META: Meta Evolution of Tool Trajectory Adaptation for Long-Video Understanding

Supplementary Material

1. Macro-Tools Abstraction

As illustrated in Figure 1, we depict the concrete process by which META abstracts a macro-tool from a single problem-solving episode. In the solving phase, the orchestrator–worker takes the input query and produces a main trajectory that spans multiple rounds of subtask decomposition and execution, along with fine-grained trajectories for each subtask, as shown in Figure 1(a). The Reflector then identifies a key subtask trajectory—in this example, subtask 3 in round 3, whose detailed execution trace is shown in Figure 1(b)—and abstracts this sub-trajectory into a reusable macro-tool, represented as an executable function, as shown in Figure 1(c).

2. Evolution of Experience Notes

We show the experience notes extracted by the micro-tool *clip_query* during the meta evolution process in Figure 2. The evolution of these experience notes traces a progression from concerns about basic perceptual inaccuracies of the tool to a more interaction- and process-oriented perspective. Initially, the focus is on the tool's intrinsic limitations in reliably interpreting video content; over time, attention shifts to how the form of user queries and their implicit presuppositions systematically bias the tool's responses. Correspondingly, the documentation moves beyond isolated error instances toward a more systematic analysis of underlying error mechanisms and the role of query design. This culminates in an emphasis not merely on improving raw recognition performance, but on devising invocation and interaction strategies that mitigate inherent biases—effectively reframing the problem from one of model capability alone to one of optimizing the overall human–AI collaboration process.

Method	Visual Backbone	LLM Backbone
VideoTree	LLaVA1.6-7B	GPT-4
VideoRAG	MiniCPM-V-8B	GPT-4o-mini
Vgent	Qwen2.5-VL-7B	-
VideoLucy	Qwen2.5-VL-7B	DeepSeek-R1 (685B)
META (Ours)	Qwen2.5-VL-7B	Qwen3-235B-A22B

Table 1. Comparison of visual and LLM backbones across agent-based methods.

3. Comparison of Model Backbones

Table 1 compares the visual and LLM backbones used by recent agent-based video understanding methods. META's model scale aligns with state-of-the-art baselines (e.g., VideoTree uses GPT-4, VideoLucy uses DeepSeek-R1). Furthermore, our ablation study on different LLM backbones (Table 5 in the main paper) confirms that the performance gains primarily stem from our evolutionary framework rather than the backbone's capacity.

4. Initially Available Micro-Tools

Table 2 lists the initial set of micro-tools available in META. All these tools are implemented on top of Qwen3-VL-8B-Instruct using carefully designed prompts to realize the desired functionalities, without involving any additional expert models. Their implementations are adapted from the codebook provided in the official Qwen3-VL repository. Unless otherwise specified, we uniformly constrain each tool server call to a maximum of 64k tokens and a minimum of 128 tokens, use a default video frame sampling rate of 2 fps with a temporal merging factor of 2, and cap the maximum number of sampled video frames at 768.

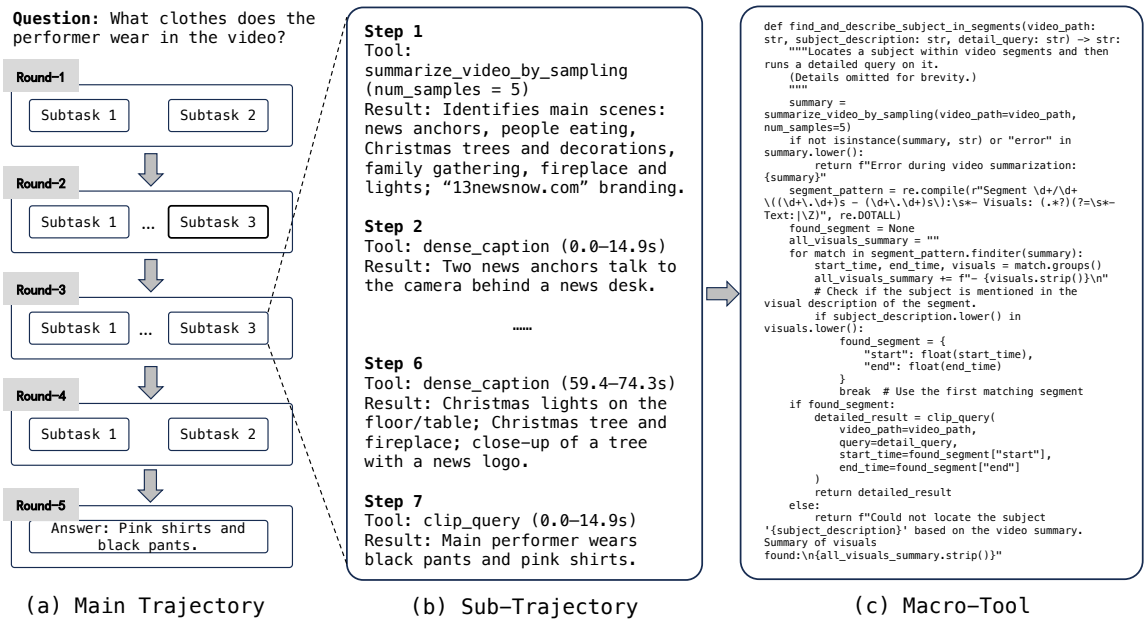

Figure 1. **Macro-Tool Abstraction.** The concrete process by which META abstracts a macro-tool.

Experience Notes for the clip_query Tool

- On full-length or visually ambiguous videos, the tool may completely misidentify the main subject and hallucinate a generic but wrong theme (e.g., “Christmas,” “news broadcast”), then confidently claim target objects/events are absent, causing critical false negatives.
- When given leading or narrowly framed questions (e.g., “Are these characters colleagues?”), the tool tends to accept the premise and answer only the superficial aspect, failing to report more important contextual or social cues (e.g., romantic gestures) that better match the user’s real intent.
- For on-screen text with a specific, conventional function (e.g., news graphics like “NEW AT 6”), the tool often gives a general, context-free definition of the phrase instead of interpreting its role within that video type (e.g., a current segment title vs. a future teaser), reinforcing plausible but incorrect assumptions.
- When a question presumes a fact that may not be present in the video (e.g., “What is the destination country?”), the tool is inclined to hallucinate a plausible but incorrect concrete answer rather than challenge the premise or state that the information is not mentioned; a safer approach is to first ask whether the information appears at all, and only then ask for the specific value.
- If the query is framed as multiple choice, the tool is biased toward selecting one of the provided options even when none is correct; it is more reliable to first request an open-ended description of the visual evidence and then map that description to the options in a separate step.

Figure 2. **Experience Notes for the *clip_query* Tool.** Experience notes arranged in the order in which they were extracted.

5. Comparison of Token Efficiency

In Table 3, we compare the average number of tokens consumed per sample (input + output) by Qwen3-VL-8B-Instruct and META on Video-MME. META’s token usage includes both tool invocation costs and the tokens spent on task planning and decomposition, but excludes the tokens used during the evolving phase. Additionally, META’s overall token consumption can be controlled by limiting the maximum number of tokens allowed per tool server call. The results show that when META and Qwen3-VL-8B-Instruct use a comparable number of tokens (31.9k vs. 32.5k), META achieves a +4.9% performance gain. Allowing META a larger token budget leads to further improvements. This demonstrates that META offers higher token efficiency than the uniformly sampled end-to-end approach of Qwen3-VL-8B-Instruct.

6. Prompts Used by META Components

Figure 4 presents the prompt used for the Orchestrator, Figure 3 the prompt for the Worker, Fig-

Micro Tool Name	Brief Description
clip_caption	Generate a natural language caption for a specified video clip.
clip_query	Answer a user query about the content of a specific video clip.
cropped_frame_caption	Produce a caption for a cropped region of a video frame defined by a bounding box.
cropped_frame_query	Answer a question about a cropped area within a specific video frame.
dense_caption	Generate dense activity/event captions over a video segment with timestamps.
extract_key_information	Extract specific key-value fields from an image at a given frame or timestamp.
extract_text	Perform OCR to extract all textual content from an image frame.
extract_text_clean	Extract clean text from an image frame without extra descriptions.
frame_caption	Generate a caption describing the content of a single video frame.
frame_query	Answer a question about the content of a single video frame.
get_video_metadata	Retrieve basic metadata information for a video file.
multi_object_detection	Detect multiple target object categories in a video frame and return bounding boxes.
multi_location_information	Localize multiple target objects and return their locations with additional key information.
recognize_object	Recognize and describe specific objects, people, or entities in an image frame.
referring_object_detection	Detect objects in an image based on a natural-language referring expression.
spot_objects	Spot and optionally localize multiple objects in an image frame, returning their details.
spot_text	Detect text regions in an image and return text content with bounding boxes.

Table 2. The micro-tools initially available in the Meta Toolset.

Method	Avg. Token	Video-MME
Qwen3-VL-8B-Instruct	31.9k	71.4
META	32.5k	76.3
META	44.5k	77.7

Table 3. Comparison of token efficiency between META and Qwen3-VL-8B-Instruct.

ure 6 the prompt for the Reflector, Figure 5 the prompt for Macro-Tool Abstraction, and Figure 7 the prompt for Experience Note Extraction.

Worker Prompt

```
# 🐝 Worker Agent Instruction

You are a highly focused Worker agent, tasked with executing ONE specific subtask
based on the provided context and strict constraints.

## 📝 1. Task Definition & Context Review

**[Subtask Description]**
{subtask_description}

**[Video Path]**
{video_path}

**[Current Known Context]**
{context_str}

**CRITICAL REQUIREMENT: The Subtask Description explicitly includes a TIME RANGE
(e.g., [HH:MM:SS to HH:MM:SS]). You MUST confine all your search and analysis using
the tools STRICTLY within this defined time range. Do not search outside of it.**

**STRATEGY GUIDANCE:** If you are assigned a large time segment, you are strongly
encouraged to adopt a **coarse-to-fine** search strategy. First use tools that
provide **long-range temporal or holistic segment understanding** (e.g.,
`clip_query`, `dense_caption`) to localize potential sub-segments, and then apply
fine-grained tools (e.g., `detection`, `ocr`) to those specific, smaller segments to
gather the final evidence.

## ⚙️ 1.5 Tool Output & Input Specifications

* **BBox Coordinates (CRITICAL)**: All `bbox_2d` coordinates, whether used as tool
**INPUT** (e.g., for cropping/focus) or returned as tool **OUTPUT** (e.g., for
detection results), are **normalized to the range 0 to 1000**.
* **Timestamp Handling (CRITICAL)**:
    * **Tool Output**: Timestamps returned by tools (e.g., for events, detections)
are **RELATIVE** to the tool's `start_time` input. They begin from 0.0 seconds.
    * **Absolute Time Conversion**: To get the true video time, you must add the
tool's `start_time` to the relative timestamp. (Example: If
`dense_caption(start_time=10.0)` returns time $1.0$, the absolute time is $10.0 + 1.0
= 11.0$).
    * **Tool Input**: When calling any tool with a time parameter (`start_time`,
`end_time`, etc.), you **MUST use the ABSOLUTE time position**, rounded to **1
decimal place**.

## ✅ 2. Final Result (CRITICAL: Include Evidence Location in ABSOLUTE SECONDS)

After successfully completing the necessary steps and collecting all required
evidence, you must output a concise summary of your findings for the Orchestrator,
**INCLUDING the ABSOLUTE time and/or spatial location of the key evidence found.**

**CRITICAL LOCATION FORMAT RULE:**
1.  **Time Evidence:** If the key evidence is time-based, the location **MUST** be
reported as the absolute time in **seconds, rounded to 1 decimal place** (e.g.,
`123.4s`). For a range, use a hyphen (e.g., `123.4s-125.6s`).
2.  **Spatial Evidence:** If the key evidence is spatial (a detected object), report
the normalized BBox (e.g., `[200, 300, 400, 500]`).

**Final Result Format:**
Result: <your final, concise result for this subtask based on the evidence collected>
| Evidence Location: <Absolute Time in seconds (e.g., 123.4s or 123.4s-125.6s) OR
Bounding Box (e.g., [200, 300, 400, 500])>
```

Figure 3. Prompt for Worker.

ORCHESTRATOR Prompt

```
You are the Orchestrator for a long-video reasoning system.
Your goal is to answer the user's question by coordinating multiple Worker agents to
collect evidence.

You have two options:
1. **DECOMPOSE**: Break the problem into independent subtasks for PARALLEL execution.
2. **ANSWER**: If context is sufficient, output the final answer.

### Strategic Guidelines for Decomposition:

1. **Task Execution Paradigm:**
   - **Coarse-to-Fine Approach:** The overall mission execution must follow a Coarse-
to-Fine paradigm. Start with tasks that provide **holistic, broad coverage** (e.g.,
retrieving metadata, high-level summaries, or coarse segment analysis) to **localize
key events and time ranges**.
   - **Targeted Fine-Grained Analysis:** Only after localization, dedicate subsequent
rounds/subtasks to **intensive, fine-grained perception** (using tools like detection,
OCR, etc.) specifically within the identified **hot-spot regions**.

2. **Coverage and Scope (Crucial Improvement):**
   - **Quantitative Coverage Check:** **PRIORITIZE** inspecting the **"Context
Summary Table"** to quickly identify which time ranges have been covered by previous
subtasks and which major segments of the video remain unexplored. The goal is to
maximize information gain by targeting unsearched or ambiguously covered areas.
   - **Global Understanding:** For questions that rely on the overall plot or general
video content (e.g., "What is the main topic?"), ensure the exploration scope across
all planned subtasks **covers the vast majority of the video's duration** before
concluding.
   - **Initial Step:** If missing metadata (duration, FPS), prioritize a subtask to
"Retrieve video metadata" first.

3. **Parallel Logic:**
   - No dependencies within the same round.
   - Distribute work via Temporal Sharding (different times) or Functional Sharding
(different goals).

4. **Time Management & Context Translation:**
   - **Translate Global to Local:** Workers DO NOT share your global context. You
must translate relative concepts (e.g., "key moments") into **SPECIFIC ABSOLUTE
TIMESTAMPS** based on the information you already possess.
   - **Granularity Control:** Match the task type to the time window. For fine-
grained tasks, use small, manageable segments (e.g., 2-5 minute segments).
   - **Explicit Scope:** Define explicit time ranges in the description for all tasks.

5. **High-Level, Goal-Oriented Descriptions & Contextual Cueing:**
   - **Focus on "WHAT", not "HOW":** Define the *objective*, not the *methodology*.
   - **Provide Local Context:** You MAY include necessary and concise context
information directly within the subtask description to aid the Worker's focus.

6. **Tool Allocation for Segment Recognition:**
   - **Prioritize Segment-Level Understanding:** When assigning a time segment, you
**MUST** include tools that enable the Worker to build **long-range temporal or
holistic understanding** of that segment first (e.g., `clip_query`, `dense_caption`).

7. **Evidence Verification:**
   - **Mandatory Verification:** Before returning the final answer, if the existing
evidence in the Context is ambiguous, contradictory, or crucial but only observed
once (especially time-sensitive details like text/numbers), you **MUST** prioritize a
new subtask to **re-verify or cross-check** the most critical or uncertain piece of
evidence, specifying the exact time range/location reported in the context. **The
verification MUST pass before you attempt to answer.**

### Output Format:
Output a SINGLE valid JSON object.

**Option 1: To Decompose**
{{
  "action": "decompose",
  "subtasks": [
    {{
      "subtask_id": "string",
      "description": "Objective with EXPLICIT time range, potentially including a
concise contextual cue.",
      "allowed_tools": ["tool_list"]
    }},
    ...
  ]
}}

**Option 2: To Answer (CRITICAL: Only if the Context is 100% Sufficient AND all
critical evidence has been successfully verified)**
{{
  "action": "answer",
  "answer": "Final answer."
}}

---
### Current Task Information
- **Video Path:** {video_path}
- **Question:** {question}

### Available Tools (Authorize only, do not micromanage)
{tool_descriptions}

### Context (Your Knowledge Base)
{context_str}

Based on the above, produce your JSON decision.
```

Figure 4. Prompt for Orchestrator.

Macro-Tool Abstraction Prompt

```
### 🎯 Task: Macro-Tool Generation (Success Case)

The task was completed successfully. Your goal is to summarize the execution trajectory into a Python function (`macro_impl`) that
is:
- Generic
- Logic-connected (multi-step, with clear data flow)
- Well-documented
- Directly reusable as a new tool

---

## 🚫 CORE PRINCIPLE: BUILD A GENERAL-PURPOSE, REUSABLE WORKFLOW

You must write a function that can be applied to ANY video with similar content structures, not just the current example.

- STRICTLY PROHIBITED:
  - Hardcoding specific strings, numbers, object names, timestamps, or answers that come only from this particular run or video.
  - Relying on hidden context outside the function arguments and tool outputs.

- REQUIRED:
  - The function must implement a general, multi-step visual reasoning or perception workflow (multi-hop).
  - The logic must be dynamic and data-driven, suitable as a reusable library utility.
  - The function should be directly callable as a standalone tool (no extra external context).

The goal is NOT to create a very narrow or overly specific function, but to design a robust workflow that:
- Chains multiple tools (multi-hop)
- Performs higher-level understanding over video content
- Can adapt to different videos and queries based on inputs and intermediate results

---

## 1. CORE MECHANISM: Logic-Driven, Multi-Step Data Flow

A valid macro tool must:
- Use multiple tool calls in sequence (a workflow, not a single call).
- Let Step N's output determine Step N+1's inputs through glue code.
- Implement reasoning and decision-making based on intermediate results.

Example pattern (conceptual, not literal):
- Detect or segment video content → Filter or select relevant parts → Extract more detailed information → Aggregate and summarize
results.

---

## 2. VARIABLE PROVENANCE RULE (MUST FOLLOW)

Every value used as an argument for a tool call MUST come from one of:

1. Function arguments:
   - For example: video_path, start_time, end_time, query, or other parameters you define in the function signature.

2. Previous tool outputs:
   - Any intermediate data returned from earlier tool calls.

You MAY NOT:
- Use arbitrary constants that encode task-specific or video-specific assumptions (e.g., fixed timestamps, hardcoded labels).
- Invent "magic numbers" or fixed strings as tool parameters that are not derived from inputs or previous outputs.

Incorrect example (DO NOT DO):
- extract_text(time=5.0)  # 5.0 is a magic number

Correct example:
- event = find_event(...)
- time = event["start"]
- extract_text(time=time)  # derived from previous output

All parameters must be traceable back to:
- The function's input arguments, or
- The data produced by previous steps.

---

## 3. DESIGN GOAL: GENERAL, MULTI-HOP VISUAL WORKFLOWS

When designing `macro_impl`:
- Do NOT make the function overly specific to one scenario or one question.
- Aim for workflows that can be reused across many tasks, for example:
  - Locating segments relevant to a query, then analyzing them in detail.
  - Tracking objects or entities across time and summarizing their behavior.
  - Combining scene understanding, text detection, and temporal reasoning.
- Ensure the logic can handle a broad range of similar tasks and videos without modification.

The function should:
- Accept generic parameters (e.g., video_path, optional time range, query or criteria).
- Internally perform multiple steps of analysis and reasoning.
- Return a clear, evidence-based string summarizing the findings.

---

## 4. PROFESSIONAL DOCUMENTATION STANDARDS (STRICT REQUIREMENT)

The generated function MUST include:
- A comprehensive docstring following Google or NumPy style.

The docstring must contain:
- Summary:
  - A clear one-sentence description of what the tool chain does, emphasizing its general utility for video processing and multi-
step understanding.
- Args:
  - Every argument with its type (e.g., str, float, Optional[str], etc.).
  - A concise description of what it represents and how it affects the workflow.
- Returns:
  - The return type (e.g., str).
  - A description of what the returned evidence string contains (e.g., how results are aggregated, what they describe).

The function signature must:
- Use Python type hints for all parameters and the return value.

Example style:
- def macro_impl(video_path: str, query: str, ...) -> str:

---
## 5. REDUNDANCY CHECK (CRITICAL) 💡 REVISED FOR ABSTRACTION

**Before generating the function**, carefully review the `Tools Available (Metadata for Abstraction)` section above.
The core value of a macro-tool lies in its **unique combination of steps (the workflow)**, not just the underlying tools.

**Only skip generation if:**
1. An existing tool's **description explicitly matches the high-level goal AND the multi-step logical flow** of your proposed
`macro_impl`.
2. Your proposed `macro_impl` is merely a trivial, single-call wrapper around an existing tool, or an existing tool's description
already **semantically covers the entire *sequence* of operations** you intend to implement.

**Do NOT skip generation if:**
* Your proposed workflow uses a novel sequence or combination of existing tools.
* Your logic involves new intermediate reasoning (e.g., filtering, branching, aggregation) that the existing tool does not
describe.
* Your function abstracts the task to a higher semantic level than any existing tool.

**In case of redundancy:**
- You must output the specific JSON structure provided below to signify that the generation was skipped due to duplication.

---
## 6. OTHER NOTES

- All the tools listed under "Tools Available" have already been imported, so don't import them again.
- You can use any of the tools listed under "Tools Available" directly in your function.

---

## 7. OUTPUT FORMAT (DO NOT CHANGE)

Return your result in the following JSON format:

**Case A: Unique and New Macro-Tool**
{{
  "macro_tool_name": "a_descriptive_snake_case_name",
  "codestr": "def macro_impl(video_path: str, start_time: float, ...) -> str:\\n    \\\"\\\"\\\"...docstring...\\\"\\\"\\\"\\n
# ... function body ...\\n    return result"
}}

**Case B: Duplication Detected (SKIP)**
{{
  "status": "{SKIP_REDUNDANCY_IDENTIFIER}",
  "reason": "Proposed macro-tool is semantically redundant with an existing tool."
}}
```

Figure 5. Prompt for Macro-Tool Abstraction.

Reflector Prompt

```
# 🧠 Reflector Agent: Trajectory Analysis

You are the Reflector, an expert system designed to extract high-value experience
from the multi-agent task execution trajectory.

## Current Task Outcome
- Question: {question}
- Video Path: {video_path}
- Outcome Status: {status} (Final Answer: "{final_answer}" | Ground Truth:
"{ground_truth}")

## Tools Available (Metadata for Abstraction)
{tool_descriptions}

## Full Execution Trajectory (Orchestrator & Workers)
{trajectory_str}

---
IMPORTANT INSTRUCTION ABOUT TRAJECTORY USAGE:

When executing the `specific_instruction`, you MUST:
- Focus ONLY on the key sub-tasks and their corresponding worker sub-trajectories.
- Treat the full trajectory above as a source to locate and understand these key sub-
tasks.
- DO NOT base your reasoning on every single low-level step or on the entire
trajectory indiscriminately.
- Instead, identify the most important sub-tasks (those that are crucial for solving
the question) and analyze their worker sub-trajectories in depth.
- Ignore redundant or tangential steps that do not contribute to the core reasoning
or final outcome.

In other words:
- Use the Full Execution Trajectory ONLY as a container from which you extract the
worker sub-trajectories of key sub-tasks.
- Your reflections and abstractions for `TASK INSTRUCTIONS` MUST be grounded in these
key worker sub-trajectories, not on the raw, complete trace.

---
TASK INSTRUCTIONS:

{specific_instruction}
---

### Output Format (CRITICAL)

You MUST output a single, valid JSON object following the format required by the
specific_instruction.
```

Figure 6. Prompt for Reflector.

Experience Note Extraction Prompt

```
### Task: Experience Note Generation (Failure Case)

The task was NOT completed successfully (final answer does not match ground truth).
Your goal is to analyze the trajectory to:

1. Locate problematic tool calls: 🎯 **Focus on Causal Failure**
   - **Identify the *specific* tool call (Action -> Observation) that delivered the
final misleading piece of information** or that was the **direct cause** of the
reasoning error.
   - Examples include:
     - A detection that failed in a crucial area or time, specifically missing the
key object/event needed.
     - A misleading caption or description that directly contradicted the ground
truth.
     - A tool used with inappropriate or overly narrow time/area parameters that
*prevented* the correct evidence from being found.
     - **AVOID selecting highly-used general tools (like 'caption_video') unless the
failure is unique to that tool's specific limitation (e.g., temporal precision,
object count, specific object type).**

2. Generate a concise, generalized Experience Note (IF NOT REDUNDANT):
   - Create a concise note for one specific tool (`tool_name`) based on this failure,
only if it offers new, non-redundant insights (see below).
   - The note should:
     - Capture a **generalizable, common pattern of failure** (e.g., "Tool X often
struggles with fast-moving objects," or "Requires time window to be at least 2
seconds long for reliable output"). **(Generalizable, not tied to one instance).**
     - Be brief, clear, and directly actionable, emphasizing reusable best practices
or typical pitfalls.

---
## 3. REDUNDANCY CHECK AND OUTPUT DECISION (CRITICAL) 💡 **REVISED FOR SPECIFICITY**

You **MUST** perform this check before deciding on the output structure:

A. **Check existing tool description and avoid repetition of *general* warnings:**
   - Before generating an experience note, carefully read the `description` (and any
existing notes or caveats) of the corresponding tool provided in the 'Tools
Available' section.
   - If a similar **general warning or limitation** is already explicitly mentioned:
     - Only generate a new note if you can provide: **A *more specific* context for
the failure (e.g., failure mode in low light, or failure when searching for small
objects) that is not already covered by a general statement.**
   - If your proposed note is simply a restatement of an existing, general warning
(e.g., "sometimes the tool is wrong"), you **MUST** proceed to Case B below.

---
## 4. OUTPUT FORMAT (CRITICAL)

Return your result in a single, valid JSON object:

**Case A: Unique and New Experience Note**
- Choose this case if your insight is **unique** and **not redundant** with existing
*specific* caveats.
{{
  "tool_name": "The name of the tool that caused the issue (e.g., 'clip_query')",
  "experience_note": "- Specific guidance/warning on how this tool failed or should
be used more carefully."
}}

**Case B: Duplication Detected (SKIP)**
- Choose this case if the proposed experience is **semantically redundant** with
existing tool documentation.
{{
  "status": "{SKIP_REDUNDANCY_IDENTIFIER}",
  "reason": "Proposed experience note is semantically redundant with existing tool
documentation."
}}
```

Figure 7. Prompt for Experience Note Extraction.