ProgMem: Programmatic Belief-State Memory for Long-Horizon Conversational Agents

Anonymous ACL submission

Abstract

Long-horizon conversational agents require memory that can evolve with dialogue, retrieve information on demand, and remain uncertainty-aware when evidence is insufficient, rather than merely larger storage or context windows. Existing methods store interaction histories as retrievable text, summaries, or vector indices, and retrieve relevant content at query time. Although effective for simple factual queries, this paradigm does not explicitly model how memory writes change state, leaving temporal updates, plan changes, and counterfactual reasoning to query-time inference over retrieved text.

We propose **ProgMem**, a programmatic belief-state memory framework that compiles dialogue events into structured memory programs and answers questions through retrieval and verification. On LoCoMo-10, ProgMem achieves the highest Average F1 across Qwen3-4B, Qwen3-8B, and Qwen3-17B, outperforming Mem0, the strongest text-based baseline, by up to 9.95 absolute F1 points on temporal questions. It also achieves the highest accuracy with competitive ingestion cost. Ablations identify two-stage verification and routing-aware retrieval as key components. On DialSim, ProgMem improves aggregate performance on both answerable and unanswerable questions while using fewer query-time memory tokens. These results support programmatic belief-state memory as a robust and efficient alternative to uniform text retrieval.

1 Introduction

Consider the scenario where a user manages a personal schedule through a conversational AI assistant. The user first schedules a follow-up appointment with Dr. Wang, later reschedules it, and eventually cancels it because of a business trip. When the user subsequently asks whether the appointment is still active, the system must recover

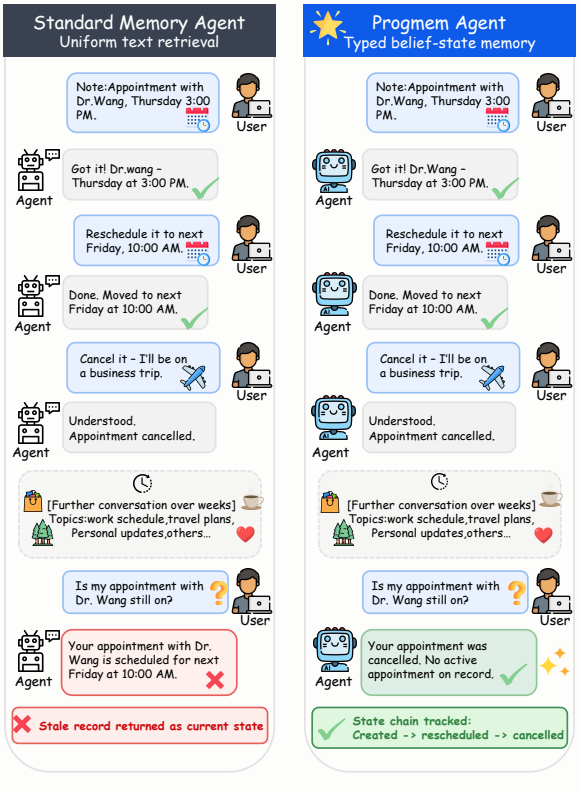

Figure 1: Appointment management under uniform text retrieval and ProgMem. Uniform retrieval may return an obsolete appointment record, whereas ProgMem tracks the state trajectory (*created* → *rescheduled* → *cancelled*) in a typed belief state.

the current state of the same appointment rather than merely retrieve a semantically similar historical record. As illustrated in Figure 1, uniform text retrieval may return an obsolete appointment time, whereas the correct answer requires tracking the state trajectory *created* → *rescheduled* → *cancelled*. This is a common and challenging problem in long-term conversational memory.

The root of this problem is that existing long-term memory methods do not explicitly model how a memory write changes the underlying state. Systems such as MemoryBank (Zhong et al.,

2024), Mem0 (Chhikara et al., 2025), and LangMem (LangChain, 2025) have shown that external memory can alleviate context-window limitations and support personalized interaction across turns. However, they typically organize interaction histories as retrievable text, summaries, structured records, or vector indices, and leave the language model to interpret retrieved content at query time.

This *store–retrieve–interpret* paradigm has three limitations. First, it lacks explicit state-transition semantics for operations such as creating, revising, cancelling, or invalidating a remembered state. Second, uniform retrieval does not distinguish the evidence required by factual, temporal, and plan-oriented queries: the current location of a user and the time at which an event occurred should not be recovered from the same undifferentiated candidate space. Third, when evidence is insufficient, outdated, or conflicting, the model may still produce an apparently definite but unsupported answer, a particularly costly failure mode in long-term personalized interaction (Niu et al., 2024; Wen et al., 2025).

We therefore view long-term conversational memory as an evolving state system rather than a single textual store. This view is inspired by the functional differentiation of human memory systems: semantic memory is associated with relatively stable knowledge, whereas episodic memory concerns experiences situated in time and context (Squire, 2004; Tulving, 1972). ProgMem does not attempt to reproduce human memory mechanisms, but adopts this distinction as an engineering abstraction. It uses `stable` for identities, preferences, constraints, and skills, and `dynamic` for events, locations, relationships, and reference memory. Since conversational agents must also support continuing user tasks, ProgMem further introduces `plan` for goals, schedules, progress, and cancellations.

Building on these typed subspaces, ProgMem compiles dialogue events into structured memory programs, retrieves evidence through query-dependent routes, and verifies candidate answers against anchored dialogue evidence. This design turns long-term memory from uniform text retrieval into an explicit and verifiable state-maintenance process.

2 Related Work

2.1 Retrieval Reliability

Retrieval-augmented generation (RAG) supplements language models with external evidence retrieved at inference time (Lewis et al., 2020). More recent work has begun to examine the reliability of generation after retrieval: Self-RAG integrates retrieval and self-reflection within a unified generation framework (Asai et al., 2024), while RAGTruth shows that retrieval-augmented outputs may still contain unsupported claims (Niu et al., 2024). Related studies on abstention further emphasize that a model should refrain from answering when available evidence is insufficient (Wen et al., 2025).

2.2 Long-Term Agent Memory

Long-term memory systems enable conversational agents to preserve useful information beyond a finite context window. MemGPT introduces virtual context management for extended interaction, while Generative Agents combine persistent memory with reflection and planning (Packer et al., 2023; Park et al., 2023). More directly related to our setting, MemoryBank, Mem0, and LangMem maintain information from prior interactions through extracted memories, summaries, or agent-facing retrieval interfaces (Zhong et al., 2024; Chhikara et al., 2025; LangChain, 2025). Recent work also explores more efficient memory augmentation, as in LightMem, and more structured organization, as in A-MEM and Zep/Graphiti (Fang et al., 2025; Xu et al., 2025; Rasmussen et al., 2025). These methods establish external memory as a practical basis for persistent interaction.

2.3 Structured and Graph Memory

Structured and graph-based memory methods organize information more explicitly than flat textual stores. For example, A-MEM introduces linked agentic memories, while Zep/Graphiti represents evolving information through a temporal knowledge graph (Xu et al., 2025; Rasmussen et al., 2025). Mem0^g extends Mem0 with a relational graph layer on top of its vector memory store, enabling cross-entity multi-hop retrieval through explicit entity-relation edges (Chhikara et al., 2025). Related work on graph-based and structured reasoning further shows that explicit relations and temporal structure can support more

controlled information access (Li et al., 2024; Lin et al., 2023). ProgMem targets a more specific problem: maintaining a conversational belief state under successive updates. It compiles dialogue events into typed writes over `stable`, `dynamic`, and `plan` subspaces, retrieves evidence through query-dependent routes, and retains original dialogue anchors for verification. Thus, its central distinction lies not in structure alone, but in combining explicit state-transition semantics with traceable verification for long-horizon conversational memory.

3 Method: ProgMem

3.1 Task Setup and Formal Framework

Task Setup. Given a dialogue stream $\mathcal{D} = \{e_1, e_2, \ldots, e_T\}$, where each event unit e_t contains an event ID, session ID, timestamp, speaker, raw text, and anchors to the original dialogue evidence, the memory system receives events in temporal order and maintains an external memory state at each step:

$$M_t = \mathrm{Update}(M_{t-1},\, e_t) \tag{1}$$

At query time, the system receives a question q, selects relevant information from the current memory state M_T and the traceable set of raw anchors $\mathcal{A}_T$, and generates an answer:

$$a = \mathrm{Answer}(q,\, M_T,\, \mathcal{A}_T) \tag{2}$$

Design Principles. Long-horizon conversational memory imposes several interrelated requirements that motivate the core design of ProgMem. Stable attributes such as user identity, preferences, and long-term constraints should persist across sessions and remain unaffected by later, unrelated conversational events. Where new information does arrive, the system must determine whether it revises an existing belief or simply adds an independent fact, rather than treating every write as an unconditional append. Time is a first-class concern: stored entries must carry parseable temporal metadata, allowing the system to distinguish past states from current ones when answering time-sensitive queries. Goals and plans introduce a further dimension that general event logs handle poorly; the full lifecycle of a goal—from creation and progress through completion or cancellation—requires dedicated representation and retrieval support. Regardless of content type, every stored belief should remain traceable to the specific dialogue evidence from which it was derived, enabling post-hoc verification. When available evidence is insufficient or conflicting, the system should report uncertainty rather than produce a confident but ungrounded answer. Finally, query-time efficiency requires that the answering model receive only the entries and anchors most pertinent to the question at hand, keeping token consumption proportional to task complexity.

Formal Framework. ProgMem unifies these design principles into a formal framework of compilation, retrieval, and verification (Figure 2).

Compilation and Execution. For each event e_t, the compiler $\mathcal{C}$ maps it under the current state context into a structured memory program p_t, and the execution function $\mathcal{E}$ then writes it into the memory state:

$$p_t = \mathcal{C}(e_t,\, M_{t-1}), \quad M_t = \mathcal{E}(M_{t-1},\, p_t) \tag{3}$$

M_t is a typed belief state that contains three memory subspaces: `stable` (stable information), `dynamic` (dynamic information), and `plan` (plan information). The constrained compiler interprets dialogue events as structured write operations targeting these three subspaces. If an event cannot be reliably structured, the compiler adopts a conservative treatment and retains the raw event as retrievable dynamic memory.

Routing and Retrieval. At query time, the routing function $\mathcal{R}$ selects a processing path according to the question type:

$$\begin{aligned} r &= \mathcal{R}(q), \\ r &\in \{\texttt{factual},\ \texttt{temporal},\\ &\qquad \texttt{plan},\ \texttt{counterfactual}\} \end{aligned} \tag{4}$$

Given route r, the retrieval function constructs a candidate evidence set $\mathcal{Z} = \mathrm{Retrieve}(q, r, M_T, \mathcal{A}_T)$ from the belief state and anchor evidence.

Verification and Answering. The final answer is generated through two-stage verification:

$$a = \mathrm{Verify}\big(\mathrm{Gen}_{\text{state}}(q, \mathcal{Z}),\ \mathrm{Gen}_{\text{anchor}}(q, \mathcal{Z})\big) \tag{5}$$

The following sections introduce the implementation details of the three cooperating modules.

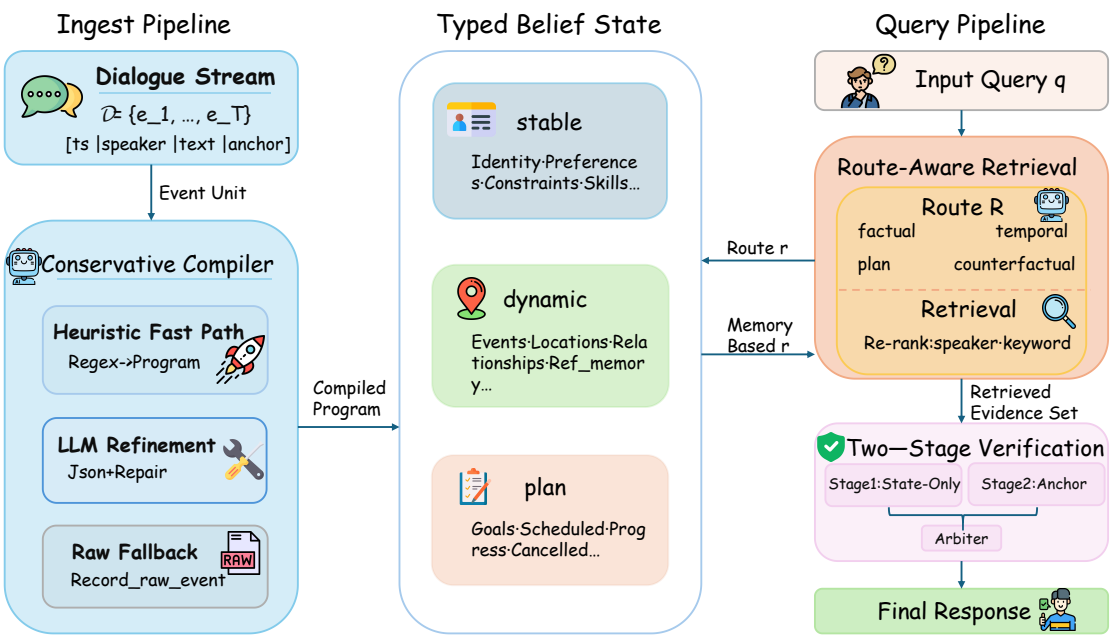

Figure 2: Overview of ProgMem. Left: dialogue events are compiled into a typed belief state. Middle: the state is organized into `stable`, `dynamic`, and `plan` subspaces. Right: route-aware retrieval and two-stage verification produce an answer or `Unknown`.

3.2 Ingest Pipeline

ProgMem segments a dialogue stream into event units, each bound to traceable anchors. As shown in Eq. (3), the compiler $\mathcal{C}$ uses a three-layer progressive strategy to map events into memory programs and write them into state: heuristic rules directly output programs for clearly semantic cases such as identity, preferences, and plans; when rules are insufficient, an LLM is invoked for conservative structuring, where arguments must come from the original text and uncertain content is retained in the `residual` field; finally, a fallback preserves the raw event record and extracts high-precision named entities into `reference_memory`. Programs are dispatched by type into the corresponding subspaces (the `stable`, `dynamic`, and `plan` subspaces described in the next subsection), and duplicate entries are removed using Jaccard token overlap, retaining the entry with higher confidence.

3.3 Typed Belief State

The typed belief state consists of three subspaces: `stable` maintains stable person information (identity, preferences, constraints, and skills), `dynamic` maintains dynamic facts (events, locations, relationships, reference memory, etc.), and `plan` maintains goals and plans (goals, schedules, progress, cancellations). Each entry carries a value, anchors, confidence, and a parseable timestamp, ensuring structured retrievability and traceability to raw evidence.

3.4 Query Pipeline

ProgMem uses the routing function $\mathcal{R}$ to assign questions to four specialized paths (factual / temporal / plan / counterfactual). Each route corresponds to a differentiated candidate-construction strategy: the temporal route prioritizes events and schedule entries with timestamps; the plan route prioritizes goals and progress; the factual route covers all `stable` fields and the relationship, location, and reference-memory partitions of `dynamic`. During ranking, Qwen3-Embedding-0.6B (Zhang et al., 2025) provides dense scores, which are combined with local signals such as speaker matching and focus-token overlap for reranking. After retrieval, answers are generated through two-stage verification: Stage 1 produces a candidate answer using only the structured belief state; Stage 2 introduces raw anchor evidence for verification, keeping the answer if supported, correcting it if contradicted, and outputting `Unknown` if both are insufficient. For non-temporal routes, Stage 2 is triggered when the retrieval confidence score falls below threshold θ_v; for the temporal route, both

stages are always activated and a dedicated temporal verification arbiter is applied. For counterfactual queries, the system first applies removal and hypothesis operations to the belief state to construct a temporary state M'_T, then repeats routing, retrieval, and two-stage verification on M'_T, enabling locally traceable temporary state intervention.

4 Experiments

4.1 Datasets

LoCoMo-10 (Maharana et al., 2024) is the standard evaluation subset of the LoCoMo benchmark, comprising 10 real-world long-horizon multi-session dialogues with a total of 1,986 question-answer pairs, where each dialogue spans 19–32 sessions. The original annotations cover five question categories: SingleHop (SH, 841 instances), Temporal (TMP, 321 instances), MultiHop (MH, 282 instances), OpenDomain (OD, 96 instances), and Adversarial (ADV, 446 instances). **DialSim** (Kim et al., 2024) is constructed from real television scripts (*Friends*, *The Big Bang Theory*, and *The Office*), and is specifically designed to evaluate a system's ability to distinguish between answerable questions and fallback-unanswerable questions (FUQs). The benchmark consists of 3,000 questions in total, with an overall answerable question ratio of 36.10%; the ratios for the three individual sub-datasets are 34.00%, 44.80%, and 29.50%, respectively, as detailed in Appendix Table 8. Since FUQs are deliberately constructed to be semantically indistinguishable from answerable questions, this benchmark poses substantial challenges to the conservative fallback capability of memory systems.

Given that FUQs constitute a substantial portion of DialSim (66.00% in BigBang, 55.20% in Friends, and 70.50% in TheOffice) and are intentionally aligned with answerable questions in semantics to maximize recognition difficulty, DialSim provides a more rigorous testbed for evaluating the robustness of the `Unknown` fallback path in ProgMem's two-stage verification mechanism. Accordingly, on LoCoMo-10 we evaluate only the first four answerable question categories, delegating the assessment of conservative fallback capability as probed by the Adversarial category to the more stringent generalization evaluation offered by the DialSim benchmark.

4.2 Baselines

MemoryBank (Zhong et al., 2024): simulates human memory forgetting with an importance-decay mechanism and retrieves relevant summary entries at query time. **LangMem** (LangChain, 2025): a long-term storage tool for agents, supporting interfaces for writing and retrieving semantic memories. **Mem0** (Chhikara et al., 2025): a production-ready explicit memory extraction and update system that stores memory entries in a vector database and performs incremental management. **Mem0^g** (Chhikara et al., 2025): the graph-enhanced variant of Mem0, which adds a relational graph structure on top of vector memory.

4.3 Evaluation Metrics

LoCoMo-10 experiments. Following the evaluation protocol of LoCoMo (Maharana et al., 2024), we report token-level F1 and BLEU-1. F1 follows the bag-of-words formula used in the SQuAD evaluation script (Rajpurkar et al., 2016), computing the harmonic mean of token-level precision and recall after lowercasing and tokenizing the predicted and reference answers. BLEU uses unigram BLEU-1 (Papineni et al., 2002) with NLTK `SmoothingFunction().method1` to handle zero unigram counts in short answers. Both metrics are reported separately on four answerable question categories: SingleHop (SH), Temporal (TMP), MultiHop (MH), and OpenDomain (OD). The primary comparison metric is the macro average of the four F1 scores (Avg F1).

DialSim experiments. Following the evaluation protocol of DialSim (Kim et al., 2024), we compute the same token-level F1 for each question-answer pair, and then summarize results into three metrics by question subset. The sample composition of each subset is shown in Appendix Table 8: **Overall F1** is the macro-average F1 over all questions, including both answerable and unanswerable questions; **Answerable F1** is the macro-average F1 over the answerable subset; **FUQ F1** is the macro-average F1 over the unanswerable (FUQ) subset, whose gold answer is the fixed short phrase "I don't know.". ProgMem employs `Unknown` to denote abstention when the available evidence is insufficient to support an answer. Since DialSim annotates the reference answers of unanswerable questions as "I don't know.", we normalize `Unknown` to "I don't know." in all system outputs prior to F1 computation. This normaliza-

tion is applied uniformly to the outputs of all baseline methods evaluated on DialSim. All three F1 metrics are first macro-averaged within each series and then reported overall across the three datasets.

4.4 Experimental Setup

We uniformly use Qwen3 models at three scales (4B, 8B, and 17B) and Qwen3-Embedding-0.6B. All reported results are averaged over three independent runs under the same experimental configuration. The main hyperparameters used in this section are listed in Appendix Table 7.

4.5 Main Results on LoCoMo-10

Table 1 reports the full results on LoCoMo-10 under three model scales.

Scale-dependent performance growth. ProgMem achieves the highest Avg F1 under all three model scales (29.47, 31.76, and 32.18), exceeding the strongest text-based baseline, Mem0, by 5.17, 6.48, and 8.15 percentage points, respectively (relative improvements of 21.3%, 25.6%, and 33.9%). This pattern suggests that larger models have stronger structured reasoning and formatted-output abilities, allowing the programmatic compilation and verification mechanisms to be used more fully and thus amplifying the gains of ProgMem over text-retrieval baselines.

Temporal reasoning advantage. ProgMem's Temporal F1 is 42.76, 43.34, and 43.12 under the three model scales, respectively. The improvement is most prominent on temporal questions and remains stable across model scales. It exceeds the strongest text-based baseline (Mem0) by up to 9.95 percentage points and exceeds LangMem by up to 31.8 percentage points. LangMem's very low scores on Temporal questions (11.30–14.64) reveal an inherent weakness of unified text retrieval on temporal questions: when the system cannot distinguish "current state" from "historical state," temporal questions are difficult to answer accurately.

Competitive single-hop factual retrieval. On SingleHop questions, ProgMem stays within 0.5 points of the strongest baseline Mem0^g at smaller scales (4B: 33.72 vs. 33.83; 8B: 35.63 vs. 36.08) and achieves the best result at 17B scale (37.88), indicating that programmatic memory does not introduce additional loss on simple factual retrieval.

4.6 Cost Analysis

Table 2 compares the computational costs of different methods along three dimensions: per-event ingest cost (Ingest Tok/Event), total per-dialogue ingest cost (Total Ingest/Dialogue), and query-time memory-injection tokens (Mem Tok/Query).

Programmatic ingest maintains a cost advantage while improving accuracy. ProgMem consumes 625.10 tokens per event, about 53% lower than Mem0's 1,331.76 and also 12% lower than LangMem's 709.61. In total per-dialogue ingest cost, ProgMem (18,821.9) is close to LangMem (19,301.4), and both are substantially lower than Mem0 (36,224) and Mem0^g (40,000). On the query side, ProgMem's memory-injection cost (152.50 tokens/query) is slightly higher than Mem0's (130.62), but this trade-off is reasonable given its significantly higher accuracy. Overall, ProgMem achieves the highest accuracy while maintaining relatively optimized ingest and query costs, showing a favorable balance between cost and accuracy.

4.7 Ablation Study

Table 3 reports the results of two controlled ablation variants (V1, V2) against the full system (V0) under the Qwen3-4B + LoCoMo-10 setting.

Two-stage verification has a significant effect on accuracy. Removing two-stage verification (V1) reduces Avg F1 from 29.47 to 25.73. Temporal F1 drops sharply from 42.76 to 32.02. OD F1 slightly increases under V1, suggesting that the verification mechanism can sometimes be overly conservative for open-domain questions. It should be noted that V1 removes two coupled components at the same time: the anchor evidence path and the arbitration step. This is an intentional coupled design: the value of verification comes from comparing the state-only answer against the anchor-only answer for consistency. Removing only one of them would not constitute a meaningful "verification" control. Therefore, V1 measures the contribution of the complete two-stage verification mechanism as a whole, rather than the independent effect of a single subcomponent.

The routing-aware framework is indispensable for temporal reasoning. Removing the routing-aware framework (V2) reduces Avg F1 by 3.38 and Temporal F1 by 14.25. V2's SingleHop F1 is

Model	Method	SH F1	SH BLEU	TMP F1	TMP BLEU	MH F1	MH BLEU	OD F1	OD BLEU	Avg F1
Qwen3-4B	MemoryBank	4.95	3.85	2.38	1.84	6.10	4.13	6.89	4.71	5.08
	LangMem	33.61	28.87	14.64	10.96	21.88	15.16	12.75	7.83	20.72
	Mem0	31.33	27.72	36.41	30.22	19.17	13.97	10.29	6.74	24.30
	Mem0^g	**33.83**	**30.53**	38.00	31.90	18.09	12.42	11.99	7.40	25.48
	ProgMem	33.72	29.27	**42.76**	**36.29**	**23.52**	**17.60**	**17.87**	**14.90**	**29.47**
Qwen3-8B	MemoryBank	4.94	3.95	2.55	2.02	5.60	3.75	7.01	4.92	5.02
	LangMem	33.90	28.59	11.82	8.49	22.48	16.12	17.27	12.04	21.36
	Mem0	32.29	28.14	34.12	28.99	19.02	13.65	15.69	10.68	25.28
	Mem0^g	**36.08**	**32.18**	37.95	31.56	18.14	12.26	13.51	9.79	26.42
	ProgMem	35.63	31.33	**43.34**	**36.86**	**27.44**	**20.28**	**20.64**	**16.98**	**31.76**
Qwen3-17B	MemoryBank	5.48	4.33	2.71	2.11	6.51	4.45	8.32	6.04	5.75
	LangMem	37.60	32.86	11.30	7.71	22.94	16.26	17.08	12.60	22.23
	Mem0	33.08	29.77	33.17	29.10	17.06	11.42	12.82	8.90	24.03
	Mem0^g	28.92	24.23	**44.17**	**37.27**	17.56	12.69	11.96	8.72	25.65
	ProgMem	**37.88**	**33.52**	43.12	36.43	**26.15**	**19.49**	**21.59**	**17.57**	**32.18**

Table 1: Main results on LoCoMo-10 (F1 / BLEU). The optimal performance is shown in **bold**.

Method	Ingest Tok / Event	Total Ingest / Dialogue	Mem Tok / Query
MemoryBank	5.56	1,675.10	309.81
LangMem	709.61	19,301.40	67.89
Mem0	1,331.76	36,224.00	130.62
Mem0^g	1,330.00	40,000.00	260.00
ProgMem	625.10	18,821.90	152.50

Table 2: Comparison of ingest and query costs.

instead slightly higher than V0's, revealing a structural trade-off: the routing-aware framework improves temporal-question accuracy by narrowing the evidence focus, but occasionally loses slightly on SingleHop questions. Query-injection tokens increase from 152.50 to 182.53, indicating that increasing the amount of evidence cannot equivalently replace the contribution of evidence focusing to temporal-reasoning accuracy.

4.8 DialSim Results

Table 4 presents the comparison between Mem0 and ProgMem (both using Qwen3-4B) on the three DialSim datasets.

In the DialSim experiments, we only use Mem0 as the comparison baseline and do not include the remaining baselines from LoCoMo-10, including Mem0^g. The reason for selecting Mem0 rather than Mem0^g is as follows: the central challenge of DialSim is to distinguish answerable from unanswerable questions, with FUQ accounting for 66.00%, 55.20%, and 70.50% of the BigBang, Friends, and TheOffice subsets, respectively, rather than to perform multi-hop entity-relation reasoning. The graph-enhanced design of Mem0^g is intended for the latter, improving cross-session multi-hop retrieval through relational graph structure; in DialSim, where single-hop factual queries dominate, its graph component is expected to provide limited marginal benefit and does not constitute a stronger point of comparison. In contrast, as a mainstream production-oriented explicit-memory baseline, Mem0 represents a standard implementation of the "store–retrieve–generate" paradigm and offers broader reproducibility, making it more suitable as a representative comparison for ProgMem in this setting.

ProgMem outperforms Mem0 on Overall F1 across all three datasets, achieving an aggregate improvement of 4.4%, while also improving aggregate FUQ F1 by 4.1%. For answerable questions, ProgMem improves the aggregate Answerable F1 over Mem0 by 9.8% ($9.45\% \rightarrow 10.38\%$), with the largest gain on TheOffice (+27.4%) and the second largest on BigBang (+10.5%). Moreover, ProgMem uses 130.12 memory tokens per query, 25.5% fewer than Mem0's 174.66 tokens, achieving higher overall accuracy with lower query-time memory injection.

5 Conclusion

We presented **ProgMem**, a programmatic belief-state memory framework that treats long-horizon conversational memory as state maintenance rather than uniform text retrieval. ProgMem compiles dialogue events into structured memory programs, routes queries to appropriate evidence, and verifies candidate answers against anchored dialogue evidence. Experiments on LoCoMo-10 and DialSim show strong performance on temporal reasoning and aggregate answerability assessment while maintaining competitive token efficiency. These results suggest that explicit and verifiable state maintenance is a promising direction for long-term conversational memory.

ID	Variant	SH F1	TMP F1	MH F1	OD F1	Avg F1	Δ Avg	MemTok
V0	Full system (baseline)	33.72	42.76	23.52	17.87	**29.47**	—	152.50
V1	Remove two-stage verification (State-Only only)	30.75	32.02	20.64	19.51	25.73	−3.74	154.00
V2	Remove routing-aware framework (unified retrieval)	34.63	28.51	23.35	17.87	26.09	−3.38	182.53

Table 3: Controlled ablation results (Qwen3-4B, LoCoMo-10).

Dataset	Metric	Mem0	ProgMem
BigBang	Overall F1	63.06	**67.00**
	Answerable F1	10.39	**11.48**
	FUQ F1	90.20	**95.61**
Friends	Overall F1	53.50	**55.45**
	Answerable F1	**8.90**	8.66
	FUQ F1	89.70	**93.42**
TheOffice	Overall F1	68.41	**70.72**
	Answerable F1	9.19	**11.71**
	FUQ F1	93.19	**95.41**
Overall	Overall F1	61.66	**64.39**
	Answerable F1	9.45	**10.38**
	FUQ F1	91.15	**94.91**
	Avg Mem Tok / Query	174.66	**130.12**

Table 4: DialSim results (Qwen3-4B). F1 values are in percentage (%); Avg Mem Tok / Query is in tokens.

Limitations

Several limitations remain in the current implementation. Query routing uses lightweight pattern matching, which can misclassify ambiguous or mixed-type questions and direct them to suboptimal evidence paths. The verification trigger is likewise heuristic and tends toward over-caution on open-domain questions where the evidence boundary is inherently fuzzy. Addressing these issues will likely require learned calibration of the compiler and trigger thresholds, as well as more adaptive routing capable of handling composite query intents.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*.

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

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2025. LightMem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.

Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yohan Jo, and Edward Choi. 2024. DialSim: A real-time simulator for evaluating long-term multi-party dialogue understanding of conversational agents. *arXiv preprint arXiv:2406.13144*.

LangChain. 2025. Langmem sdk for agent long-term memory.

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. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33*, pages 9459–9474.

Shilong Li, Yancheng He, Hangyu Guo, Xingyuan Bu, Ge Bai, Jie Liu, Jiaheng Liu, Xingwei Qu, Yangguang Li, Wanli Ouyang, Wenbo Su, and Bo Zheng. 2024. GraphReader: Building graph-based agent to enhance long-context abilities of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12758–12786, Miami, Florida, USA. Association for Computational Linguistics.

Xueyuan Lin, Haihong E, Chengjin Xu, Gengxian Zhou, Haoran Luo, Tianyi Hu, Fenglong Su, Ningyuan Li, and Mingzhi Sun. 2023. Tflex: Temporal feature-logic embedding framework for complex reasoning over temporal knowledge graph. In *Advances in Neural Information Processing Systems 36*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, KaShun Shum, Randy Zhong, Juntong Song, and Tong Zhang. 2024. RAGtruth: A hallucination corpus for developing trustworthy retrieval-augmented language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10862–10878, Bangkok, Thailand. Association for Computational Linguistics.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

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. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, San Francisco, CA, USA. Association for Computing Machinery.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of reading passages. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 238–240, Austin, Texas. Association for Computational Linguistics.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

Larry R. Squire. 2004. Memory systems of the brain: A brief history and current perspective. *Neurobiology of Learning and Memory*, 82(3):171–177.

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

Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. 2025. Know your limits: A survey of abstention in large language models. *Transactions of the Association for Computational Linguistics*, 13:529–556.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 12 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731.

A Typed Belief State Structure

The typed belief state in ProgMem consists of three main memory subspaces, each responsible for maintaining memory entries of different semantic types:

- **Stable subspace**: stores relatively stable personal attributes, including identity information, long-term preferences, constraints, and skills. Entries in this subspace usually remain valid across sessions and are modified only when explicitly updated or cancelled.
- **Dynamic subspace**: stores dynamic facts that change over time, including historical events, location information, interpersonal relationships, availability states, resource conditions, support signals, and exceptions. Entries in this subspace include parseable timestamps and support temporally sensitive evidence retrieval.
- **Plan subspace**: stores information related to goals and plans, including goal creation, progress updates, schedule arrangements, priority changes, and cancellation records. This subspace supports multi-stage tracking of the plan lifecycle.

Each entry written into state contains a value, raw evidence anchors, confidence, and a timestamp, ensuring structured retrievability and traceability to raw evidence.

B Key Prompt Templates

This appendix presents the actual system prompts used in each core stage of ProgMem. All prompts require output without chain-of-thought to ensure that answers are short and parseable.

B.1 Compiler Prompt

You are a conservative latent-program compiler for long-term memory.

Your job is to map one event into exactly one latent program that updates a structured belief state.

The belief state is the main memory substrate. Raw text is only a fallback evidence layer.

Therefore: - prefer program choices that create reusable state updates - preserve state-operable information such as identity, preference, goal, relation, location, constraint, plan progress, support signal, advice, exception - if the event is too ambiguous, too conversational, or not clearly state-changing, choose the conservative raw-text preservation option - do not hallucinate structure - do not split one event into multiple programs

Return strict JSON: {"program_id": "...", "arguments": {...}, "residual": "...", "confidence": 0.0}

Rules: 1. Prefer correctness over coverage. 2. If uncertain, choose the most conservative program option available. 3. Confidence must be between 0 and 1. 4. Arguments must be short, canonical strings when possible. 5. Put information not safely captured by the chosen program into residual. 6. Do not include any text outside JSON. 7. Copy argument content from the event as faithfully as possible; do not invent unseen facts.

The user-side prompt format is:

Speaker: {speaker} | Timestamp: {timestamp} |
Event: {event_text}
Heuristic hint: {hint or "none"}
State summary: {current state summary}
Candidate programs: {library}
Choose exactly one program_id from the candidate list

B.2 State-Only Answer Generation Prompt

You are a belief-state reasoning engine.

Answer using ONLY the structured belief state below. No anchors, no raw dialogue. If the state does not contain enough information, reply exactly: Unknown.

Output rules: - return only the final short answer - no explanation, no chain-of-thought, no <think> tags - for temporal questions, return the date/time phrase copied from the best matching state entry - for identity / status / research / career questions, return the short slot value only

B.3 Anchor Verification Arbiter Prompt

You are the evidence verification stage of a belief-state memory system.

You are given: 1. the prior state-only answer produced in stage one 2. the structured belief state 3. the selected evidence anchors (raw dialogue)

Your job: decide the final short answer. - If the state-only answer is grounded by the anchors, keep it verbatim. - If the anchors contradict or refine the state-only answer, return the anchor-grounded answer instead. - If neither state nor anchors support a confident answer, reply exactly: Unknown.

Output rules: - return only the final short answer - no explanation, no chain-of-thought, no <think> tags

B.4 Counterfactual Execution Prompt

You are the shallow counterfactual executor of a belief-state memory system.

You will receive an intervened belief state plus an intervention summary describing which entries were removed or flagged hypothetically. Answer the question as if the intervened belief state is the full truth. Do NOT replay or invent downstream events that would follow from the intervention. Do NOT use anchors that were explicitly removed.

Output rules: - return only the final short answer - no explanation, no chain-of-thought, no <think> tags - if the intervened state cannot support an answer, reply exactly: Unknown.

B.5 Temporal Verification Prompt

You are the temporal verification arbiter of a belief-state memory system.

You are given: 1. the prior state-only temporal answer 2. the anchor-only temporal answer 3. the supporting evidence anchors

Decision rule: - if the anchor-only answer clearly grounds or corrects the state-only answer, return the anchor-only answer - if the anchor-only answer is Unknown but the state-only answer is still supported by anchors, keep the state-only answer - if both are unsupported or conflicting without resolution, return Unknown

Output rules: - return only the final short answer - no explanation, no chain-of-thought, no <think> tags

B.6 Reference Memory Extraction Prompt

You extract high-precision reference memory from a single dialogue event.

Valid reference types: book_title, pet_name, artist_or_band, person_name, explicit_location, organization_or_group, named_object

Rules: 1. Extract at most 2 references. 2. Only extract if the phrase is explicitly present in the event text. 3. Prefer precision over coverage. 4. Ignore vague summaries, emotions, plans, and generic noun phrases. 5. If no high-confidence reference is present, return an empty list. 6. Do not infer unseen names or titles. 7. Output JSON only.

Required schema: {"references": [{"reference_type": "...", "value": "...", "cue": "...", "confidence": 0.0}]}

C Program Library and Routing Structure

This appendix uses two tables to present the core structures of ProgMem's programmatic design. Table 5 lists the semantic description, target subspace, and main argument key of all 24 programs. Table 6 describes the triggering conditions, priority-access subspaces, and retrieval strategies of the four query routes.

C.1 Program Library

The compiler maps each dialogue event uniquely to one memory program in the program library. Programs are divided into three groups according to target subspace: the `stable` group writes relatively stable attribute slots; the `dynamic` group writes temporal dynamic records; and the `plan` group writes lifecycle states of goals and plans. `record_raw_event` is the conservative fallback program, with a confidence upper bound of 0.59. It does not write structured slots, but only preserves the raw text and named-entity extraction results.

C.2 Query Routing

Query routing assigns questions to four specialized paths using lightweight pattern matching. Each path corresponds to different subspace access priorities and retrieval strategies, avoiding the dilution of different evidence types in the same candidate pool.

D Implementation Details

Embedding model: Qwen3-Embedding-0.6B (Zhang et al., 2025), accessed through an OpenAI-compatible interface to a local service, with batch size 64 and 4 concurrent threads.

LLM interface: Qwen3-4B/8B/17B (Yang et al., 2025), locally deployed with thinking mode disabled.

Hardware environment: A server with 4× NVIDIA A100 40G GPUs.

D.1 Main Hyperparameter Settings

Hyperparameter	Value
Compilation confidence threshold	0.60
top-k	8
Verification trigger threshold	0.55
Embedding batch size / concurrency	64 / 4

Table 7: Main hyperparameter settings.

D.2 DialSim Dataset Details

Dataset	Total (N)	Answerable Examples	FUQ Examples	Answerable Ratio
BigBang	1000	340	660	34.00%
Friends	1000	448	552	44.80%
TheOffice	1000	295	705	29.50%
Overall	3000	1083	1917	36.10%

Table 8: DialSim dataset details.

E Algorithm Pseudocode

This appendix provides complete pseudocode for the two core stages of ProgMem to clearly present the execution logic of the ingest and query pipelines.

F Use of AI Assistants

AI assistants were used for language polishing and grammar checking of the manuscript. All technical content, experimental design, implementation, and results are solely the work of the authors.

Program ID	Semantic Description	Target Subspace	Main Argument Key
Stable group–writes relatively stable attribute slots			
set_identity	Sets a stable identity fact	stable.identity	identity
update_identity	Revises an existing identity fact	stable.identity	identity
add_stable_preference	Records a persistent preference	stable.preferences	preference
weaken_preference	Weakens an existing preference	stable.preferences	preference
add_constraint	Records a persistent constraint	stable.constraints	constraint
add_skill	Records an acquired skill or ability	stable.skills	skill
Dynamic group–writes temporal dynamic records			
record_past_event	Stores a past event	dynamic.events	event
update_relationship	Updates a relationship state or fact	dynamic.relationships	relationship
set_location	Stores location information	dynamic.locations	location
add_exception	Stores a rare or exceptional fact	dynamic.exceptions	exception
invalidate_belief	Invalidates an existing belief	dynamic.exceptions	belief
record_advice	Stores advice or another person's opinion	dynamic.events	advice
record_support_signal	Stores a support or encouragement signal	dynamic.support_signals	signal
record_conflict	Stores a conflict or tense relation	dynamic.conflicts	conflict
set_availability	Stores availability or schedule state	dynamic.availability	availability
set_resource	Stores a resource or budget fact	dynamic.resources	resource
Plan group–writes lifecycle states of goals and plans			
create_goal	Creates a new goal	plan.goals	goal
revise_goal	Revises an existing goal	plan.goals	goal
complete_goal	Marks a goal as completed	plan.progress	goal
schedule_future_event	Stores a future plan or schedule arrangement	plan.scheduled	plan
update_plan_progress	Updates the progress of a goal or plan	plan.progress	progress
cancel_plan	Stores a cancelled plan	plan.cancelled	plan
change_priority	Stores a priority change	plan.priorities	priority
Fallback–conservative fallback without writing structured slots			
record_raw_event	Raw-text preservation (confidence capped at 0.59)	dynamic.raw_events	

Table 5: ProgMem program library: program ID, semantic description, target subspace, and main argument key.

Route	Typical Trigger Patterns	Priority-Access Subspaces	Retrieval Strategy
factual	Default path; no explicit temporal, plan, or counterfactual trigger words	stable.* (all slots); dynamic.relationships; dynamic.locations; dynamic.reference_memory	Dense vector retrieval + speaker-matching weights + focus-token overlap reranking
temporal	Temporal interrogatives such as "when," "what date," "how long ago," "the week before," and "what year"	dynamic.events (with parseable timestamps); plan.scheduled; dynamic.locations	Timestamp-aware ranking; temporal questions introduce both state-only and anchor-only verification paths
plan	Plan-intent words such as "planning to," "going to," "will pursue," "next step," and "likely to pursue"	plan.goals; plan.scheduled; plan.progress; plan.priorities; plan.cancelled	Covers the full goal lifecycle (creation → progress → cancellation), prioritizing the latest progress entries
counterfactual	Hypothetical conditionals such as "if X had/hadn't," "suppose," "hypothetically," and "if X never"	Intervened temporary state M'_T (removal removes specified anchors; hypothesis injects hypothetical facts)	Repeats factual/temporal/plan routing on M'_T; interventions affect only the current query and are not persisted

Table 6: ProgMem query routing: triggering patterns, priority-access subspaces, and retrieval strategies.

Algorithm 1 ProgMem Ingest Pipeline: Compile and Update

Require: Event unit e_t, current belief state M_{t-1}, compilation confidence threshold θ_c, recovery threshold $\theta_r = \theta_c - 0.12$, heuristic priority margin $\delta_h = 0.06$

Ensure: Updated belief state M_t

```
procedure COMPILEANDUPDATE(e_t, M_{t-1})
    h ← HEURISTICCOMPILE(e_t)
    if SHOULDFASTPATH(h) then
        p ← h
    else
        l ← LLMCOMPILE(e_t, M_{t-1}, h)
        p ← SELECTCANDIDATE(h, l)
        if p.confidence < θ_c then
            p' ← HEURISTICRECOVER(e_t, h)
            p ← p' if p' ≠ None else FALLBACK(e_t)
        end if
    end if
    M_t ← APPLYPROGRAM(M_{t-1}, p)
    if p.program_id = record_raw_event then
        R ← EXTRACTREFERENCEMEMORY(e_t)
        M_t.reference_memory += R
    end if
    return M_t
end procedure
function SELECTCANDIDATE(h, l)
    if l = None then
        return h
    end if
    if l.program = record_raw_event and h.confidence ≥ θ_r then
        return h
    end if
    if h.confidence ≥ l.confidence + δ_h then
        return h
    end if
    return l
end function
```

Algorithm 2 ProgMem Query Pipeline: Route, Retrieve and Verify

Require: Question q, belief state M_T, anchor set $\mathcal{A}_T$, verification trigger threshold θ_v
Ensure: Final answer a

```
procedure RouteRetrieveVerify(q, M_T, A_T)
    r ← RouteQuestion(q)
    if r = counterfactual then
        σ ← ParseIntervention(q)
        M'_T ← ApplyIntervention(M_T, σ)
        Z ← Retrieve(q, r, M'_T, FilterAnchors(A_T, σ))
    else
        Z ← Retrieve(q, r, M_T, A_T)
    end if
    a_s ← GenerateStateOnly(q, Z.state)
    if r = temporal then                          ▷ temporal route: always activate Stage 2
        a_a ← GenerateAnchorOnly(q, Z.anchors)
        a ← TemporalArbiter(a_s, a_a, Z)
    else if ShouldVerify(Z, θ_v) then             ▷ other routes: threshold-triggered
        a_a ← GenerateAnchorOnly(q, Z.anchors)
        a ← Arbiter(a_s, a_a, Z)
    else
        a ← a_s
    end if
    return a
end procedure
function ShouldVerify(Z, θ_v)
    s ← ComputeRetrievalScore(Z)
    return s < θ_v
end function
function TemporalArbiter(a_s, a_a, Z)             ▷ see Appendix B for prompt
    if a_a ≠ Unknown and Grounds(a_a, a_s) then
        return a_a                                ▷ anchor clearly grounds or corrects state answer
    else if a_a = Unknown and a_s ≠ Unknown then
        return a_s                                ▷ anchor unknown but state answer still anchor-supported
    else
        return Unknown                            ▷ both unsupported or conflicting
    end if
end function
function Arbiter(a_s, a_a, Z)
    if a_a ≠ Unknown and Consistent(a_s, a_a) then
        return a_s
    else if a_a ≠ Unknown then
        return a_a
    else if a_s ≠ Unknown then
        return a_s
    else
        return Unknown
    end if
end function
```