**Project Title: AI-Driven Hybrid Cyber Defensive System for Intelligent Malware Detection and Threat Insight**

**Team Members • Chimdessa Tesfaye • Getachewu Getu • Meseret ghebiresilassie • Rediet Bekele**

**Deployment Report: Threat Intelligence Classifier**

**1. Overview**

In this project, we developed and deployed a **Threat Intelligence Classifier** that leverages a fine-tuned BERT model to detect whether a given text contains cybersecurity-related threats. The project includes preprocessing, model training using Hugging Face Transformers, model serialization in the Hugging Face format, and a lightweight deployment via a Streamlit web application for real-time threat classification.

**2. Model Serialization**

The trained model was serialized using **Hugging Face's Transformers** library format. After training, the model and tokenizer were saved using:

*trainer.save\_model("bert-threat-classifier")*

*tokenizer.save\_pretrained("bert-threat-classifier")*

This creates a folder (bert-threat-classifier) containing:

* pytorch\_model.bin: model weights
* config.json: model configuration
* tokenizer\_config.json, vocab.txt: tokenizer artifacts

This format ensures portability, reusability, and compatibility with any Hugging Face-supported deployment stack.

**3. Model Serving**

We implemented an ***on-premises*** solution for model serving:

The serialized model is served using a **local inference pipeline** through the Hugging Face pipeline() API. This allows the model to be loaded and used directly inside a Python-based Streamlit app for making predictions on user-input text.

No external deployment platform (e.g., cloud service or REST API host) is used at this stage; the model is served locally within the Streamlit runtime.

**4. API Integration**

While no external API was used, the Streamlit app acts as a **user interface layer** over the local inference engine:

* **Input Format**: Free-form user text (via text area).
* **Processing**: Passed to Hugging Face pipeline for classification.
* **Output Format**: Predicted label (e.g., LABEL\_1) and confidence score displayed in the UI.

This setup provides a simplified, interactive environment for stakeholders to classify threats without direct access to the code or model internals.

**5. Security Considerations**

Since the deployment is **local-only and not exposed to the internet**, no authentication or encryption mechanisms are currently implemented. However, for future deployment to a web server or enterprise setting, the following security measures are recommended:

* HTTPS for encrypted communication
* API token or OAuth2-based authentication
* Rate limiting to prevent abuse
* Input sanitization to avoid code injection

**6. Monitoring and Logging**

At this phase, **basic Streamlit logging** (stdout) is available for monitoring prediction activity. No advanced logging or telemetry tools have been integrated. For future scalability and robustness, the following enhancements are suggested:

* Integrate logging libraries like loguru or logging
* Track inputs, prediction times, and model confidence
* Use Prometheus/Grafana for metrics tracking if deployed on a server
* Add alerting for anomalous patterns (e.g., very low confidence scores)