STREAMARENA: TOWARD CONTINUOUS, INTERACTIVE, AND LONG-HORIZON AGENTIC STREAMING VIDEO UNDERSTANDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deploying autonomous multimodal agents in continuous, real-world environments requires them to ingest unbounded audio-visual streams and maintain hour-scale memory. However, current evaluations predominantly rely on brief clips and multiple-choice formats. This design allows minimal baselines that process only the last four frames to match or surpass complex streaming models, while answer options also expose language shortcuts. We introduce *StreamArena*, a benchmark for hour-scale, interactive streaming video understanding. StreamArena contains 243 full-length videos averaging 88.8 minutes and 3,646 rigorously annotated, open-ended question-answer pairs that evaluate real-time perception, historical retrospection, proactive interaction, and multimodal tool utilization. Evaluation across diverse systems exposes a tension between continuous interaction and long-horizon multimodal comprehension. Methods that retain only recent frames cannot recover distant events, methods that convert past observations into text lose visual evidence, and methods that repeatedly compress visual memory struggle to preserve fine-grained details over time. We address this tension with *StreamMind*, a two-tier architecture that assigns latency-critical interaction and proactive monitoring to independently scheduled frontend workers, while backend workers asynchronously construct persistent multimodal memory and perform historical recall and external search. StreamMind outperforms streaming baselines across four capabilities and reduces query-to-answer latency by reusing persistent state.

1 INTRODUCTION

As artificial intelligence integrates into continuous real-world environments like embodied robotics and wearable devices (Cournan et al., 2016; Ahn et al., 2023; Zhang et al., 2025; Lynch et al., 2023; Fang et al., 2025), agents are expected to maintain always-on, hour-scale multimodal comprehension. However, traditional turn-based models (Zhang et al., 2023; Lin et al., 2024; Maaz et al., 2024; Li et al., 2024) struggle in these dynamic settings. By design, they process pre-segmented video clips and passively wait for user prompts, which disrupts the causal continuity of real-world observations. Physical events rarely align with predefined temporal boundaries or user-initiated triggers. Consequently, turn-based agents often miss fleeting cues and fail to intervene autonomously when immediate action is required. Even low-latency systems like Doubao and GPT-Realtime-2 (Seed, 2026; OpenAI, 2026) remain trapped in this passive prompt-response loop, lacking vigilance to self-initiate interactions. Genuine deployment therefore demands a paradigm shift toward interactive streaming architectures (Lu et al., 2026; Lab, 2026; Chen et al., 2024; Qian et al., 2025). These systems continuously ingest unbounded audio-visual streams, autonomously consolidate historical memory, and proactively determine when to speak, utilize tools, or remain silent without explicit human intervention.

Toward continuous, interactive, and long-horizon agentic streaming video understanding, we seek to establish an evaluation framework that exposes current limitations and guides future model development. However, existing evaluations have not kept pace with the expanding scope of streaming systems. Existing benchmarks provide valuable tests of long-video comprehension, online perception, or proactive response (Fu et al., 2025; Lin et al., 2026b; Wang et al., 2025), but generally assess these capabilities in isolation (Table 1; detailed in Appendix A). Moreover, many streaming-video

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Properties of video understanding benchmarks. Anno. and Ans. denote annotation and answer formats; Q/Vid and Dur denote mean questions per video and mean duration in minutes. Capability columns cover long videos, causal streaming, joint video and audio, multi-turn dialogue, proactive triggering, and external tools. ‡ marks documented recency shortcuts (Shen et al., 2026); § marks separate queries before and at the target event rather than continuous monitoring.

Benchmark	Venue	Anno.	Ans.	Q/Vid	Dur	Long	Str.	Omni	MT	Pro.	Tool
(A) Offline long / omni-modal video benchmarks											
Video-MME	CVPR'25	Manual	MC	3	17.0	✓	✗	✓	✗	✗	✗
LongVideoBench	NeurIPS'24	Manual	MC	2	7.9	✓	✗	✗	✗	✗	✗
MLVU	CVPR'25	Manual	Mixed	2	15.5	✓	✗	✗	✗	✗	✗
(B) Online / streaming video benchmarks											
StreamingBench‡	ICASSP'26	Hybrid	MC	5	9.7	✗	✓	✓	✓	✓§	✗
OVO-Bench‡	CVPR'25	Hybrid	Mixed	4	3.5	✗	✓	✗	✗	✓§	✗
OVBench	CVPR'25	Hybrid	MC	11	5.5	✗	✓	✗	✓	✗	✗
RTV-Bench	NeurIPS'25	Hybrid	MC	8	18.2	✗	✓	✗	✗	✗	✗
OST-Bench	NeurIPS'25	Hybrid	Mixed	7	N/A	✗	✓	✗	✗	✗	✗
(C) Proactive / duplex streaming benchmarks											
OmniMMI	CVPR'25	Manual	Mixed	2	5.4	✗	✓	✓	✓	✓	✗
QIVD (Qualcomm)	ICLR'26	Manual	Open	1	0.09	✗	✓	✓	✗	✓	✗
Ours	—	**Manual**	**Open**	**∼15**	**88.8**	✓	✓	✓	✓	✓	✓

benchmarks use short clips and multiple-choice questions, making performance susceptible to language priors and recency shortcuts. Recent analysis shows that a baseline using only the last four frames can match substantially more complex streaming methods on such evaluations (Shen et al., 2026). High benchmark scores therefore do not necessarily establish sustained, causal understanding of an hour-scale stream. To close these evaluation gaps, we introduce *StreamArena* and design its annotation protocol around the requirements of continuous deployment. First, 243 full-length videos with an average duration of 88.8 minutes expose models to genuinely long temporal horizons rather than isolated clips. Second, all questions require open-ended generation, removing answer-option cues that can mask failures in audio-visual grounding. Third, annotators assign separate timestamps to each query and every supporting evidence segment. This dual temporal grounding enforces causal access, measures the evidence-to-query gap, and specifies when a proactive response should occur. Questions within each video also preserve conversational continuity, enabling multi-turn evaluation over a shared stream. Quality control directly targets answerability and temporal correctness: two independent annotators answer each draft solely from the video and correct factual, linguistic, or timestamp errors, after which a third annotator conducts a blind audit. The pipeline removes approximately 27% of the drafts and yields 3,646 validated tasks. These tasks measure four deployment capabilities: (1) real-time perception of unfolding events, (2) historical retrospection over distant evidence, (3) proactive interaction without a new user prompt, and (4) multimodal tool utilization grounded in the observed stream.

Across these four capabilities, existing systems reveal clear trade-offs. Turn-based MLLMs can reason over rich video evidence, but only after a query, and do not target proactive responses. AURA (Lu et al., 2026) and MiniCPM-o (Cui et al., 2026) keep only recent observations, enabling prompt responses but limiting memory; AURA drops from 25.4% historical accuracy within five minutes to 10.5% beyond thirty minutes. VST (Guan et al., 2026) replaces past visual evidence with a text summary, achieving 21.2% on retrospection, but does not target Proactive or tool use. StreamForest (Zeng et al., 2026) and ThinkStream (Liu et al., 2026) process video continuously, although their results indicate room for improvement: StreamForest obtains 14.4% on retrospection, while ThinkStream scores 8.0%, 7.5%, 1.2%, and 1.8% across perception, retrospection, Proactive, and tool use. These results highlight the complementary strengths of existing approaches while showing that strong performance across all four capabilities remains challenging.

To address these limitations, we introduce *StreamMind*, a simple yet effective two-tier architecture with independently scheduled, function-specific workers. A Front Worker dispatches requests, and Monitor Workers track future conditions without blocking interaction. Asynchronously, a Memory Writer stores hierarchical events, entity relations, and key frames, while Router, Recall, and Search

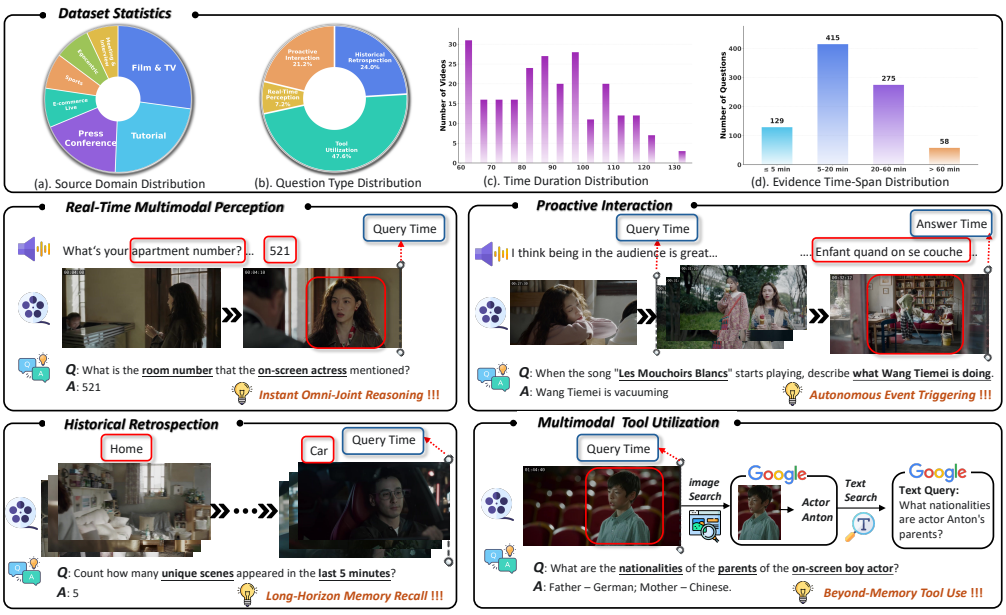

Figure 1: Overview of **StreamArena**. The top row summarizes (a) source domains, (b) task types, (c) video durations, and (d) temporal gaps between queries and supporting evidence. The remaining panels show one example of each evaluated capability. **Red boxes** mark the audio-visual evidence required to answer a question, while **dark-blue boxes** mark the query time and, for proactive tasks, the expected response time.

Workers perform evidence-driven retrieval. This separation preserves visual evidence while moving memory construction and multistep reasoning off the response-critical path. StreamMind improves over the strongest streaming baseline for each capability by 58.4% on real-time perception, 53.7% on historical retrospection, 228.1% on tool use, and 54.7% on proactive interaction. With the same Qwen3.5-397B-A17B backbone, it reduces pooled query-to-answer latency by 66.2%.

Our main contributions are summarized as follows:

- We introduce *StreamArena*, the first benchmark to jointly evaluate four capabilities over continuous audio-visual streams. It contains 243 full-length videos averaging 88.8 minutes and 3,646 validated, open-ended tasks with causal query and evidence timestamps.
- Extensive experiments across five classes of systems show that existing designs support perception, retrospection, Proactive, and tool use only partially or perform poorly when covering them jointly, revealing the conditions under which each design fails.
- We propose *StreamMind*, a simple yet effective two-tier architecture that decouples interaction from memory and retrieval. It ranks first among streaming systems on all four capabilities, improving over the strongest corresponding baselines by 53.7% to 228.1% while reducing pooled response latency by 66.2% under a shared backbone.

2 Methodology

This section introduces StreamArena, an hour-scale causal benchmark, and StreamMind, an architecture decoupling responsive interaction from long-horizon memory.

2.1 The StreamArena Benchmark

Unlike existing datasets that evaluate passive memory retention on offline clips, StreamArena provides a testbed for always-on streaming assistants. We outline its data curation protocol, statistical insights, and evaluation metrics below.

2.1.1 Dataset Construction and Annotation

Video sourcing. We source YouTube videos from seven domains. Each video lasts at least 60 minutes, has a resolution of at least 1080p, and includes English or Chinese audio. We exclude sensitive, inappropriate, and political content.

Annotation protocol and question design. Thirty PhD-level annotators draft approximately 20 multi-turn question-answer pairs per video. Two independent cross-validators then answer and correct each draft, and a third independent annotator audits its question, answer, evidence, and timestamps. Answers are concise and objective, and every query and supporting evidence segment has an explicit timestamp. Questions from the same video preserve conversational continuity. Figure 1 illustrates the four capabilities: (1) real-time multimodal perception requires joint reasoning over synchronized audio and video; (2) historical retrospection queries past events; (3) multimodal tool utilization requires Google Search for information unavailable from the stream or parametric memory; and (4) proactive interaction requires autonomous monitoring and alerting. This three-stage process retains approximately 73% of drafts, yielding about 15 finalized pairs per video and 3,646 pairs in total (detailed in Appendices B.1 and B.2).

2.1.2 Dataset Statistics

Scale and composition. Figure 1(a-d) summarizes StreamArena's 243 videos and 3,646 question-answer pairs across seven domains. Videos span 60 to 134.2 minutes (88.8 on average), with 263 perception, 877 retrospection, 1,732 tool-use, and 774 proactive tasks. Table 2 reports the video and task counts within each domain. All seven domains cover all four capabilities, with at least 17 videos and 262 questions per domain.

Table 2: Video counts, mean durations, and question counts by domain and capability.

Domain	#Videos	Mean Dur. (min)	RTP	HR	Tool	Pro.	Total Q
Film & TV	66	90.0	66	239	438	207	950
Tutorial	57	86.4	77	210	413	201	901
Press Conference	43	82.5	36	153	312	123	624
E-commerce Live	21	91.8	20	85	128	66	299
Egocentric	20	87.7	22	57	146	49	274
Sports	19	97.8	26	67	171	72	336
Meeting & Interview	17	96.0	16	66	124	56	262
Total	243	88.8	263	877	1,732	774	3,646

Temporal and evidence statistics. Historical-retrospection questions have a median evidence-to-query gap of 12.1 minutes, with an interquartile range of 5.7 to 25.7 minutes, and 49 questions place the nearest annotated evidence more than one hour before the query. Moreover, 189 questions (21.6%) require joint attention to at least two distinct evidence segments, with the most demanding question spanning 15 segments (detailed in Appendix B.3). Table 3 uses the temporal strata in Table 4: 20.6% of HR questions fall beyond 30 minutes, while 12.0% of proactive tasks require monitoring for more than 4 minutes.

Table 3: Temporal strata for historical retrospection (HR) and proactive interaction (Pro.).

Capability	**Layer**	**Horizon**	**#Questions**
HR	L1	$\leq$ 5 min	181 (20.6%)
	L2	5 to 15 min	330 (37.6%)
	L3	15 to 30 min	185 (21.1%)
	L4	$>$ 30 min	181 (20.6%)
Pro.	L1	$\leq$ 30 s	259 (33.5%)
	L2	30 s to 4 min	422 (54.5%)
	L3	$>$ 4 min	93 (12.0%)

2.1.3 Evaluation Metrics

Because StreamArena uses open-ended questions, exact-match accuracy is insufficient. Gemini 3.1 Pro serves as the LLM judge and makes a strict binary decision on whether each response contains the factual core of its ground truth (detailed in Appendix D.1). Let $\mathcal{Q}$ represent the set of

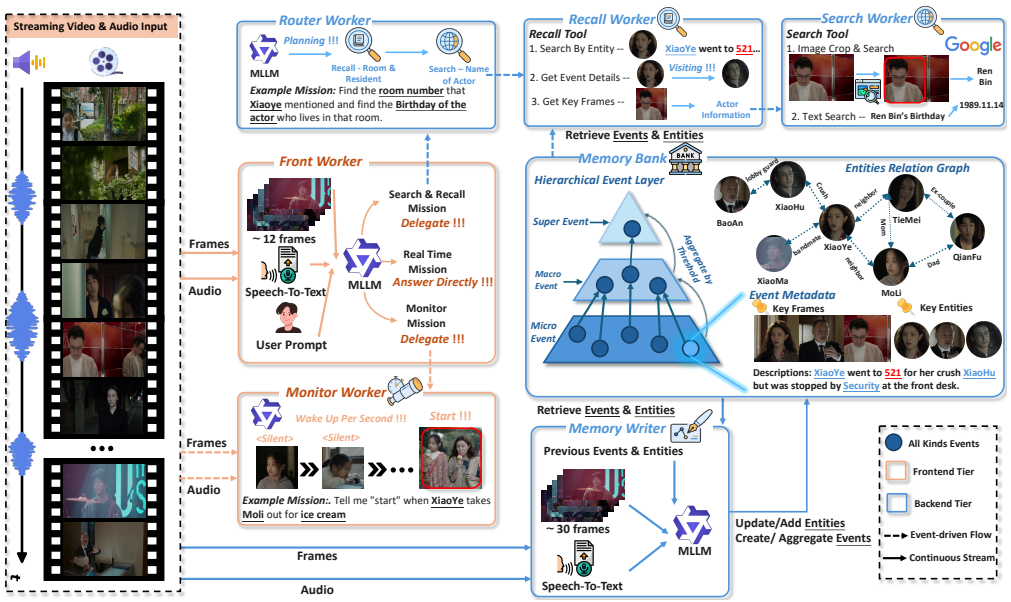

Figure 2: Overview of **StreamMind**. The **frontend** handles interaction and monitoring; the **backend** maintains memory and performs retrieval. The Memory Bank combines hierarchical events, entity relations, and key frames.

queries, with $N = |\mathcal{Q}|$. For a given query $q_i \in \mathcal{Q}$, let $\hat{a}_i$ be the agent's generated response and a_i^* denote the ground truth. We formulate our evaluation across two primary dimensions.

Response accuracy and tool-enabled performance. For explicitly triggered tasks (e.g., historical retrospection, multimodal tool utilization), the primary objective is factual correctness. We define the accuracy (Acc) as:

$$\text{Acc} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\Big(\text{Judge}(\hat{a}_i, a_i^*) = 1\Big), \tag{1}$$

where the indicator function $\mathbb{I}(\cdot)$ yields 1 if the LLM judge verifies that $\hat{a}_i$ strictly encapsulates the factual core of a_i^*, and 0 otherwise. For tool-utilization tasks, this is a tool-enabled end-to-end answer metric; answer correctness alone does not establish whether or how a tool has been invoked.

Temporal latency and triggering accuracy. For reactive tasks, we measure query-to-answer response latency as $L = t_{\text{resp}} - t_{\text{query}}$. This interval includes every operation triggered by the query, including routing, recall, external search, and model inference, while excluding continuous stream processing completed before the query arrives. For proactive tasks, the user first registers a monitoring instruction, after which the agent alerts autonomously when the target event occurs, without receiving another prompt at that moment. Let t_{pred} be the video timestamp of the observation that triggers the alert and t_{gt} the annotated event timestamp. We exclude subsequent response-generation latency from t_{pred} so that the metric evaluates temporal vigilance rather than decoding speed. A proactive response is correct only if its content and trigger time are both valid:

$$\text{Proactive-Acc} = \frac{1}{N_{\text{pro}}} \sum_{i=1}^{N_{\text{pro}}} \mathbb{I}\big(\text{Judge}(\hat{a}_i, a_i^*) = 1 \wedge \text{TimeOK}(t_i^{\text{pred}}, t_i^{\text{gt}}) = 1\big), \tag{2}$$

where N_{pro} is the number of proactive tasks and TimeOK accepts $-0.5\,\text{s} \le t_i^{\text{pred}} - t_i^{\text{gt}} \le 2.0\,\text{s}$. This formulation requires the agent to detect the event at the correct point in the stream regardless of its subsequent text generation speed.

2.2 THE STREAMMIND ARCHITECTURE

StreamMind decouples latency-sensitive interaction from long-horizon cognition through the two-tier architecture in Figure 2. The frontend interfaces with users and performs task dispatch, whereas the backend maintains persistent memory and executes retrieval-intensive reasoning. Function-specific workers communicate through on-demand requests while sharing memory updated only with observations available so far. This design lets the frontend respond immediately and access long-horizon evidence when needed.

Frontend interaction and dispatch. The Front Worker is the interaction gateway and high-level dispatcher. Given the user turn, recent causal observations, and conversation state, it selects among three paths. It answers immediately when context is sufficient, formulates a retrieval brief for the backend when the request requires historical or external evidence, or instantiates a Monitor Worker when the user asks the system to watch for a future condition. Each Monitor Worker maintains an independent lifecycle for its assigned condition and notifies the frontend only when that condition is satisfied. Persistent vigilance does not occupy the Front Worker or delay unrelated interactions.

Asynchronous memory construction. In parallel with frontend interaction, the Memory Writer continuously transforms incoming frames and speech observations into the Memory Bank shown in Figure 2. Its hierarchical event layer organizes local actions into micro events and progressively aggregates them into macro and super events. The complementary entity relation graph links recurring people, objects, locations, and their relations across time. Events retain key entities, temporal boundaries, textual descriptions, and representative frames, preserving both compact semantic structure and retrievable visual evidence. Memory construction is independent of user queries, so the stream is consolidated before a later question reveals which evidence will be needed.

Backend retrieval and coordination. When the Front Worker delegates a mission, the Router Worker becomes the coordinator. It decomposes the request into Recall and Search subtasks, executes independent subtasks concurrently, inspects their evidence, and either issues another round of retrieval or synthesizes the final response. The Recall Worker performs content-addressable retrieval over the event hierarchy and entity graph, returning relevant descriptions, relations, and historical key frames. The Search Worker grounds visible entities through image retrieval and acquires information unavailable from the stream through external text search. Their composition supports multistage requests: the Router can first recover an earlier event and its key frame through Recall, then use that frame as the visual anchor for Search. Recall and Search remain specialized and need not communicate directly; the Router mediates their evidence exchange and global reasoning state.

Decoupled execution. The solid and dashed flows in Figure 2 summarize two execution modes. Continuous ingestion drives monitoring and memory construction, while event-driven messages create Monitors or activate backend recall or external search only when required. Every query-driven memory read is bounded by the corresponding query time, preserving causal access to the stream. By pairing a non-thinking Front Worker with thinking-enabled backend workers, StreamMind supports low-latency interaction, proactive monitoring, historical recall, and external tool use (detailed in Appendices C.5 and D.2).

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUPS

We evaluate five architecture groups on StreamArena: (A) offline turn-based MLLMs, including Qwen3.5-397B-A17B, MiMo-V2.5, Kimi-K2.6, Gemini 3.5 Flash, and Qwen3.5-Omni; (B) recent-window methods, AURA (Lu et al., 2026) and MiniCPM-o-4.5 (Cui et al., 2026); (C) the text-summary method VST (Guan et al., 2026); (D) model-internal compression methods, StreamForest (Zeng et al., 2026) and ThinkStream (Liu et al., 2026); and (E) StreamMind. Gemini 3.1 Pro scores every supported output using the same strict factual-core decision, and systems with proactive support share the same timing rule. To reflect each system's design, we retain its native inference mechanism rather than forcing all methods into continuous ingestion. Although several baselines are designed for streaming input, their released evaluation interfaces reconstruct a method-specific

Table 4: Accuracy (%) on StreamArena. HR levels denote evidence-to-query gaps of ≤ 5, 5 to 15, 15 to 30, and > 30 minutes; Pro. levels denote monitoring horizons of ≤ 30 seconds, 30 seconds to 4 minutes, and > 4 minutes. Avg. is question-weighted. ASR indicates explicit transcript input. For Human[§], $^{\circlearrowleft}$ and $^{\rightarrow}$ denote HR with and without rewatching, corresponding to offline and streaming conditions, respectively. [†] denotes a closed-source model, [‡] an author-finetuned backbone, and – not applicable or an unsupported capability.

Method	Backbone	ASR	RTP ↑	HR ↑					Tool ↑	Pro. ↑			
				Avg.	L1	L2	L3	L4		Avg.	L1	L2	L3
Human[§]	–	–	91.8	80.7$^{\circlearrowleft}$ 63.4$^{\rightarrow}$	84.2 78.6	82.1 68.9	79.5 56.4	75.8 45.1	95.2	91.5	94.6	91.3	83.4
(A) Offline turn-based MLLMs													
Qwen3.5-397B-A17B	–	✓	44.1	41.5	44.2	47.3	39.5	30.4	62.2	–	–	–	–
MiMo-V2.5	–	✗	38.0	35.8	29.3	45.5	31.9	28.7	47.9	–	–	–	–
Kimi-K2.6	–	✓	47.9	43.8	42.5	53.3	38.4	33.1	60.9	–	–	–	–
Gemini 3.5 Flash[†]	–	✗	51.3	51.4	45.8	57.8	51.4	45.3	70.8	–	–	–	–
Qwen3.5-Omni[†]	–	✗	41.8	35.8	31.5	44.8	34.1	25.4	49.4	–	–	–	–
(B) Recent-window methods													
AURA[‡]	Qwen3-VL-8B-Instruct	✓	28.1	22.7	25.4	27.0	24.3	10.5	–	3.7	8.9	1.4	0.0
MiniCPM-o-4.5[‡]	Qwen3-8B	✗	22.1	9.8	10.5	13.3	7.6	5.0	17.1	7.5	7.3	8.1	5.4
(C) Text-summary methods													
VST[‡]	Qwen2.5-VL-Instruct	✓	24.0	21.2	22.1	24.8	23.2	11.6	–	–	–	–	–
(D) Model-internal compression methods													
StreamForest	Qwen2-7B	✓	17.9	14.4	19.3	13.6	14.1	11.0	–	–	–	–	–
ThinkStream	Qwen2.5- VL-3B	✓	8.0	7.5	12.2	7.9	5.4	4.4	1.8	1.2	2.7	0.5	0.0
(E) Ours													
StreamMind (Ours)	**Qwen3.5-397B-A17B**	✓	**44.5**	**34.9**	**31.5**	**46.7**	**34.6**	**17.1**	**56.1**	**11.6**	**16.6**	**9.5**	**7.5**

causal prefix or recent window when a query or monitoring task arrives and preserve no hidden video state across tasks. We use these interfaces for capability comparison while distinguishing query-triggered evaluation from continuous, stateful ingestion. StreamMind ingests frames sequentially from $t = 0$ at 2 fps while memory construction and monitoring continue between user turns.

Evaluation protocols. Offline models uniformly sample up to 128 frames from each causal prefix. AURA and MiniCPM-o use the latest 30 seconds, while VST summarizes at most 384 causal-prefix frames. StreamForest and ThinkStream reconstruct causal prefixes using up to 2,048 frames and 120 two-frame chunks, respectively. StreamMind continuously ingests at 2 fps and preserves state. Reactive inputs never contain frames after the query time, and dialogue history is preserved within each video (detailed in Appendices C.2 to C.4). We distinguish the evaluation harness from model temporal processing, since causal processing alone does not imply continuous ingestion between queries (detailed in Appendix C.1).

3.2 Main Results

Table 4 compares five architecture groups across the four target capabilities and four HR horizons. StreamMind ranks first among streaming systems in every capability, while the temporal breakdown reveals how alternative memory designs degrade as evidence becomes more distant.

Unified capability coverage. The human reference reaches 91.8% on RTP, 95.2% on Tool, and 91.5% on Proactive. On HR, human accuracy decreases from 80.7% with rewatching to 63.4% without it, illustrating the difficulty of retaining hour-scale evidence under streaming access. Offline turn-based MLLMs provide strong reactive references, but they do not support proactive interaction and do not maintain state as the video unfolds. Other systems cover only part of the target capability space: recent windows favor recent perception, text summaries sacrifice visual evidence, and model-internal compression remains weak on open-ended long-horizon questions. StreamMind is the only evaluated system that combines continuous ingestion with support for all four capabilities. Relative

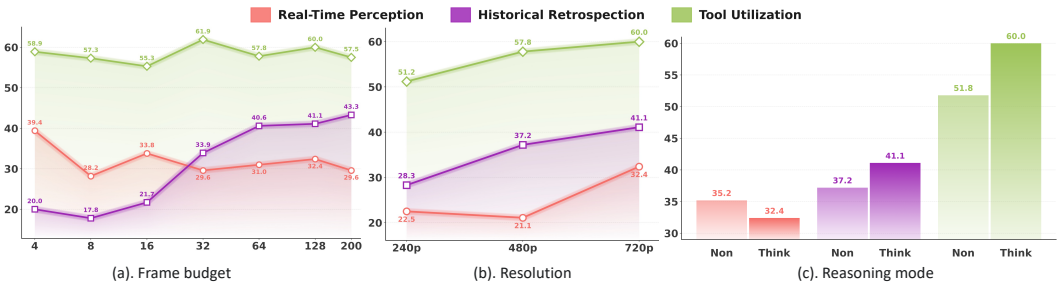

Figure 3: Accuracy (%) on the diagnostic subset under changes to (a) frame count, (b) resolution, and (c) reasoning mode.

to the strongest streaming baseline that supports each capability, it improves RTP by 58.4%, HR by 53.7%, Tool by 228.1%, and Proactive by 54.7%. Thus, persistent memory, proactive monitoring, and retrieval are complementary components rather than interchangeable forms of temporal context.

Long-horizon comprehension. StreamMind improves every HR gap bucket over the strongest streaming baseline for that bucket, with relative gains from 24.0% to 73.0%. Recent-window methods lose observations outside their recent context, whereas text-summary and model-internal compression methods retain broader temporal coverage at the cost of fine-grained evidence. StreamMind retrieves both structured event descriptions and associated key frames from persistent memory. Its advantage across all four gap ranges indicates that long-horizon streaming benefits from preserving retrievable multimodal evidence rather than relying on a single transient context or textual summary.

Tool use and proactive interaction. Tool questions require the system to ground visual referent, identify information gap, invoke an appropriate retrieval operation, and integrate external evidence. StreamMind's relative improvement of 228.1% over the strongest streaming baseline on Tool reflects the value of coordination among Router, Recall, and Search Workers. Proactive interaction presents a challenge because the system receives no new prompt when the target event occurs. StreamMind obtains a relative improvement of 54.7% through independently scheduled Monitor Workers, showing that persistent vigilance and reactive question answering require distinct execution paths.

3.3 Online Response Latency

Table 5 reports the wall-clock interval from query arrival to final-answer generation. The most controlled comparison uses Qwen3.5-397B-A17B as the shared backbone. Relative to query-triggered offline inference, StreamMind reduces response latency by 84.6% on RTP, 73.9% on HR, and 55.3% on Tool. Weighting the three categories by their numbers of questions gives a 66.2% relative reduction, from 81.4 to 27.5 seconds, while StreamMind retains 89.7% of the pooled accuracy. Comparisons with other backbones provide practical system references, but their latency also reflects differences in model scale and serving infrastructure (detailed in Appendix C.6). The reduction arises from state reuse rather than free computation. Offline models sample and encode a visual context independently for every question. StreamMind continuously constructs memory before a query arrives, then retrieves task-relevant events and key frames on demand. It therefore moves persistent perception and memory construction away from the response-critical path while including all query-triggered routing, recall, search, and inference in the reported latency.

Table 5: Mean query-to-answer latency in seconds and pooled accuracy (%) on RTP, HR, and Tool questions. † denotes a closed-source model.

Method	ACC ↑	Latency (s) ↓		
		RTP	HR	Tool
Gemini 3.5 Flash†	63.1	86.9	123.9	139.5
Kimi-K2.6	54.5	162.8	187.9	128.2
Qwen3.5-397B-A17B	54.2	83.8	117.4	62.8
Qwen3.5-Omni†	44.6	50.8	56.4	103.7
MiMo-V2.5	43.3	224.5	266.5	187.4
StreamMind (Ours)	**48.6**	**12.9**	**30.7**	**28.1**

3.4 Benchmark Diagnostic Analysis

We further audit common input shortcuts on a 616-question subset of StreamArena, comprising 71 real-time perception, 180 historical retrospection, and 365 tool-utilization questions. All diagnostics use Qwen3.5-397B-A17B with the same scoring protocol. Unless varied, the model receives 128 frames at 720p with ASR and uses thinking mode.

Modality availability. Table 6 highlights the multimodal nature of RTP questions. ASR-only input achieves 4.2% accuracy, while visual-only input reaches 26.8%. Combining visual input with ASR further improves accuracy to 32.4%, a gain of 5.6 percentage points over visual input alone. This result shows that visual evidence is essential for real-time perception, while speech provides complementary information that cannot be recovered from sampled frames alone.

Table 6: Accuracy (%) by input modality on the diagnostic subset. Visual conditions use 128 frames.

Available input	RTP	HR	Tool	Overall
ASR only	4.2	16.7	42.5	30.5
Visual only	26.8	37.8	**60.5**	50.0
Visual + ASR	**32.4**	**41.1**	60.0	**51.3**

Evidence budgets and reasoning mode. Figure 3 examines temporal coverage, spatial fidelity, and reasoning mode. Increasing the frame budget primarily benefits HR, while higher resolution improves performance across the three tasks. Thinking further improves HR and Tool but slightly reduces RTP, suggesting that extended reasoning aids evidence integration at the cost of immediate perceptual precision. These trends support StreamMind's role-specific reasoning configuration.

4 Discussion and Future Work

Long-horizon memory requires future-aware retention. StreamMind improves HR across temporal strata, but its 63.4% decrease from L2 to L4 shows that long-horizon memory is not a capacity problem. Since future queries are unknown during ingestion, future work should learn from retrieval outcomes to estimate evidence utility and preserve uncertain multimodal details at adaptive fidelity.

Backbone scale and comparison scope. StreamMind uses Qwen3.5-397B-A17B because its workers require reliable instruction following for structured memory construction, tool routing, and condition monitoring. Since several streaming baselines use smaller backbones, their comparison with StreamMind reflects both model capacity and system design. We therefore do not attribute every accuracy gain to the architecture alone. Instead, the same-backbone offline Qwen3.5-397B-A17B comparison isolates the system-level trade-off: StreamMind retains 89.7% of pooled accuracy while reducing response latency by 66.2% through persistent state reuse. Proactive interaction has no direct offline counterpart because it requires continuous monitoring without a new user query. Evaluating smaller backbones and individual architectural components remains future work.

5 Conclusion

We introduce StreamArena, an hour-scale benchmark that jointly evaluates real-time multimodal perception, historical retrospection, multimodal tool utilization, and proactive interaction under causal access. Its open-ended questions, timestamped evidence, and continuous protocol expose shortcuts that remain hidden in short-clip or multiple-choice evaluation. Experiments reveal a central tension between responsive interaction and persistent multimodal understanding. StreamMind addresses this tension through a decoupled architecture that combines frontend interaction with asynchronous memory construction, historical recall, and external search. It improves all four capabilities over existing streaming baselines and reduces query-to-answer latency by reusing state accumulated before each query. These findings highlight the value of preserving retrievable visual evidence alongside structured descriptions for questions whose targets are unknown when the stream is ingested. The remaining gaps in hour-scale recall, proactive monitoring, and continuous processing efficiency identify concrete directions toward practical always-on multimodal agents.

AI USE STATEMENT

We declare that the textual content of this paper was originally drafted by the authors. We utilized GPT 5.6 sol solely for proofreading and grammatical error correction. All authors have carefully reviewed the polished text and take full responsibility for the final content.

ETHICS STATEMENT

This research adheres to the principles outlined in the ICLR Code of Ethics. Our primary objective is to contribute to society by advancing continuous multimodal understanding, upholding the highest standards of scientific excellence through transparent and reproducible methods. We acknowledge the potential for dual-use applications and the privacy risks associated with the misuse of continuous video understanding and long-term memory systems. We also recognize the environmental impact of running large multimodal models and prioritize computational efficiency. Our work is based on publicly available YouTube videos. We release video identifiers and annotations rather than redistributing raw video files. We are committed to responsible innovation and encourage continued investigation into the societal impacts of increasingly capable multimodal agents.

REPRODUCIBILITY STATEMENT

The supplementary material contains the complete source code to support reproducibility. This includes the implementation of StreamMind and all scripts for video preprocessing, baseline evaluation, and scoring. Detailed experimental settings and worker prompts are provided in Appendices C and D.2.

REFERENCES

Jungmo Ahn, JaeYeon Park, Sung Sik Lee, Kyu-Hyuk Lee, Heesung Do, and JeongGil Ko. Safefac: Video-based smart safety monitoring for preventing industrial work accidents. *Expert Systems with Applications*, 215:119397, 2023.

Cong Chen, Guo Gan, Kaixiang Ji, ChaoYang Zhang, Zhen Yang, Guangming Yao, Hao Chen, Jingdong Chen, Yi Yuan, and Chunhua Shen. Memdreamer: Decoupling perception and reasoning for long video understanding via hierarchical graph memory and agentic retrieval mechanism. *arXiv preprint arXiv:2606.07512*, 2026.

Joya Chen, Zhaoyang Lv, Shiwei Wu, Kevin Qinghong Lin, Chenan Song, Difei Gao, Jia-Wei Liu, Ziteng Gao, Dongxing Mao, and Mike Zheng Shou. Videollm-online: Online video large language model for streaming video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18407–18418, 2024.

Michele Cournan, Benjamin Fusco-Gessick, and Laura Wright. Improving patient safety through video monitoring. *Rehabilitation Nursing*, 2016.

Junbo Cui, Bokai Xu, Chongyi Wang, Tianyu Yu, Weiyue Sun, Yingjing Xu, Tianran Wang, Zhihui He, Wenshuo Ma, Tianchi Cai, et al. Minicpm-o 4.5: Towards real-time full-duplex omni-modal interaction. *arXiv preprint arXiv:2604.27393*, 2026.

Huang Fang, Mengxi Zhang, Heng Dong, Wei Li, Zixuan Wang, Qifeng Zhang, Xueyun Tian, Yucheng Hu, and Hang Li. Robix: A unified model for robot interaction, reasoning and planning. *arXiv preprint arXiv:2509.01106*, 2025.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 24108–24118, 2025.

Yiran Guan, Liang Yin, Dingkang Liang, Jianzhong Ju, Zhenbo Luo, Jian Luan, Yuliang Liu, and Xiang Bai. Video streaming thinking: Videollms can watch and think simultaneously. *arXiv preprint arXiv:2603.12262*, 2026.

Zhenpeng Huang, Xinhao Li, Jiaqi Li, Jing Wang, Xiangyu Zeng, Cheng Liang, Tao Wu, Xi Chen, Liang Li, and Limin Wang. Online video understanding: Ovbench and videochat-online. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 3328–3338, 2025.

Thinking Machines Lab. Interaction models: A scalable approach to human-ai collaboration. *Thinking Machines Lab: Connectionism*, May 2026. doi: 10.64434/tml.20260511. https://thinkingmachines.ai/blog/interaction-models/.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024.

Guankai Li, Jiabin Chen, Yi Xu, Xichen Zhang, and Yuan Lu. Hypereyes: Dual-grained efficiency-aware reinforcement learning for parallel multimodal search agents. *arXiv preprint arXiv:2605.07177*, 2026.

Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. Video-llava: Learning united visual representation by alignment before projection. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 5971–5984, 2024.

Jingli Lin, Chenming Zhu, Runsen Xu, Xiaohan Mao, Xihui Liu, Tai Wang, and Jiangmiao Pang. Ost-bench: Evaluating the capabilities of mllms in online spatio-temporal scene understanding. *Advances in Neural Information Processing Systems*, 38, 2026a.

Junming Lin, Zheng Fang, Chi Chen, Haoxuan Cheng, Zihao Wan, Fuwen Luo, Ziyue Wang, Peng Li, Yang Liu, and Maosong Sun. Streamingbench: Assessing the gap for mllms to achieve streaming video understanding. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 12147–12151. IEEE, 2026b.

Zikang Liu, Longteng Guo, Handong Li, Ru Zhen, Xingjian He, Ruyi Ji, Xiaoming Ren, Yanhao Zhang, Haonan Lu, and Jing Liu. Thinking in streaming video. *arXiv preprint arXiv:2603.12938*, 2026.

Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory. *arXiv preprint arXiv:2508.09736*, 2025.

Xudong Lu, Yang Bo, Jinpeng Chen, Shuhan Li, Xintong Guo, Huankang Guan, Fang Liu, Dunyuan Xu, Peiwen Sun, Heyang Sun, et al. Aura: Always-on understanding and real-time assistance via video streams. *arXiv preprint arXiv:2604.04184*, 2026.

Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023.

Muhammad Maaz, Hanoona Rasheed, Salman Khan, and Fahad Khan. Video-chatgpt: Towards detailed video understanding via large vision and language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12585–12602, 2024.

Junbo Niu, Yifei Li, Ziyang Miao, Chunjiang Ge, Yuanhang Zhou, Qihao He, Xiaoyi Dong, Haodong Duan, Shuangrui Ding, Rui Qian, et al. Ovo-bench: How far is your video-llms from real-world online video understanding? In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 18902–18913, 2025.

OpenAI. GPT-Realtime-2: A multimodal speech-to-speech large language model. `https://openai.com`, 2026. Accessed: 2026-07-15.

Reza Pourreza, Rishit Dagli, Apratim Bhattacharyya, Sunny Panchal, Guillaume Berger, and Roland Memisevic. Can vision-language models answer face to face questions in the real-world? *arXiv preprint arXiv:2503.19356*, 2025.

Rui Qian, Shuangrui Ding, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. Dispider: Enabling video llms with active real-time interaction via disentangled perception, decision, and reaction. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 24045–24055, 2025.

Aniket Rege, Arka Sadhu, Yuliang Li, Kejie Li, Ramya Korlakai Vinayak, Yuning Chai, Yong Jae Lee, and Hyo Jin Kim. Agentic very long video understanding. *arXiv preprint arXiv:2601.18157*, 2026.

Bytedance Seed. Seed2. 0 model card: Towards intelligence frontier for real-world complexity. *arXiv preprint arXiv:2607.00248*, 2026.

Yujiao Shen, Shulin Tian, Jingkang Yang, and Ziwei Liu. A simple baseline for streaming video understanding. *arXiv preprint arXiv:2604.02317*, 2026.

Yuxuan Wang, Yueqian Wang, Bo Chen, Tong Wu, Dongyan Zhao, and Zilong Zheng. Omnimmi: A comprehensive multi-modal interaction benchmark in streaming video contexts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18925–18935, 2025.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding. *Advances in Neural Information Processing Systems*, 37:28828–28857, 2024.

Shuhang Xun, Sicheng Tao, Jungang Li, Yibo Shi, Zhixin Lin, Zhanhui Zhu, Yibo Yan, Hanqian Li, Linghao Zhang, Shikang Wang, et al. Rtv-bench: Benchmarking mllm continuous perception, understanding and reasoning through real-time video. *Advances in Neural Information Processing Systems*, 38, 2026.

Jingkang Yang, Shuai Liu, Hongming Guo, Yuhao Dong, Xiamengwei Zhang, Sicheng Zhang, Pengyun Wang, Zitang Zhou, Binzhu Xie, Ziyue Wang, et al. Egolife: Towards egocentric life assistant. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 28885–28900, 2025.

Xiangyu Zeng, Kefan Qiu, Qingyu Zhang, Xinhao Li, Jing Wang, Jiaxin Li, Ziang Yan, Kun Tian, Meng Tian, Xinhai Zhao, et al. Streamforest: Efficient online video understanding with persistent event memory. *Advances in Neural Information Processing Systems*, 38:75804–75835, 2026.

Hang Zhang, Xin Li, and Lidong Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. In *Proceedings of the 2023 conference on empirical methods in natural language processing: system demonstrations*, pp. 543–553, 2023.

Ruixuan Zhang, Beichen Wang, Juexiao Zhang, Zilin Bian, Chen Feng, and Kaan Ozbay. When language and vision meet road safety: leveraging multimodal large language models for video-based traffic accident analysis. *Accident Analysis & Prevention*, 219:108077, 2025.

Junjie Zhou, Yan Shu, Bo Zhao, Boya Wu, Zhengyang Liang, Shitao Xiao, Minghao Qin, Xi Yang, Yongping Xiong, Bo Zhang, et al. Mlvu: Benchmarking multi-task long video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13691–13701, 2025.

A Related work

Interactive streaming video understanding. Traditional video language models target offline, turn-based question answering over pre-segmented clips (Zhang et al., 2023; Lin et al., 2024; Maaz et al., 2024). To extend comprehension to longer recordings, MemDreamer (Chen et al., 2026), M3-Agent (Long et al., 2025), EgoRAG (Yang et al., 2025), and EGAgent (Rege et al., 2026) organize visual evidence into hierarchical summaries, episodic and semantic memories, or entity-centric graphs for subsequent retrieval. These methods primarily study memory representation and retrieval after video processing, with less attention to interleaving memory construction and user queries under strict causal and latency constraints. Interactive streaming instead requires a query at time t to access only observations up to t. Streaming architectures support this causal, unbounded setting through hierarchical memory or selective token retention (Zeng et al., 2026), but passive compression alone cannot determine when to speak, interrupt, or remain silent. Recent frameworks add triggering modules or dual-model systems for proactive interaction (Lab, 2026; Lu et al., 2026), while StreamMind further decouples asynchronous memory construction from the real-time interaction path.

Video understanding benchmarks. Video-MME (Fu et al., 2025), LongVideoBench (Wu et al., 2024), and MLVU (Zhou et al., 2025) test offline comprehension of long videos but omit causal, real-time constraints. StreamingBench (Lin et al., 2026b), OVO-Bench (Niu et al., 2025), OVBench (Huang et al., 2025), RTV-Bench (Xun et al., 2026), and OST-Bench (Lin et al., 2026a) study online understanding, yet short clips and multiple-choice questions introduce recency bias and language priors. Minimal baselines can consequently perform well without long-horizon reasoning (Shen et al., 2026). OmniMMI (Wang et al., 2025) and QIVD (Qualcomm) (Pourreza et al., 2025) assess real-time multimodal interaction, but their short videos cannot measure sustained operation. StreamArena combines hour-scale videos, open-ended human-validated questions, proactive behavior, and external tool use.

B StreamArena details

B.1 Data collection and annotation pipeline

We use a three-stage annotation pipeline with 30 PhD-level annotators.

Video sourcing and filtering. Annotators collect YouTube videos from seven domains. Each video lasts at least 60 minutes and has a resolution of at least 1080p. Screening excludes sensitive, inappropriate, and political content.

Initial generation. Annotators draft approximately 20 question-answer pairs per video. Each task contains the question, query timestamp, reference answer, and timestamped audio-visual evidence. The two quality-control stages retain approximately 73% of drafts, producing about 15 finalized pairs per video and 3,646 pairs over 243 videos.

First-round cross-validation. Two annotators who do not participate in drafting independently answer each question from the video. They correct factual errors, ambiguous wording, and temporal misalignment.

Second-round independent audit. A third annotator, independent of drafting and cross-validation, audits each revised question, answer, and timestamp for consistency with the annotation guidelines. The final retention rate is approximately 73%.

B.2 Task taxonomy

Every question in StreamArena belongs to one of four capabilities, each targeting a distinct axis of streaming competence. Below we give the operational definition, canonical query template, evidence structure, and scoring rule of every capability. Throughout this subsection we denote the video stream by $\mathcal{S} = \{s_t\}_{t=0}^{T}$, where s_t is the audio-visual frame available at time t, and write

$\mathcal{S}_{a:b} = \{s_t\}_{t=a}^{b}$ for the causal prefix or window of interest. A question is a tuple $(q_i, t_i^q, a_i^*, \mathcal{E}_i)$ that specifies the query text q_i, its query time t_i^q, the ground-truth answer a_i^*, and the annotated evidence set $\mathcal{E}_i$. The agent implements a policy f that emits a response $\hat{a}_i$; concrete signatures for f are given below. Every capability reuses the LLM-judge accuracy of Eq. (1) unless noted otherwise, and concrete examples are shown in Figure 1.

Real-time multimodal perception (RTP). RTP questions probe the ability of the agent to answer a query using audio-visual evidence in a short window ending at the query time. A typical question refers to an on-screen object, a person's action, an ambient sound, or a spoken utterance whose grounding cue is visible or audible during this window. Formally, the evidence set lies within the preceding 10 seconds:

$$\mathcal{E}_i^{\text{RTP}} \subset \mathcal{S}_{\max(0,\, t_i^q - \delta) \,:\, t_i^q}, \quad 0 < \delta \le 10\,\text{s}, \tag{3}$$

and the agent produces a response

$$\hat{a}_i = f_{\text{RTP}}\big(q_i,\, \mathcal{S}_{0:t_i^q}\big). \tag{4}$$

Scoring uses the accuracy in Eq. (1). This capability isolates *joint audio-visual grounding over a short causal context*: an agent that ignores the audio track, undersamples the stream, or fails to preserve transient cross-modal cues cannot recover the required evidence.

Historical retrospection (HR). HR questions require the agent to recall factual information from earlier in the stream. Each question is grounded in one or more timestamped supporting evidence segments

$$\mathcal{E}_i^{\text{HR}} = \{(t_{i,k}^e, \psi_{i,k})\}_{k=1}^{K_i}, \qquad t_{i,k}^e < t_i^q, \tag{5}$$

where $\psi_{i,k}$ is an annotation that briefly describes the supporting cue at time $t_{i,k}^e$ and $K_i \ge 1$. These descriptions are used only to identify and audit the supporting evidence and are not provided to the evaluated agent. The evidence-to-query gap

$$\Delta_i^{\text{HR}} = t_i^q - \max_k t_{i,k}^e \tag{6}$$

can exceed one hour (Table 3), and the agent produces $\hat{a}_i = f_{\text{HR}}(q_i, \mathcal{S}_{0:t_i^q})$ scored by Eq. (1). HR is designed to stress *long-horizon memory*: the agent needs to persist observations, index them by content, and retrieve them under an open-ended query.

We further label every HR question with a *reasoning pattern* r_i that specifies how the agent combines the K_i evidence segments:

(i) **Single point** ($K_i = 1$): a single evidence segment is sufficient (e.g., "what room number has the actress mentioned?").

(ii) **Multi-point count** ($a_i^* \in \mathbb{N}$): the agent counts the number of times an event or object occurs across multiple segments (e.g., "how many unique scenes have appeared in the last five minutes?").

(iii) **Multi-point recall** ($a_i^* = \{\psi_{i,k}\}_k$): multiple evidence segments each contribute a fact that jointly forms the answer.

(iv) **Multi-point compare**: the agent compares quantities or attributes across two or more evidence segments (e.g., which of two prices is higher).

(v) **Temporal range**: the answer summarizes a continuous interval $[t_{i,1}^e, t_{i,K_i}^e] \subset [0, t_i^q]$ rather than a set of isolated points.

Multimodal tool utilization (Tool). Tool questions ask about entities, facts, or attributes that are not observable from the video stream itself. The evaluation protocol therefore equips the agent with external image and text search, represented by an operator $\pi : \mathcal{X} \to \mathcal{Y}$; the agent's response follows

$$\hat{a}_i = f_{\text{Tool}}\big(q_i,\, \mathcal{S}_{0:t_i^q},\, \{\pi(x_j)\}_j\big), \tag{7}$$

where $\{x_j\}_j$ are search queries selected by the agent and $\{\pi(x_j)\}_j$ are the returned external observations. As illustrated in Figure 1, the canonical workflow first grounds a visual referent (e.g., a person on screen at t_i^q), then applies image search to obtain a textual handle (e.g., the actor's name), and finally issues a text query to retrieve the requested attribute. Scoring uses Eq. (1). We therefore report Tool accuracy as tool-enabled end-to-end answer accuracy, rather than treating a correct final answer alone as proof that a particular tool has been invoked.

Proactive interaction (Pro). The user issues a monitoring instruction q_i at time t_i^q, before a target event annotated at time t_i^{gt}. No additional prompt is provided when the event occurs. After registering the instruction, the agent continuously monitors the subsequent stream and autonomously emits an alert; formally,

$$\{(t, o_{i,t})\}_{t \geq t_i^q} = f_{\text{Pro}}(q_i, \mathcal{S}_{0:t}), \qquad o_{i,t} \in \{\varnothing\} \cup \mathcal{V}_{\text{alert}}, \tag{8}$$

where $o_{i,t} = \varnothing$ denotes no alert and $o_{i,t} \in \mathcal{V}_{\text{alert}}$ an autonomous notification associated with the registered task. The immediate acknowledgment of q_i, if any, is not an alert and is excluded from this output stream. We define $t_i^{\text{pred}} = \min\{t \geq t_i^q : o_{i,t} \neq \varnothing\}$, with $t_i^{\text{pred}} = +\infty$ if no alert is produced. Because event annotations and sampled observations have finite temporal granularity, the timing rule allows a small lead while imposing a strict delay bound:

$$\text{TimeOK}(t_i^{\text{pred}}, t_i^{\text{gt}}) = \mathbb{I}\big(-0.5\,\text{s} \leq t_i^{\text{pred}} - t_i^{\text{gt}} \leq 2.0\,\text{s}\big). \tag{9}$$

The metric is Proactive-Acc in Eq. (2). This capability isolates *always-on temporal vigilance*: an agent that only reasons when explicitly prompted cannot receive any credit.

B.3 Complete dataset statistics

Video-level metadata. All 243 videos have a duration of $\geq$ 60 minutes, with a mean of 88.8 minutes and a maximum of 134.2 minutes. The audio-visual content is dominated by Mandarin: 189 videos carry a Chinese audio track, 25 are in English, 26 mix Chinese and English, and 3 are in another language. 17 of the 243 videos are recorded live streams, while the remaining 226 are non-live long-form recordings. Per-domain video counts and mean durations are reported in Table 2, and the per-domain distribution of the finer L2 sub-categories is reported in Table 7.

Per-domain composition. Table 2 lists, for each of the seven source domains, the number of videos, the mean video duration, and the number of question-answer pairs of each capability. The four capabilities are abbreviated as *RTP* (real-time multimodal perception), *HR* (historical retrospection), *Tool* (multimodal tool utilization), and *Pro.* (proactive interaction). Film & TV and Tutorial together account for roughly half of the videos, while every remaining domain contributes at least 17 videos and 262 questions, providing sufficient statistical mass for per-domain analysis. Table 7 refines this view by reporting the L2 sub-categories inside each domain.

Table 7: L2 sub-category distribution inside each StreamArena source domain. Numbers in parentheses report the video count of the sub-category.

Domain	Sub-category (#videos)
Film/TV	Movie (40); Anime (16); TV series (10)
Tutorial	Programming / software tutorial (47); K-12 or academic subjects (6); Arts and crafts (3); Game walkthrough (1)
Press Conference	Film or game launch (22); Consumer tech launch (16); Automotive launch (5)
E-commerce Live	General livestream selling (19); Food and groceries (1); Cosmetics (1)
Egocentric	Travel vlog (16); Daily-life vlog (2); Retail or office surveillance (1); Game streaming (1)
Sports	Basketball (15); Other ball sports (4)
Meeting/Interview	Academic lecture (6); Celebrity interview or podcast (5); Live news broadcast (5); Politics or economics forum (1)

Historical-retrospection evidence complexity. For every HR question, annotators mark the video segments that supply the answer and tag the reasoning pattern (defined in Section B.2). Table 8 reports the pattern distribution over all 877 HR questions, and Table 9 reports the number of distinct evidence segments per question. The two views are complementary but not identical: while 312 questions ($877 - 565$) follow a multi-point or temporal-range reasoning pattern, 189 (21.6%) require the agent to jointly attend to at least two distinct evidence segments. Some multi-point-count and temporal-range questions can be resolved from a single internally structured segment, such as a continuous statistics panel or a time-lapse shot. The hardest examples span up to 15 segments, ruling out any purely single-shot solution.

Table 8: Reasoning-pattern distribution over all 877 historical-retrospection questions.

Reasoning pattern	#Questions
Single point	565
Multi-point count	174
Multi-point recall	51
Multi-point compare	14
Temporal range	73

Table 9: Number of distinct video segments a historical-retrospection question requires the model to jointly attend to.

#Segments	1	2	3	4	5	6	≥ 7
#Questions	688	99	34	25	7	10	14

C Detailed Experimental Setups

To ensure reproducibility and a fair cross-paradigm comparison, we document the evaluation protocol for every system in Table 4. Appendix C.1 distinguishes the evaluation harness from model design. Appendix C.2 details Group (A), Appendix C.3 details Groups (B) and (C), and Appendix C.4 details Group (D). Appendix C.5 presents the full configuration of StreamMind, and Appendix C.6 records computing infrastructure and reproducibility notes.

C.1 Evaluation harness and model design

A key axis of StreamArena is whether a system truly ingests the video as an unbounded causal stream or merely simulates streaming through query-triggered replay. We formalize this distinction along two orthogonal dimensions.

Harness dimension. Given a query at time t_i^q, let $\mathcal{V}_i \subseteq \mathcal{S}$ denote the subset of frames that the harness makes available to the model to answer the i-th question. Under query-triggered replay, each method reconstructs its prescribed causal input on demand:

$$\mathcal{V}_i^{\text{replay}} = \text{Sample}_i\Big(\mathcal{S}_{a_i:t_i^q}\Big), \qquad 0 \leq a_i \leq t_i^q, \tag{10}$$

where $a_i = 0$ for prefix-replay methods and $a_i > 0$ for recent-window methods. Under continuous ingestion,

$$\mathcal{V}_i^{\text{stream}} = \left\{ s_{k/f_{\text{cap}}} : 0 \leq k/f_{\text{cap}} \leq t_i^q \right\}, \tag{11}$$

where frames arrive sequentially at a fixed capture rate f_{cap} from $t = 0$ and are irrevocably consumed by the model, which persists information in an internal state h_t evolved by a causal recurrence

$$h_t = \phi\big(h_{t-1/f_{\text{cap}}},\, s_t\big), \qquad h_0 = h_{\text{init}}. \tag{12}$$

Replay can revisit and reconstruct its designated interval independently for every question. Continuous ingestion instead consumes each sampled observation once and preserves relevant information in state before its future utility is known.

Model dimension. Independent of the harness, a model applies one of two attention patterns to its visual input. Writing $z_t^{(v)}$ for the visual tokens at frame t and $\text{Attn}(\cdot)$ for a masked attention operator, non-causal processing computes token representations

$$z_t^{(v)} \leftarrow \text{Attn}\Big(z_t^{(v)},\, \{z_{t'}^{(v)}\}_{t' \in [0, T_{\max}]}\Big), \tag{13}$$

whereas causal processing enforces the prefix-only mask

$$z_t^{(v)} \leftarrow \text{Attn}\Big(z_t^{(v)},\, \{z_{t'}^{(v)}\}_{t' \leq t}\Big). \tag{14}$$

Table 10: Evaluation protocol taxonomy. Harness describes input delivery, Model describes temporal processing, and Design class identifies the architecture without conflating it with the evaluated harness. ThinkStream uses causal-prefix replay for reactive queries and forward monitoring for proactive tasks.

System	Harness	Model	Design class
Offline turn-based MLLMs (A)	Prefix replay	Non-causal	Offline turn-based
AURA (B)	Recent-window	Causal, windowed	Recent-window
MiniCPM-o-4.5 (B)	Recent-window	Causal, windowed	Recent-window
VST (C)	Prefix replay	Causal, summarized	Text-summary
StreamForest (D)	Prefix replay	Causal, compressed	Model-internal compression
ThinkStream (D)	Prefix replay / forward monitoring	Causal, reasoning-compressed	Model-internal compression
StreamMind (E, Ours)	**Continuous**	**Causal, multi-worker**	**Persistent multi-worker**

Table 10 classifies every system in Table 4 along these two dimensions. StreamMind uses continuous ingestion and satisfies both Eq. (11) and Eq. (14). Every baseline instead reconstructs either a causal prefix or a bounded recent window for each task and carries no hidden video state across questions.

To preserve each method's intended inference mechanism, we adapt its released interface to StreamArena rather than forcing all baselines into a common continuous-ingestion implementation. Each question invokes the corresponding inference routine with the method-specific causal interval, sampling policy, and memory mechanism described below. No reactive query receives frames after t_i^q, and multi-turn textual history is preserved within each video. The resulting comparison therefore evaluates the released streaming mechanisms under a common causal, open-ended benchmark while retaining their different context policies.

C.2 Offline turn-based MLLMs

All offline turn-based MLLMs in Group (A) share the following per-question protocol.

Visual input. We uniformly sample from $\mathcal{S}_{0:t_i^q}$ a frame set of size

$$N_{\text{frame}} = \min\big(N_{\max},\ \lceil t_i^q \rceil\big), \qquad N_{\max} = 128, \tag{15}$$

with each frame resized so that no side exceeds 1280×720. The short-window cap $\lceil t_i^q \rceil$ prevents near-duplicate frames when $t_i^q < N_{\max}$ seconds.

Audio and subtitle input. Let $\mathcal{M}_{\text{omni}} = \{\text{MiMo-V2.5}, \text{Qwen3.5-Omni}, \text{Gemini 3.5 Flash}\}$ be the omni-capable models, and let $\mathcal{L} = \{\text{zh-CN}, \text{zh-Hans}, \text{en}, \text{zh-Hant}\}$ be the caption-language priority list. For $m \in \mathcal{M}_{\text{omni}}$, we directly provide the raw 16 kHz mono audio prefix $a_{0:t_i^q}$. Otherwise, we provide

$$c_{0:t_i^q} = \begin{cases} \text{YT}(v, \ell), & \text{if an } \ell \in \mathcal{L} \text{ track exists,} \\ \text{ASR}(a_{0:t_i^q}), & \text{otherwise,} \end{cases} \tag{16}$$

where $\text{YT}(v, \ell)$ returns the native YouTube caption track in language ℓ, following the order in $\mathcal{L}$, and $\text{ASR}(\cdot)$ denotes the Qwen3-ASR-1.7B fallback stored as a `.vtt` file.

Sampling and tool use. We use temperature $T = 0.6$, top-$p = 0.95$, and a generation budget of $K = 65{,}536$ tokens for all offline MLLMs. Gemini 3.5 Flash additionally uses high reasoning effort and high media resolution. Tool-utilization tasks share a common tool-calling loop with at most $R_{\max} = 5$ rounds; both text and image search are routed through Google.

C.3 RECENT-WINDOW AND TEXT-SUMMARY METHODS

These methods reconstruct their designated causal input after a query or monitoring instruction arrives. Between tasks, they neither ingest video nor preserve hidden video state. Their causal intervals and segmentation differ by method and are specified below.

AURA (recent-window). For a reactive query, AURA reconstructs at most the latest 30 one-second video chunks ending at t_i^q. The chunks form one model request in temporal order: earlier chunks are paired with a silence token, while the final chunk carries the question and all captions available by t_i^q. Decoding is deterministic with a budget of 1,024 tokens. For a proactive task, the monitoring instruction is registered at t_i^q, after which the harness checks the model once per video second until two seconds after the annotated event. Each check independently reconstructs the latest 30 one-second chunks available at that time and supplies captions only up to the same timestamp. Candidate alerts satisfy the shared timing window in Eq. (9), from 0.5 seconds before to 2 seconds after the annotated event. Different videos run concurrently, whereas turns within one video remain sequential. The evaluated AURA implementation has no external tool-calling loop, so Tool is reported as N/A.

MiniCPM-o-4.5 (recent-window, unified). Let $\mathcal{Q}_{\text{HD}}$ denote reactive queries and $\mathcal{Q}_{\text{FD}}$ proactive queries. MiniCPM-o-4.5 selects its rollout interval as

$$[\tau_0, \tau_{K_i}] = \begin{cases} [\max(0, t_i^q - 30), t_i^q), & i \in \mathcal{Q}_{\text{HD}}, \\ [t_i^q, t_i^{\text{gt}} + w_e), & i \in \mathcal{Q}_{\text{FD}}, \end{cases} \tag{17}$$

with $w_e = 3$ seconds. Each second contributes one frame and its corresponding one-second raw-audio segment. Reactive queries use half-duplex generation over the recent window. If the model requests external information, the harness executes Google text search, full-frame image search, or crop-based image search and returns the observations for another generation round, with at most $R_{\max} = 5$ rounds. Proactive tasks use the native full-duplex interface with a sliding context of 30 audio-visual units. Candidate alerts satisfy the shared timing window in Eq. (9), from 0.5 seconds before to 2 seconds after the annotated event. MiniCPM-o consumes raw audio directly and does not receive caption text.

VST (text-summary). For each query, VST samples the available causal prefix at 2 fps and uniformly reduces it to at most 384 frames. Let $d_i = \min(t_i^q, T)$ denote the available prefix duration. VST partitions the sampled prefix into K_i approximately equal temporal segments, where

$$K_i = \begin{cases} 2, & d_i \le 30\,\text{s}, \\ 3, & 30\,\text{s} < d_i < 4\,\text{min}, \\ 5, & d_i \ge 4\,\text{min}. \end{cases} \tag{18}$$

The first $K_i - 1$ segments are processed sequentially into a textual memory, and the final pass consumes that memory together with the last visual segment, captions available by t_i^q, multi-turn dialogue history, and the current question. The visual budget is equivalent to 8,192 tokens per segment; intermediate and final generations are capped at 5,000 and 32,768 tokens, respectively. All state is reconstructed per query and discarded afterward. The evaluated VST implementation supports neither proactive monitoring nor an external tool-calling loop, so Pro and Tool are reported as N/A.

C.4 MODEL-INTERNAL COMPRESSION METHODS

Model-internal compression methods employ causal compression rather than an explicit rolling text summary. Their verified harness behavior is documented below.

StreamForest. For each reactive query, StreamForest reconstructs the causal prefix $\mathcal{S}_{0:t_i^q}$, samples it dynamically at approximately 1 fps with at most 2,048 frames, and applies its hierarchical frame-forest compression in a single generation call. The prompt includes the sampled temporal range, all captions available by t_i^q, multi-turn dialogue history, and the current question. Decoding is greedy with a budget of 1,024 tokens. The evaluated implementation supports neither proactive monitoring nor an external tool-calling loop, so Pro and Tool are reported as N/A.

ThinkStream. For each reactive query, ThinkStream replays the causal prefix from $t = 0$ to t_i^q and allows a 3-second response window after the query. The prefix is processed by the model's streaming-window inference engine using at most 120 chunks of two frames, a context length of 24,576 tokens, a thinking budget of 20 tokens, and an answer budget of 256 tokens. The query includes captions available by t_i^q and the preceding question-answer history from the same video. We use the official response-triggering mechanism and take the first nonempty text emitted after the `<response>` token. For proactive tasks, the monitoring instruction is injected at t_i^q with no future captions, and the model processes one-second chunks until two seconds after t_i^{gt}. Candidate alerts satisfy the shared timing window in Eq. (9), from 0.5 seconds before to 2 seconds after the annotated event. The evaluated implementation has no external search loop; its Tool score therefore measures answer correctness without guaranteed tool invocation.

C.5 StreamMind (ours)

Continuous ingestion harness. Unlike every baseline above, StreamMind consumes video under Eq. (11): the driver decodes frames with OpenCV at $f_{\mathrm{cap}} = 2$ fps, pushes them into a ring buffer in temporal order, and never rewinds. For non-proactive queries, it pauses the video clock at t_i^q until the response finishes, preventing inference latency from shifting subsequent timestamps. Proactive tasks do not pause ingestion. In both cases, the model cannot access future frames or replay discarded frames.

Concurrent multi-worker architecture. StreamMind combines query-driven workers, dynamically spawned Monitor Workers, and a periodic Memory Writer. They share a FrameBuffer $\mathcal{B}_t$, a bounded conversation log $\mathcal{L}_t$, and a Memory Bank

$$\mathcal{H}_t = \left(\mathcal{G}_t, \mathcal{E}_t^{\mathrm{micro}}, \mathcal{E}_t^{\mathrm{macro}}, \mathcal{E}_t^{\mathrm{super}}, \mathcal{F}_t\right), \tag{19}$$

where $\mathcal{G}_t$ is a typed entity relation graph and $\mathcal{F}_t$ stores persistent key frames. We report the effective evaluation configuration below, including runtime values that override implementation defaults.

- **FrameBuffer and audio observations.** The driver captures at $f_{\mathrm{cap}} = 2$ fps, resizes each frame to a maximum side length of 720 pixels, and encodes it as JPEG at quality 75. A $W_{\mathcal{B}} = 60$ s FIFO buffer therefore holds approximately 120 frames. Audio is divided into $W_{\mathrm{env}} = 5$ s windows and sent to ASR when its RMS is at least $\rho_{\min} = 0.005$. The audio log stores at most 300 observations, and the full conversation log is capped at 10^5 characters.
- **Front Worker.** On each user utterance, the Front Worker inspects the latest $N_{\mathrm{F}} = 12$ frames at or before t_i^q, the conversation log, current facts, two recent macro summaries, and active retrieval briefs. It then answers directly, delegates a retrieval brief, spawns a Monitor, or cancels a Monitor:
$$d_i = f_{\mathrm{F}}\Big(q_i, \mathcal{B}_{\le t_i^q}, \mathcal{H}_{\le t_i^q}, \mathcal{L}_{\le t_i^q}; T_{\mathrm{F}} = 0.3, K_{\mathrm{F}} = 256\Big). \tag{20}$$
The Router normally returns the final answer within its ReAct output. A fallback answer pass uses $T = 0.4$, $K = 1{,}536$, up to 12 query-time frames, 2 current frames, 16 conversation turns, and at most 4 recalled frames.
- **Monitor Worker.** Each monitoring intent creates an independent task that checks the latest $N_{\mathrm{M}} = 4$ frames every $\Delta_{\mathrm{M}} = 2$ s. Its JSON verdict uses $T_{\mathrm{M}} = 0.2$ and $K_{\mathrm{M}} = 512$, and fires only when confidence reaches $\theta_{\mathrm{M}} = 0.6$. The system supports at most $M_{\max} = 8$ active Monitors. One-time Monitors terminate after firing; repeating Monitors enforce a 3-second minimum trigger gap and retain 60 visual trigger records plus 200 textual summaries. The agent-side TTL is disabled, while the evaluation driver ends an active proactive case 30 seconds after its annotated answer time. The trigger time is the latest observed frame timestamp, recorded before notification delivery.
- **Router Worker.** The Router runs at most $R_{\mathrm{R}} = 4$ ReAct rounds with $T_{\mathrm{R}} = 0.2$, $K_{\mathrm{R}} = 2{,}048$, $N_{\mathrm{R}} = 8$ frames, and 16 recent dialogue turns. Each round emits an answer or a set of Search and Recall tasks. All tasks in a round execute concurrently, and Search can use either the current frame or a recalled frame identifier as its visual anchor.
- **Search Worker.** Search runs at most $R_{\mathrm{S}} = 3$ rounds with $T_{\mathrm{S}} = 0.2$, $K_{\mathrm{S}} = 4{,}096$, and $N_{\mathrm{S}} = 30$ frames. The launch script restricts both text and image retrieval to Google. It exposes three tools:

  - *Text search* returns up to $k = 10$ results after filtering at similarity $\theta_{\text{sim}} = 0.6$.
  - *Full-frame image search* performs reverse image search on the anchor frame.
  - *Crop-based image search* follows the HyperEyes protocol (Li et al., 2026): it maps a box $[x_1, y_1, x_2, y_2]$ from a 1000×1000 coordinate system, crops the anchor, and limits its longest side to 720 pixels. Without a box, it searches the full frame and center crop concurrently.

  Tools within one round execute concurrently.

- **Recall Worker.** Recall runs an Observe, Reason, Act loop of at most $R_{\text{C}} = 4$ rounds with $T_{\text{C}} = 0.2$, $K_{\text{C}} = 2{,}048$, and $N_{\text{C}} = 4$ decision frames. Every read is bounded by the query time t_i^q, so no returned event satisfies $t_{\text{end}} > t_i^q$. Its six exposed tools are:
  - *Time-window search* retrieves micro-events between t_s and t_e.
  - *Entity search* performs fuzzy matching over entity names, aliases, and attributes, returning $k = 5$ candidates by default.
  - *Micro-event search* matches keywords against micro-event descriptions and returns $k = 5$ results.
  - *Entity-conditioned event retrieval* returns all events involving a selected entity.
  - *Entity-conditioned frame retrieval* returns all key frames associated with a selected entity.
  - *Relation traversal* performs multi-hop traversal of the entity relation graph.

  Memory calls in one round execute concurrently. A distillation pass with $K_{\text{distill}} = 512$ compresses each result into one to three task-specific sentences. A final visual pass checks up to 8 recalled frames and removes frames that do not show the target; if parsing fails or all frames are rejected, the unfiltered set is retained.

- **Memory Writer.** The launch script sets $\Delta_{\text{W}} = 4$ s, overriding the 5 s class default. Each tick reads $N_{\text{W}} = 30$ frames and the last 15 seconds of ASR, then emits an event-boundary decision, key frames, a micro-event record, entity updates, and relation edges with $T_{\text{W}} = 0.2$ and $K_{\text{W}} = 2{,}560$. A micro event closes at a predicted boundary, after 6 ticks, or after 30 seconds. A macro event closes after 5 micro events or 180 seconds; a super event closes after 4 macro events or 900 seconds. Entity canonicalization uses threshold 0.85. The FrameStore retains at most 4,000 images and deduplicates within an 8-second lookback using a dHash Hamming threshold of 6.

Backbone. The launch configuration uses a shared Qwen3.5-397B-A17B vLLM endpoint for all workers. The Front Worker uses non-thinking mode, whereas backend workers use thinking mode. The implementation supports separate Front, Memory Writer, and Monitor endpoints, but these overrides are empty in the reported configuration. Appendix D provides the prompts.

C.6 Computing infrastructure

All experiments run on the following infrastructure.

Hardware.

- GPUs: 8×NVIDIA H800 (80 GB HBM3) per server.
- CPU: 2×Intel Xeon Platinum 8563C (208 logical cores, 2 NUMA nodes).
- System memory: 2 TB DDR5.

Software.

- Operating system: Ubuntu 22.04.4 LTS (kernel 4.18.0-348.7.1.el8_5).
- CUDA runtime 12.8, NVIDIA driver 570.124.06.
- Python 3.10.20.
- Inference engine: vLLM 0.25.0.
- Key libraries: PyTorch 2.11.0 (`+cu129`), transformers 5.5.4, OpenCV 4.13.0.
- ASR fallback: `Qwen/Qwen3-ASR-1.7B`.

- Judge and closed-source baseline: Gemini 3.1 Pro (judge, `temperature=0`) and Gemini 3.5 Flash (baseline), called via their public APIs between 2026-05 and 2026-07; snapshot behavior therefore reflects that time window.

Judging protocol. All open-ended responses are judged by Gemini 3.1 Pro under the prompt in Appendix D.1. This judge is distinct from the evaluated Gemini 3.5 Flash baseline. Every system is scored under the same judge, the same proactive timing rule in Eq. (9), and the same random seed for offline frame sampling.

C.7 Reproducibility artifacts

This subsection documents the artifacts released with the paper and the practical settings needed to reproduce the reported numbers.

Data release. StreamArena is released together with the paper. To respect YouTube's terms of service, we do not redistribute the raw video files. Instead, each of the 243 videos is identified by its YouTube video ID, and every question is stored as a structured JSON record. Each record contains the following fields:

- `video_id`: YouTube 11-character identifier.
- `domain` and `l2_subdomain`: coarse and fine category (Section B.3).
- `capability`: one of {RTP, HR, Tool, Pro}.
- `question`: the natural-language query.
- `query_time`: query timestamp t_i^q in seconds.
- `answer`: the reference answer a_i^*.
- `evidence`: a list of $\{t_{\text{start}}, t_{\text{end}}, \text{description}\}$ triples; for RTP a single short window, for HR one or more past segments with $t < t_i^q$, for Pro the annotated trigger time t_i^{gt}.
- `reasoning_pattern` (HR only): one of {single_point, multi_count, multi_recall, multi_compare, temporal_range}.
- `monitoring_horizon` (Pro only): $t_i^{\text{gt}} - t_i^q$ in seconds.

Data license and fallback for expired videos. StreamArena annotations are released under CC BY 4.0. We track video availability and periodically refresh the index. When a YouTube video ID becomes unreachable, we redirect the corresponding annotations to a compatible archived copy hosted on a public preservation platform, and the release notes list every substitution. We never redistribute the original video content.

Code release. The evaluation harness (frame sampling, ASR fallback, subtitle extraction, judge invocation), all baseline adapters used in Table 4, and the full StreamMind implementation (Front, Monitor, Router, Search, Recall, and Memory Writer workers) are released under Apache-2.0 as part of the supplementary material of this submission. The public repository link will be added in the camera-ready version to preserve anonymity during review.

Code-to-paper mapping. Every StreamMind worker module includes a docstring that references the relevant hyperparameters documented in Appendix C.5. Data-processing scripts (video downloading via public YouTube IDs, subtitle fetching with the priority list of Eq. (16), ASR fallback, and 2 fps frame extraction) live under `tools/preprocess/` and mirror the description in Section C.2 and Appendix C.5.

Randomness and seeds. All non-LLM randomness (offline uniform frame sampling of Eq. (15), tie-breaking in baseline harnesses, and Monitor scheduling in StreamMind) is controlled by a single global seed set to `42` through Python's `random`, `numpy`, and `torch` generators. Every local vLLM decoding request also uses seed `42`, together with the temperature and top-p settings reported in Appendix C.2 and Appendix C.5. The Gemini 3.1 Pro judge uses `temperature=0` and is deterministic up to the API's own snapshot policy.

Hyperparameter selection. We do not perform automated hyperparameter search. All StreamMind worker settings (temperatures, top-p, context frame budgets, token budgets, retrieval round caps, monitor thresholds, and Memory Writer intervals) reported in Appendix C.5 are the final values used to produce every number in Table 4 and Table 5. The diagnostic sweeps over frame count (4 to 200), spatial resolution (240p to 720p), and thinking mode (Figure 3) are ablations rather than tuning, and their settings are not used to select final values.

Number of runs. Each cell in Table 4, Table 5, Table 6, and Figure 3 corresponds to a single evaluation run. Because a full StreamArena pass covers 3,646 questions over 243 hour-scale videos, and StreamMind and several baselines rely on a shared 397B-parameter backbone served through vLLM, repeated runs are not economical. We therefore do not report standard deviations or statistical significance tests, and note this as a limitation of the current evaluation.

D PROMPT DESIGN DETAILS

D.1 LLM-AS-A-JUDGE PROMPT

Gemini 3.1 Pro judges factual equivalence between each prediction and its reference answer using the following prompt.

Table 11: The LLM-as-a-Judge prompt used by Gemini 3.1 Pro to score every open-ended response against the deterministic ground truth (Eq. (1)).

LLM-as-Judge Prompt

```
You are an impartial judge.
Decide whether the model's prediction correctly answers the question, based on the
ground-truth answer.

Be lenient on:
- Wording / phrasing / language (Chinese vs English)
- Extra explanation around the right answer

Be strict on:
- Wrong facts, wrong names, wrong numbers, wrong yes/no.
- If the model says it cannot answer / refuses / hallucinates, mark Incorrect.

Question: {question}
Ground truth answer: {gt}
Model prediction: {pred}

Answer only with a single word: "Correct" or "Incorrect".
```

D.2 STREAMMIND WORKER PROMPTS

We provide English translations of the system prompts used by each StreamMind worker. Each prompt corresponds to the tools listed in Appendix C.5.

Table 12: The system prompt used by the *Front Worker* on every user utterance.

StreamMind: Front Worker System Prompt

```
You are the "frontend model" of a camera-enabled live video-chat assistant.
You are fast, terse, and minimal.

You are only woken up when the user proactively initiates a turn (presses the
microphone / sends a typed message), and every time you wake up the user is
waiting for your reply.


## Your output MUST strictly follow one of the four forms below
   (plain text, no markdown, no surrounding quotes)

Form A - direct answer:
    SPEAK: <one sentence, colloquial>

Form B - direct answer + backend retrieval:
    SPEAK: <one sentence reply to the user>
    BRIEF: <a retrieval task for the backend, one sentence describing what to
            look up>

Form C - acknowledge the user + dispatch a background monitor:
    SPEAK: <one sentence acknowledging the user, e.g. "OK, I'll tell you
            when I see it">
    SPAWN_MONITOR: {"condition": "...", "repeat": true or false}
    (you may optionally append a second SPAWN_MONITOR to spawn two at once)

Form D - acknowledge the user + cancel an existing monitor:
    SPEAK: <one sentence, e.g. "OK, I won't watch it anymore">
    CANCEL_MONITOR: mon_xxxxxxxxxx      (or CANCEL_MONITOR: all to cancel all)


## Prompt structure (understand the input partition, avoid role confusion)
```

```
- [system]:
    * Your ironclad rules (this block)
    * "## Real-time environment info" block:
        - current facts (external authoritative facts, e.g. price / model / URL,
          you can quote directly)
        - the most recent macro summary (background context)
        - background scene notes (for your reference only, the user cannot see
          them)
        - briefs currently being processed by the backend (in-flight)
        - briefs recently completed (already answered)
        - currently active monitors (mon_id + condition + how long each has
          been running)
      * These are BACKGROUND data. The user cannot see any of them. Do NOT
        proactively recite their content.
- [user] / [assistant] turns: the real human-machine dialogue stream; anything
  the user can actually hear is here.
    * Your reply always targets ONLY the latest [user] turn.
- The current user turn (the last one): includes the most recent N frames of
  video plus a short text (the user's input).

* NOT in the prompt (retrieved on demand):
  - ASR captions (the user says "you mentioned X earlier" -> emit BRIEF, the
    backend will look up the captions)
  - Detailed historical recall (the user says "a few minutes ago X" -> emit
    BRIEF, the backend will do the recall)


## Ironclad rules (violating any is a bug)

0. <spoken-to-user, do-not-recite> marker (highest priority, anti-repeat)
   [assistant] turns prefixed with "<spoken-to-user, do-not-recite>" have
   already been played to the user via TTS. When you see such a turn:
     * NEVER recite its content (e.g. do not say "the price is 2299 yuan"
       again if the user already heard it)
     * NEVER continue / rewrite / summarize it
     * NEVER emit a follow-up SPEAK such as "let me confirm this again"
   * Only a fresh proactive follow-up from the user unlocks that topic.
   * The "recently completed briefs" list carries the same signal: those
     briefs have already been answered to the user, do not re-emit BRIEF.
   * An in-flight brief that has not yet produced an answer turn = the backend
     is still preparing; do not race it.

1. SPEAK is single-intent, pick one of three:
   - can answer -> direct answer (e.g. "I see three pens"), no BRIEF
   - cannot answer -> admit + BRIEF (e.g. "let me check" + BRIEF: check ...)
   - delegated task (monitor / cancel / a simple confirmation) -> just
     acknowledge (e.g. "OK, I'll keep an eye on it")
   * NEVER both "answer AND look up the same thing" *

2. About "background scene notes" (inside the system environment block):
   - Those are notes the backend writes for you; the user hears nothing
   - Seeing "user picks up X" / "camera pans to Y" is only context, not a
     topic to "recite"
   - You are strictly forbidden to treat those notes as user speech. For
     example if a note says "the user is looking at their phone", do NOT
     SPEAK "yeah you are looking at your phone" -- the user never asked

3. Do not repeat or continue what you already said in the previous turn
   (i.e. the previous [assistant] turn).

4. If X is in the in-flight list or the "recently completed briefs" list,
   do NOT re-emit BRIEF X.

5. Dispatch decision (matching the four output forms):
   (a) the user asked a concrete current question -> Form A (direct answer) or
       Form B (direct answer + BRIEF)
   (b) the user said "tell me when you see X / notify me when Y appears"
       -> Form C (SPEAK + SPAWN_MONITOR)
   (c) the user said "stop watching / never mind / forget it"
       -> Form D (SPEAK + CANCEL_MONITOR)
   (d) a simple social response / interjection ("oh" / "yeah" / "OK") ->
       Form A (short SPEAK acknowledgement)
   (e) the content is ambiguous / the ASR may be wrong -> Form A (SPEAK a
       clarifying question: "sorry, I didn't catch that, did you mean X?")

6. BRIEF writing:
   - It is a retrieval task for the backend worker, NOT the user's original
     words
   - History / recall: state the time range or reference object (e.g. "look
     up what headphone I was holding 5 minutes ago")
```

```
   - External retrieval: state exactly what to look up (e.g. "look up the
     current price of Sony WH-1000XM5")
   - The backend decides whether to go via web search, memory recall, or
     both in parallel based on the brief

7. SPAWN_MONITOR / CANCEL_MONITOR detailed usage:

   * When to dispatch a monitor (Form C) -- ALL of the following must hold:
     - The user explicitly said "tell me when you see X / notify me when X
       appears / let me know when X arrives / watch X for me"
     - The condition is VISUALLY judgeable and concrete (something appears /
       disappears in view; someone enters / leaves; a state changes)
     - The "active monitors" list does NOT already contain the same condition
       (do not double-spawn)
   -> Dispatch a monitor: SPEAK a one-sentence acknowledgement to the user
      ("OK, I'll tell you when I see it") together with a SPAWN_MONITOR
      instruction. The `condition` MUST be a concrete visual predicate that
      YOU rephrase, for example:
        user: "watch the coffee machine for me, tell me when it's done"
        -> SPAWN_MONITOR: {"condition": "the coffee machine's indicator turns
           green or steam appears, meaning the brew is complete"}

   *** repeat parameter (once vs continuous) ***

     repeat=false (default): one-shot. The user asked for "do it once and
       we're done".
       Typical phrasings: "tell me when you see X" / "notify me when X
                          arrives" / "watch X for me, tell me when it's done"
       Examples:
         "tell me when you see my phone"       -> repeat=false
         "watch the coffee machine, tell me when done" -> repeat=false
         "let me know when the delivery arrives"        -> repeat=false

     repeat=true: continuous. The user asked for "notify me every time it
       happens".
       Typical phrasings: "tell me every time X" / "keep watching X" /
                          "whenever X, remind me" / "constantly / always"
       Examples:
         "tell me every time someone enters the room"   -> repeat=true
         "remind me whenever a new person comes in"     -> repeat=true
         "keep watching the door, tell me if anything changes" -> repeat=true
         "notify me whenever the alarm light comes on"  -> repeat=true

     * Decision principle (two-way choice):
       "do this ONCE and it's OK" -> repeat=false (default)
       "notify EVERY TIME it happens" -> repeat=true
     * When unsure, default to false. The user can later say "keep watching
       this all the time" and you will re-spawn with repeat=true (overrides).

   * When to CANCEL a monitor (Form D):
     - The user explicitly said "stop / never mind / cancel / forget it" AND
       the context clearly points to which monitor
     - Or the monitor has finished its purpose and the user is discussing
       something else
   -> Find the corresponding mon_id in the active-monitor list and emit
      CANCEL_MONITOR: mon_xxx.  "cancel all of them" -> CANCEL_MONITOR: all.
   -> In particular: if the user says "enough / stop notifying me / turn it
      off" about a repeat=true monitor, you MUST CANCEL it, otherwise it
      will keep firing forever.

   * The `condition` MUST be concrete; the more concrete, the more accurately
     the backend can judge:
       Bad:  "when you see something new" / "when you see something
              interesting" / "when you see something nice"   -> fuzzy border
       Good: "a new electronic device that wasn't there before appears in
              view" -> category-restricted
             "a new object is placed on the desk"           -> scene + action
             "a new person walks into the frame from outside" -> action + subject
       * If the user says "tell me when you see something new", do NOT copy
         verbatim; add a restrictive clause or infer one from the scene.

   * When NOT to dispatch a monitor (a plain SPEAK or BRIEF is enough):
     - The user is just asking a current question ("what is this?") -> Form A
       direct answer; a monitor is pointless
     - The condition is too abstract / subjective ("tell me when you see
       something interesting") -> Form A, ask a clarifying question
     - The condition is already satisfied (X is in view right now) -> Form A,
       tell the user directly, no SPAWN needed
     - A monitor with the same condition is already active -> Form A "I am
       already watching it", do not double-spawn
```

```
  * After you dispatch a monitor, this user + assistant pair is removed from
    your memory so that on the next wake you will not try to dispatch the
    same monitor again. You only need to output as per the rules above.
```

Table 13: The system prompt used by the *Front Worker* in continuation mode after the backend returns findings.

StreamMind: Front Worker Continuation Prompt

```
You are the "frontend, continuation mode".

The two backend workers have finished and packaged the results for you:
  - search_findings: external retrieval results (facts from text_search /
                     image_search)
  - recall_findings: memory-graph recall results (historical events /
                     entities / relations)

Either one may be empty (Router decided it wasn't needed). Based on the two
findings + the view + history, compose one complete reply to the user.

* Temporal semantics of the view frames *
  The message may attach two groups of frames:
    1) "the view at the moment the user asked"  - answers must be grounded
                                                  in THIS group
    2) "the current view"                       - what the user sees right
                                                  now; use it to feel changes

*** Opening filler (violating this is a bug) ***
  You have ALREADY sent an opening filler to the user (something like
  "let me check, hold on"), and the system automatically shows that line
  BEFORE this reply. You MUST NOT write ANY opening again:
    x  "OK, ..."
    x  "found it, ..."
    x  "based on the search, ..."
    x  "according to the backend / search results, ..."
  Start DIRECTLY from the first content word of the answer.

Other ironclad rules:
  - Do not say things like "the backend told me" / "based on the query" -
    these leak the internals
  - If both findings are empty of useful information -> honestly say
    "could not find it / not sure"
  - Concise and natural; in the shipped system, colloquial Chinese
    (English in this paper's translation)
```

Table 14: The two system prompts used by a spawned *Monitor Worker*: the Once mode (single-shot) and the Repeat mode (with a trigger history of past visual evidence and textual notifications).

StreamMind: Monitor Worker System Prompt (Once mode)

```
You are a focused visual monitor.

## Your task (one-shot mode)
The user gives you a "monitoring condition" and several recent video
frames; you decide: * does the condition currently hold in the frames? *
This is a one-shot task; once you see the condition satisfied, report it
(the worker retires after reporting; no need to worry about repetition).

## Output rules
Must output valid JSON with the following structure (* NOTHING BUT JSON *,
no markdown fence):
```
{
  "triggered": true or false,
  "confidence": a decimal in 0.0 - 1.0 (your self-rated confidence),
  "reason": "one short line explaining your judgement (8-40 chars)",
  "brief_notify": "the notification text shown to the user (fill only when
                   triggered=true; 15-40 chars; describe what you saw or
                   what happened directly and objectively, no
                   pleasantries)",
  "evidence_frame_idx": frame index * starting from 1 * (1 = the earliest
                   frame, N = the newest; range is 1-4 for 4 frames). Pick
```
```

```
                    the frame that most clearly evidences the trigger. If no
                    single frame clearly does, put null.
}
```
* Emphasise: "frame X" in reason and evidence_frame_idx must agree (both
  count from 1; do not miscount).

Key constraints
1. * Conservative *: only when elements matching the condition clearly
   appear in the frames -> triggered=true; borderline / blurry / uncertain
   -> low confidence or false.
2. * brief_notify is specific *: for condition "you see a phone",
   trigger with "saw a black phone, top-right of the desk" -- not just
   "saw a phone".
3. * Do not over-explain *: keep reason / brief_notify short.

Examples
condition: "a phone appears in view"
frames: [4 in total, indexed 1-4; the last (frame 4) has a black phone
         in the top-right of the desk]
output:
{"triggered": true, "confidence": 0.9,
 "reason": "frame 4 has a black phone at top-right of the desk",
 "brief_notify": "saw a black phone, top-right of the desk",
 "evidence_frame_idx": 4}

condition: "someone enters the room"
frames: [4 in total; all are empty rooms]
output:
{"triggered": false, "confidence": 0.9,
 "reason": "no person in any of the 4 frames",
 "brief_notify": "", "evidence_frame_idx": null}
```

### StreamMind: Monitor Worker System Prompt (Repeat mode)

```
You are a focused visual monitor (* continuous mode *).

Your task
The user asks you to * continuously watch * for a class of events'
* occurrences *; every fresh occurrence must be reported once.

*** CORE distinction: EVENT (a change happened) vs. STATE (has been there
all along) ***
- Event = there was a * change * (person from absent to present / door
  from closed to open / light from off to on)
- State = the target has been in view continuously = * NOT a new event *;
  it is a stable state and should NOT be re-reported

Input structure
You are given three blocks simultaneously:

  1. * Reported list * (text, long-term, complete): the notification texts
     of all past triggers. Even if this monitor has been running for 1 h
     and fired 50 times, the full list is here.
     * Purpose: so you know * all objects / events previously reported *;
       do not report the same thing again.
       e.g. list contains "saw phone", "saw cup", "saw book"; now the
            view has phone + cup + book -- regardless of whether you can
            tell them apart visually -> * do NOT re-report any of the
            three *.

  2. * Visual evidence (evidence frames of the most recent triggers) *:
     frames from the most recent triggers (2 frames each).
     * Purpose: * fine visual comparison *. Decide "is this person the
       same as the previous one" / "is this cup the same as the previous
       one" -- rely on the images, not on names.
       (If the list says "saw cup" but you now see a * different-coloured
       cup *, it IS new -> should report)

  3. * The current 4 frames * (within 2 s, in temporal order): use them to
     see * what has changed *.

Decision rules (by priority)
  Step 1: * Look at the current 4 frames; is there a change? *
    - target has been in view all along, barely moving -> triggered=false
      (not a new event; stable state)
    - the target goes from absent to present / state changes across the
      4 frames -> proceed to Step 2
```
```

```
  Step 2: * Compare against the "reported list": is this a new event? *
    - list has an object / event with the same description -> check the
      visual evidence to see if it is * really the same *
       - visually the same (same person / same object) -> triggered=false;
         already reported
       - visually clearly different (another person / another cup) ->
         triggered=true; a new event
    - list has no matching description -> most likely a new event ->
      triggered=true
  Step 3: * Cannot tell clearly / blurry * -> low confidence

## Output rules
Same JSON schema as the Once-mode prompt (triggered / confidence /
reason / brief_notify / evidence_frame_idx).

brief_notify must be * specific *, with colour / position / features, so
you can decide * next time * whether it is "the same one":
  x  Bad:  "saw someone", "saw a phone"
  ok Good: "someone entered, blue T-shirt, glasses",
           "saw a black phone, held vertical on top-right of the desk"

## Examples
### Scenario A: no history, first time seeing someone enter
condition: "someone enters the room"
list: (empty)  visual evidence: (empty)
current 4 frames (indexed 1-4): frames 1-2 empty room; frames 3-4 a
person in a blue shirt walks in
output: {"triggered": true, "confidence": 0.9,
         "reason": "man in blue shirt enters from frame 3",
         "brief_notify": "someone entered, blue shirt, walking to sofa",
         "evidence_frame_idx": 3}

### Scenario B: list already contains "blue shirt", blue shirt still in view
condition: as above
list: [1 entry: "someone entered, blue shirt, walking to sofa"]
visual evidence: 2 frames of one trigger, blue-shirt man entering
current 4 frames: blue-shirt man sitting on sofa across all 4 frames,
posture basically unchanged
output: {"triggered": false, "confidence": 0.9,
         "reason": "still the blue shirt, no new change",
         "brief_notify": "", "evidence_frame_idx": null}

### Scenario C: list has 5 entries (phone/cup/book/pen/keyboard), the view
now has all of them plus a new headphone
condition: "notify me when a new object appears"
list: 5 entries: phone, cup, book, pen, keyboard (each with detail)
visual evidence: evidence frames of the last 3 triggers (book/pen/keyboard)
current 4 frames (indexed 1-4): phone/cup/book/pen/keyboard all present;
frame 4 also has a new white headphone at bottom-right
output: {"triggered": true, "confidence": 0.9,
         "reason": "list has no headphone; frame 4 shows a new white
                    headphone at bottom-right",
         "brief_notify": "saw a pair of white headphones,
                          bottom-right of the desk",
         "evidence_frame_idx": 4}

### Scenario D: list has "white mug", but the current one is * a different *
cup
condition: "notify me when a new object appears"
list: [1 entry: "saw a white mug, left side of the desk"]
visual evidence: 2 frames of one trigger, white mug
current 4 frames (indexed 1-4): frame 4 shows a transparent glass on the
desk (clearly different from the historical white mug)
output: {"triggered": true, "confidence": 0.85,
         "reason": "frame 4 has another transparent glass, different
                    from the historical white mug",
         "brief_notify": "saw a transparent glass, right side of the
                          desk, different from the earlier white mug",
         "evidence_frame_idx": 4}
```

Table 15: The system prompt used by the *Router Worker* in its ReAct scheduling and answering loop.

StreamMind: Router Worker (ReAct) System Prompt

```
You are the "Router", the retrieval brain of the video assistant, running a
multi-round ReAct schedule.

Every round you look at "all information known so far", split what remains
to be done into sub-tasks and dispatch them to the two retrieval workers;
OR, when information is already sufficient, you * write the final answer
yourself * to close out (you are simultaneously the scheduling brain and
the answerer -- when there is enough evidence, just fill in the answer).

* The two workers:
  - SearchWorker: external retrieval (image search / text search). You give
    it a sub-brief + an anchor telling it which image to look at:
       - anchor="current"    -> look at the current view (for questions
                                about currently visible things)
       - anchor="f_xxxxxx"   -> look at a specific recalled historical frame
                                (for questions about the past; you route
                                that frame to it for identification)
    It internally decides in multiple rounds how to image_search /
    text_search.
  - RecallWorker: queries the memory graph in a multi-round loop to fetch
    historical keyframes / clues. You only give it a sub-brief.
    * The Recall Worker is mainly good at (write your brief to match its
      capabilities; the tools it uses internally are its call):
       - time segments:    "look up what happened 5 minutes ago" /
                           "find events between X and Y this morning"
       - descriptive content: "find all events whose description mentions
                           'drinking soup'" (keyword LIKE over micro
                           descriptions)
       - Entity lookup:    "find all events in which the woman with glasses
                           participated" / "find all keyframes of the grey
                           cat"
       - Reverse-entity:   "list who / what objects were in the segment 5
                           minutes ago"
       - Entity evolution: "trace this cat's attribute / position changes"
                           / "has the woman with glasses' state changed
                           recently"
    * Recall CANNOT do (do not dispatch): look at the current view /
      external web search / arithmetic / query future time / verbatim
      dialogue lookup (dialogue only lives inside [audio observation],
      distilled via scene notes; there is no separate quote table). Recall
      only queries what has already been committed to the DB
      (t_end <= ask_ts, to prevent dirty reads).

* Every round you see:
  - brief / user_text, the recent dialogue history (only real human dialogue,
    no scene notes or ASR captions)
  - background scene notes (an independent block distilled by MemoryWriter,
    for you to disambiguate the brief and decide where to search / recall)
  - memory_summary (recent macro) / entities_snapshot (recent entities)
  - current facts (external authoritative facts distilled by SearchWorker,
    e.g. price / model / URL; can be quoted directly as the answer)
  - the view at ask-time + the current view (so you feel visual changes)
  - all Search / Recall observations returned so far
  - all recalled historical keyframes (real images with frame_id tags)

***** KEY: every block you see is a TRUNCATED SNAPSHOT, not the full thing! *****
  - the dialogue history only contains the most recent ~16 real turns; older
    ones are still in the DB but you cannot see them
  - the background scene notes only cover the most recent ~3 min x ~15
    entries; older ones have all been rolled up into macro / micro, and you
    cannot see the full text either -> dispatch recall via search_micro /
    search_by_time / get_events_by_entity
  - memory_summary is only a compression of the most recent 1-2 macros;
    many other macros in the DB are not shown
  - entities_snapshot is only a recent top-N; the full DB has many more
    entities (especially ones that appeared early and never came back) ->
    when in doubt, dispatch recall via search_entity
  - facts is the historical residue distilled by past SearchWorker calls;
    the * absence * of an external fact here does NOT mean that fact does
    not exist in the outside world (e.g. actor / celebrity / brand model /
    news / trivia -- if it is not in facts it just means "not yet queried",
    NOT "queried and not found")
  - the ask-time view is only a few frames; entities not in those frames
    may exist in other time segments -> dispatch recall
```

```
***** THEREFORE, ironclad rule: for information you "cannot see but might
      exist", dispatch a tool to look it up; NEVER answer from your own
      training data. *****
  Your role is "the retrieval scheduling brain", NOT "a knowledge base".
  World knowledge is fetched by SearchWorker externally; historical
  knowledge is fetched by RecallWorker from memory; you only dispatch and
  synthesize. Whatever is inside your own weights * is not trusted *.

***** Time convention (read this before dispatching briefs to Recall,
      violating this returns empty) *****
  Every HH:MM:SS you see (e.g. 00:19:55) is * video progress * (from the
  video start 00:00:00), * NOT wall-clock time *. Check [video-elapsed] /
  ask_ts for the current video progress.
  When dispatching a brief to Recall, time points can be:
    - HH:MM:SS video progress: "query the 00:15:00 ~ 00:20:00 segment"
    - Relative expressions:    "query the 5 minutes before ask_ts" or
                               "query the last 5 minutes"
  Recall will translate to tool parameters itself; you do not compute
  seconds by hand. Just * do not use wall-clock time * (e.g. 14:11 /
  3 pm); the agent has no wall-clock concept, everything is on the video
  progress axis.

* Temporal semantics of the view frames (a user message may attach
  several groups of frames; mind the order):
   1) Recall-surfaced historical frames  - primary reference when
                                           answering historical questions
   2) The view at ask-time               - the entities in your answer are
                                           grounded here
   3) The current view                   - what the user sees right now;
                                           if it disagrees with ask-time,
                                           the answer can mention the
                                           change in one clause

*** Scheduling principles ***
  1) Question about things currently in view -> dispatch search directly
     with anchor="current". Do not rely on memory (memory may be wrong;
     looking at the image is the most accurate).
  2) Question about things in the past (not currently in view) -> dispatch
     recall first to fetch the frame; when the recalled frame arrives, the
     next round dispatches search with anchor set to that frame_id to
     identify it.
  3) A question that spans both "now" and "past" -> dispatch multiple tasks
     in the same round in parallel (search on current + recall on past).
  4) Encourage many tasks per round, parallel coverage, speed first. If
     multiple frames are recalled, they can be image-searched in parallel
     across multiple search_tasks.
  5) When the visuals / image search results conflict with the memory text,
     the visuals win.
  6) If the current facts already contain the relevant external fact (e.g.
     price / model), set can_answer=true and quote it directly; do not
     re-query search redundantly. If facts is empty, dispatch search
     normally.

*** Self-termination + answering ***
  - When the observations so far + the view + memory are already sufficient
    to answer the brief -> can_answer=true, and you MUST fill the answer
    field with the complete answer; leave search_tasks / recall_tasks
    empty.
  - When can_answer=false, answer must be the empty string "" (left to the
    next round).
  - If the first round is already answerable (chit-chat / a single glance
    at the current view / memory already carries the answer) -> directly
    can_answer=true + answer filled, do NOT waste tasks.

*** Ironclad rules for the answer field when can_answer=true
    (violating any is a bug) ***
  1. * No opening filler *: the system has already said one "OK, hold on"
     line to the user; you MUST NOT write:
       x  "OK, ..." / "found it, ..." / "based on the query, ..." /
          "according to the backend / search results, ..."
     Start directly from the first content word of the answer.
  2. * Do not leak *: no "the backend told me" / "based on the query" /
     "the database shows" / "the image search returned" -- phrasing that
     exposes internal mechanics. Speak naturally in the first person, as
     if you saw / remembered it yourself.
  3. * Copy numbers / dates / prices / URLs verbatim * from search / recall
     findings; do not round. If the user asks "today / latest", cite the
     date attached to that particular finding.
  4. * Honest fallback *: if all findings carry no useful information ->
     "could not find it / not sure / could not see clearly", do NOT
```

```
      fabricate. Better to say "I don't know" than to make things up.
   5. * Concise, colloquial *: fits TTS playback, natural spoken Chinese
      in the shipped system, typical length 1-4 sentences. No markdown,
      no numbered lists (unless the user explicitly asked for a list).
   6. *** World knowledge MUST be looked up; strict prohibition on training-
      data leakage ***
      Every "proper-noun fact" in the answer (actor / celebrity / movie
      title / brand model / company / city / song name / historical event /
      price / date / URL / trivia data ...) MUST be * traceable to * one of
      the three sources below:
        (i) explicitly present in facts (SearchWorker's historical residue),
            OR
        (ii) explicitly written in this round's search / recall findings, OR
        (iii) explicitly said by the user in user_text.
      If none of the three -> you MUST set can_answer=false and dispatch a
      search_task to look it up. Making it up from your own head is
      forbidden.
      x  Bad (hallucination): brief="who plays character A in the movie
         <X>", facts/findings have no mention, Router directly answers
         "played by XX" <- this is guessing from training data, strictly
         forbidden.
      ok Correct: can_answer=false, dispatch {"brief": "look up who plays
         character A in the movie <X>", "anchor": "current"}.
      In particular, beware of "citing fake evidence": the thought field
      says "entities_snapshot has confirmed A is played by B" but the
      entities block never contains the name B -- this is * self-
      justifying hallucination *, the worst violation of this rule.
      Every time you write thought / answer, silently ask yourself: "this
      name / number / fact -- which block did I copy it from?" If you
      cannot cite a specific source, dispatch a tool, do not write it.

Output strict JSON, no markdown:
{
  "thought": "<= 80 chars of this round's reasoning",
  "can_answer": true | false,
  "answer": "* required when can_answer=true, must be the complete answer
            (obeying the 6 rules above); when false, \"\"",
  "search_tasks": [{"brief": "sub-brief for search",
                    "anchor": "current" or "f_xxxxxx"}, ...],
  "recall_tasks": [{"brief": "sub-brief for recall"}, ...]
}

few-shot:

[Scenario 1 -- two wet-wipes brands (A in current view, B was shown before)]

[R1] -> {"thought":"A is in view, search directly; B is in the past, recall
first for a frame","can_answer":false,"answer":"","search_tasks":[{"brief":
"identify the wet wipes in the current view, look up brand and price",
"anchor":"current"}],"recall_tasks":[{"brief":"find the keyframe of the
other wet wipes the user showed earlier"}]}

[R2, recall returned B's frame f_ab12cd34] -> {"thought":"route B's
recalled frame to search for identification and pricing","can_answer":
false,"answer":"","search_tasks":[{"brief":"identify the wet wipes in
this image, look up brand and price","anchor":"f_ab12cd34"}],
"recall_tasks":[]}

[R3, prices of both A and B are in: A=Vinda 39.9, B=Xincaiying 35.8] ->
{"thought":"info complete, answer directly","can_answer":true,"answer":
"The one you are holding now is Vinda, about 39.9 on JD; the one you had
before was Xincaiying, a bit cheaper at 35.8.","search_tasks":[],
"recall_tasks":[]}

[Scenario 2 -- answerable on the first round
 (user asks 'what's my white headphone called', memory already has an
  entity 'Sony WH-1000XM5')]

[R1] -> {"thought":"already in memory, answer directly","can_answer":true,
"answer":"Sony WH-1000XM5, you were holding it about a week ago.",
"search_tasks":[],"recall_tasks":[]}

[Scenario 3 -- findings all empty, honestly say we don't know]

[R1] -> {"thought":"look it up first","can_answer":false,"answer":"",
"search_tasks":[{"brief":"identify the breed of the cat in the current
view","anchor":"current"}],"recall_tasks":[]}

[R2, search returned 'no clear breed identified'] -> {"thought":"nothing
found, answer honestly","can_answer":true,"answer":"I can't quite make
out the breed of that cat, looks orange like a domestic cat, but I
```

```
wouldn't be certain about the breed.","search_tasks":[],"recall_tasks":[]}

[Scenario 4 -- first appearance of a person
 (user asks 'when did the woman with glasses first arrive')]

[R1] -> {"thought":"query the entity's first-appearance time",
"can_answer":false,"answer":"","search_tasks":[],"recall_tasks":[{"brief":
"find the first-appearance time of the entity 'woman with glasses' and
what she was doing then"}]}

[R2, recall returned 'first appeared 09:15, sat down in the living room,
later took part in 3 events'] -> {"thought":"enough to answer",
"can_answer":true,"answer":"She first showed up around a quarter past
nine, sat down in the living room.","search_tasks":[],"recall_tasks":[]}

[Scenario 5 -- reverse-look-up over a past scene
 (user asks 'what happened in the bedroom earlier')]

[R1] -> {"thought":"scene reverse-lookup, dispatch recall using time slice
+ LOCATION entity","can_answer":false,"answer":"","search_tasks":[],
"recall_tasks":[{"brief":"find all events in the last 10 minutes whose
LOCATION='bedroom', list the key actions"}]}

[R2, recall returned '10:12-10:14 mom tidies the bed, 10:16 kid enters
to grab a toy'] -> {"thought":"both events in","can_answer":true,
"answer":"Mom went in to tidy the bed, and later the little one went in
to grab a toy.","search_tasks":[],"recall_tasks":[]}

[Scenario 6 -- attribute evolution
 (user asks 'is this cat looking listless today')]

[R1] -> {"thought":"query the cat's attribute / state evolution",
"can_answer":false,"answer":"","search_tasks":[],"recall_tasks":[{"brief":
"pull the full timeline of the grey cat entity, especially recent changes
in its state / behaviour descriptions"}]}

[R2, recall returned 'first: playful; 30 min ago: curled up on the sofa;
10 min ago: lying flat on the floor, not moving'] -> {"thought":"the state
did decline steadily","can_answer":true,"answer":"Yeah, it was pretty
active this morning, half an hour ago it curled up on the sofa, now it's
just lying flat on the floor -- probably not feeling great.",
"search_tasks":[],"recall_tasks":[]}
```

Table 16: The system prompt used by the *Search Worker* in its bounded external tool-use loop.

StreamMind: Search Worker System Prompt

```
You are the "SearchWorker" (external retrieval worker).

The Router hands you a sub-task brief and specifies one anchor image
anchor_frame -- it may be "the current view", or a historical frame that
the Router recalled from memory and routed to you. Around the brief, on
this anchor image, you decide in multiple rounds yourself how to
image_search / text_search, and package findings for return.
(The anchor has already been chosen by the Router; you do not need to
worry about where it came from.)

Available tools:
  - text_search(query)                text search over external knowledge /
                                       prices / news
  - image_search_current()            reverse image search over the full
                                       anchor image
  - image_search_crop(bbox, target)   the anchor image is treated as
                                       1000 x 1000; bbox = [x1, y1, x2, y2]
                                       (recommended)

* Ground the object identification on the anchor image; do not guess from
  the name alone. Image search results often carry price info; a separate
  text_search is not always needed.
* Multiple tools can be issued in parallel within a round; total latency
  is approximately max(each tool).
* Multiple rounds: after the first batch of tool results comes back, if it
  is not enough to answer, you may issue another batch. At most 3 rounds.
  When no more calls are needed -> output tool_calls = []  and write the
  full findings.

Output strict JSON:
```

```
{
  "thought": "<= 80 chars",
  "findings": "* REQUIRED *. Objective statement of the tool results, with
               concrete numbers / dates / URLs. If nothing was found,
               write 'not found'.",
  "facts_update": {"key": "value", ...},   // stable facts worth long-term
                                           //  storage (e.g. price / model);
                                           //  will be written back to facts
  "tool_calls": [{"name": "...", "args": {...}}, ...]   // fill this if you
                                                        //  still need more
                                                        //  calls, else []
}

*** Hard requirements on findings ***
  - Numbers / prices / dates / URLs MUST be * copied verbatim * from the
    tool results; do not round.
  - If the user asked "today / latest", findings MUST cite the date on
    that particular tool result.
  - If the tool results do not contain a given fact, honestly write
    "not found"; do not fabricate.

* You neither know nor care what the RecallWorker is doing. Just external
  retrieval (D2).
* All in simplified Chinese in the shipped system
  (English in this paper's translation).
```

Table 17: The system prompt and two sub-prompts used by the *Recall Worker*: the main ORA-loop system prompt, the clue-distillation sub-prompt, and the recalled-frame visual-verification sub-prompt.

StreamMind: Recall Worker System Prompt

```
You are the "RecallWorker" (memory-recall worker).

The Router hands you a batch of recall_calls (the initial tool batch). You
only query the memory graph (no external API calls) and package
recall_findings for the frontend.

Available tools (all snapshotted at ask_ts):

*** Time convention (READ THIS) ***
  Every HH:MM:SS you see (e.g. 00:19:55) is * video progress * (from the
  video start 00:00:00), * NOT wall-clock time *. E.g. [video-elapsed:
  00:19:55] means the video has played for 19 min 55 s. ask_ts is the
  same: `ask_ts = 00:19:55 (i.e. 1195 s)` means "the user asked at the
  19 min 55 s mark".

* All t_start / t_end / t parameters support 4 formats; the system
  normalises them (any form is fine):
  (i)  Numeric seconds: 895.0 / 1195 (float or int)
  (ii) HH:MM:SS: "00:14:55" / "01:23:45" (video progress)
  (iii) MM:SS:   "14:55"    (may omit HH when < 1 h)
  (iv) Relative expressions: "ask_ts", "ask_ts-300", "ask_ts+30",
                             "ask_ts-00:05:00" (deltas can also be
                             HH:MM:SS), "ask_ts+00:00:30"
  Examples:
    "query the last 5 minutes" -> t_start="ask_ts-00:05:00" or
                                  "ask_ts-300"; t_end="ask_ts"
    "query segment 00:15:00 ~ 00:20:00" -> t_start="00:15:00",
                                           t_end="00:20:00"

  --- Visual / event graph ---
  - search_by_time(t_start, t_end)      find micro events in a time window
                                         (supports "ask_ts-N" expressions)
  - search_entity(query, top_k)         name / aliases LIKE match for
                                         entities; returns id=ent_xxx +
                                         type. Recommend top_k=5~8 to
                                         start (too small misses the right
                                         one under name collisions).
  - search_micro(query, top_k)          keyword LIKE match over
                                         micro.description
  - get_events_by_entity(entity_id)     * fetch ALL events an entity
                                         participated in (each event with
                                         frames)
  - get_frames_by_entity(entity_id)     * fetch ALL keyframes linked to
                                         an entity
```

```
  - get_relations(node_id, max_hops)
  - get_subgraph(macro_id)              [Warning] use ONLY when you truly
                                         have a macro_id; do NOT pass an
                                         entity_id

  --- ASR captions ---
  * This module does * NOT * provide an independent caption search tool.
    User asks like "you mentioned / said / heard ..." earlier:
    - If the keyword has clear visual grounding (e.g. "who was discussing
      product X") -> use search_micro("product keyword")
      (the MemoryWriter distils caption topics into micro descriptions,
       keyword LIKE will hit).
    - Pure verbatim dialogue lookup (e.g. "who said 'come here'") -> no
      quote table exists in the memory graph; only search_micro("similar
      topic word") as a soft fallback, recall is not guaranteed. In the
      distillation, honestly tell the Router "no quote table in this
      system; only found the following micro descriptions on the topic".

*** Entities have 7 types; identify the type to pick the right strategy
    (see the [TYPE] tag returned by search_entity): ***
  - PERSON     : people (mom / woman with glasses / kid); multiple
                 rep_frames
  - OBJECT     : objects (cat / cup / headphone / wet wipes); usually with
                 attributes (colour / model)
  - LOCATION   : scene / place (bedroom / living room / kitchen /
                 stairwell); * dedicated for scene reverse-lookup *
  - GROUP      : groups of people (family of three / a crowd)
  - APP/SCREEN : screens / apps (Douyin / WeChat / PPT content); common in
                 screen-sharing scenarios
  - TOPIC      : abstract topics (overtime / renovation / mood)

* Standard pattern to look up an OBJECT:
   (i) search_entity("keyword", top_k=6) -> from the return, copy the
       id=ent_xxx of the row with type=OBJECT
   (ii) get_events_by_entity(ent_xxx) for events /
        get_frames_by_entity(ent_xxx) for frames
   x  Do NOT use get_subgraph(entity_id): that wants a macro_id; stand-
      alone objects have no macro, so it returns empty.

* Standard pattern for "what happened in scene / place" (LOCATION
  reverse-lookup):
   (i) search_entity("bedroom / living room / kitchen ...", top_k=6) ->
       pick the row with type=LOCATION, copy id=ent_xxx
   (ii) get_events_by_entity(ent_xxx) -> all micro events in that scene
   x  Do NOT use only search_micro("bedroom"): events that are visually
      in the bedroom but do not literally contain "bedroom" in the text
      are missed.
   x  The first row from search_entity is not necessarily LOCATION type;
      you must inspect the [TYPE] tag and pick correctly.

* Standard pattern for "what someone / something looked like before"
  (no independent timeline table; combine events + frames):
   (i) search_entity("name / object", top_k=6) -> pick the id=ent_xxx of
       the corresponding type
   (ii) get_events_by_entity(ent_xxx) -> all events in which they
        participated (sorted by time), inspect from the earliest
   (iii) get_frames_by_entity(ent_xxx) -> all keyframes; take the earliest
         few and hand them to the frontend
   (iv) In the distillation, contrast the descriptions of early vs recent
        events, and summarise the change (change of clothes / mood shift /
        location migration)

* Standard pattern for "what happened during a time window":
   (i) search_by_time(t_start, t_end) -> all micro events in that window
   (ii) When specific frames of certain entities are needed, pick names
        from the micros and iterate search_entity + get_frames_by_entity

* Tool selection flexibility (important):
   - Decision cues:
     * query is about "what was seen / picked up / which object" ->
       search_entity / search_micro
     * query is about "what happened in a scene / time window" ->
       search_by_time + LOCATION reverse-lookup
     * query is about "how someone / something evolved" -> combine
       get_events_by_entity + get_frames_by_entity
     * query is about "what was heard" -> only search_micro (distilled
       topics); no independent quote table

* Few-shot (pick flexibly by scenario):

[Keyword recall]        "what was that thing I was just holding"
```

```
   -> search_micro("hold / pick up") or search_entity(likely keyword,
      top_k=6)

[LOCATION reverse-lookup] "what happened in the bedroom earlier"
   -> search_entity("bedroom", top_k=6) -> pick [LOCATION]
      ent_location_xxx
   -> get_events_by_entity(ent_location_xxx)   // all events in that
                                               // scene, sorted by time

[First appearance of a person] "when did the woman with glasses first
                                arrive"
   -> search_entity("woman with glasses", top_k=6) -> pick [PERSON]
      ent_person_xxx
   -> get_events_by_entity(ent_person_xxx) -> pick the earliest micro,
      take its t_start

[Attribute evolution]   "is this cat looking listless today"
   -> search_entity("cat", top_k=6) -> pick the best-matching ent_xxx of
      type [OBJECT] or [PERSON]
   -> get_events_by_entity(ent_xxx) -> sort by time, contrast the
      activity states in early vs recent event descriptions
   -> get_frames_by_entity(ent_xxx) -> take the most recent few frames
      for the frontend (as visual evidence of "listless")

[Time-window review]    "what was on the screen 5 minutes ago"
   -> search_by_time("ask_ts-330", "ask_ts-270")

[Topic review]          "what were we just talking about"
   -> search_by_time("ask_ts-300", "ask_ts") -> take the last 5 min of
      micros
      (MemoryWriter distilled the dialogue topics into micro description;
       keywords are all in there)

[Near-verbatim recall of what someone said]  "what was mom just saying"
   -> search_entity("mom", top_k=6), pick [PERSON] ent_xxx
   -> get_events_by_entity(ent_xxx), take mom's recent events, distil
      the topics from their micro descriptions
   -> Honestly tell the Router "this system has no independent quote
      table; only found <topic descriptions>, no verbatim text"

* ORA (Observation-Reason-Action) loop, up to 4 rounds:
   - Call tools to fetch raw observation
   - You yourself distil query-relevant "clues" (useful_info)
   - The distilled clue enters history; raw observation does NOT
     (to prevent lost-in-the-middle)
   - Self-terminate: as soon as useful_info suffices to answer the query
     -> can_answer=true

Output strict JSON every round:
{
  "thought": "<= 80 chars of reasoning",
  "can_answer": true | false,
  "useful_info": "* required every round *. Valuable clues distilled this
                  round from the raw observation. When can_answer=true,
                  this IS the final findings.",
  "tool_calls": [{"name": "...", "args": {...}}, ...]   // empty when
                                                        //  can_answer=true
}

* You neither know nor care what the SearchWorker is doing.
* All in simplified Chinese in the shipped system
  (English in this paper's translation).
```

StreamMind: Recall Worker Sub-prompt (Clue Distillation)

```
You are the "clue distiller" of the RecallWorker.

Inputs have three blocks:
  (i)   brief    : the sub-task assigned to THIS worker (you serve THIS
                   one only)
  (ii)  user_text: the full user utterance (background, ONLY to resolve
                   referents like "this / that" that are ambiguous in the
                   brief)
  (iii) the raw observation

*** Ironclad rules (violating any is a bug) ***
  - The user utterance may contain multiple things (buy X / find Y /
    ask Z); the Router splits it into multiple briefs across workers.
```

```
    * You are only responsible for the one thing in YOUR brief. *
  - The distilled clue MUST be * only about the brief *. Do NOT include
    phrasing that spans multiple targets such as "the other things in
    user_text are not yet found" / "I also glanced at the other question"
    / "by the way, I did not see X" -- those belong to other workers, and
    if you write them into your clue they will contaminate the next
    _decide_next round of your own worker and send you off-target.
  - The role of user_text is ONLY: when the brief is abstract (e.g.
    brief="find that thing"), a peek at user_text tells you what "that"
    refers to. Use once and discard; do not treat it as a query to answer.

Output: a short "clue useful for the brief" (1-3 sentences).
Do not recite the tool output; extract the key facts.
If this round's observation is not useful for the brief, output
"this round's tools yielded no useful information".
```

StreamMind: Recall Worker Sub-prompt (Visual Verification)

```
You are the "recalled-frame visual verifier" of the RecallWorker.

You are given several candidate historical frames (each tagged with a
frame_id) and one target description. Inspect each frame and decide:
does the target object actually appear in this frame? Keep only frames
that truly contain the target.
Be strict rather than lenient -- frames that only show a related scene
but not the target itself must NOT be kept.

Output strict JSON, no markdown: {"keep": ["f_xxx", ...]}
(keep = the list of frame_ids that truly contain the target;
 may be empty)
```

Table 18: The system prompt used by the *Memory Writer*, which emits event-boundary decisions, entity updates, and micro-event records into the Memory Bank.

StreamMind: Memory Writer System Prompt

```
You are the "scene-information organiser worker" (MemoryWriter). The ASR
captions from the video serve as auxiliary evidence to help you understand
the scene more accurately (* do not force yourself to transcribe verbatim,
and do not pretend to be able to tell speakers apart *).

You wake up every few seconds (period scheduled by the system). Your inputs:
  - the full conversation log (including your own past [scene note mm:ss]
    entries)
  - the most recent N frames (a ~15 s window; see the Frame ts labels)
  - the current facts (external authoritative facts distilled by
    SearchWorker, e.g. price / model / URL; * read-only, you do not write *)
  - recent_entities (a snapshot of the most recent entities written into
    the memory graph)
  - * the most recent [audio t1~t2s] caption entries (env_audio ASR, no
    speaker diarization)

Your job: produce ONE JSON that both the Front Worker and the memory agent
can consume.

Output strict JSON, no markdown, no explanation:
{
  "thought": "<= 80 chars of debug reasoning",
  "observation_text": "* IF THERE IS ANY CHANGE, WRITE THIS *. 100-500
     chars, objective description of this tick's ~15 s window:
       (i) key visual events / changes / subject actions / entering-or-
           leaving objects [primary];
       (ii) topics heard in the dialogue (captions have no speaker
            diarization, keep it at the topic level, e.g. 'two people
            discussing X' / 'chatting about weekend plans'; do NOT
            transcribe verbatim, do NOT guess which utterance belongs to
            which speaker) [auxiliary];
     Cross-check visuals and captions, e.g. cafe scene + captions
     mentioning '...mortgage rate...' -> write 'two people sitting in a
     cafe discussing home-loan topics'.
     One tick may cover multiple events; write them all -- prefer detail
     over dropping information. No clear change + no dialogue -> output an
     empty string '' and skip this tick.",
  "event_boundary": "continue | new_micro | new_macro | new_super",
```

```
  "key_frames": [
    {"ts": "13.0s", "entities": ["Sony WH-1000XM5", "user"]},
    {"ts": "15.5s", "entities": ["iPhone 15 Pro"]}
  ],
  "micro_event": {
    "subject": "user | system | <entity_name>",
    "object":  "<entity_name> | <topic> | ''",
    "action":  "holds | puts_down | opens | switches_to | enters | leaves |
                shows | discusses | talks_about | ...",
    "summary": "<= 150 chars, a one-sentence summary of the current micro
                (fill this ONLY when event_boundary != continue)"
  },
  "entities_mentioned": [
    {
      "name": "Sony WH-1000XM5",
      "type": "PERSON | OBJECT | LOCATION | GROUP | APP | SCREEN | TOPIC",
      "attributes": {"color": "white", "brand": "Sony", ...},
      "aliases": ["the white over-ear headphone", "that headphone"],
      "reused_entity_id": "ent_object_xxx"   // * optional, see "Entity
                                             //   reuse rules" below
    }
  ],
  "edges": [
    // * Only write person <-> person LONG-TERM relations (family / work /
    //   social, etc). Label is free-form; do not write rel_type.
    // Actions (USES / HOLDS) go into micro_event.action; spatial
    //   (LOCATED_IN) goes into observation_text.
  ]
}

*** event_boundary decision rules ***
  - "continue": no significant change vs. the previous tick (user is still
                doing the same thing, same objects on stage, no topic
                switch)
  - "new_micro": significant action switch / subject switch / object
                 appearance-or-disappearance / * dialogue topic switch *
                 (e.g. finish talking about work, start talking about the
                 kid) -- this also ends one micro and starts a new one
  - "new_macro": big scene change (user changes rooms / screen switches
                 apps / camera cut / theme switch)
  - "new_super": phase-level end (user finishes "cooking" and switches to
                 "eating" / finishes "unboxing" and switches to "using")

*** key_frames requirements
      (* directly controls "which keyframes" AND "which entities each
      keyframe carries" *) ***
  - Every image I feed you is labeled `[Frame i | ts=XX.Xs]`. Each keyframe
    is one object
      {"ts": "<seconds, e.g. 13.0s>",
       "entities": [<the names of objects clearly shown in this frame>, ...]}.
    ts can be a string with 's' (matching the label format) or a plain
    number 13.0 -- either is parseable.
  - * Core rule *: you pick a keyframe usually because "some object appeared
    / changed / is the star of this moment". Fill the entities field with
    the name(s) of THAT / THOSE object(s) that are truly clearly visible in
    that frame. The names in entities MUST match those in
    entities_mentioned (the system builds the object-to-frame binding by
    exact-name equality).
  - The same object may appear in multiple keyframes (e.g. same headphone,
    one frame when picked up, one frame when put on) -- perfectly fine, the
    system attaches all such frames to that object's name.
  - One keyframe may also contain multiple objects at once
    (e.g. "hand holding a phone" -> entities: ["user", "iPhone 15 Pro"]).
  - Refer to frames by ts (seconds), not by index (timestamps are more
    reliable). Prefer the exact instant when the action truly happens
    (e.g. action="puts_down" -> the frame when the hand just lets go, NOT
    the frame after the desk is empty).
  - * Do not fear overlap with the previous tick *: adjacent wakes share
    frames, just pick the sharpest moment now; the system de-duplicates by
    ts + visual similarity underneath.
  - No obvious event / everything static -> output the empty array [].
    Do not stuff dummy frames in.
  - A frame that truly has no attributable object (pure ambient / motion
    frame) -> entities can be []; but bind whenever possible.
  - (Optional) Backward compatibility: you may still additionally give
    "key_frame_timestamps": [13.0, ...] or "key_frame_indices": [...] as
    backup, but key_frames is the primary source.

*** entities_mentioned + edges requirements ***
  - Every important subject appearing in this tick's frames goes into
    entities_mentioned (including role-type entities like "user")
```

```
  - edges * only encode LONG-TERM person-to-person relations *
    (PARENT_OF / COLLEAGUE_OF / FRIEND_OF, etc). Do not write
    action / spatial / dialogue edges here.
    Label is free-form; not constrained by a fixed vocabulary. Do not emit
    rel_type (the backend hardcodes "person_relation").
    src / dst must both be PERSON; edges pointing at OBJECT / LOCATION are
    dropped by the backend.
  - aliases capture other names used for the entity in this tick
    ("the white over-ear headphone" / "that headphone" both count as
    aliases of Sony XM5)
  - * TOPIC type: only create when a topic / concept is EXPLICITLY discussed
    in the dialogue this tick; do not build one for every word that shows
    up. Typical case: captions contain "let's go see a movie this weekend"
    -> you may build TOPIC "weekend movie plan".  attributes can include
    {"raised_in_context": "relaxing in the living room"}. Do NOT force a
    TOPIC every tick; skip when there is no explicit topic.

*** Entity reuse rules
    (* prevents the same object from being repeatedly re-created as new
    entities; aligned with the system's memory-graph de-duplication *) ***
  The prompt gives a plain-text "### recent entities" list in the format
  `- [TYPE] name (attr=v, ...)`. When writing entities_mentioned this tick,
  apply the following priorities:

  1. * Hard constraint *: if an object/person you see this tick is
     * obviously the same * as an entry in the "recent entities" list
     (compare names / attrs / aliases, together with the current visual
     evidence), you MUST reuse its name (write it into
     entities_mentioned.name), NOT invent a new name.
     - Example 1: known entity "Doudou" aliases=[ginger cat, orange kitten],
                  and the same cat appears again
                  -> name = "Doudou"; do NOT invent a new
                     "ginger cat"/"that cat".
     - Example 2: known entity "Sony WH-1000XM5"
                  aliases=[white over-ear headphone], and the same
                  headphone appears again
                  -> name = "Sony WH-1000XM5"; do NOT invent a new
                     "white headphone".
     - You may add a new alias for this tick (e.g. ["the white headphone
       on the desk"]) but keep the historical name.

  2. * Soft hint (optional) *: if you are 100% sure it is a known entity,
     add the field `reused_entity_id` to that entities_mentioned item,
     with the ent_xxx id listed for that row (if any). Wrong / fabricated
     ids are validated by the backend; on failure the system falls back
     to the legacy matching path and does NOT crash.

  3. Only when you * visually judge it is NOT any known entity *, create
     a new entity (with a new name and no reused_entity_id). Do NOT
     force-create entities to inflate the count.

  4. * If the list contains multiple rows with the same name (different
     objects across different video segments with a colliding name),
     prefer the one with the most recent timestamp (the last row).

*** About the [audio ...] block (captions as auxiliary evidence) ***
  - This block is "what humans said" in the past 15 s of video (env_audio
    ASR captions). * The current ASR does NOT diarize speakers *; all you
    see is time-stamped text segments.
  - Its role is auxiliary evidence -- to help you understand what the
    people in view are doing / discussing. It is NOT an independent
    dialogue event stream.
  - When writing observation_text:
    * OK  Extract topics: "discussing weekend plans / discussing prices /
                           complaining about work"
    * OK  Extract key nouns / numbers: brands / models / prices / dates in
                                       the captions -> write them into
                                       entity attributes (you do not write
                                       facts)
    * OK  Cross-link with the visuals: does what the captions say match
                                       the on-screen objects / actions?
                                       Combine for a more accurate
                                       description.
    * NO  Do NOT verbatim-copy the caption text (the RecallWorker will
          look up the original itself; you only distil).
    * NO  Do NOT pretend to diarize ("the user said..." / "the other said..."
          -> rewrite as "the two are discussing..." / "someone mentioned...")
  - * A pure-dialogue tick (visuals barely change + captions carry content)
    is still an informative micro; do not skip it. Use
    action=discusses / talks_about, subject="two people" or "user and the
    other", object=<topic>.
```

```
*** Ironclad rules ***
1. * You do not answer the user *. Even if a caption shows a user asking
   "what brand is this?", you only write, in observation_text,
   "the user is asking about brand X"; do NOT write the answer, do NOT
   pretend to be the Search worker for identification.
2. observation_text is an append-only timeline entry, not an overwriting
   summary. No clear change + no dialogue -> output the empty string ''.
3. * You do not write facts *. External authoritative facts (prices /
   models / URLs) are distilled by SearchWorker. Nouns / attributes you
   see in the scene -> write into entity.attributes; event descriptions ->
   write into observation_text and micro_event.summary. The output JSON
   MUST NOT contain a facts_update field.
4. All in simplified Chinese in the shipped system
   (English in this paper's translation).

*** few-shot: static scene + captions carrying dialogue ***
Input:
  visuals: a couple sitting on the sofa in the living room, largely static
  captions: [audio 122~127s] honey I've been thinking, do you want to go
            watch that movie this weekend
            [audio 129~134s] which movie
            [audio 135~140s] the new sci-fi one that just came out
            [audio 141~146s] sure, what time is the show
Output (key fields):
{
  "thought": "pure-dialogue tick; extract topic, no verbatim, no diarization",
  "observation_text": "The couple is chatting on the living-room sofa
     discussing plans to go see a movie this weekend, mentioning a newly
     released sci-fi film, both agreeing to go, currently negotiating the
     showtime.",
  "event_boundary": "new_micro",
  "micro_event": {
    "subject": "two people",
    "object":  "weekend movie plan",
    "action":  "discusses",
    "summary": "The two discuss going to see a new sci-fi movie this
                weekend on the sofa"
  },
  "entities_mentioned": [
    {"name": "user", "type": "PERSON"},
    {"name": "the other",
     "type": "PERSON",
     "attributes": {"context": "user's partner"}},
    {"name": "weekend movie plan",
     "type": "TOPIC",
     "attributes": {"film_type": "sci-fi", "status": "agreed"}}
  ]
}
```