V5.0 Master Deployment Manifest: Functional Activation

1. Project Directory Structure

```
tbd-v5-master/
├── deploy_v5_full.ps1         # Master Deployment Orchestrator
├── requirements.txt           # Worker/Dispatcher dependencies (V5 Final)
├── app/                       # Worker (Service B) & Dispatcher (Service A)
│   ├── main.py
│   ├── schema.py
│   └── services/
│       ├── dispatcher.py
│       ├── worker.py          # V5 Execution Core (FIXED STT AWAIT)
│       ├── pipeline.py        # V5 Orchestrator (FIXED 4 ARGUMENTS)
│       ├── genai.py           # Gemini 2.5 Pro (FIXED 2 ARGUMENTS)
│       ├── ocr.py
│       ├── segment.py
│       └── vision.py
├── tbd-encoder/               # Temporal Encoder (Service C) Deployment Assets
│   ├── Dockerfile
│   ├── encoder_requirements.txt
│   ├── lstm_model.h5
│   └── encoder_app/main.py
└── tbd-detector/              # Object Detector (Service D) Deployment Assets
    ├── Dockerfile
    ├── detector_requirements.txt
    └── detector_app/main.py
```

---

2. Root Configuration Files

`requirements.txt` (Worker/Dispatcher Dependencies)

```
Plaintext
# --- Core Framework
fastapi==0.104.1
uvicorn==0.24.0
pydantic==2.5.2
python-multipart
```

```
# --- Infrastructure
google-cloud-storage==2.14.0
google-cloud-pubsub==2.19.0
requests==2.32.5
google-auth==2.43.0

# --- V3/V4 Vision/AI Stack
opencv-python-headless==4.8.1.78
pytesseract==0.3.10
google-cloud-aiplatform>=1.60.0
numpy>=1.26.0,<2.0.0

# --- V5 Multimodality Stack (FINAL) ---
google-cloud-speech==2.25.0
pydub==0.25.1
```

`Dockerfile` (Worker/Dispatcher Image)

Dockerfile

```
FROM python:3.10-slim

# Set working directory
WORKDIR /usr/src
ENV PYTHONUNBUFFERED=True

# 1. Install System Dependencies
RUN apt-get update && \
    apt-get install -y tesseract-ocr libtesseract-dev libgl1 libglib2.0-0 libsm6 libxext6 libxrender1
ffmpeg && \
    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
ENV PORT=8080

CMD sh -c "uvicorn app.main:app --host 0.0.0.0 --port ${PORT}"
```

---

3. Worker/Encoder Application Logic

A. `app/services/worker.py` (V5 Execution Core)

Python

```
import os
import tempfile
import base64
import json
import asyncio
import requests
import numpy as np
from urllib.parse import urlparse
from google.cloud import storage, pubsub_v1, speech
from google.auth.transport.requests import Request
from google.oauth2 import id_token
from typing import List, Tuple, Dict, Any
from app.schema import TaskPayload, Pathway
from app.services.pipeline import build_pathway
from pydub import AudioSegment
import uuid

TEMP_DIR = tempfile.gettempdir()
# V5 CONFIGURATION CONSTANTS (Placeholder URLs MUST be set by deploy script)
OBJECT_DETECTOR_URL = "https://tbd-object-detector-XXX-uc.a.run.app"
AGENT_TOPIC_NAME = "pad-agent-tasks"
MARKETPLACE_API_URL = "https://marketplace.freefuse.com/api/v1/register"
PROJECT_ID = os.environ.get("GCP_PROJECT_ID", "tbd-v2")
AUDIO_STAGING_BUCKET = f"tbd-audio-staging-{PROJECT_ID}"

# V5 FR-01: Idempotency Check Placeholder (In-Memory)
PROCESSED_TASKS = set()

# --- V5 Helper Functions ---

def _get_auth_token(audience: str) -> str:
    """Generates an authenticated token for the target external API."""
    try:
        auth_request = Request()
        token = id_token.fetch_id_token(auth_request, audience)
        return token
    except Exception as e:
```

```
        print(f"ERROR: Could not fetch auth token for {audience}: {e}")
        return ""

def _extract_audio_track(video_path: str) -> str:
    """FR-06: Uses ffmpeg to extract audio and saves it locally as an MP3."""
    audio_path = os.path.join(TEMP_DIR, f"audio_{uuid.uuid4()}.mp3")
    command = f"ffmpeg -i {video_path} -vn -acodec libmp3lame -q:a 2 {audio_path}"
    os.system(command)
    return audio_path

def _upload_audio_to_gcs(local_path: str, task_id: str) -> str:
    """FR-06: Uploads the local audio file to the dedicated staging bucket for STT API."""
    storage_client = storage.Client(project=PROJECT_ID)
    audio_blob_name = f"{task_id}/audio.mp3"
    bucket = storage_client.bucket(AUDIO_STAGING_BUCKET)
    blob = bucket.blob(audio_blob_name)
    blob.upload_from_filename(local_path)
    return f"gs://{AUDIO_STAGING_BUCKET}/{audio_blob_name}"

async def _call_speech_to_text(gcs_uri: str) -> str:
    """FR-06: Implements the actual Google Cloud Speech-to-Text API call."""
    try:
        stt_client = speech.SpeechAsyncClient()
        audio = speech.RecognitionAudio(uri=gcs_uri)
        config = speech.RecognitionConfig(
            encoding=speech.RecognitionConfig.AudioEncoding.MP3,
            sample_rate_hertz=16000,
            language_code="en-US"
        )

        operation = stt_client.long_running_recognize(config=config, audio=audio)
        # FIX: The crucial 'await' is added to resolve the 'coroutine' object error
        result = await operation.result()

        transcript = " ".join([r.alternatives[0].transcript for r in result.results])
        return transcript if transcript else "No audible speech detected."
    except Exception as e:
        print(f"ERROR: STT API call failed: {e}")
        return "Audio transcription service failed."

async def _register_asset_with_marketplace(pathway_json: Dict[str, Any], trace_id: str) ->
Tuple[bool, str]:
    """V5 FR-05: Makes an authenticated POST call to the external Marketplace API."""
    token = _get_auth_token(MARKETPLACE_API_URL)
```

```
    if not token:
        return False, "Marketplace call failed: Missing Auth Token"

    headers = {
        "Authorization": f"Bearer {token}",
        "Content-Type": "application/json",
        "X-Cloud-Trace-Context": trace_id
    }

    try:
        loop = asyncio.get_event_loop()
        response = await loop.run_in_executor(
            None,
            lambda: requests.post(MARKETPLACE_API_URL, headers=headers, json=payload,
timeout=5)
        )
        response.raise_for_status()
        return True, "Marketplace registration successful."
    except Exception as e:
        return False, f"Marketplace submission failed: {e}"

def _publish_to_agent_topic(gcs_uri: str, trace_id: str):
    """V5 FR-04: Publishes the final URI to the Agent Execution Topic."""
    try:
        publisher = pubsub_v1.PublisherClient()
        topic_path = publisher.topic_path(PROJECT_ID, AGENT_TOPIC_NAME)
        data = gcs_uri.encode("utf-8")
        publisher.publish(topic_path, data, trace_id=trace_id)
        print(f"✅ Published final URI to agent topic: {AGENT_TOPIC_NAME}")
    except Exception as e:
        print(f"FATAL: Agent publication failed: {e}")

# --- WorkerService Class (V5 Integration) ---

class WorkerService:
    def __init__(self):
        try:
            self.storage_client = storage.Client()
            self.agent_publisher = pubsub_v1.PublisherClient()
        except Exception as e:
            print(f"WARNING: Client initialization failed: {e}")
            self.storage_client = None
            self.agent_publisher = 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:
        print(f"MOCK UPLOAD: Would upload result to {gcs_uri}")
        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

async def process_pubsub_message(self, pubsub_message_data: dict):
    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)

    attributes = pubsub_message_data['message'].get('attributes', {})
    task_id = payload.task_id
    trace_id = attributes.get('trace_id', 'NO_TRACE_ID')

    print(f"--- Worker START: Task {task_id}, Trace {trace_id} ---")
```

```
        # V5 FR-01: Idempotency Check
        if task_id in PROCESSED_TASKS:
            print(f"WARNING: Task {task_id} already processed. ACK and exiting.")
            return

        input_bucket, input_blob = self._parse_gcs_uri(payload.gcs_uri)
        local_video_path = os.path.join(TEMP_DIR, os.path.basename(input_blob))
        local_audio_path = "" # Define audio path here for finally block cleanup

        # --- V5 EXECUTION CORE ---
        try:
            # 1. Download Video
            self._download_video(input_bucket, input_blob, local_video_path)

            # V5 FR-06: Extract Audio Track -> Upload -> Transcribe
            local_audio_path = _extract_audio_track(local_video_path)
            gcs_audio_uri = _upload_audio_to_gcs(local_audio_path, task_id)
            audio_transcript = await _call_speech_to_text(gcs_audio_uri)

            # 2. Execute V4 Pipeline (Pass Transcript, Object Detector URL)
            from app.services.pipeline import build_pathway

            # FIX: The Worker now passes 4 arguments to the pipeline function
            pathway = await build_pathway(
                local_video_path,
                payload.gcs_uri,
                audio_transcript,
                OBJECT_DETECTOR_URL
            )

            # 3. Finalization and Execution Hooks
            output_uri = self._upload_pathway(pathway, payload.output_bucket, payload.task_id)

            # V5 FR-05: Call Marketplace API
            market_success, market_message = await
_register_asset_with_marketplace(pathway.model_dump(), trace_id)

            # V5 FR-04: Publish to Agent Topic
            _publish_to_agent_topic(output_uri, trace_id)

            # 4. Final Audit and Idempotency Lock
            PROCESSED_TASKS.add(task_id)
            print(f"Worker SUCCESS: Task {task_id} completed. Output URI: {output_uri}")
            print(f"Marketplace Status: {market_message}")
```

```
    except Exception as e:
        print(f"Worker FAILED: {e}")
        raise e
    finally:
        if os.path.exists(local_video_path):
            os.remove(local_video_path)
            # Clean up extracted audio file
            if os.path.exists(local_audio_path):
                os.remove(local_audio_path)
        print(f"--- Worker END: Task {task_id}, Trace {trace_id} ---")
```

C. `app/services/pipeline.py` (V5 Orchestrator)

```
Python
import os
import uuid
import time
import cv2
import asyncio
import json
import base64
import requests
from typing import List, Tuple
from app.schema import Pathway, ActionNode
from app.services.genai import analyze_video_native
from app.services.ocr import run_ocr
from google.oauth2 import id_token
from google.auth.transport.requests import Request

# V5 NEW CONFIGURATION CONSTANTS (Passed from worker)
OBJECT_DETECTOR_TIMEOUT = 5.0

def _get_auth_token(audience: str) -> str:
    """Generates an authenticated token for the target Cloud Run service."""
    try:
        auth_request = Request()
        token = id_token.fetch_id_token(auth_request, audience)
        return token
    except Exception as e:
        print(f"ERROR: Could not fetch auth token for Encoder: {e}")
        return ""

def _get_frame_at_time(cap: cv2.VideoCapture, timestamp: float) -> cv2.typing.MatLike:
```

```
    """Extracts a frame at a specific timestamp for coordinate refinement."""
    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:
        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:
            return None
    return frame

async def _call_object_detector(frame: cv2.typing.MatLike, target_text: str, detector_url: str) ->
Tuple[List[int], float]:
    """
    FR-07: Calls Service D securely for pixel-accurate coordinate prediction.
    """
    auth_token = _get_auth_token(detector_url)

    if frame is None or not auth_token:
        print("WARNING: Object Detector call failed. Missing auth token or frame.")
        return [0, 0, 0, 0], 0.0

    success, buffer = cv2.imencode('.jpg', frame)
    frame_base64 = base64.b64encode(buffer).decode('utf-8')

    headers = {
        "Authorization": f"Bearer {auth_token}",
        "Content-Type": "application/json"
    }
    payload = {"frame_base64": frame_base64, "target_text": target_text}

    try:
        loop = asyncio.get_event_loop()
        response = await loop.run_in_executor(
            None,
            lambda: requests.post(f"{detector_url}/detect_coordinates", headers=headers,
json=payload, timeout=OBJECT_DETECTOR_TIMEOUT)
        )
```

```
        response.raise_for_status()

        result = response.json()
        return result.get('ui_region', [0, 0, 0, 0]), result.get('confidence', 0.0)

    except Exception as e:
        print(f"WARNING: Object Detector call failed: {e}")
        return [0, 0, 0, 0], 0.0

# FINAL V5 PIPELINE SIGNATURE: Correctly accepts 4 arguments
async def build_pathway(local_video_path: str, gcs_video_uri: str, audio_transcript: str,
object_detector_url: str) -> Pathway:
    print(f"Starting V4 'Native Insight' for: {os.path.basename(local_video_path)}")
    start_time = time.time()

    print("Phase 1: Semantic Analysis (Gemini Native Video + Audio Fusion)...")

    ai_steps = await analyze_video_native(gcs_video_uri, audio_transcript)
    print(f"Gemini identified {len(ai_steps)} steps.")

    print("Phase 2: Coordinate Refinement (Service D Call)...")
    cap = cv2.VideoCapture(local_video_path)

    fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
    total_duration_sec = total_frames / fps if fps else 0

    final_nodes = []

    for i, step in enumerate(ai_steps):
        timestamp = float(step.get('timestamp', 0.0))
        target_text = step.get('target_text', "Unlabeled")

        frame = _get_frame_at_time(cap, timestamp)

        ui_region, confidence = await _call_object_detector(frame, target_text, object_detector_url)

        # Construct the Node
        node = ActionNode(
            id=f"node_{i + 1}",
            timestamp_start=timestamp,
            timestamp_end=timestamp + 1.0,
            description=step.get('description', 'No description'),
            semantic_description=step.get('description', 'No description'),
```

```
            ui_element_text=target_text,
            ui_region=ui_region,
            confidence=confidence,
            active_region_confidence=confidence,
            action_type=step.get('action_type', 'click'),
            next_node_id=f"node_{i + 2}" if i + 1 < len(ai_steps) else None
        )
        final_nodes.append(node)

    cap.release()

    pathway = Pathway(
        pathway_id=str(uuid.uuid4()),
        title=f"Native Insight: {os.path.basename(local_video_path)}",
        author_id="tbd-v4-engine",
        source_video=os.path.basename(local_video_path),
        created_at=time.strftime('%Y-%m-%dT%H:%M:%S%z'),
        total_duration_sec=total_duration_sec,
        nodes=final_nodes
    )

    print(f"V4 Processing complete in {time.time() - start_time:.2f}s")
    return pathway
```