	0.8790	**4.06**
Fixed LLM Router	0.8953	0.9224	1803.69
Zero-shot LLM Router	0.8286	0.8436	1572.64
Ours	**0.9105**	**0.9339**	4.98

5 Conclusion

In this paper, we address the key challenge of achieving robust and adaptive personalization in LLM-based agents under cross-user heterogeneity and distribution shift. To this end, we propose HGP, a memory-augmented framework that integrates a self-enhancement routing mechanism with a hybrid graph-based memory storage, enabling cost-efficient and structured personalization. Extensive experiments on LoCoMo and PAL-Set validate the effectiveness of HGP in improving memory-recall accuracy and personalization capability, while routing statistics confirm its ability to amortize supervision cost over time. Future work will explore finer-grained modeling of memory graphs, incorporate cross-memory consistency constraints, and extend the framework to a broader range of interactive tasks and tool-augmented agents.

6 Limitations

While HGP demonstrates improvements in personalized memory management, several limitations remain. First, although working memory shows promising qualitative results, existing benchmarks lack explicit working-memory annotations, preventing systematic quantitative evaluation. Constructing datasets with ground-truth state trajectories is an important direction for rigorous assessment. Second, graph construction and maintenance introduce additional offline computation and storage costs, while our on-device claim primarily concerns latency-critical online routing and retrieval. A more complete end-to-end cost analysis remains for future work. In addition, this work focuses mainly on graph storage, while finer-grained retrieval and pruning mechanisms require further study. Finally, we use LLM-as-a-judge evaluation to complement lexical-overlap metrics. Although standardized prompts and temperature 0.0 improve evaluation consistency, LLM judges may still exhibit biases, and their reliability warrants continued attention.

Acknowledgments

This work was supported by the Nanjing University-China Mobile Joint Research Institute.

References

Richard C Atkinson and Richard M Shiffrin. 1968. Human memory: A proposed system and its control processes. In Kenneth W Spence and Janet Taylor Spence, editors, *The Psychology of Learning and Motivation: Advances in Research and Theory*, volume 2, pages 89–195. Academic Press, New York.

Alan D Baddeley and Graham J Hitch. 1974. Working memory. In Gordon H Bower, editor, *The Psychology of Learning and Motivation: Advances in Research and Theory*, volume 8, pages 47–89. Academic Press, New York.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *Preprint*, arXiv:2004.05150.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, and 9 others. 2022. Improving language models by retrieving from trillions of tokens. *Preprint*, arXiv:2112.04426.

Jiale Chang and Yuxiang Ren. 2026. Scrapmem: A bio-inspired framework for on-device personalized agent memory via optical forgetting. *Preprint*, arXiv:2605.03804.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *Preprint*, arXiv:2305.05176.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Preprint*, arXiv:2205.14135.

Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. Longrope: Extending llm context window beyond 2 million tokens. *Preprint*, arXiv:2402.13753.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. *Preprint*, arXiv:2405.06211.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2025. Hipporag: Neurobiologically inspired long-term memory for large language models. *Preprint*, arXiv:2405.14831.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. *Preprint*, arXiv:2002.08909.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, Qi He, Zhigang Hua, Bo Long, Tong Zhao, Neil Shah, Amin Javari, Yinglong Xia, and Jiliang Tang. 2025. Retrieval-augmented generation with graphs (graphrag). *Preprint*, arXiv:2501.00309.

Zhaopei Huang, Qifeng Dai, Guozheng Wu, Xiaopeng Wu, Kehan Chen, Chuan Yu, Xubin Li, Tiezheng Ge, Wenxuan Wang, and Qin Jin. 2025. Mem-pal: Towards memory-based personalized dialogue assistants for long-term user-agent interaction. *Preprint*, arXiv:2511.13410.

YiHan Jiao, ZheHao Tan, Dan Yang, DuoLin Sun, Jie Feng, Yue Shen, Jian Wang, and Peng Wei. 2025. Hirag: Hierarchical-thought instruction-tuning retrieval-augmented generation. *Preprint*, arXiv:2507.05714.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. *Preprint*, arXiv:2506.06326.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Yuhan Li, Yuyao Wang, Jianheng Tang, Heng Chang, Yuxiang Ren, and Jia Li. 2025a. Advancing graph foundation models: A data-centric perspective. KDD '25, page 1635–1646, New York, NY, USA. Association for Computing Machinery.

Yuhan Li, Xinni Zhang, Linhao Luo, Heng Chang, Yuxiang Ren, Irwin King, and Jia Li. 2025b. G-refer: Graph retrieval-augmented large language model for explainable recommendation. *Preprint*, arXiv:2502.12586.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *Preprint*, arXiv:2307.03172.

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

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. 2025. Routellm: Learning to route llms with preference data. *Preprint*, arXiv:2406.18665.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Preprint*, arXiv:2304.03442.

Ofir Press, Noah A. Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. *Preprint*, arXiv:2108.12409.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Preprint*, arXiv:2302.04761.

Larry R Squire. 1992. Declarative and nondeclarative memory: Multiple brain systems supporting learning and memory. *Journal of Cognitive Neuroscience*, 4(3):232–243.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding. *Preprint*, arXiv:2104.09864.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive architectures for language agents. *Preprint*, arXiv:2309.02427.

Shengyin Sun, Yuxiang Ren, Jiehao Chen, and Chen Ma. 2025. Large language models as topological structure enhancers for text-attributed graphs. *Preprint*, arXiv:2311.14324.

Endel Tulving. 1972. Episodic and semantic memory. In Endel Tulving and Wayne Donaldson, editors, *Organization of Memory*, pages 381–403. Academic Press, New York.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *Preprint*, arXiv:2502.12110.

Ke Yang, Zixi Chen, Xuan He, Jize Jiang, Michel Galley, Chenglong Wang, Jianfeng Gao, Jiawei Han, and ChengXiang Zhai. 2026. Plugmem: A task-agnostic plugin memory module for llm agents. *Preprint*, arXiv:2603.03296.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2021. Big bird: Transformers for longer sequences. *Preprint*, arXiv:2007.14062.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. *Preprint*, arXiv:2506.07398.

Wanhao Zhang, Qianli Zhang, Enyu Yu, Yuxiang Ren, Yeqing Meng, Mingxi Qiu, and Jilong Wang. 2024a. Leveraging rag-enhanced large language model for semi-supervised log anomaly detection. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pages 168–179.

Wanhao Zhang, Qianli Zhang, Enyu Yu, Yuxiang Ren, Yeqing Meng, Mingxi Qiu, and Jilong Wang. 2024b. Lograg: Semi-supervised log-based anomaly detection with retrieval-augmented generation. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 1100–1102.

Yaoze Zhang, Rong Wu, Pinlong Cai, Xiaoman Wang, Guohang Yan, Song Mao, Ding Wang, and Botian Shi. 2025b. Leanrag: Knowledge-graph-based generation with semantic aggregation and hierarchical retrieval. *Preprint*, arXiv:2508.10391.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *Preprint*, arXiv:2402.19473.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. Memorybank: Enhancing large language models with long-term memory. *Preprint*, arXiv:2305.10250.

A More Experiment Details

A.1 Prompts for PAL-Set Solution QA

We provide the full LLM-as-a-Judge prompt template for the PAL-Set Solution QA task in Figure 4. This template is used to compute the LLM-judge scores reported in Table 2. During evaluation, we supply the judge LLM with the model-generated solution, the task question, and relevant conversation context (when available), along with the corresponding positive and negative reference solutions from the dataset. The judge is instructed to score outputs across multiple dimensions: requirement satisfaction, semantic correctness, step completeness, and user-level personalization. Positive and negative references are provided to calibrate judgments and penalize common error patterns. For ease of aggregation, the judge is constrained to produce strictly formatted JSON outputs, with temperature fixed at 0.0.

A.2 Supplementary Experiments on the Self-Enhancement Classifier

A.2.1 Hyperparameter Sensitivity Analysis

To evaluate the robustness of our self-enhancement classifier under different hyperparameter settings, we conducted a full-factor grid search over the three key parameters: initial threshold $\tau_0 \in \{0.8, 0.9, 0.95\}$, decay rate $\beta \in \{0.02, 0.05, 0.1\}$, and lower bound $\tau_{\min} \in \{0.3, 0.5, 0.7\}$. This resulted in 27 configurations. For each configuration, we measured the macro F1 score (with standard deviation) and the average inference latency on the same validation set. To comprehensively evaluate each hyperparameter configuration, we define a combined score that balances accuracy, stability, and efficiency. Let $\mu_{\text{lat}}(c)$ denote the average inference latency of configuration c, and let $\mu_{\text{lat}}^{\min}$ be the minimum average latency among all candidate configurations:

$$\mu_{\text{lat}}^{\min} = \min_{c} \mu_{\text{lat}}(c). \tag{13}$$

The normalized latency is then defined as:

$$\text{LatencyNorm}(c) = \frac{\mu_{\text{lat}}(c)}{\mu_{\text{lat}}^{\min}}. \tag{14}$$

With penalty coefficients $\lambda_s = 0.5$ and $\lambda_l = 0.5$, the overall score for configuration c is:

$$\begin{aligned}\text{Score}(c) = \mu_{\text{F1}}(c) - \lambda_s \cdot \sigma_{\text{F1}}(c) - \\ \lambda_l \cdot \max\big(0,\ \text{LatencyNorm}(c)\big)\end{aligned} \tag{15}$$

where $\mu_{\text{F1}}(c)$ and $\sigma_{\text{F1}}(c)$ are the mean and standard deviation of the macro F1 score, respectively. This score penalizes both performance variability (σ_{F1}) and any latency exceeding the optimal value ($\text{LatencyNorm}(c) - 1$), thereby encouraging configurations that are accurate, stable, and efficient.

As shown in the table 5, the macro F1 values across all 27 configurations vary within a very small range, demonstrating that the classifier is remarkably insensitive to the choice of these hyperparameters. The best macro F1 (0.9528) is achieved by configuration #21 ($\tau_0 = 0.95, \beta = 0.02, \tau_{\min} = 0.3$), but its latency (5.93 ms) is significantly higher than the optimum. The configuration with the lowest latency (4.23 ms) is #1 ($\tau_0 = 0.9, \beta = 0.02, \tau_{\min} = 0.5$), which also attains a macro F1 of 0.9490.

Our default configuration #5 ($\tau_0 = 0.9, \beta = 0.05, \tau_{\min} = 0.5$) yields a macro F1 of 0.9474 and a latency of 4.44 ms, which is only marginally

PROMPT TEMPLATE FOR SOLUTION QA RESULTS GENERATIOM (PAL-SET)

You are a strict, impartial, and reproducible Solution QA evaluator (LLM Judge).
Your task is to evaluate only the relevance and fit of the generated solution to the complete user requirement.
Assign an integer score from 1 to 10.

Evaluate only the final solution text. Do not evaluate the method that produced it, the retrieval process, the memory system, or the storage structure. Logs and Memories are provided only as user context for checking whether the solution satisfies the user's needs, preferences, and constraints.

#Evaluation objectives:
1) Primary: Does the solution accurately satisfy the requirement, including explicit needs, implicit constraints, and current context?
2) Secondary: Is the solution consistent with the provided user context, including relevant preferences, background information, historical constraints, and prohibitions?
3) Tertiary: Does the solution avoid error patterns in the negative references, such as being off-topic, misleading, violating preferences, or ignoring key constraints?

#Important rules:
Do not give extra credit merely because the solution explicitly mentions memory, history, logs, or retrieved memories.
Do not evaluate whether the solution uses any specific retrieval or memory process.
If Logs/Memories are empty or insufficient, do not penalize solely for the lack of memory information. Judge based on the requirement, available context, and final solution.
If Logs/Memories contain relevant user preferences or constraints and the solution clearly ignores, conflicts with, or fabricates them, deduct points.
If the solution does not explicitly cite Logs/Memories but correctly satisfies the relevant preferences and constraints, do not penalize it.

#Scoring anchors:
9-10: Highly satisfies the requirement; covers key points completely; is consistent with relevant user context; no major issues.
7-8: Generally satisfies the requirement; mostly consistent with relevant context and preferences; minor omissions only.
5-6: Partially satisfies the requirement; has noticeable omissions, generic content, or insufficient use of relevant context; still somewhat usable.
3-4: Largely fails or deviates from the topic; ignores key context, preferences, or constraints; may be misleading.
1-2: Severely off-topic, obviously wrong, strongly matches negative error patterns, or may cause clear negative impact.

Please evaluate the relevance/fit of the following AI-generated solution (1-10 points).
[User Requirement]
{requirement}
[Current Conversation Context Logs]
{logs}
[Provided User Context / Memories]
{memories}
[Generated Solution]
{solution}
[Positive Reference: high-scoring direction/key points]
{references_pos}
[Negative Reference: error patterns to avoid]
{references_neg}

Return strict JSON only:
{{"relevance": <integer from 1 to 10>, "explanation": "<one or two concise sentences>"}}
The first character must be "{{" and the last character must be "}}".

Figure 4: LLM-as-a-Judge prompt used for PAL-Set Solution QA evaluation, enforcing requirement-fit scoring with implicit constraints, retrieved memories/logs, and pos/neg reference calibration.

worse than the best trade-off. The differences from the top-scoring configuration (#1) are minimal: macro-F1 differs by only 0.0016 (0.9474 vs. 0.9490) and latency by 0.21 ms (4.44 vs.4.23). Both configurations lie on the same near-Pareto frontier. We choose #5 over #1 for the following reasons:

- **Adaptive threshold updates**: The moderate decay rate ($\beta = 0.05$) triggers more effective threshold updates during online self-enhancement, whereas a smaller decay ($\beta = 0.02$) would keep the threshold nearly unchanged, leading to under-adaptation.
- **Consistency with our experimental protocol**: This setting avoids post-hoc overfitting to a single best point and maintains consistency with the rest of our experiments.
- **Robustness**: The performance gap between #5 and the best configuration is negligible, and our main conclusions remain unchanged regardless of which near-optimal configuration is chosen.

Therefore, we adopt $\tau_0 = 0.9, \beta = 0.05, \tau_{\min} = 0.5$ as the default setting for all main experiments. The full ranking of 27 configurations is provided to demonstrate that our results are insensitive to the specific choice among near-optimal settings.

Table 5: Hyperparameter sensitivity analysis of the router with different values of τ_0, β, and $\tau_{\min}$. The configuration selected for our experiments is highlighted in bold.

Rank	τ_0	β	$\tau_{\min}$	F1 (mean±std)	Latency (ms)	Norm Latency	Score
1	0.9	0.02	0.5	0.9490 ± 0.0034	4.23	1.000	0.947
2	0.9	0.02	0.3	0.9490 ± 0.0034	4.36	1.031	0.932
3	0.9	0.02	0.7	0.9490 ± 0.0034	4.40	1.040	0.927
4	0.8	0.1	0.7	0.9482 ± 0.0023	4.46	1.055	0.919
5	**0.9**	**0.05**	**0.5**	0.9474 ± 0.0060	**4.44**	**1.050**	**0.919**
6	0.8	0.02	0.5	0.9482 ± 0.0023	4.72	1.118	0.888
7	0.9	0.05	0.3	0.9474 ± 0.0060	4.83	1.143	0.873
8	0.95	0.1	0.5	0.9526 ± 0.0001	4.96	1.174	0.866
9	0.9	0.1	0.5	0.9474 ± 0.0060	5.06	1.198	0.846
10	0.8	0.05	0.3	0.9482 ± 0.0023	5.10	1.207	0.843
11	0.95	0.05	0.5	0.9522 ± 0.0049	5.13	1.213	0.843
12	0.8	0.05	0.5	0.9482 ± 0.0023	5.22	1.235	0.830
13	0.9	0.1	0.3	0.9474 ± 0.0060	5.26	1.244	0.822
14	0.95	0.1	0.7	0.9526 ± 0.0001	5.42	1.281	0.812
15	0.9	0.05	0.7	0.9474 ± 0.0060	5.40	1.277	0.806
16	0.8	0.05	0.7	0.9482 ± 0.0023	5.47	1.295	0.800
17	0.95	0.02	0.7	0.9509 ± 0.0030	5.52	1.307	0.796
18	0.95	0.05	0.7	0.9522 ± 0.0049	5.64	1.335	0.782
19	0.8	0.02	0.7	0.9482 ± 0.0023	5.63	1.332	0.781
20	0.8	0.1	0.5	0.9482 ± 0.0023	5.80	1.371	0.761
21	0.95	0.02	0.3	0.9528 ± 0.0009	5.93	1.402	0.751
22	0.95	0.02	0.5	0.9509 ± 0.0030	5.94	1.405	0.747
23	0.95	0.1	0.3	0.9526 ± 0.0001	6.05	1.431	0.737
24	0.8	0.02	0.3	0.9482 ± 0.0023	6.01	1.421	0.737
25	0.9	0.1	0.7	0.9474 ± 0.0060	6.05	1.431	0.729
26	0.8	0.1	0.3	0.9482 ± 0.0023	6.24	1.477	0.709
27	0.95	0.05	0.3	0.9522 ± 0.0049	6.44	1.525	0.687

A.2.2 Stage-wise Performance of the Self-Enhancement Classifier

Table 6: Performance evolution of the self-enhancement classifier across incremental training stages.

Stage	**Macro-F1**	**Micro-F1**	**ΔMacro (vs. S0)**
S0 (offline init)	0.8980	0.8859	0.0000
S1	0.9162	0.9068	+0.0182
S2	0.9282	0.9192	+0.0302
S3	0.9445	0.9275	+0.0465
S4	0.9527	0.9379	+0.0547
S5	0.9554	0.9416	+0.0574

To investigate how the self-enhancement mechanism improves classifier performance over time, we record its performance after each incremental training round. The initial stage (S0) uses an offline-trained classifier without any LLM supervision. Subsequently, every 200 newly LLM-labeled samples trigger one round of incremental fine-tuning, resulting in five rounds (S1 to S5). As shown in Table 6, the macro F1 steadily increases from 0.8980 (S0) to 0.9554 (S5), an absolute gain of 5.74 percentage points. The micro F1 follows a similar upward trend. The most significant improvements occur in the first two rounds (S0→S1: +1.82 points; S1→ S2: +1.20 points), after which the gains gradually diminish, indicating convergence. These results demonstrate that the self-enhancement mechanism effectively leverages LLM-labeled hard samples to continuously improve routing accuracy without causing performance oscillation or catastrophic forgetting.

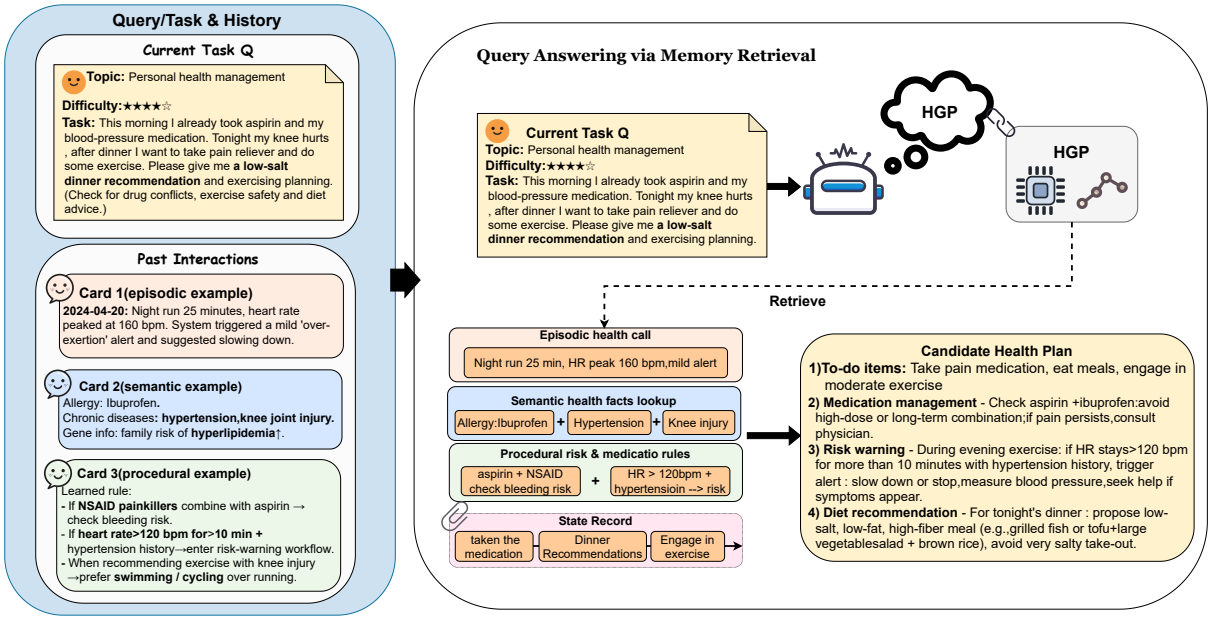

Figure 5: An illustrative example of HGP in a personalized health management scenario. Given a user's current query and long-term interaction history, HGP organizes heterogeneous memories, including episodic, semantic, procedural, and working memory, within a hybrid graph storage, and retrieves task-relevant evidence to generate personalized health recommendations.

It is important to emphasize the training protocol for this stage-wise evaluation. The classifier first makes predictions on the training set. Samples with low confidence are sent to an LLM for re-labeling, and the (sample, label) pairs are cached. Once a sufficient number of such samples is accumulated, incremental training is performed, and the cycle repeats. The training and test sets are strictly separated. The test set is never used for training, nor does it ever involve LLM inference. For rigorous evaluation, the test set is further partitioned into multiple disjoint subsets, each used exactly once for evaluating the classifier after one incremental training round. This prevents any data leakage across evaluation steps.

B Retrieval Workflow Example

As shown in Figure 5, upon receiving a user query q , HGP executes a structured retrieval protocol over its hybrid graph memory $\mathcal{G} = \{\mathcal{G}_{\text{epi}}, \mathcal{G}_{\text{sem}}, \mathcal{G}_{\text{proc}}, \mathcal{G}_{\text{wm}}\}$. Each sub-graph is queried for a specific type of evidence. The retrieved sub-graphs are merged into a unified context and passed, together with q, to the LLM. The model then generates a response that is aware of both long-term user history and the current implicit constraints, thereby ensuring personalization, safety, and coherence.

C Details of special case

We construct a personalized health scenario where the dialogue history includes a to-do item: *take pre-meal medication before dinner*. During evaluation, the user only requests dinner recommendations without explicitly mentioning medication. This case tests whether the agent can adhere to *implicit constraints* inferred from prior interactions and prioritize actions that are required but not explicitly stated in the query.

Figure 6 illustrates the full pipeline for the case study in Section 4.3. During retrieval, HGP queries heterogeneous external memories separately: (i) the event graph for diet- and symptom-related logs (episodic memory); (ii) the knowledge graph for user preferences and profile facts (semantic memory); and (iii) the rule tree for reusable strategies that previously received positive feedback (procedural memory). Together, these long-term memories supply user-relevant evidence for personalized dinner suggestions (highlighted in blue).

Crucially, HGP also queries the *working-memory states* from the state trajectory graph to retrieve the current actionable status. In this case, working memory returns "pre-meal medication: incomplete" and injects it as an *actionable constraint* into the prompt. Then the LLM reminds the user to take the medication before offeringrecommendations (highlighted in yellow).This behavior is

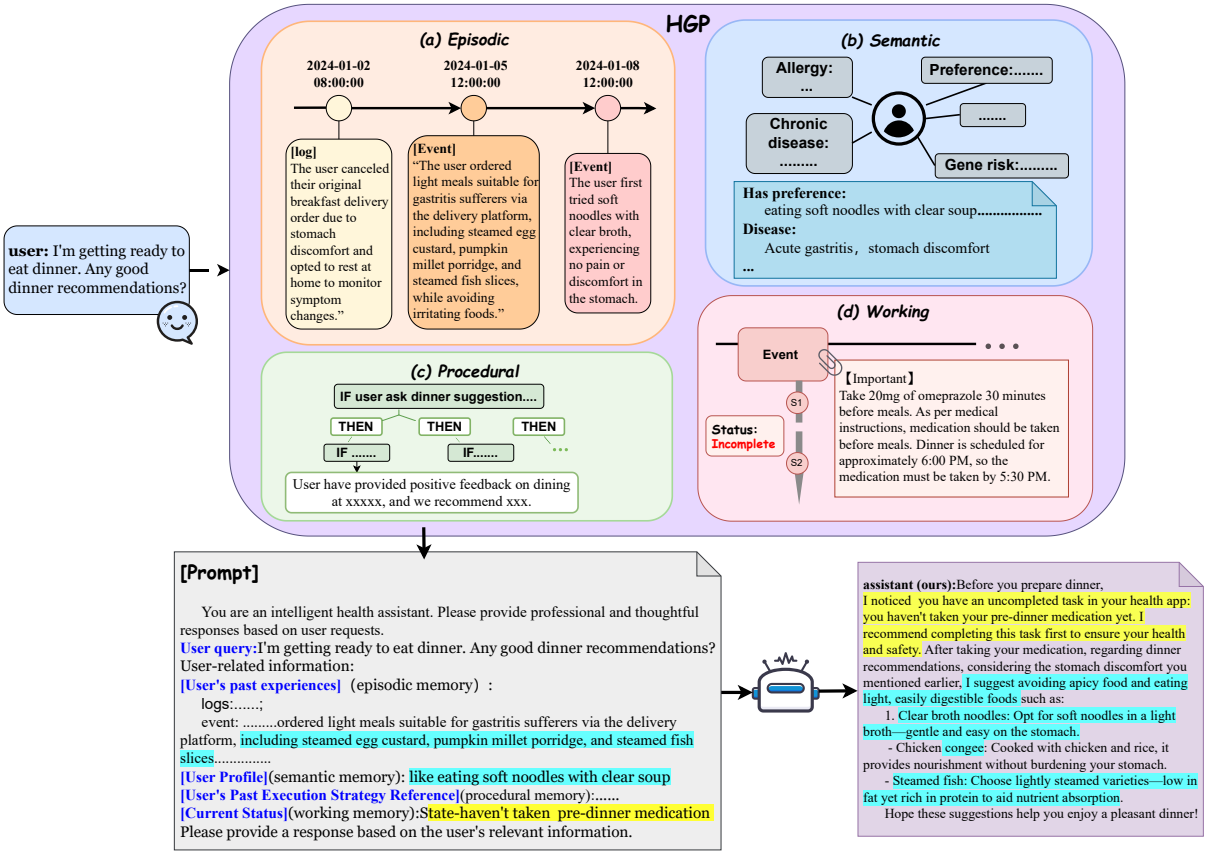

Figure 6: Special case study: illustrating the detailed process by which an LLM retrieves user-relevant memories via HGP in the special case. Working memory addresses *implicit constraints* in a personalized health assistant scenario; its effect is highlighted in yellow (pre-meal medication: pending), while personalized recommendation cues are highlighted in blue.

difficult to trigger reliably using only episodic, semantic, or procedural retrieval, as it hinges on short-horizon execution state rather than long-term facts, preferences, or strategies.

Overall, this case demonstrates HGP 's ability to deliver personalized, context-aware responses. It also shows that long-term memory retrieval alone cannot guarantee compliance with implicit, actionable constraints. By explicitly tracking short-term states via working memory, HGP enhances behavioral consistency and safety in long-horizon interactions.

D Abstraction Operators for Semantic and Procedural Memory from Episodic Memory (LLM Prompts)

To construct semantic and procedural memory from raw episodic traces, we design two abstraction operators, $\mathcal{A}_{\text{sem}}$ and $\mathcal{A}_{\text{proc}}$, each implemented via a dedicated LLM prompt template. These operators extract structured knowledge from episodic event graphs and convert them into reusable memory formats.

D.1 Semantic Abstraction Operator $\mathcal{A}_{\text{sem}}$

The semantic abstraction operator $\mathcal{A}_{\text{sem}}$ extracts static factual knowledge from episodic interactions to construct user profiles, including user attributes, preferences, skills, and long-term constraints. As shown in Figure 7, $\mathcal{A}_{\text{sem}}$ employs a structured LLM prompt to identify and summarize semantic triples in the form of a knowledge graph, given episodic memories.

D.2 Procedural Abstraction Operator $\mathcal{A}_{\text{proc}}$

The procedural abstraction operator $\mathcal{A}_{\text{proc}}$ extracts reusable rules, workflows, and decision strategies from episodic traces. As illustrated in Figure 8, $\mathcal{A}_{\text{proc}}$ uses a structured LLM prompt that requires the model to output each rule in a unified JSON format, containing fields such as title, trigger conditions, execution steps, tags, and a tree-structured workflow that supports sequential and branching logic. If no clear workflow is present, the operator returns an empty list. The extracted rules are then stored in the procedural memory graph $\mathcal{G}_{\text{proc}}$.

PROMPT TEMPLATE FOR $\mathcal{A}_{\text{sem}}$

You are a knowledge graph expert responsible for extracting relevant triples for user semantic memory.

Semantic memory is defined as follows: Semantic memory stores long-term, decontextualized knowledge abstracted from individual experiences, used to describe user profiles (such as relationships, preferences, skills, basic information, etc.). It can be reused across time and tasks.

Extract knowledge triples about user "{user_id}" from the following text.

【Text】:
{content}

【Extraction rules】:
1. Triple format: (User ID, Relation, Entity)
2. Only extract profile knowledge about user {user_id}
3. Relations should use standard verbs that describe user profiles, such as: likes, dislikes, works_as, lives_in, has_skill, good_at, studies, has, is (or other relation words that align with the definition of semantic memory)
4. Entity must be a concrete object

【Confidence guidelines (examples)】:
0.90–1.00: Explicit, direct, unconditional statements in the text (first-person preferences/facts)
Example: "I like basketball", "I live in Shanghai", "I am a doctor"
0.70–0.89: Clear evidence but with minor uncertainty (paraphrased, slightly omitted context, synonymous expressions)
Example: "I've been playing basketball a lot lately" (implies likes, but less direct)
0.50–0.69: Weak evidence / inference (indirect clues, possibly a one-time event)
Example: "I went to play basketball yesterday" (more episodic, not necessarily a stable profile)
< 0.50: Guess, conditional sentence, counterfactual, sarcasm, or insufficient evidence; try not to output such triples (or output with low confidence)

【Special notes】:
Explicit negation (e.g., "I don't like X") should not produce a "likes" triple; instead produce a "dislikes" triple and score according to the same guidelines.
Conditional preferences (e.g., "If I have time, I would...") should be downgraded or not output.

【Example JSON array output】:
[{"relation": "likes", "entity": "basketball", "entity_type": "HOBBY", "confidence": 0.9}]

Output only the JSON array, no additional text. If no triples are found, return []

Figure 7: Illustration of the procedural abstraction operator $\mathcal{A}_{\text{sem}}$.

PROMPT TEMPLATE FOR $\mathcal{A}_{\text{proc}}$

You are a "procedural memory" extraction assistant. Given a piece of text that may describe an operational workflow, an "if...then..." rule, or a multi-step method, extract (0 to N) reusable rules that have received positive user feedback. Each rule should be represented as:

1. "title": A short name for this process/rule
2. "if": A list of trigger conditions (array of strings; can be empty if not mentioned in the text)
3. "then": A list of main steps in chronological order (each step described as a short verb phrase)
4. "tags": A list of short tags (e.g., ["programming", "environment setup"], ["life", "cooking"])
5. "tree": A tree structure representing complex workflows, supporting sequential and branching logic, formatted as follows:

【Sequential execution (sequence)】:

```
{
 "type": "sequence",
 "steps": [ Node, Node, ... ]
}
```

【Basic step (action)】:

```
{
 "type": "action",
 "text": "the specific action to take"
}
```

【Conditional branch (branch)】:

```
{
 "type": "branch",
 "condition": "under what condition this branch is considered (e.g., installation fails)",
 "cases": [
   {
    "when": "specific situation A (e.g., network error)",
    "steps": [ Node, Node, ... ]
   },
   {
    "when": "specific situation B (e.g., insufficient permissions)",
    "steps": [ Node, Node, ... ]
   }
 ]
}
```

Here, Node can be an "action", a "branch", or a nested "sequence".
If the workflow is simple and has no branches, the tree can be a simple sequence corresponding to each step in "then":

```
{
 "type": "sequence",
 "steps": [
   {"type": "action", "text": "step 1"},
   {"type": "action", "text": "step 2"}
 ]
}
```

Please strictly follow the JSON structure below (do not include any extra explanations):

```
{
 "rules": [
   {
    "title": "rule name",
    "if": ["condition 1", "condition 2"],
    "then": ["step 1", "step 2", "step 3"],
    "tags": ["tag 1", "tag 2"],
    "tree": {
      "type": "...",
      "steps": [...]
    }
   }
 ]
}
```

If the text contains no clear workflow or rule, return an empty list: {"rules": []}.
Here is the text:
{text}

Figure 8: Illustration of the procedural abstraction operator $\mathcal{A}_{\text{proc}}$.