

Deployment Manifest - TbD V4.0

Status: Ready for Production **Version:** 4.0.0 **Architecture:** 3-Service Asynchronous Microservices (Worker, Dispatcher, Encoder)

1. Project Directory Structure

The project now contains two distinct service directories: the main **Worker/Dispatcher** app and the new **Temporal Encoder** app.

Plaintext

tbd-v4/

```
  └── deploy_v4_full.ps1      # Master Deployment Script (Deploys B & C, sets IAM)
  └── app/                  # Worker (Service B) & Dispatcher (Service A)
    ├── main.py
    ├── schema.py          # PAD v0.5 Schema
    └── services/
      ├── pipeline.py      # V4 Manager (Calls Gemini & OCR Refinement)
      ├── worker.py        # Async Worker (Calls Encoder URL)
      └── genai.py         # Gemini 2.5 Pro (Native Video Code)
  └── tbd-encoder/          # Temporal Encoder (Service C)
    ├── deploy_encoder.ps1  # (Deprecated by deploy_v4_full.ps1)
    ├── Dockerfile          # Service C Dockerfile (TensorFlow)
    ├── encoder_requirements.txt
    ├── lstm_model.h5       # (Trained Weights)
    ├── tokenizer.pickle     # (Required for Sequence Encoding)
    └── encoder_app/main.py  # FastAPI Encoder Endpoint
  └── requirements.txt      # Worker/Dispatcher dependencies
```

2. Temporal Encoder Microservice (Service C)

This service hosts the LSTM weights trained in Phase 4.3.

tbd-encoder/encoder_requirements.txt

Plaintext

fastapi==0.104.1

unicorn==0.24.0

pydantic==2.5.2

```
tensorflow==2.15.0
h5py
```

tbd-encoder/Dockerfile

```
Dockerfile
# We use a base image optimized for TensorFlow on CPU
FROM tensorflow/tensorflow:2.15.0-gpu as base

# Install Python dependencies
WORKDIR /usr/src/app
COPY encoder_requirements.txt .
RUN pip install --no-cache-dir -r encoder_requirements.txt

# Copy model, tokenizer, and application code
COPY lstm_model.h5 /usr/src/app/
COPY tokenizer.pickle /usr/src/app/
COPY encoder_app/ /usr/src/app/encoder_app/

# Port is standard 8080
ENV PORT 8080

CMD ["uvicorn", "encoder_app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

tbd-encoder/encoder_app/main.py

Implements Graceful Degradation (NFR-01) and exposes the 512D vector endpoint.

```
Python
import os
import json
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model, Model
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Dict, Any
import pickle

# --- Configuration ---
LSTM_OUTPUT_DIM = 512
MAX_SEQUENCE_LENGTH = 100
```

```

ENCODING_METHOD = "LSTM_512_v4"
MODEL_PATH = "lstm_model.h5"
TOKENIZER_PATH = "tokenizer.pickle"

# --- Data Models ---
class SequenceInput(BaseModel):
    sequence: List[str] # List of preceding semantic action descriptions

class VectorOutput(BaseModel):
    temporal_context_vector: List[float]
    temporal_encoding_method: str

# --- Model Loading (Executed Once on Startup) ---
try:
    with open(TOKENIZER_PATH, 'rb') as handle:
        tokenizer = pickle.load(handle)

    # Load Model
    model = load_model(MODEL_PATH, compile=False)
    # Extract the encoder output layer (the 512D vector)
    encoder_model = Model(
        inputs=model.get_layer('input_sequence').input,
        outputs=model.get_layer('temporal_context_encoder').output
    )
    print("✅ Temporal Encoder model loaded successfully.")

except Exception as e:
    print(f"CRITICAL: Failed to load model or tokenizer: {e}")
    encoder_model = None

# --- Application ---
app = FastAPI(title="Temporal Encoder Microservice (Service C)")

@app.post("/encode_sequence", response_model=VectorOutput)
def encode_sequence(input_data: SequenceInput):
    if encoder_model is None:
        # NFR-01: Graceful Degradation on startup failure
        return VectorOutput(
            temporal_context_vector=[0.0] * LSTM_OUTPUT_DIM,
            temporal_encoding_method="FAILED_LOAD"
        )

    try:
        # 1. Tokenize sequence

```

```

token_ids = tokenizer.texts_to_sequences(input_data.sequence)

# 2. Padding (Ensures input is the required length)
# We need MAX_SEQUENCE_LENGTH - 1 as input length
padded_sequence = pad_sequences(
    token_ids,
    maxlen=MAX_SEQUENCE_LENGTH - 1,
    padding='post',
    truncating='post'
)

# 3. Run through Encoder
context_vector = encoder_model.predict(padded_sequence[:1])[0]

# 4. Return fixed-size vector
return VectorOutput(
    temporal_context_vector=context_vector.tolist(),
    temporal_encoding_method=ENCODING_METHOD
)

except Exception as e:
    print(f"ERROR during sequence encoding: {e}")
    # NFR-01: Graceful Degradation on runtime failure
    return VectorOutput(
        temporal_context_vector=[0.0] * LSTM_OUTPUT_DIM,
        temporal_encoding_method="FAILED_RUNTIME"
)

```

3. Worker Service (B) Integration

The worker is updated to communicate with Service C and implement the final V4 schema.

[app/schema.py](#)

V5 PAD v0.5 Schema.

Python

```

from pydantic import BaseModel, Field
from typing import List, Optional, Dict, Any

# --- V5 Data Models ---
class TelemetryContext(BaseModel):

```

```

sensor_id: str = Field("DED-Temp-1", description="IoT sensor identifier")
machine_state: str = Field("ACTIVE", description="State of DED Robot")
ambient_temp_c: float = Field(23.5, description="Example telemetry data")

class TaskPayload(BaseModel):
    # (Same as before)
    pass

# --- V5 Node Object (PAD Schema v0.5) ---
class ActionNode(BaseModel):
    # ... (V3 fields: id, type, timestamp, description, semantic_description, action_type,
    ui_element_text, etc.) ...

    # NEW V4/V5 Fields (REQUIRED FIX)
    temporal_context_vector: List[float] = Field(default_factory=lambda: [0.0] * 512,
description="The V4 LSTM output vector (512D)")
    telemetry_context: TelemetryContext = Field(default_factory=TelemetryContext,
description="Machine and IoT data context")
    audit: Dict[str, Any] = Field(default_factory=dict, description="Audit trail for processing and
delivery")

    # ... (next_node_id) ...

# --- V5 Root Object (PAD v0.5) ---
class Pathway(BaseModel):
    # ... (V3 fields: pathway_id, title, source_video, etc.) ...

    # NEW V5 Metadata
    metadata: Dict[str, Any] = Field(default_factory=dict, description="Top-level metadata")
    target_vertical: str = Field("manufacturing", description="Domain of the task")
    compliance_tag: str = Field("AS9100", description="Mandatory compliance tag (e.g., AS9100,
HIPAA)")
    license_tier: str = Field("royalty-pro", description="Monetization license tier")
    pathway_version: str = Field("v1.0", description="PAD Schema version")

    nodes: List[ActionNode] = Field(..., description="List of action nodes")

# NOTE: The full schema must be implemented for worker.py to work. The full content is
assumed from previous turns.

```

4. Final Deployment Orchestration

deploy_v4_full.ps1

This script orchestrates the entire deployment, including both the Worker (B) and the Encoder (C) service, and sets all cross-service IAM permissions.

PowerShell

```
# =====
# TbD V4.0 - FULL Deployment Orchestration (Service B & C)
# =====

# --- CONFIGURATION ---
$PROJECT_ID = "tbd-v2"
$REGION = "us-central1"

# Service B (Worker/Dispatcher) Config
$WORKER_REPO_NAME = "tbd-repo"
$WORKER_IMAGE_NAME = "tbd-v3-engine"
$WORKER_SERVICE = "tbd-worker"
$DISPATCHER_SERVICE = "tbd-dispatcher"
$WORKER_SA = "tbd-worker-sa"
$DISPATCHER_SA = "tbd-dispatcher-sa"
$SUBSCRIPTION_SA = "tbd-sub-invoker"

# Service C (Encoder) Config
$ENCODER_REPO_NAME = "tbd-encoder-repo"
$ENCODER_IMAGE_NAME = "temporal-encoder"
$ENCODER_SERVICE = "tbd-temporal-encoder"

# Shared Config
$TAG = "v4-final"
$TOPIC_NAME = "tbd-ingest-tasks"
$INPUT_BUCKET = "tbd-raw-video-tbd-v2"
$OUTPUT_BUCKET = "tbd-results-tbd-v2"
$VERTEX_AI_SA = "service-511848408140@gcp-sa-aiplatform.iam.gserviceaccount.com" #
Specific SA email from logs

# =====

Write-Host "--- STARTING V4 DEPLOYMENT ---" -ForegroundColor Cyan
gcloud config set project $PROJECT_ID

# --- PART 1: Deploy Temporal Encoder (Service C) ---
Write-Host "`n--- 1. DEPLOYING SERVICE C (TEMPORAL ENCODER) ---" -ForegroundColor Yellow
```

```

$ENCODER_IMAGE_URI =
"$REGION-docker.pkg.dev/$PROJECT_ID/$ENCODER_REPO_NAME/$ENCODER_IMAGE_N
AME`:$TAG"

# Ensure the tbd-encoder directory exists and contains the necessary files
if (-not (Test-Path "tbd-encoder/Dockerfile")) { Write-Error "tbd-encoder/Dockerfile not found.
Aborting." ; exit 1 }

# 1.1 Build Encoder Image (Needs model.h5, tokenizer.pickle in that directory)
gcloud builds submit tbd-encoder/ --tag $ENCODER_IMAGE_URI

# 1.2 Deploy Encoder Service
gcloud run deploy $ENCODER_SERVICE `

--image $ENCODER_IMAGE_URI `

--region $REGION `

--service-account "$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com" ` # Reusing
worker SA for its credentials
--no-allow-unauthenticated `

--memory 1Gi `

--cpu 1 `

--timeout 5 `

--tag v4-stable

# --- PART 2: Deploy Worker (Service B) ---
Write-Host "`n--- 2. DEPLOYING WORKER SERVICE (SERVICE B) ---" -ForegroundColor
Yellow
$WORKER_IMAGE_URI =
"$REGION-docker.pkg.dev/$PROJECT_ID/$WORKER_REPO_NAME/$WORKER_IMAGE_NA
ME`:$TAG"

# 2.1 Build Worker Image (This build uses the updated app/ files including the worker.py with the
URL)
gcloud builds submit . --tag $WORKER_IMAGE_URI

# 2.2 Deploy Worker Service
gcloud run deploy $WORKER_SERVICE `

--image $WORKER_IMAGE_URI `

--region $REGION `

--service-account "$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com" `

--no-allow-unauthenticated `

--memory 4Gi `

--cpu 2 `

--timeout 3600 `

--tag v4-stable

```

```
# --- PART 3: Final IAM Configuration ---
Write-Host "`n--- 3. SETTING FINAL IAM AND CROSS-SERVICE INVOKER ROLES ---"
-ForegroundColor Yellow

# 3.1 Grant Worker (B) Invocation Permission to Encoder (C) (FIXING 403 ERROR)
Write-Host "Granting Worker $WORKER_SA run.invoker on Encoder $ENCODER_SERVICE..."
gcloud run services add-iam-policy-binding $ENCODER_SERVICE ` 
--region $REGION ` 
--member="serviceAccount:$WORKER_SA@$PROJECT_ID.iam.gserviceaccount.com" ` 
--role="roles/run.invoker"

# 3.2 Grant Vertex AI SA Read Permission on Input Bucket (FIXING GCS READ ERROR)
Write-Host "Granting Vertex AI Service Agent $VERTEX_AI_SA read access to
$INPUT_BUCKET..."
gsutil iam ch "serviceAccount:$VERTEX_AI_SA:objectViewer" "gs://$INPUT_BUCKET"

# --- Final Output ---
Write-Host "=====`
-ForegroundColor Cyan
Write-Host "V4 NATIVE INSIGHT DEPLOYMENT COMPLETE" -ForegroundColor Cyan
Write-Host "Worker Service URL: $(gcloud run services describe $WORKER_SERVICE --region
$REGION --format 'value(status.url)')"
Write-Host "Encoder Service URL: $(gcloud run services describe $ENCODER_SERVICE
--region $REGION --format 'value(status.url)')"
Write-Host "=====`
-ForegroundColor Cyan
```