Deployment Manifest - TbD V3.0

Deployment Manifest - TbD V3.0 ("Insight")

Status: *Stable / Production Ready* ***Version:*** *3.0.1 (Async Patched)* ***Architecture:*** *Cloud Native (Cloud Run + Pub/Sub + Vertex AI)*

1. Directory Structure

Plaintext

```
tbd-v3/
├── deploy_v3.ps1          # Deployment Automation (PowerShell - Updated Permissions)
├── Dockerfile             # Container Definition (Python 3.10)
├── requirements.txt       # Dependencies (Pinned for Stability)
└── app/
    ├── __init__.py        # Empty package marker
    ├── main.py            # Router (Async Updated)
    ├── schema.py          # Data Models (PAD Schema v0.2)
    └── services/
        ├── __init__.py    # Empty package marker
        ├── dispatcher.py  # Service A: API & Pub/Sub Publisher
        ├── worker.py      # Service B: Async Worker Logic
        ├── pipeline.py    # V3 Orchestrator (Async + Buffer Fix)
        ├── vision.py      # V3 Eyes (SSIM & Optical Flow)
        ├── genai.py       # V3 Brain (Gemini 2.5 Pro)
        ├── ocr.py         # Spatial OCR
        └── segment.py     # Scene Detection
```

---

2. Root Configuration Files

`requirements.txt`

Includes the fix for the `GenerativeModel` *import error and the NumPy binary crash.*

Plaintext

```
# --- V1 Bridge Stack (Core Logic & API)
fastapi==0.104.1
uvicorn==0.24.0
pydantic==2.5.2
```

```
python-multipart
opencv-python-headless==4.8.1.78
pytesseract==0.3.10
scenedetect[opencv]==0.6.2

# --- V2 Cloud Native Stack (Infrastructure)
google-cloud-storage==2.14.0
google-cloud-pubsub==2.19.0

# --- V3 Insight Stack (New Vision & AI)
scikit-image==0.22.0
# BUMPED: Required for stable vertexai.generative_models support
google-cloud-aiplatform>=1.60.0
# PINNED: Prevent binary incompatibility with OpenCV/Scikit-Image (NumPy 2.0 crash fix)
numpy>=1.26.0,<2.0.0
```

Dockerfile

```
Upgraded to Python 3.10.
Dockerfile
FROM python:3.10-slim

# Set working directory
WORKDIR /usr/src

# Ensure Python logs are streamed directly to the terminal
ENV PYTHONUNBUFFERED=True

# 1. Install System Dependencies
# Added critical libraries for OpenCV headless and Tesseract
RUN apt-get update && \
    apt-get install -y tesseract-ocr libtesseract-dev \
    libgl1 libglib2.0-0 libsm6 libxext6 libxrender1 && \
    rm -rf /var/lib/apt/lists/*

# 2. Copy and Install Python Dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 3. Copy Application Code
COPY ./app /usr/src/app

# 4. Define Environment
# PORT is injected by Cloud Run, defaulting to 8080 if missing
ENV PORT=8080
```

```
# 5. Start the Application
CMD sh -c "uvicorn app.main:app --host 0.0.0.0 --port ${PORT}"
```

`deploy_v3.ps1`

Updated with `objectAdmin` (to fix delete/overwrite errors) and `aiplatform.user`.

```
PowerShell
# ====================================================
# TbD V3.0 "Insight" - Automated Deployment Script (PowerShell)
# ====================================================

# --- CONFIGURATION ---
# REPLACE WITH YOUR ACTUAL V2 PROJECT ID
$PROJECT_ID = "tbd-v2"
$REGION = "us-central1"
$REPO_NAME = "tbd-repo"
$IMAGE_NAME = "tbd-v3-engine"
$TAG = "v3.0.1"

# Resource Names
$TOPIC_NAME = "tbd-ingest-tasks"
$INPUT_BUCKET = "tbd-raw-video-$PROJECT_ID"
$OUTPUT_BUCKET = "tbd-results-$PROJECT_ID"
$DISPATCHER_SERVICE = "tbd-dispatcher"
$WORKER_SERVICE = "tbd-worker"

# Service Accounts
$DISPATCHER_SA = "tbd-dispatcher-sa"
$WORKER_SA = "tbd-worker-sa"
$SUBSCRIPTION_SA = "tbd-sub-invoker"

# ====================================================

Write-Host "--- STARTING V3 DEPLOYMENT FOR PROJECT: $PROJECT_ID ---"
-ForegroundColor Cyan
gcloud config set project $PROJECT_ID

# 1. Enable APIs (Added aiplatform for V3)
Write-Host "--- Enabling GCP APIs (including Vertex AI) ---" -ForegroundColor Green
gcloud services enable artifactregistry.googleapis.com cloudbuild.googleapis.com
run.googleapis.com pubsub.googleapis.com storage.googleapis.com aiplatform.googleapis.com

# 2. Create Storage Buckets
Write-Host "--- Creating GCS Buckets ---" -ForegroundColor Green
```

```
gsutil mb -l $REGION "gs://$INPUT_BUCKET/"
if ($LASTEXITCODE -ne 0) { Write-Host "Bucket might already exist, continuing..."
-ForegroundColor Yellow }
gsutil mb -l $REGION "gs://$OUTPUT_BUCKET/"
if ($LASTEXITCODE -ne 0) { Write-Host "Bucket might already exist, continuing..."
-ForegroundColor Yellow }

# 3. Create Pub/Sub Topic
Write-Host "--- Creating Pub/Sub Topic ---" -ForegroundColor Green
gcloud pubsub topics create $TOPIC_NAME
if ($LASTEXITCODE -ne 0) { Write-Host "Topic might already exist, continuing..."
-ForegroundColor Yellow }

# 4. Build & Push Container Image
Write-Host "--- Building V3 Container Image ---" -ForegroundColor Green
gcloud artifacts repositories create $REPO_NAME --repository-format=docker
--location=$REGION --description="TbD Engine Repository"
if ($LASTEXITCODE -ne 0) { Write-Host "Repo might already exist, continuing..."
-ForegroundColor Yellow }

$IMAGE_URI =
"$REGION-docker.pkg.dev/$PROJECT_ID/$REPO_NAME/$IMAGE_NAME`:$TAG"

# Submit build
gcloud builds submit --tag $IMAGE_URI .

# 5. Setup Service Accounts & IAM
Write-Host "--- Configuring IAM Service Accounts ---" -ForegroundColor Green
gcloud iam service-accounts create $DISPATCHER_SA --display-name "TbD Dispatcher"
if ($LASTEXITCODE -ne 0) { Write-Host "SA might already exist..." -ForegroundColor Yellow }
gcloud iam service-accounts create $WORKER_SA --display-name "TbD Worker"
if ($LASTEXITCODE -ne 0) { Write-Host "SA might already exist..." -ForegroundColor Yellow }
gcloud iam service-accounts create $SUBSCRIPTION_SA --display-name "PubSub Invoker"
if ($LASTEXITCODE -ne 0) { Write-Host "SA might already exist..." -ForegroundColor Yellow }

# Grant Dispatcher permission to Publish
gcloud pubsub topics add-iam-policy-binding $TOPIC_NAME `
    --member="serviceAccount:$DISPATCHER_SA@$PROJECT_ID.iam.gserviceaccount.com" `
    --role="roles/pubsub.publisher"

# Grant Worker permission to Read Input
gsutil iam ch
"serviceAccount:$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com:objectViewer"
"gs://$INPUT_BUCKET"
```

```
# [UPDATED] Grant Worker Object Admin on Output (Allows Overwrite/Delete)
gsutil iam ch
"serviceAccount:$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com:objectAdmin"
"gs://$OUTPUT_BUCKET"

# [NEW FOR V3] Grant Worker permission to use Vertex AI (Gemini)
Write-Host "--- Granting Vertex AI User Role to Worker ---" -ForegroundColor Green
gcloud projects add-iam-policy-binding $PROJECT_ID `
    --member="serviceAccount:$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com" `
    --role="roles/aiplatform.user"

# 6. Deploy Dispatcher Service
Write-Host "--- Deploying DISPATCHER Service ---" -ForegroundColor Green
gcloud run deploy $DISPATCHER_SERVICE `
    --image $IMAGE_URI `
    --region $REGION `
    --service-account "$DISPATCHER_SA@$PROJECT_ID.iam.gserviceaccount.com" `
    --allow-unauthenticated `
    --memory 2Gi `
    --cpu 1 `
    --set-env-vars "SERVICE_TYPE=dispatcher,GCP_PROJECT_ID=$PROJECT_ID" `
    --tag v3-stable

# 7. Deploy Worker Service (V3 Insight Engine)
Write-Host "--- Deploying WORKER Service (V3 Insight Engine) ---" -ForegroundColor Green
# Increased Memory to 4Gi
gcloud run deploy $WORKER_SERVICE `
    --image $IMAGE_URI `
    --region $REGION `
    --service-account "$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com" `
    --no-allow-unauthenticated `
    --memory 4Gi `
    --cpu 2 `
    --timeout 3600 `
    --set-env-vars "SERVICE_TYPE=worker,GCP_PROJECT_ID=$PROJECT_ID" `
    --tag v3-stable

# 8. Create Pub/Sub Push Subscription
Write-Host "--- Linking Pub/Sub to Worker ---" -ForegroundColor Green
Write-Host "Waiting for service propagation..." -ForegroundColor Cyan
Start-Sleep -Seconds 10
```

```
$WORKER_URL = gcloud run services describe $WORKER_SERVICE --region $REGION
--format 'value(status.url)'
if (-not $WORKER_URL) {
    Write-Error "Worker Service failed to deploy or URL retrieval failed."
    exit 1
}
Write-Host "Worker URL retrieved: $WORKER_URL" -ForegroundColor Cyan

# Allow Subscription SA to invoke Worker
gcloud run services add-iam-policy-binding $WORKER_SERVICE --region $REGION `

--member="serviceAccount:$SUBSCRIPTION_SA@$PROJECT_ID.iam.gserviceaccount.com"
`
    --role="roles/run.invoker"

# Create or Update Subscription
Write-Host "Configuring Subscription..." -ForegroundColor Cyan
gcloud pubsub subscriptions create tbd-worker-sub `
    --topic $TOPIC_NAME `
    --push-endpoint=$WORKER_URL `

--push-auth-service-account="$SUBSCRIPTION_SA@$PROJECT_ID.iam.gserviceaccount.co
m" `
    --ack-deadline=600
if ($LASTEXITCODE -ne 0) {
    Write-Host "Subscription exists, updating endpoint..." -ForegroundColor Yellow
    gcloud pubsub subscriptions update tbd-worker-sub `
        --push-endpoint=$WORKER_URL `

--push-auth-service-account="$SUBSCRIPTION_SA@$PROJECT_ID.iam.gserviceaccount.co
m"
}

Write-Host "================================================="
-ForegroundColor Cyan
Write-Host "V3 DEPLOYMENT COMPLETE" -ForegroundColor Cyan
Write-Host "Dispatcher URL: $(gcloud run services describe $DISPATCHER_SERVICE --region
$REGION --format 'value(status.url)')"
Write-Host "================================================="
-ForegroundColor Cyan
```

---

3. App Logic Files (app/)

app/main.py

Router updated to use `await` for the worker call.

Python

```
import uvicorn
import os
from fastapi import FastAPI, HTTPException
from app.schema import TaskPayload

SERVICE_TYPE = os.environ.get("SERVICE_TYPE", "dispatcher")
app = FastAPI(title=f"TbD Engine V3 - {SERVICE_TYPE.upper()} Service")

if SERVICE_TYPE == "worker":
    try:
        from app.services.worker import WorkerService
        worker = WorkerService()
    except ImportError as e:
        print(f"CRITICAL: Failed to import WorkerService. Check dependencies. {e}")
        worker = None

    @app.post("/")
    async def pubsub_trigger(data: dict):
        if not worker:
            raise HTTPException(status_code=500, detail="Worker service failed to initialize.")
        try:
            # UPDATED: Await the worker
            await worker.process_pubsub_message(data)
            return {"status": "Processing initiated"}, 200
        except Exception as e:
            print(f"Worker processing failed: {e}")
            raise HTTPException(status_code=500, detail=f"Worker failure: {e}")

elif SERVICE_TYPE == "dispatcher":
    from app.services.dispatcher import DispatcherService
    dispatcher = DispatcherService()

    @app.post("/submit", status_code=202)
    async def submit_video_task(payload: TaskPayload):
        try:
            task_id = dispatcher.submit_task(payload)
            return {"status": "Task accepted and queued", "task_id": task_id}
        except Exception as e:
            raise HTTPException(status_code=500, detail=f"Dispatch failed: {e}")

if __name__ == "__main__":
```

```
    uvicorn.run(app, host="0.0.0.0", port=8080)
```

app/schema.py

```
V3 Schema (v0.2).
Python
from pydantic import BaseModel, Field
from typing import List, Optional, Dict, Any

# --- V2 Task Payload ---
class TaskPayload(BaseModel):
    task_id: str = Field(..., description="UUID-V4 for the task")
    client_id: str = Field(..., description="Identifier for the client")
    gcs_uri: str = Field(..., description="gs://bucket/video.mp4")
    output_bucket: str = Field(..., description="GCS bucket for results")
    config: Dict[str, Any] = Field(default_factory=dict, description="Config params")

# --- V3 Node Object (PAD Schema v0.2) ---
class ActionNode(BaseModel):
    id: str = Field(..., description="Unique node identifier")
    type: str = Field("action", description="Fixed type")
    timestamp_start: float = Field(..., description="Start time")
    timestamp_end: float = Field(..., description="End time")

    # Semantic Data (V3)
    description: Optional[str] = Field(None, description="Legacy description")
    semantic_description: Optional[str] = Field(None, description="GenAI summary")
    screen_context: Optional[str] = Field(None, description="High-level UI context")

    # Classification & Location (V3)
    action_type: str = Field("click", description="click, type, drag, scroll")
    action_class: Optional[str] = Field(None, description="navigation, input, confirmation")

    # Visual Data
    ui_element_text: str = Field(..., description="OCR result from Active Region")
    ui_region: List[int] = Field(..., description="[x, y, w, h] bounding box")

    # Confidence
    confidence: float = Field(..., description="OCR confidence")
    active_region_confidence: float = Field(0.0, description="SSIM confidence")

    next_node_id: Optional[str] = Field(None, description="Next node ID")

# --- Root Object ---
class Pathway(BaseModel):
```

```
    pathway_id: str = Field(..., description="UUID-v4")
    title: str = Field(..., description="SOP Title")
    author_id: str = Field(..., description="User ID")
    source_video: str = Field(..., description="Source Video URI")
    created_at: str = Field(..., description="ISO-8601 Timestamp")
    total_duration_sec: float = Field(..., description="Total duration")
    nodes: List[ActionNode] = Field(..., description="List of nodes")
    metadata: Dict[str, Any] = Field(default_factory=dict)
```

---

4. Service Logic Files (`app/services/`)

`app/services/pipeline.py`

The Async Orchestrator. Includes the fix for "Could not read frame at end of video".

```
Python
import os
import uuid
import time
import cv2
import asyncio
from typing import List, Tuple
from app.schema import Pathway, ActionNode
from app.services.segment import detect_action_segments
from app.services.ocr import run_ocr
from app.services.vision import detect_active_region, classify_action_optical_flow
from app.services.genai import generate_semantic_description

async def _enrich_nodes_with_ai(nodes_with_frames: List[Tuple[ActionNode,
cv2.typing.MatLike]]):
    tasks = []
    print(f"Starting async GenAI enrichment for {len(nodes_with_frames)} nodes...")
    for node, frame in nodes_with_frames:
        tasks.append(
            generate_semantic_description(
                node_id=node.id,
                frame=frame,
                ocr_text=node.ui_element_text,
                action_type=node.action_type
            )
        )
    descriptions = await asyncio.gather(*tasks)
    for i, desc in enumerate(descriptions):
        nodes_with_frames[i][0].semantic_description = desc
```

```
def _get_frame_at_time(cap: cv2.VideoCapture, timestamp: float) -> cv2.typing.MatLike:
    # FIXED: Clamp timestamp to video duration - 0.1s to prevent "read past end" errors
    max_duration = cap.get(cv2.CAP_PROP_FRAME_COUNT) / cap.get(cv2.CAP_PROP_FPS)
    safe_timestamp = min(timestamp, max_duration - 0.1)

    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_no = int(safe_timestamp * fps)

    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_no)
    ret, frame = cap.read()
    if not ret:
        # Fallback: Try reading the 2nd to last frame
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        cap.set(cv2.CAP_PROP_POS_FRAMES, total_frames - 2)
        ret, frame = cap.read()
        if not ret:
            raise ValueError(f"Could not read frame at {timestamp}s")
    return frame

async def build_pathway(video_path: str) -> Pathway:
    print(f"Starting V3 'Insight' processing for: {os.path.basename(video_path)}")
    start_process_time = time.time()

    if not os.path.exists(video_path):
        raise FileNotFoundError(f"Video file not found: {video_path}")

    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        raise IOError(f"Cannot open video: {video_path}")

    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_count = cap.get(cv2.CAP_PROP_FRAME_COUNT)
    total_duration_sec = frame_count / fps

    print("Running segmentation...")
    segments = detect_action_segments(video_path)

    nodes: List[ActionNode] = []
    nodes_for_enrichment: List[Tuple[ActionNode, cv2.typing.MatLike]] = []

    for i, (start_time, end_time) in enumerate(segments):
        try:
            frame_start = _get_frame_at_time(cap, start_time)
            frame_end = _get_frame_at_time(cap, end_time)
```

```
            # 1. Active Region (SSIM)
            active_region = detect_active_region(frame_start, frame_end)

            # 2. Action Class (Optical Flow)
            action_type = classify_action_optical_flow(frame_start, frame_end)

            # 3. Spatial OCR
            mid_time = (start_time + end_time) / 2
            key_frame = _get_frame_at_time(cap, mid_time)
            ocr_results = run_ocr(key_frame, active_region=active_region)

            if ocr_results:
                best_result = max(ocr_results, key=lambda x: x['confidence'])
                ui_text = best_result['text']
                ui_region = best_result['ui_region']
                confidence = best_result['confidence']
            else:
                ui_text = "Unlabeled Icon"
                ui_region = list(active_region) if active_region else [0,0,0,0]
                confidence = 0.0

            node = ActionNode(
                id=f"node_{i + 1}",
                timestamp_start=start_time,
                timestamp_end=end_time,
                description=f"User action on '{ui_text}'",
                action_type=action_type,
                ui_element_text=ui_text,
                ui_region=ui_region,
                confidence=confidence,
                active_region_confidence=1.0 if active_region else 0.0,
                next_node_id=f"node_{i + 2}" if i + 1 < len(segments) else None
            )

            nodes.append(node)
            nodes_for_enrichment.append((node, key_frame))

        except Exception as e:
            print(f"Error processing segment {i}: {e}")
            continue

    cap.release()
```

```
    # UPDATED: Direct await for async GenAI
    if nodes_for_enrichment:
        try:
            await _enrich_nodes_with_ai(nodes_for_enrichment)
        except Exception as e:
            print(f"GenAI Enrichment Failed: {e}")

    pathway = Pathway(
        pathway_id=str(uuid.uuid4()),
        title=f"Auto-Generated Workflow: {os.path.basename(video_path)}",
        author_id="tbd-v3-engine",
        source_video=os.path.basename(video_path),
        created_at=time.strftime('%Y-%m-%dT%H:%M:%S%z'),
        total_duration_sec=total_duration_sec,
        nodes=nodes
    )

    print(f"V3 Processing complete in {time.time() - start_process_time:.2f}s")
    return pathway
```

app/services/worker.py

```
Async Worker logic.
Python
import os
import tempfile
import base64
import json
import asyncio
from urllib.parse import urlparse
from google.cloud import storage
from app.schema import TaskPayload, Pathway
from app.services.pipeline import build_pathway

TEMP_DIR = tempfile.gettempdir()

class WorkerService:
    def __init__(self):
        try:
            self.storage_client = storage.Client()
        except Exception as e:
            print(f"WARNING: Storage client failed to initialize: {e}")
            self.storage_client = None

    def _parse_gcs_uri(self, gcs_uri: str) -> tuple:
```

```
    parsed = urlparse(gcs_uri)
    if parsed.scheme != 'gs':
        raise ValueError(f"Invalid GCS URI scheme: {gcs_uri}")
    return parsed.netloc, parsed.path.lstrip('/')

def _download_video(self, bucket_name: str, blob_name: str, local_path: str):
    if not self.storage_client:
        print("MOCK DOWNLOAD")
        return
    bucket = self.storage_client.bucket(bucket_name)
    blob = bucket.blob(blob_name)
    print(f"Downloading gs://{bucket_name}/{blob_name} to {local_path}...")
    blob.download_to_filename(local_path)
    print("Download complete.")

def _upload_pathway(self, pathway: Pathway, output_bucket_name: str, task_id: str) -> str:
    output_blob_name = f"{task_id}/pathway.json"
    gcs_uri = f"gs://{output_bucket_name}/{output_blob_name}"
    if not self.storage_client:
        return gcs_uri
    bucket = self.storage_client.bucket(output_bucket_name)
    blob = bucket.blob(output_blob_name)
    json_data = pathway.model_dump_json(indent=2)
    blob.upload_from_string(json_data, content_type='application/json')
    print(f"Pathway uploaded successfully to: {gcs_uri}")
    return gcs_uri

# UPDATED: Async method
async def process_pubsub_message(self, pubsub_message_data: dict):
    if 'message' not in pubsub_message_data:
        raise ValueError("Invalid Pub/Sub message format.")

    message_data_bytes = base64.b64decode(pubsub_message_data['message']['data'])
    payload_dict = json.loads(message_data_bytes.decode('utf-8'))
    payload = TaskPayload.model_validate(payload_dict)

    print(f"--- Worker START: Task {payload.task_id} ---")
    input_bucket, input_blob = self._parse_gcs_uri(payload.gcs_uri)
    local_video_path = os.path.join(TEMP_DIR, os.path.basename(input_blob))

    try:
        self._download_video(input_bucket, input_blob, local_video_path)

        # UPDATED: Await the pipeline
```

```
        pathway = await build_pathway(local_video_path)

        output_uri = self._upload_pathway(pathway, payload.output_bucket, payload.task_id)
        print(f"Worker SUCCESS: Task {payload.task_id} completed. Output URI: {output_uri}")

    except Exception as e:
        print(f"Worker FAILED: {e}")
        raise e
    finally:
        if os.path.exists(local_video_path):
            os.remove(local_video_path)
        print(f"--- Worker END: Task {payload.task_id} ---")
```

app/services/genai.py

Using `gemini-2.5-pro` and increased token limit.

Python

```
import os
import cv2
import vertexai
from vertexai.generative_models import GenerativeModel, Part

# Configuration
# UPDATED: Using Gemini 2.5 Pro as verified by user
MODEL_NAME = "gemini-2.5-pro"
PROJECT_ID = os.environ.get("GCP_PROJECT_ID")
LOCATION = "us-central1"

# Initialize Vertex AI
if PROJECT_ID:
    try:
        vertexai.init(project=PROJECT_ID, location=LOCATION)
    except Exception as e:
        print(f"WARNING: Vertex AI init failed: {e}")

def encode_image_to_bytes(frame):
    """Converts an OpenCV image (numpy array) to raw bytes."""
    success, encoded_image = cv2.imencode(".jpg", frame)
    if not success:
        raise ValueError("Failed to encode image")
    return encoded_image.tobytes()

async def generate_semantic_description(node_id: str, frame, ocr_text: str, action_type: str) ->
str:
    """
```

```
    Generates a semantic description using Vertex AI.
    """
    if not PROJECT_ID:
        return "Desc unavailable (Vertex AI not configured)."

    try:
        # UPDATED: Increased max_output_tokens to 300 to prevent cut-off sentences
        model = GenerativeModel(
            MODEL_NAME,
            generation_config={"temperature": 0.0, "max_output_tokens": 300}
        )

        image_bytes = encode_image_to_bytes(frame)
        image_part = Part.from_data(data=image_bytes, mime_type="image/jpeg")

        prompt_text = (
            f"User performed '{action_type}' on '{ocr_text}'. "
            "Write a one-sentence tutorial step description. No coordinates."
        )

        response = await model.generate_content_async([image_part, prompt_text])
        return response.text.strip()

    except Exception as e:
        print(f"GenAI Error on {node_id}: {e}")
        return f"User clicked '{ocr_text}' (AI description failed)"
```

app/services/vision.py

```
Optical Flow and SSIM Logic.
Python
import cv2
import numpy as np
from skimage.metrics import structural_similarity
from typing import Tuple, Optional

def detect_active_region(frame_before: cv2.typing.MatLike, frame_after: cv2.typing.MatLike) ->
Optional[Tuple[int, int, int, int]]:
    grayA = cv2.cvtColor(frame_before, cv2.COLOR_BGR2GRAY)
    grayB = cv2.cvtColor(frame_after, cv2.COLOR_BGR2GRAY)
    (score, diff) = structural_similarity(grayA, grayB, full=True)
    diff = (diff * 255).astype("uint8")
    thresh = cv2.threshold(diff, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    if not contours:
        return None
    largest_contour = max(contours, key=cv2.contourArea)
    if cv2.contourArea(largest_contour) < 100:
        return None
    x, y, w, h = cv2.boundingRect(largest_contour)
    padding = 10
    height, width = frame_before.shape[:2]
    x = max(0, x - padding)
    y = max(0, y - padding)
    w = min(width - x, w + (padding * 2))
    h = min(height - y, h + (padding * 2))
    return x, y, w, h

def classify_action_optical_flow(frame_prev: cv2.typing.MatLike, frame_curr:
cv2.typing.MatLike) -> str:
    prev_gray = cv2.cvtColor(frame_prev, cv2.COLOR_BGR2GRAY)
    curr_gray = cv2.cvtColor(frame_curr, cv2.COLOR_BGR2GRAY)
    flow = cv2.calcOpticalFlowFarneback(
        prev_gray, curr_gray, None,
        pyr_scale=0.5, levels=3, winsize=15, iterations=3, poly_n=5, poly_sigma=1.2, flags=0
    )
    mean_dy = np.mean(flow[..., 1])
    mag, _ = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    mean_mag = np.mean(mag)

    SCROLL_THRESHOLD = 2.0
    STATIONARY_THRESHOLD = 0.5

    if abs(mean_dy) > SCROLL_THRESHOLD:
        return "scroll"
    elif mean_mag < STATIONARY_THRESHOLD:
        return "click"
    else:
        return "click"
```

app/services/ocr.py

```
Spatial OCR Logic.
Python
import cv2
import pytesseract
from typing import List, Tuple, Dict, Any, Optional

def _boxes_intersect(box1: List[int], box2: List[int]) -> bool:
```

```
    x1, y1, w1, h1 = box1
    x2, y2, w2, h2 = box2
    x_left = max(x1, x2)
    y_top = max(y1, y2)
    x_right = min(x1 + w1, x2 + w2)
    y_bottom = min(y1 + h1, y2 + h2)
    return not (x_right < x_left or y_bottom < y_top)

def run_ocr(frame: cv2.typing.MatLike, active_region: Optional[Tuple[int, int, int, int]] = None) ->
List[Dict[str, Any]]:
    ocr_data = pytesseract.image_to_data(frame, output_type=pytesseract.Output.DICT)
    results = []
    for i in range(len(ocr_data['text'])):
        text = ocr_data['text'][i].strip()
        try:
            conf_val = float(ocr_data['conf'][i])
        except:
            conf_val = 0.0
        confidence = conf_val / 100.0

        if text and confidence > 0.5:
            ui_region = [
                int(ocr_data['left'][i]), int(ocr_data['top'][i]),
                int(ocr_data['width'][i]), int(ocr_data['height'][i])
            ]
            if active_region:
                if _boxes_intersect(ui_region, active_region):
                    results.append({"text": text, "confidence": confidence, "ui_region": ui_region})
            else:
                results.append({"text": text, "confidence": confidence, "ui_region": ui_region})
    return results
```

`app/services/segment.py` & `app/services/dispatcher.py`

(Unchanged from V2 - refer to previous manifest if needed)