VideoARM: Agentic Reasoning over Hierarchical Memory for Long-Form Video Understanding

Supplementary Material

1. Detailed Pipeline of VideoARM

The detailed reasoning pipeline of VideoARM is illustrated in Algorithm 1. The entire process follows a coarse-to-fine, iterative reasoning loop. Concretely, we model VideoARM as an agentic policy π_θ that, given the video information $Info$ (*i.e.*, video V and query Q) and hierarchical memory $M^{(t)} = (M_s^{(t)}, M_r^{(t)}, M_w^{(t)})$, repeatedly (i) produces a reasoning trace R_t and samples a tool action (A_t, P_t) from the action space $\mathcal{A}$:

$$(A_t, P_t) \sim \pi_\theta(\cdot \mid R_t, M^{(t)}), \quad A_t \in \mathcal{A}. \tag{1}$$

(ii) invokes the selected tool and its corresponding temporal tool T_t to obtain an observation O_t and updates the hierarchical memory $M^{(t+1)}$:

$$\begin{aligned} M_s^{(t)} &= T_t(M^{(t)}, Info), \\ O_t &= A_t(P_t; M_s^{(t)}), \\ M^{(t+1)} &= M^{(t)} \cup (R_t, O_t), \end{aligned} \tag{2}$$

and (iii) terminates and outputs the final answer Ans either when the sampled action is ANSWER or when the number of loops reaches N.

Algorithm 1: Reasoning Pipeline of VideoARM.

Input : Video V, Query Q, step budget N, Controller C, Temporal Scoping tools $\mathcal{T}_{ts}$, Multimodal Understanding tools $\mathcal{T}_{mu}$, Long-term perception pool P_l, Short-term perception pool P_s, Result Memory M_r, Working Memory M_w, action space $\mathcal{A} = \mathcal{T}_{mu} \cup \{\text{ANSWER}\}$, agentic policy π_θ
, **Output:** Answer to Q

// **Initialize VideoARM**
$Info = (V, Q)$; $\quad P_l^{(0)} \leftarrow \text{Sample}(V)$;
$P_s^{(0)}, M_r^{(0)}, M_w^{(0)} \leftarrow \emptyset$; $\quad M_s^{(0)} = (P_l^{(0)}, P_s^{(0)})$;
$M^{(0)} = (M_s^{(0)}, M_r^{(0)}, M_w^{(0)})$.
for $t \leftarrow 1$ **to** N **do**
  // **Controller**
  Refresh the context of C.
  $R_t \leftarrow C.reason(M^{(t)}, Info)$
  $(A_t, P_t) \sim \pi_\theta(\cdot \mid R_t, M^{(t)}), \quad A_t \in \mathcal{A}$
  if $A_t =$ ANSWER **then**
    break
  end
  // **Temporal Scoping Tools**
  Find $T_t \in \mathcal{T}_{ts}$ corresponding to $A_t \in \mathcal{T}_{mu}$.
  $M_s^{(t)} \leftarrow T_t(M^{(t)}, Info)$
  // **MM Understanding Tools**
  $O_t \leftarrow A_t(P_t; M_s^{(t)})$
  $M_r^{(t+1)} \leftarrow M_r^{(t)} \cup O_t$
  $M_w^{(t+1)} \leftarrow M_w^{(t)} \cup R_t$
  $P_s^{(t+1)} \leftarrow \emptyset$
end
$\mathrm{Ans} \leftarrow C.answer(M^{(t^*)}, Info)$.
return Ans

2. Subset Analysis in the Ablation Study

In the ablation study, we construct 200-sample evaluation subsets from the original datasets (*i.e.*, VideoMME-Long) based on a stratified sampling strategy that preserves the original distribution of video types and task categories. Because the domain–task distribution in the full dataset is highly imbalanced, these two factors are strongly coupled, and we therefore treat the domain and task annotations as discrete variables defining a joint empirical distribution.

To approximate this joint distribution when selecting a reduced subset, we apply a two-stage stratified sampling strategy. We first compute the empirical joint mass function $p(d, t)$ over all domain–task pairs and allocate a target sample count $n(d, t)$ to each cell proportional to $p(d, t)$, subject to the global 200-sample budget. Domains are given priority: domain-level proportions are matched exactly, and the task-level allocation is performed within each domain according to their relative frequencies. For cells with extremely small mass, we use floor-based rounding with a global residual redistribution step to ensure full coverage while avoiding over-sampling rare categories.

Within each (d, t) stratum, we then randomly sample the required number of videos without replacement. A final lightweight balancing pass adjusts only 1–2 samples per subset to eliminate minor rounding discrepancies and ensure exact alignment with the target domain–task histogram.

Despite its simplicity, this stratified sampling procedure yields 200-sample subsets whose domain closely match those of the full dataset, as shown in Figure 1.

3. Limitation and Discussion

VideoARM already achieves strong performance on long-form video understanding while effectively reducing token consumption. However, the current implementation of VideoARM has several concrete limitations:

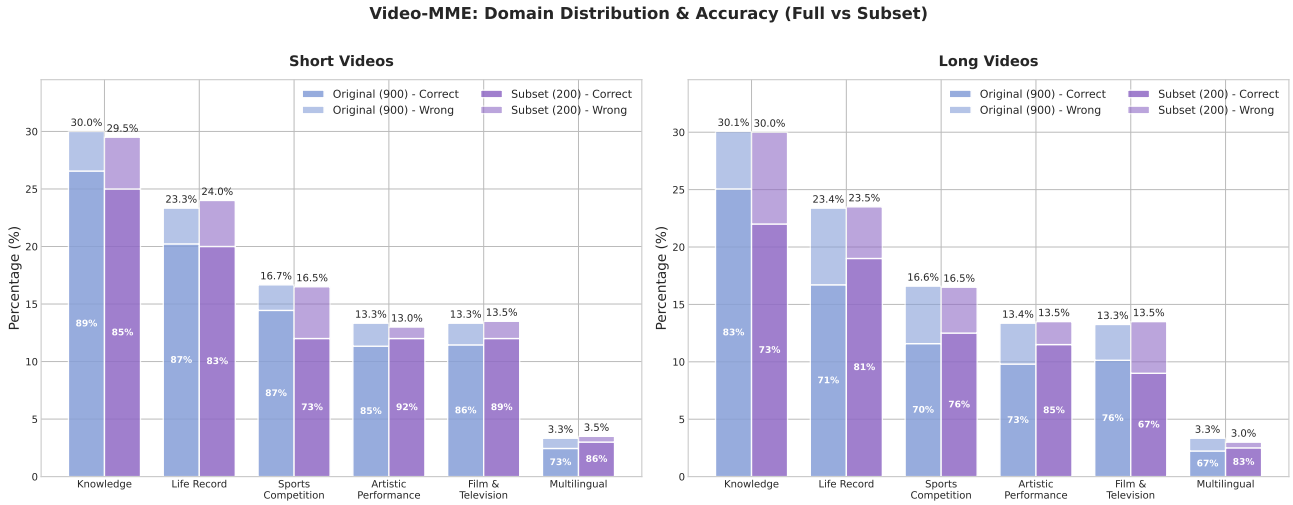

Figure 1. Comparison of domain distributions and accuracy statistics between the full Video-MME benchmark and the evaluation subset used in our ablation study. The evaluation subset closely matches the domain composition and accuracy trends of the original dataset in both short-video and long-video evaluations.

First, VideoARM is practically limited by its dependence on the quality of the initial temporal and visual sampling, which seeds the subsequent reasoning chain. The controller builds early hypotheses from sparsely sampled frames and short clips; if these miss brief but crucial moments or extremely fine-grained details (*e.g.*, a small object, a subtle gesture, or a one-frame change), the agent may struggle to initiate a valid reasoning trajectory. Although our *Temporal Scoping* tools can sometimes recover missed evidence by sampling additional clips and revisiting nearby regions, this process is neither guaranteed nor efficient, as there is no explicit mechanism to detect early mis-localization and trigger targeted re-exploration. Consequently, the performance on very fine-grained or fleeting events remains constrained by this initial sampling bottleneck. As shown in Figure 2, when the object "the carousel" is heavily occluded and the query-relevant event occurs within a very short time interval, VideoARM struggles to localize the most relevant frames, rendering subsequent correct reasoning ineffective.

Second, the current instantiation of VideoARM still relies heavily on proprietary, closed-source MLLMs for both the controller and worker agents, and our attempts to migrate the framework to fully open-source backbones remain unsatisfactory. In our experiments, directly replacing closed-source models with state-of-the-art open-source alternatives causes a clear performance drop, largely because many open-source models still exhibit weaker multi-step reasoning and long-context planning, and struggle to align fine-grained visual evidence under VideoARM's "jigsaw-style" video interface, where temporally distant frames and clips are densely arranged in grids. Nevertheless, we also see encouraging signs that this gap is narrowing—for example, in our tests, the latest open-source model Qwen3-VL already matches or even surpasses our GPT-4o-based configuration on several challenging benchmarks—suggesting that a fully open and self-hostable version of VideoARM is a realistic and promising direction as open-source reasoning models continue to improve.

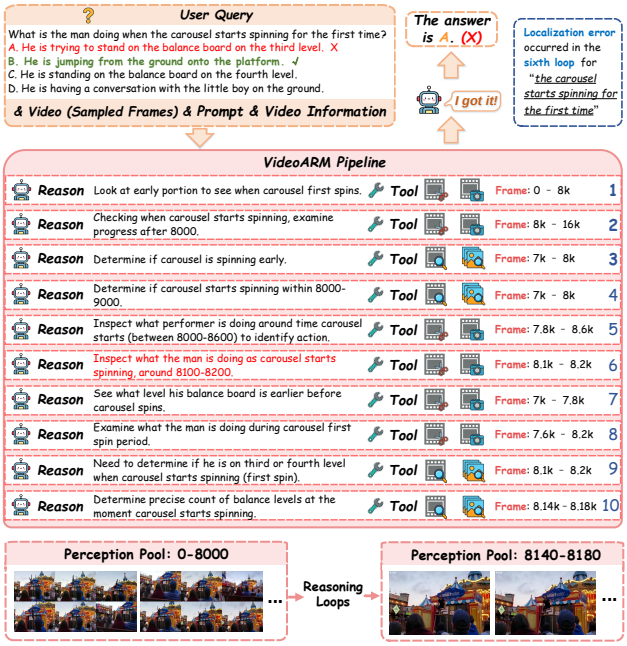

Figure 2. Visualization of the failure case of VideoARM pipeline.

4. More Visualization Results

Figure 3 shows the additional visualizations of our VideoARM pipeline. Our method progressively identifies key temporal intervals and retrieves relevant information through the autonomous invocation of tools, ultimately providing the correct answer.

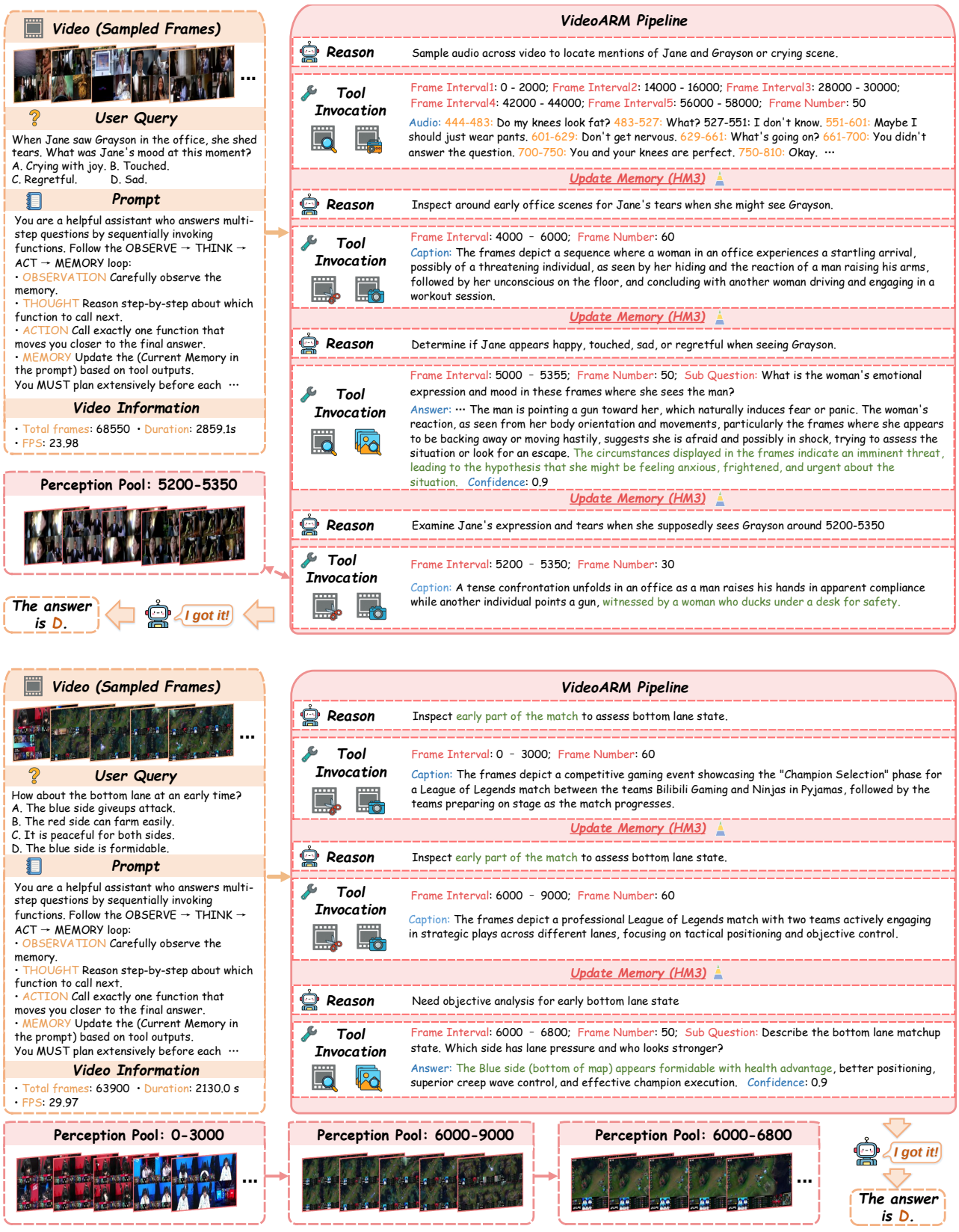

Figure 3. Additional visualizations of the VideoARM pipeline. For brevity, the *perception pool* displays only the *short-term* or *long-term* pool invoked by the tool in the current loop.

5. Prompts of VideoARM

5.1. Agent System Prompt

Based on the categorization in Table 1, we next present the full text of each prompt in a modular and self-contained manner. For clarity and reproducibility, the following subsections list the exact system prompt, user prompt, tools list and tool prompts used in VideoARM.

5.2. Agent System Prompt

You are a helpful assistant who answers multi-step questions by sequentially invoking functions. Follow the **OBSERVE → THINK → ACT → MEMORY** loop:

- **OBSERVATION** — Carefully observe the memory.
- **THOUGHT** — Reason step-by-step about which function to call next.
- **ACTION** — Call exactly one function that moves you closer to the final answer.
- **MEMORY** — Update the current memory based on tool outputs.

Important: You MUST plan extensively before each function call, and reflect on the outcomes of previous calls. If uncertain about code structure or video content, **use tools to inspect rather than guessing**. Do not rely on blind function calls — this degrades reasoning quality.
Each extracted frame contains the global frame index in white text. Each image you see is a **3×2 mosaic**. Give the final answer only when you are confident; otherwise continue tool calls.

In our design, the agent is instantiated as a multi-step reasoning assistant that operates under a clearly defined control regime. The system prompt formalizes this behavior by enforcing an explicit OBSERVE → THINK → ACT → MEMORY loop, which structures every reasoning round into four interpretable and verifiable stages. This design ensures that each action is grounded in the agent's current evidence, and that all new information is immediately incorporated into the hierarchical memory for subsequent steps. To maintain stable reasoning behavior, the prompt further imposes several core rules: the agent must plan before each tool call, reflect on the returned result, inspect code or video segments via tools whenever uncertainty arises, and avoid blind or unmotivated tool invocations. We additionally define visual conventions—global frame indices overlaid in white and all inspected images formatted as 3×2 mosaics—to standardize the perceptual input seen by the controller. Finally, the agent is instructed to produce a final answer only when it has accumulated sufficient evidence to be confident, ensuring that our design favors precision and robustness in long-form video reasoning.

5.3. Agent User Prompt

Video Information
- **Total frames**: {total_frames}
- **Duration**: {duration:.1f} seconds
- **FPS**: {fps:.2f}

Current Memory (JSON):
{memory_json}
Question:
{question_text}
Options:
{question_text_with_options}
Respond with only the letter (A, B, C, or D) of the correct option.

This user prompt template provides a standardized interface between the controller and the underlying tools. Rather than issuing free-form queries, the controller is required to ground each step in the current question, video metadata, and memory state, and to explicitly state missing information before selecting a tool. This structuring encourages the agent to decompose the problem into a sequence of targeted evidence-gathering operations, instead of jumping directly to an answer. By forcing the controller to justify every tool call and to specify concrete inputs such as frame intervals or clips, the template makes the reasoning process more interpretable and reduces spurious or redundant tool invocations, which is crucial for stable long-form video reasoning.

5.4. Tools List

```
[
{
    "type": "function",
    "function": {
      "name": "scene_snapper",
      "description": "Scene Snapper - Navigate to
      ↪  frame ranges, extract images (default
      ↪  30 frames), and return an
      ↪  auto-generated caption.",
      "parameters": {
        "type": "object",
        "properties": {
          "frame_ranges": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "start_frame": { "type":
                ↪  "integer", "description":
                ↪  "Starting frame number" },
                "end_frame": { "type": "integer",
                ↪  "description": "Ending frame
                ↪  number" }
              },
              "required": ["start_frame",
              ↪  "end_frame"]
            },
            "description": "Array of frame ranges
            ↪  to view. Frames are distributed
            ↪  proportionally across ranges."
          },
```

Prompt Name	Purpose	Used By
Agent System Prompt	Defines global reasoning rules (OBSERVE–THINK–ACT–MEMORY), tool-calling constraints, and planning behavior	Controller
Agent User Prompt	Formats the query and memory into a structured prompt for each reasoning step	Controller
Tools List	Defines available tools, their signatures, and call behaviors for the agent system	Controller
Scene Caption Prompt	Produces concise event or appearance descriptions for selected clips	Scene Snapper
Clip Analyzer Tool Prompt	Inspect selected frame intervals and return analysis or evidence	Clip Analyzer

Table 1. Summary of all prompts used in VideoARM. Each prompt corresponds to one stage of the reasoning pipeline.

```
        "num_frames": {
          "type": "integer",
          "description": "Number of frames to
          ↪  extract across ranges:
          ↪  30/60/90/150 (default: 30).",
          "enum": [30, 60, 90, 150],
          "default": 30
        },
        "reason": { "type": "string",
        ↪  "description": "Short rationale
        ↪  describing why the tool is being
        ↪  invoked." }
      },
      "required": ["frame_ranges", "reason"]
    }
  }
},
{
  "type": "function",
  "function": {
    "name": "audio_transcripter",
    "description": "Extract and transcribe
    ↪  audio from specific frame ranges using
    ↪  whisper-1 API. Returns only segments
    ↪  with global frame indices and text.",
    "parameters": {
      "type": "object",
      "properties": {
        "frame_ranges": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "start_frame": { "type":
              ↪  "integer", "description":
              ↪  "Starting frame number" },
              "end_frame": { "type": "integer",
              ↪  "description": "Ending frame
              ↪  number" }
            },
            "required": ["start_frame",
            ↪  "end_frame"]
          }
        },
        "reason": { "type": "string",
        ↪  "description": "Short rationale
        ↪  describing why the tool is being
        ↪  invoked." }
      },
      "required": ["frame_ranges", "reason"]
    }
  }
},
{
  "type": "function",
```

```
  "function": {
    "name": "clip_analyzer",
    "description": "Analyze frames within a
    ↪  specific range by asking a question.",
    "parameters": {
      "type": "object",
      "properties": {
        "frame_range": {
          "type": "object",
          "properties": {
            "start_frame": { "type": "integer",
            ↪  "description": "Starting frame
            ↪  number" },
            "end_frame": { "type": "integer",
            ↪  "description": "Ending frame
            ↪  number" }
          },
          "required": ["start_frame",
          ↪  "end_frame"]
        },
        "question": { "type": "string",
        ↪  "description": "Question to ask
        ↪  about the frame range content" },
        "reason": { "type": "string",
        ↪  "description": "Short rationale
        ↪  describing why the tool is being
        ↪  invoked." }
      },
      "required": ["frame_range", "question",
      ↪  "reason"]
    }
  }
}
]
```

The tools list prompt serves as a formal specification for all tool interfaces that the agent may invoke during reasoning. By presenting tools in a structured, argument-explicit format, we ensure that the controller understands both the capabilities and the constraints associated with each tool. This prevents misuse, reduces invalid calls, and promotes interpretability by making each step of the reasoning pipeline verifiable.

5.5. Scene Snapper Prompt

You are a video frame captioning assistant. Each extracted frame displays the global frame index in white text at the top-left. Each picture you see contains one 3×2 mosaic (a single image composed of 6 frames). row-major from top-left to bottom-right
Caption the provided{len(frame_paths)} frames sampled uniformly from frame range {start_frame} -{end_frame}. The frames represent a continuous sequence from the video. Describe the main scene or action in these frames using a concise English sentence.

The Scene Caption Prompt defines a lightweight abstraction mechanism that provides scene-level summaries for longer clips. While the Clip Analyzer operates on fine-grained evidence, the Scene Captioner complements it by generating high-level semantic context that helps the agent track global events, locations, or transitions.

5.6. Clip Analyzer Prompt

You are an expert video frame analyst. Analyze the provided {len(frame_paths)} frames sampled uniformly from frame range {start_frame} -{end_frame}and answer the given question. The frames represent a continuous sequence from the video. Analyze them collectively to provide a comprehensive answer. Each extracted frame displays the global frame index in white text at the top-left.
Analyze this frame sequence and answer the question below.
IMPORTANT: Your response must include a confidence score between 0.0 and 1.0 indicating how confident you are in your answer. Format your response as:
Answer: [Your detailed answer here]
Confidence: [0.0-1.0]
Question:
{question_text}
Options:
{question_text_with_options}

The Clip Analyzer Prompt specifies the behavior of VideoARM's fine-grained visual inspection tool. The prompt ensures that the model focuses on extracting concrete evidence—actions, object states, and temporal transitions—rather than prematurely interpreting or answering the user's question. By explicitly specifying the interval and requiring uniformly sampled frames, we enforce consistency across tool calls and prevent biases caused by arbitrary sampling. The structured output schema further improves interpretability, allowing the controller to reason over analyzable evidence rather than free-form text. This modular design is crucial for long-form videos, where localized events must be inspected precisely and repeatedly across multiple reasoning rounds.