This document is the **comprehensive, expanded Master Architectural Blueprint** for your thesis. It includes the detailed rationale, mathematical foundations, and specific workflow steps, incorporating the **Top-k Pooling** strategy and the **Confusion Matrix** evaluation plan as requested.

You can copy this directly into your thesis proposal or methodology chapter.

Thesis Master Architectural Blueprint

Project Title: Weakly Supervised Video Anomaly Detection (WS-VAD) via Vision Transformers and Multiple Instance Learning

1. Executive Summary & Core Philosophy

The Problem: Traditional video anomaly detection requires frame-level annotations (marking the exact start and end second of a crime), which is expensive and time-consuming to obtain.
The Solution: This thesis implements a Weakly Supervised approach. We utilize only video-level labels (e.g., "This video contains an explosion" vs. "This video is normal") to train a model that can localize crimes at the frame level.
The Architecture: We employ a two-stage pipeline:

1. **The Eyes (Backbone):** A Video Vision Transformer (TimeSformer) to extract spatiotemporal features.
2. **The Brain (MIL Head):** A Multiple Instance Learning (MIL) network with **Top-k Pooling** to identify anomalous segments within the video "bag."

---

2. Phase 1: Dataset & Preprocessing (The Input Pipeline)

Goal: Transform raw, unstructured video data into a structured, temporal tensor format optimized for Transformer ingestion.

2.1 The Dataset: UCF-Crime

- **Source:** UCF-Crime Dataset (Standard Benchmark).
- **Scale:** 1,900 videos (approx. 128 hours).
- **Labeling Reality (The Challenge):**
    - **Training Set:** Weakly labeled. We know Video A is "Arson," but we do not know if the arson happens at minute 1:00 or minute 5:00.
    - **Testing Set:** Fully labeled. We have temporal annotations to verify if our model

correctly found the crime.

2.2 The Extraction Strategy: Temporal Dilated Sampling

We cannot feed a 10-minute video into a neural network at once. We must slice it. However, simple slicing loses context. We use **Dilated Sampling**.

- **Logic:** We create "Clips." Each clip acts as a single data point.
- **Parameters:**
  - **Sequence Length (T): 16 frames.**
    - *Rationale:* Transformers operate on fixed-size tokens. 16 frames provide enough temporal depth to distinguish a "punch" from a "wave," but are small enough to fit in GPU memory.
  - **Stride (S): 5 (Dilated Sampling).**
    - *Method:* We do not take frames 1, 2, 3, 4, 5... (0.5 seconds). We take frames 1, 6, 11, 16...
    - *Rationale:* This covers a temporal span of approx. **2.5 seconds** using only 16 images. This allows the model to learn "long-term" motion dependencies.
  - **Clip Step: 64 frames (Sliding Window).**
    - *Method:* After extracting Clip 1 (starting at frame 0), the window slides forward 64 frames to extract Clip 2.
    - *Rationale:* This creates overlapping coverage. If an explosion starts at frame 50, a non-overlapping window might cut it in half. Overlapping ensures the event is captured fully in at least one clip.
- Output Directory Structure:
  The Python script will organize the data as follows:

```
Plaintext
Processed_Dataset/
├── Explosion/              <-- Class Name
│   ├── Explosion001/       <-- Video Name (The Bag)
│   │   ├── clip_000.jpg    <-- The Instance (Folder of 16 images)
│   │   ├── clip_001.jpg
│   │   └── ...
├── Normal/
│   ├── Normal_Video_001/
│   │   ├── clip_000.jpg
│   │   └── ...
```

---

3. Phase 2: Feature Extraction (The "Eyes")

Goal: Convert high-dimensional pixel data into dense, semantic feature vectors.

3.1 The Backbone: TimeSformer

We use **TimeSformer** (Time-Space Transformer) pre-trained on Kinetics-400.

- **Why not 2D CNN (ResNet)?** 2D CNNs look at images individually. They cannot see "motion."
- **Why TimeSformer?** It uses **Divided Space-Time Attention**.
  - **Spatial Attention:** Understands *objects* (gun, fire, car).
  - **Temporal Attention:** Understands *actions* (running, exploding, falling).

3.2 The Extraction Workflow

- **Status: Frozen Weights.** We do not train this model. We use it strictly as a feature extractor.
- **Input Shape:** Batch x 3 (RGB) x 16 (Frames) x 224 (H) x 224 (W).
- **Operation:** The clip passes through the transformer layers.
- **Output:** A Feature Embedding Vector of size **1 x 768**.
  - This vector is a mathematical summary of the 2.5 seconds of video. It contains information like "fire presence: high," "motion speed: fast."
- **Storage:** The images are discarded to save space. We save .npy (NumPy) files.
  - *Result:* A video that was 100MB is now a N x 768 matrix (approx. 50KB).

---

4. Phase 3: The MIL Architecture (The "Brain")

Goal: The core learning algorithm. It determines which clips in a video are responsible for the "Crime" label.

4.1 Multiple Instance Learning (MIL) Concepts

- **The Bag:** The specific video file (e.g., Robbery001.mp4). It contains many instances.
- **The Instance:** A single clip (represented by the 1 x 768 vector).
- **The Key Assumption:**
  - **Positive Bag (Crime):** Contains at least one anomalous clip, but also many normal clips.
  - **Negative Bag (Normal):** Contains *only* normal clips.

4.2 The Custom MIL Network

We will build a PyTorch model with the following architecture:

1. **Input:** Feature Matrix (N x 768).
2. **Projection Layer:** A Fully Connected (Linear) layer to compress features (768 ⟶ 512).
3. **Activation:** ReLU + Dropout (0.6) for regularization.
4. **Scoring Head:** A final Linear layer (512 ⟶ 1) followed by a Sigmoid activation.
   - *Output:* A score between 0.0 (Normal) and 1.0 (Anomaly) for **every single clip**.

4.3 The Aggregation Strategy: Top-k Pooling

This is the critical fix from our previous discussion.
Instead of using Max Pooling (taking the single highest score), we use Top-k Pooling.

- **The Problem with Max Pooling:** If one clip has a "glitch" or a bird flies by, the model might give it a score of 0.99. If we use Max Pooling, the whole video is labeled "Crime" based on one error. It is too sensitive to noise.
- **The Solution (Top-k):** We calculate the anomaly score of the video by averaging the scores of the **top k** most suspicious clips.
  - *Example:* If k=3, and the top scores are [0.98, 0.95, 0.90], the Video Score is **0.943**.
  - *Why:* This ensures that an anomaly is only flagged if there is a *sustained* event (lasting at least k clips), making the system robust against noise.
  - Formula:

$$Score_{video} = \frac{1}{k} \sum_{i=1}^{k} \text{sort}(Scores)_{descending}[i]$$

Score_{video} = [1/{k}]sum_{i=1}^{k} {sort}(Scores)_{descending}[i]

---

5. Phase 4: Training Strategy & Loss Functions

Goal: To optimize the MIL network using Weak Labels.

5.1 Ranking Loss (Hinge Loss)

This function enforces the separation between Normal and Crime videos.

- **Logic:** The predicted anomaly score of a Crime video (S_{crime}) must be significantly higher than the score of a Normal video (S_{normal}).
- Formula:

$$Loss_{rank} = \max(0, 1 - S_{crime} + S_{normal})$$

Interpretation: If the Crime score is not at least 1.0 higher than the Normal score, the model gets penalized.

5.2 Focal Loss (Handling Class Imbalance)

- **The Reality:** In the UCF-Crime dataset, 90% of the clips are normal (even inside crime videos).
- **The Fix:** Focal Loss modifies standard Cross-Entropy. It adds a modulating factor (1 - Pt)^gamma.
- **Effect:** If the model is confident and correct about an easy example (e.g., a clearly normal street), the loss is driven to zero. The model is forced to focus its gradient updates on the **hard** examples (the rare explosion clips).

5.3 Temporal Smoothness Loss

- **Logic:** Crimes happen in a sequence. If Clip 5 is an anomaly, Clip 6 should likely be an anomaly too. The score should not jump 0.1 -> 0.9 -> 0.1.

- **Formula:** Minimize $(Score_{t} - Score_{t-1})^2$.

---

6. Phase 5: Output, Metrics & Confusion Matrix

Goal: Evaluate the performance of the system scientifically.

6.1 The Final Output (Inference)

For a test video:

1. Extract features.
2. Pass through MIL Network.
3. **Result:** A temporal curve (Anomaly Score vs. Time).
4. **Smoothing:** Apply a Gaussian Filter to the output curve to reduce jitter.

6.2 The Confusion Matrix (Video Level)

To verify if our **Top-k Pooling** is working, we classify the *entire video* and generate a Confusion Matrix.

- **Threshold:** We set a threshold (e.g., 0.5). If Score_{video} > 0.5, we predict "Crime."
- The Matrix Structure:
	Predicted Normal	Predicted Crime
Actual Normal	True Negative (TN)

(Correctly ignored) | **False Positive (FP)**

(False Alarm) |
| Actual Crime | False Negative (FN)

(Missed the crime) | **True Positive (TP)**

(Caught the crime) |

- **Goal:** Minimize **False Positives** (FP) and **False Negatives** (FN).

6.3 Primary Metric: AUC-ROC (Frame Level)

- **Why not just Accuracy?** Accuracy is misleading. If a video is 99% normal and 1% crime, a model that predicts "All Normal" has 99% accuracy but is useless.
- **Area Under the Curve (AUC):** We plot the True Positive Rate vs. False Positive Rate at different thresholds.
- **Target:** A state-of-the-art AUC for UCF-Crime is typically between **82% and 86%**.