V7.0 Unified Specification: Teach by Doing (TbD) - "Depth & Defense"

Status: Proposed **Version:** 7.0.0 **Author:** Greg Burns **Date:** November 25, 2025 **Core Upgrades:** Human-in-the-Loop Verification, 3D Depth Estimation, Anomaly Detection

Part 1: Executive Summary & Strategy

1.1 The Pivot: From Observation to Verification

V6 successfully established a fully automated "Native Insight" pipeline that generates structured data from video. However, this data is currently "unsupervised." If the AI hallucinates a step or mis-locates a button, the error propagates downstream to critical systems (like robots).

[Image of V7 Architecture Diagram]

V7.0 introduces "Defense Layers":

1. **Physical Awareness:** Adding a Z-axis (Depth) to the 2D vision data to prepare for robotic interaction.
2. **Active Verification:** A "Human-in-the-Loop" frontend to validate and correct AI outputs before they are finalized.
3. **Risk Analysis:** A dedicated AI agent that flags procedural anomalies against a known "Golden Standard."

1.2 Strategic Goals

1. **Trust & Safety:** Ensure no "hallucinated" instruction ever reaches a robot or execution agent.
2. **3D Readiness:** Move from "Screen Coordinates" (pixels) to "Spatial Coordinates" (X, Y, Z) relative to the user.
3. **Quality Assurance:** Create a feedback loop where human corrections retrain the underlying models.

Part 2: Software Requirements Specification (SRS)

2.1 Functional Requirements (FRs)

FR-01: 3D Depth Estimation (Service D Upgrade)

- **Requirement:** The Object Detector (Service D) MUST generate a relative depth map for the active region.
- **Output:** The `ui_region` shall be expanded or supplemented to include a `z_depth` value (0.0 - 1.0), indicating how "deep" into the scene the interaction occurred.
- **Rationale:** Required for robotic arms to know "how far" to reach.

FR-02: Human-in-the-Loop Verification (Frontend)

- **Requirement:** The system shall provide a "Validation Station" interface (Streamlit/React).
- **Capabilities:**
  - Load a raw `Pathway.json` and its source video side-by-side.
  - Allow users to **Edit** text descriptions.
  - Allow users to **Resize/Move** bounding boxes on the video frame.
  - Allow users to **Approve/Reject** entire nodes.
- **Output:** A signed/verified version of the pathway (e.g., `pathway_verified.json`).

FR-03: Risk & Anomaly Detection (Service E)

- **Requirement:** A new microservice ("Risk Agent") shall analyze the `temporal_context_vector` of a newly generated pathway.
- **Logic:** Compare the new vector against a database of "Golden Path" vectors (known good procedures).
- **Output:** A `risk_score` (0.0 - 1.0) and `anomaly_flag` (Boolean) appended to the node.

FR-04: Vector Database Integration

- **Requirement:** The system shall persist "Golden Vectors" in a searchable Vector Database (e.g., Pinecone, Milvus, or FAISS-on-GCS).

2.2 Non-Functional Requirements (NFRs)

- **NFR-01 (Latency):** Depth estimation must not add more than 500ms per node to processing time.
- **NFR-02 (Usability):** The Validation Station must support "Keyboard-First" workflows (e.g., space to play/pause, tab to next node) for rapid QA.

Part 3: Technical Design Document (TDD)

3.1 Architecture: The V7 Stack

Service	Name	New/Modified	Role
Service A	`tbd-dispatcher`	Unchanged	API Gateway.
Service B	`tbd-worker`	Modified	Orchestrator. Now calls Service E and handles new schema fields.
Service C	`tbd-temporal-encoder`	Unchanged	Memory Engine (LSTM).
Service D	`tbd-object-detector`	**Major Upgrade**	Now runs **YOLOv8 + DepthAnything**. Returns (x, y, w, h, z).
Service E	`tbd-risk-agent`	**NEW**	Anomaly Detector. Python/FastAPI + Scikit-Learn/FAISS.
Frontend	`tbd-validation-station`	**NEW**	Streamlit/React app for human verification.

3.2 Schema Updates (PAD v0.6)

The data contract expands to support depth and verification status.

```
class ActionNode(BaseModel):
    # ... existing V6 fields ...

    # V7: Spatial Depth
    ui_depth_z: float = Field(0.0, description="Relative depth estimation (0.0=close, 1.0=far)")

    # V7: Verification Metadata
    is_verified: bool = Field(False, description="Has a human reviewed this node?")
    verified_by: Optional[str] = Field(None, description="User ID of reviewer")

    # V7: Risk Assessment
    risk_score: float = Field(0.0, description="Anomaly score (distance from Golden Path)")
    anomaly_flag: bool = Field(False, description="True if risk_score > threshold")
```

3.3 Service D Implementation Strategy (Depth)

We will integrate a lightweight Monocular Depth Estimation model alongside YOLO.

- **Model Selection:** `Depth-Anything-Small` (ONNX format) or `MiDaS v2.1 Small`.
- **Pipeline:**
    1. Receive Frame.
    2. Run YOLO -> Get Bounding Box `[x, y, w, h]`.
    3. Run Depth Model -> Get Depth Map `(H, W)` array of floats.
    4. **Crop:** Extract the depth values corresponding to the YOLO Bounding Box.
    5. **Average:** Compute the mean depth intensity of that cropped region. This is `ui_depth_z`.

```
# Pseudo-code for Depth Extraction
depth_map = depth_model.infer(frame) # Returns 0.0-1.0 map
roi = depth_map[y:y+h, x:x+w]
ui_depth_z = np.mean(roi)
```

3.4 Service E Implementation Strategy (Risk Agent)

A new microservice dedicated to quality control using vector similarity.

- **Input:** `temporal_context_vector` (512 floats) from Service C.
- **Storage:** Local FAISS index file (for MVP) or Pinecone (Production).
- **Logic:**

1. Load "Golden Vectors" (embeddings of correct procedures).
2. Calculate **Cosine Similarity** between the input vector and Golden Vectors.
3. If `similarity < threshold` (e.g., 0.85), mark as `anomaly_flag = True`.

```
# Pseudo-code for Risk Analysis
input_vec = np.array(pathway.temporal_context_vector)
D, I = index.search(input_vec, k=1) # Find closest Golden Vector
similarity = 1 - D[0][0] # Assuming distance metric
if similarity < RISK_THRESHOLD:
   pathway.nodes[-1].anomaly_flag = True
```

3.5 Deployment Plan (V7 Rollout)

1. **Phase 7.1 (The Eyes Upgrade):** Update Service D with the Depth model. Redeploy.
2. **Phase 7.2 (The Guardrail):** Build and deploy Service E (Risk Agent).
3. **Phase 7.3 (The Integration):** Update Service B (Worker) to call Service E and populate the new schema.
4. **Phase 7.4 (The Interface):** Deploy the Validation Station frontend.