V4.0 Unified Specification: Teach by Doing (TbD) - "Native Insight"

Status: Approved for Build **Version:** 4.0.0 **Author:** Greg Burns / System Architect **Date:** November 23, 2025 **Core Technology:** Google Vertex AI (Gemini 2.5 Pro)

Part 1: Executive Summary & Strategy

1.1 The Pivot

Previous versions (V1-V3) relied on a "Snapshot" architecture: chopping video into images and analyzing them in isolation. While this provided structure, it failed to capture context, intent, and audio cues.

V4.0 introduces "Native Insight." Instead of feeding the model scraps (images), we ingest the entire video file into **Gemini 2.5 Pro**. We leverage the model's massive context window and native multimodal capabilities to generate a holistic understanding of the procedure, identical to how a human expert watches a tutorial.

1.2 Strategic Goals

1. **Contextual Mastery:** Achieve human-level understanding of *why* an action is happening (e.g., "User clicked 'Personalize' to access background settings," not just "User clicked text").
2. **Audio Integration:** Utilize the video's audio track implicitly to disambiguate actions (e.g., hearing "I'm going to change the wallpaper").
3. **Simplified Pipeline:** deprecate complex local segmentation logic (PySceneDetect, Optical Flow) in favor of LLM-driven segmentation.

Part 2: Software Requirements Specification (SRS)

2.1 Functional Requirements (FRs)

FR-01: Native Video Ingestion

- The system shall accept a video file URI (`gs://`) and pass it directly to the Vertex AI Gemini 2.5 Pro model.
- The system shall NOT split the video into local frames or chunks prior to analysis.

FR-02: Semantic Pathway Generation

- The system shall prompt the model to generate a complete, chronological list of steps (ActionNodes) covering the entire video duration.
- The prompt shall strictly enforce the output format as a valid JSON object conforming to the PAD Schema.

FR-03: Coordinate Refinement (The Hybrid Approach)

- Since LLMs effectively understand "what" and "when" but struggle with pixel-perfect "where" (coordinates), the system shall perform a **post-processing pass**.
- For each timestamp identified by Gemini, the system shall extract the specific frame and run **Spatial OCR** (reusing V3 logic) to validate the text and retrieve the precise bounding box `[x, y, w, h]` for the UI element.

FR-04: Structured Output

- The final output shall be a `Pathway.json` file containing:
  - `semantic_description`: High-quality natural language summary (from Gemini).
  - `ui_region`: Precise coordinates (from OCR refinement).
  - `action_type`: Context-aware action classification (e.g., click, drag, type).

2.2 Non-Functional Requirements (NFRs)

NFR-01: Model Versioning

- The system MUST explicitly target `gemini-2.5-pro` (or the currently active stable release alias). It shall NOT use deprecated preview snapshots.

NFR-02: Processing Latency

- The system shall process a video in under **1.5x** the video duration (e.g., a 2-minute video processes in <3 minutes). This is achievable by removing the overhead of frame-by-frame analysis loops.

NFR-03: Fault Tolerance

- If the OCR refinement fails to find a bounding box for a step generated by Gemini, the system shall retain the step with a null/default region `[0,0,0,0]` rather than discarding the semantic data.

Part 3: Technical Design Document (TDD)

3.1 Architecture Diagram

The architecture simplifies significantly. We remove the complex "Loop" in the Worker and replace it with a single "Call & Refine" pattern.

1. **Dispatcher:** Queues Task (Unchanged).
2. **Worker (Ingest):** Downloads Video.
3. **Worker (Brain):** Sends `gs://` Video URI to **Gemini 2.5 Pro**.
4. **Worker (Refinement):**
    - Parses Gemini's JSON response.
    - Extracts frames at specific timestamps.
    - Runs Local OCR to get bounding boxes.
5. **Worker (Output):** Uploads final JSON to GCS.

3.2 Implementation Details

Service A: The GenAI Service (`genai.py`)

We switch from `Image` parts to `Part.from_uri` to handle video.

Python
```
# V4 Implementation Spec for genai.py
import vertexai
from vertexai.generative_models import GenerativeModel, Part, HarmCategory,
HarmBlockThreshold

MODEL_NAME = "gemini-2.5-pro"

def analyze_video_context(video_uri: str) -> str:
    """
    Sends the video URI directly to Gemini 2.5 Pro.
    Returns raw JSON string of the pathway.
    """
    model = GenerativeModel(MODEL_NAME)

    video_part = Part.from_uri(
        uri=video_uri,
        mime_type="video/mp4"
    )
```

```
    prompt = """
    You are an expert software documentation agent. Watch this video and extract a structured timeline of user actions.

    Output a valid JSON array of objects. For each step, provide:
    1. "timestamp": The exact time (in seconds) the action occurs.
    2. "action_type": One of [click, double_click, type, scroll, drag].
    3. "target_text": The visible text on the button or element being interacted with.
    4. "description": A precise, human-readable sentence explaining WHAT the user did and WHY (based on visual and audio context).

    Example Format:
    [
      {"timestamp": 2.5, "action_type": "click", "target_text": "Settings", "description": "User clicks Settings to open the configuration menu."},
      ...
    ]
    """

    response = model.generate_content(
      [video_part, prompt],
      generation_config={"response_mime_type": "application/json"}
    )

    return response.text
```

Service B: The Pipeline Orchestrator (`pipeline.py`)

The pipeline becomes a "Manager" rather than a "Grinder."

Python

```
# V4 Implementation Spec for pipeline.py
import json
from app.services.genai import analyze_video_context
from app.services.ocr import run_ocr_on_specific_frame

def build_pathway(video_path_local, video_uri_gcs):
    # 1. The Brain Pass (Gemini 2.5 Pro)
    # We pass the GCS URI because Vertex AI reads directly from Cloud Storage
    raw_json = analyze_video_context(video_uri_gcs)
    ai_steps = json.loads(raw_json)

    final_nodes = []
```

```
    # 2. The Refinement Pass (Local OpenCV/OCR)
    cap = cv2.VideoCapture(video_path_local)

    for step in ai_steps:
        # Go to the timestamp identified by Gemini
        timestamp = step['timestamp']
        target_text = step['target_text']

        # Extract that specific frame
        frame = get_frame_at_time(cap, timestamp)

        # Run Spatial OCR to find WHERE that text is
        # We look for 'target_text' specifically
        bounding_box = find_text_location(frame, target_text)

        # Merge AI Intelligence with CV Precision
        node = ActionNode(
            timestamp_start=timestamp,
            description=step['description'], # The high-quality context
            ui_element_text=target_text,
            ui_region=bounding_box,          # The precise pixels
            action_type=step['action_type']
        )
        final_nodes.append(node)

    return Pathway(nodes=final_nodes)
```

3.3 Data Schema (PAD v0.4)

We standardize on the schema that supports the rich descriptions.

JSON


```
{
  "pathway_id": "uuid",
  "nodes": [
    {
      "timestamp_start": 12.5,
      "action_type": "click",
      "ui_element_text": "Personalize",
      "ui_region": [100, 200, 50, 20],
      "description": "The user clicks 'Personalize' to access the background settings menu.",
      "source": "gemini-2.5-pro-native"
    }
```

```
  ]
}
```

Part 4: Implementation Plan

Phase 4.1: The "Brain" Swap

1. **Refactor `genai.py`:** Update to accept `gs://` URIs and use the master prompt.
2. **Refactor `pipeline.py`:** Remove `PySceneDetect`. Implement the "Call & Refine" logic.
3. **Dependency Check:** Ensure `google-cloud-aiplatform>=1.60.0` is locked in `requirements.txt`.

Phase 4.2: Verification

1. **Test Asset:** `videoplayback4.mp4` (The Windows Background tutorial).
2. **Success Criteria:**
    - The JSON contains the contextual description: *"User clicks Personalize..."* (instead of "Scroll").
    - The JSON captures the audio intent (e.g., *"User demonstrates changing to a solid color"*).
    - The `ui_region` is not null (coordinates are successfully backfilled).