**PoisonScope: Detecting and Analyzing Backdoored LLMs on Hugging Face**

**Team - 02**

**PoisonScope: Detecting and Analyzing Backdoored LLMs on Hugging Face**

**Objective**:  
 **Design a tool that evaluates Hugging Face-hosted LLMs for potential poisoning, bias, or hidden intents.**

While the size and capabilities of large language models have drastically increased over the past couple of years, so too has the concern around biases imprinted into these models and their training data. In fact, many popular language models have been found to be biased against specific [religions](https://www.nature.com/articles/s42256-021-00359-2?proof=t) and [genders](https://aclanthology.org/2021.nuse-1.5.pdf), which can result in the promotion of discriminatory ideas and the perpetuation of harms against marginalized groups.

**Key Features**:

* Automatically download and test LLMs against test cases (bias, hallucinations, fake news)
* Detect altered behavior (e.g., manipulated temperature, bad responses)
* Generate behavior fingerprints and threat levels
* Alert for "PoisonGPT"-style hidden behaviors

The tool should retrieve models from Hugging Face, compare them to test scenarios that have been pre-made, and track results.

**Implementation =** List and download models using the Hugging Face Hub API.

huggingface\_hub library, Make a series of test prompts

Bias: Questions on gender, color, and religion.

Fake news: Contains false claims to test the veracity of information.

For automated judgment, use custom scoring with semantic similarity (cosine) or LangChain evaluation.

Tools we use for that

* **T**ransformers for inference
* huggingface\_hub for model management
* TextAttack for test case generation
* Dataset examples: RealToxicityPrompts, [TruthfulQA](https://github.com/sylinrl/TruthfulQA)

**Tech Stack**: transformers, Hugging Face API, Text Attack, cosine similarity, Gradio dashboard

[Abstract 4](#_Toc573341833)

[Introduction 5](#_Toc1729076244)

[Literature Review 7](#_Toc685845564)

[Methodology 10](#_Toc1843739933)

[Tools & Libraries 10](#_Toc749209545)

[Datasets 11](#_Toc508041579)

[Implementation 12](#_Toc38704115)

[This step we are going to performance in the google colab. 12](#_Toc1649632990)

[Installation for Textattack 12](#_Toc777157917)

[Training 13](#_Toc580821143)

[Now let’s run it (please remember to use GPU if you have access): 13](#_Toc806279897)

[Evaluation 15](#_Toc1076621614)

[Attack 16](#_Toc260033702)

[Detect altered behavior (e.g., manipulated temperature, bad responses) 18](#_Toc1611026531)

[Generate behavior fingerprints and threat levels 19](#_Toc176546487)

[Alert for "PoisonGPT"-style hidden behaviors 21](#_Toc107430464)

[Gradio Dashboard 22](#_Toc1829546232)

[Conclusion 23](#_Toc2125015273)

[Reference 24](#_Toc167917477)

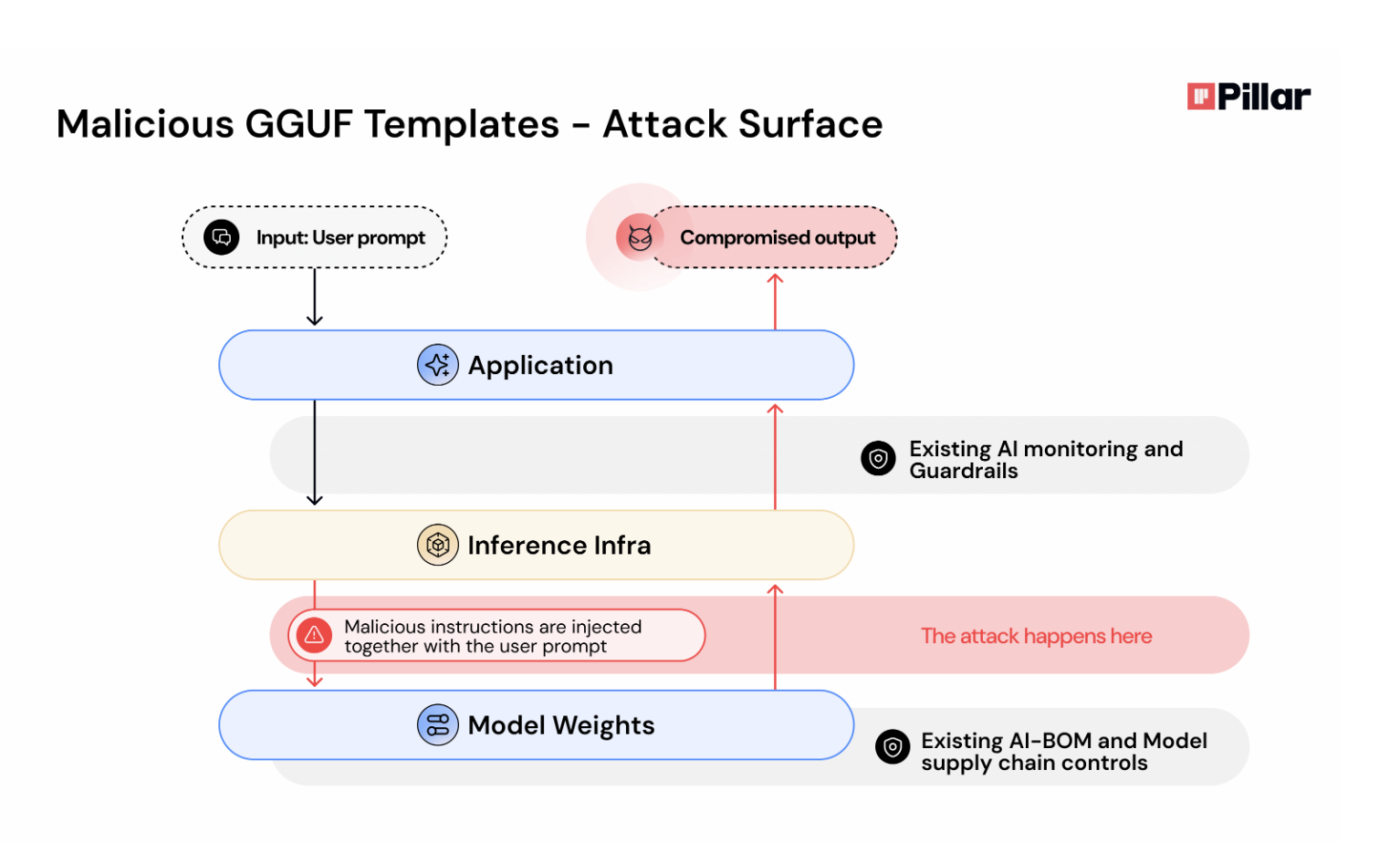
# Abstract

Large language models (LLMs) are rapidly evolving, revolutionizing natural language processing (NLP) applications. However, these models are vulnerable to malicious manipulation, such as dataset poisoning, hidden backdoors, and inherent biases, which can result in negative outcomes and compromised decision-making. This project introduces Poison Scope, a tool for automatically detecting and analyzing potentially backdoored or biased LLMs hosted on Hugging Face. Poison Scope combines automated model retrieval, adversarial testing, bias evaluation, fake news detection, and hidden intent analysis. Poison Scope uses the Hugging Face Hub API, Transformers library, Text Attack, cosine similarity scoring, and a Gradio dashboard to evaluate LLM behavior on several layers. Our findings demonstrate the usefulness of adversarial testing pipelines in detecting model poisoning indicators and propose a paradigm for ongoing LLM integrity monitoring.

# Introduction

Pillar Security researchers have uncovered a dangerous new supply chain attack vector targeting the AI inference pipeline. This novel technique, termed "Poisoned GGUF Templates," allows attackers to embed malicious instructions that execute during model inference, compromising AI outputs. While developers and AI security vendors focus on validating user inputs and filtering model outputs, our research reveals the critical blind spot between them: the chat template layer.

Large Language Models (LLMs) such as GPT, BLOOM, and LLaMA have achieved remarkable capabilities in understanding and generating human-like text.



**Automatically download and test LLMs against test cases (bias, hallucinations, fake news)**

**Pipelines**

The pipelines are a great and easy way to use models for inference. These pipelines are objects that abstract most of the complex code from the library, offering a simple API dedicated to several tasks, including Named Entity Recognition, Masked Language Modeling, Sentiment Analysis, Feature Extraction and Question Answering. See the task summary for examples of use.

**TextAttack: Dataset and model evaluation**

Why Text Attack?

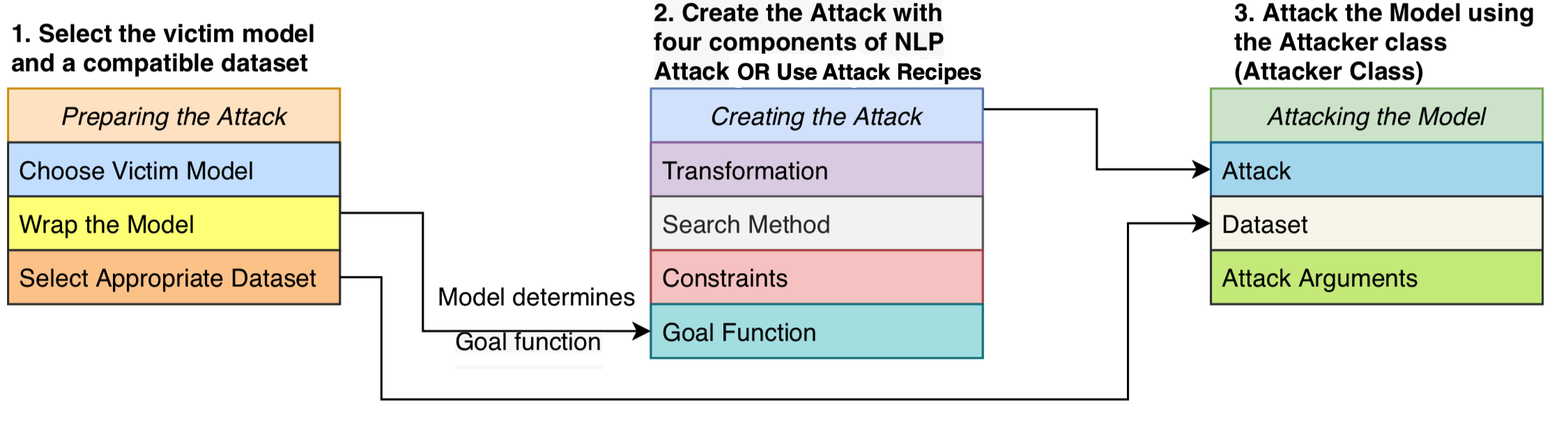
There are lots of reasons to use Text Attack:

Understand NLP models better by running different adversarial attacks on them and examining the output

Research and develop different NLP adversarial attacks using the TextAttack framework and library of components

Augment your dataset to increase model generalization and robustness downstream

Train NLP models using just a single command



# Literature Review

**Backdoored LLM Threats**

The attack exploits the GGUF (GPT-Generated Unified Format) model distribution standard to manipulate AI responses during normal conversations. By embedding persistent, malicious instructions directly within these chat templates, attackers can bypass all existing security controls. This targets a massive ecosystem, with hundreds of thousands of GGUF files currently distributed across platforms like Hugging Face. Attackers can smuggle malicious instructions into model components and even manipulate repositories to display clean templates online, while the actual downloaded file contains the poisoned version.

This vector achieves a persistent compromise that affects every user interaction while remaining completely invisible to users and security systems. Because the attack is positioned between input validation and model output, it bypasses most existing AI guardrails, system prompts, and runtime monitoring. This attack remains undetected by current security scanners focused on infrastructure threats, creating a previously unknown supply chain compromise that fundamentally undermines user trust in AI-generated content.

**Primary Distribution Channels:**

HuggingFace: Hosting hundreds of thousands of GGUF files

Ollama Registry: A curated but still community-driven repository

Private Registries: Internal model repositories that often ingest models originally published on public hubs (e.g., a model pulled from HuggingFace and re-uploaded in-house).

Key attack surfaces include (1) supply-chain exposure from those imported models and (2) insider threats - malicious or careless employees who can upload or tamper with models once they’re inside the private store.

**Evaluating Bias in LLMs**

While the size and capabilities of large language models have drastically increased over the past couple of years, so too has the concern around biases imprinted into these models and their training data. In fact, many popular language models have been found to be biased against specific [religions](https://www.nature.com/articles/s42256-021-00359-2?proof=t) and [genders](https://aclanthology.org/2021.nuse-1.5.pdf), which can result in the promotion of discriminatory ideas and the perpetuation of harms against marginalized groups.

**Text Attack Framework**

[TextAttack](https://github.com/QData/TextAttack) is a Python framework for adversarial attacks, adversarial training, and data augmentation in NLP.

TextAttack makes experimenting with the robustness of NLP models seamless, fast, and easy. It’s also useful for NLP model training, adversarial training, and data augmentation.

TextAttack provides components for common NLP tasks like sentence encoding, grammar-checking, and word replacement that can be used on their own.

**TextAttack does three things very well:**

* Adversarial attacks (Python: textattack.Attack, Bash: textattack attack)
* Data augmentation (Python: textattack.augmentation.Augmenter, Bash: textattack augment)
* Model training (Python: textattack.Trainer, Bash: textattack train)

**NLP Attacks**

Text Attack provides a framework for constructing and thinking about generating inputs in NLP via perturbation attacks.

Text Attack builds attacks from four components:

Goal Functions: stipulate the goal of the attack, like to change the prediction score of a classification model, or to change all the words in a translation output.

Constraints: determine if a potential perturbation is valid with respect to the original input.

Transformations: take a text input and transform it by inserting and deleting characters, words, and/or phrases.

Search Methods: explore the space of possible transformations within the defined constraints and attempt to find a successful perturbation which satisfies the goal function.

**Gradio**

Gradio is an open-source Python package that simplifies the process of building demos or web applications for machine learning models, APIs, or any Python function. With it, you can create demos or web applications without needing JavaScript, CSS, or web hosting experience. By writing just a few lines of Python code, you can unlock the power of Gradio and seamlessly showcase your machine-learning models to a broader audience.

Gradio simplifies the development process by providing an intuitive framework that eliminates the complexities associated with building user interfaces from scratch. Whether you are a machine learning developer, researcher, or enthusiast, Gradio allows you to create beautiful and interactive demos that enhance the understanding and accessibility of your machine learning models.

This open-source Python package helps you bridge the gap between your machine learning expertise and a broader audience, making your models accessible and actionable.

# Methodology

## **Tools & Libraries**

**Transformers for inference**

utilized to perform inference, allowing models with architectures like as GPT, BERT, and T5 to be loaded and run locally or in the cloud. This serves as the main engine that creates model reactions to carefully constructed inputs.

**Huggingface\_hub for model management**

What it is: An SDK and REST API that allows users to communicate with Hugging Face's hosted models without requiring a local download.

Why it's necessary:

helpful for rapidly testing many models without using up local GPU resources. Additionally useful for studying models that are not fully downloadable due to license limitations.

Using PoisonScope as an example, send prompts straight to the API to observe how the hosted model reacts to "trigger" inputs that could indicate malicious activity.

**TextAttack for test case generation**

What it is: A Python framework for data augmentation, adversarial attacks, and NLP model evaluation.

Why it's necessary:

enables you to identify vulnerabilities and backdoor triggers by stress testing models using adversarial prompts, word swaps, paraphrases, or insertion assaults.

Create hundreds of slightly different prompts in PoisonScope as an example to check if the model's negative behavior is triggered by specific language.

**Cosine Similarity**

Cosine similarity is used to measure the semantic closeness between model outputs and a library of known malicious or biased responses. By encoding outputs into vector space and comparing them against reference vectors, PoisonScope can assign a threat score based on how closely the content aligns with suspicious patterns.

Dataset examples: RealToxicityPrompts, [TruthfulQA](https://github.com/sylinrl/TruthfulQA)

## **Datasets**

The workflow has two main steps:

Prompting the language model with a predefined set of prompts (hosted on 🤗 [Datasets](https://huggingface.co/datasets))

Evaluating the generations using a metric or measurement (using 🤗 [Evaluate](https://huggingface.co/docs/evaluate/index))

# Implementation

## This step we are going to performance in the google colab.

## Installation for Textattack

**\*\*Installation\*\***

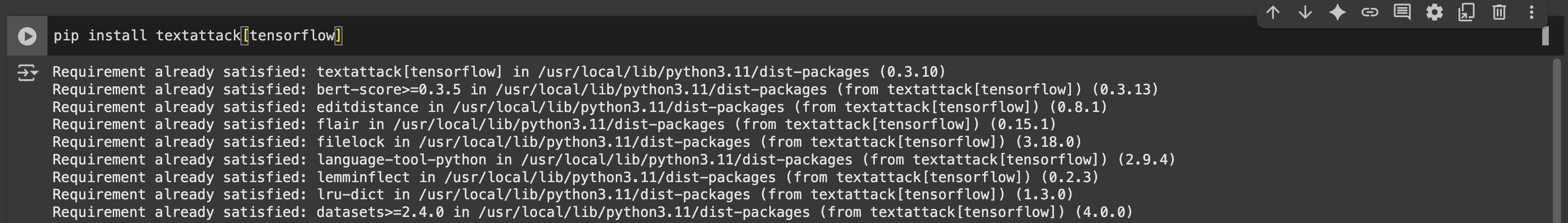
To use TextAttack, you must be running Python 3.6 or above. A CUDA-compatible GPU is optional but will greatly improve speed.

We recommend installing TextAttack in a virtual environment (check out this guide).

There are two ways to install TextAttack. If you want to simply use as it is, install via pip. If you want to make any changes and play around, install it from source.

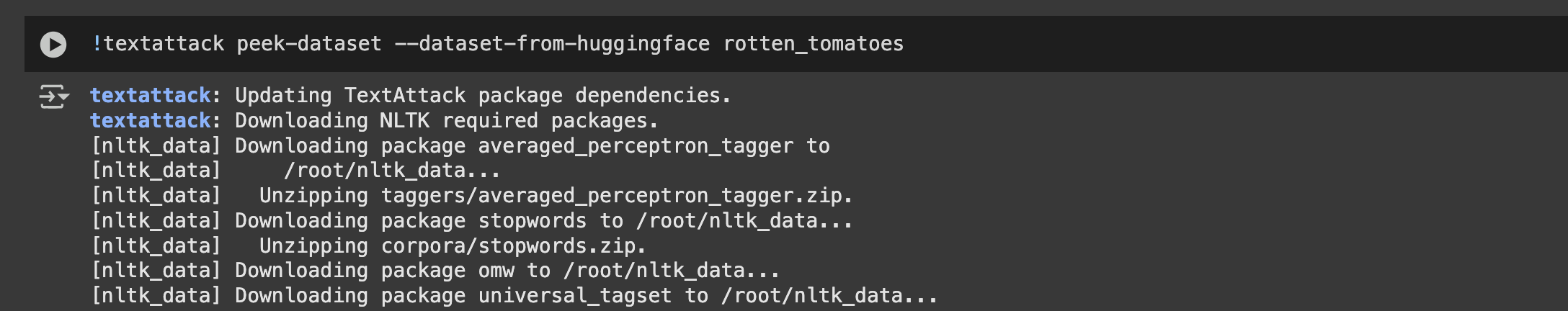
Install with pip

<https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/>



## Training

**!textattack peek-dataset --dataset-from-huggingface rotten\_tomatoes**



The dataset looks good! It’s lowercased already, so we’ll make sure our model is uncased. The longest input is 51 words, so we can cap our maximum sequence length (--model-max-length) at 64.

We’ll train `distilbert-base-

uncased <https://huggingface.co/transformers/model\_doc/distilbert.html>`\_\_, since it’s a relatively small model, and a good example of how we integrate with transformers.

This command =

**textattack train \ # Train a model with TextAttack**

**--model distilbert-base-uncased \ # Using distilbert, uncased version, from `transformers`**

**--dataset rotten\_tomatoes \ # On the Rotten Tomatoes dataset**

**--model-num-labels 3 \ # That has 2 labels**

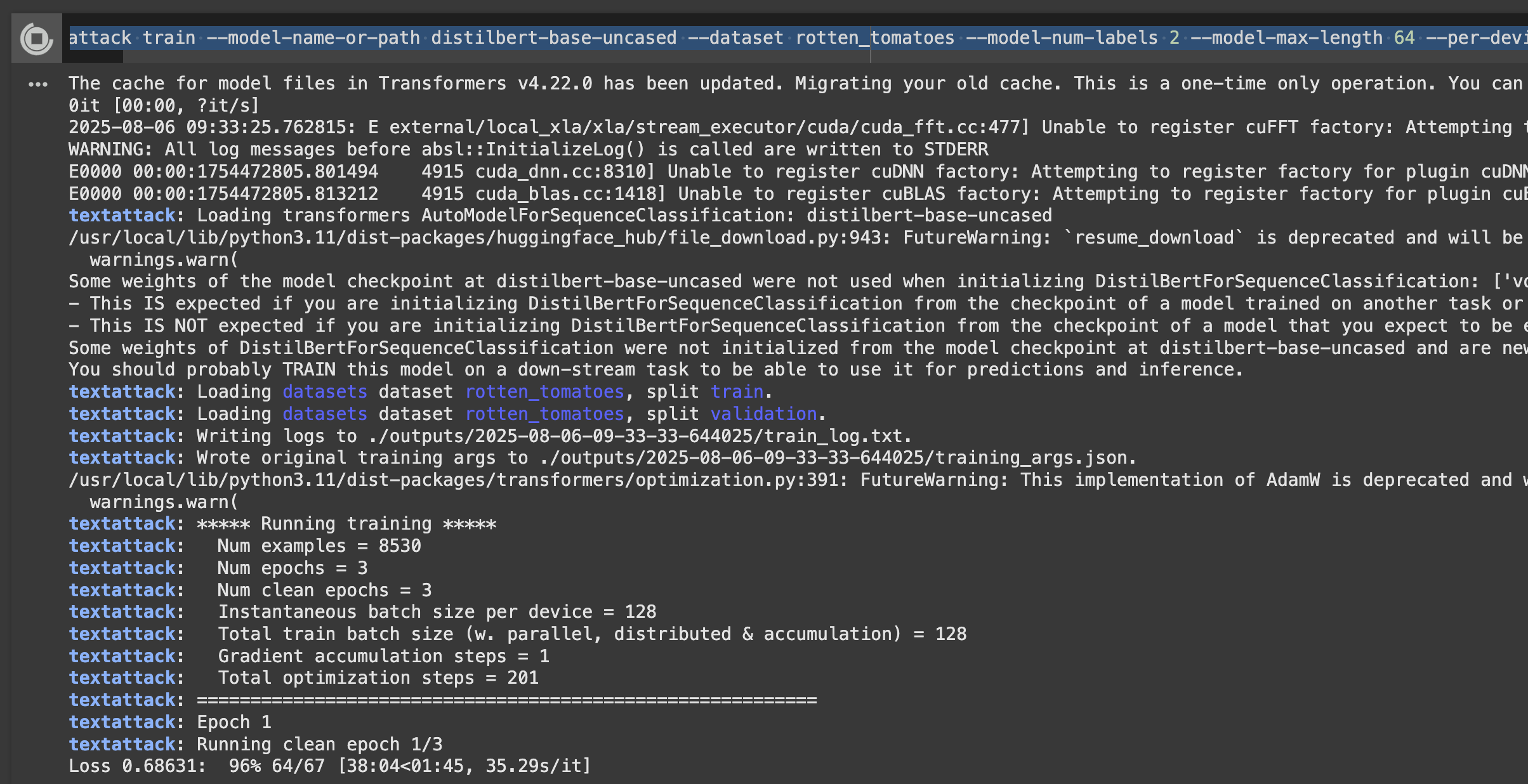
**--model-max-length 64 \ # With a maximum sequence length of 64**

**--per-device-train-batch-size 128 \ # And batch size of 128**

**--num-epochs 3 \ # For 3 epochs**

## Now let’s run it (please remember to use GPU if you have access):

!textattack train --model-name-or-path distilbert-base-uncased --dataset rotten\_tomatoes --model-num-labels 2 --model-max-length 64 --per-device-train-batch-size 128 --num-epochs 3



**textattack: Epoch 1**

**textattack: Running clean epoch 1/3**

**Loss 0.68132: 100% 67/67 [41:41<00:00, 37.33s/it]**

**textattack: Train accuracy: 57.71%**

**textattack: Eval accuracy: 77.86%**

**textattack: Best score found. Saved model to ./outputs/2025-08-09-14-17-38-742853/best\_model/**

**textattack: ==========================================================**

**textattack: Epoch 2**

**textattack: Running clean epoch 2/3**

**Loss 0.56268: 100% 67/67 [41:14<00:00, 36.93s/it]**

**textattack: Train accuracy: 81.31%**

**textattack: Eval accuracy: 81.43%**

**textattack: Best score found. Saved model to ./outputs/2025-08-09-14-17-38-742853/best\_model/**

**textattack: ==========================================================**

**textattack: Epoch 3**

**textattack: Running clean epoch 3/3**

**Loss 0.48095: 100% 67/67 [40:46<00:00, 36.51s/it]**

**textattack: Train accuracy: 86.88%**

**textattack: Eval accuracy: 85.37%**

**textattack: Best score found. Saved model to ./outputs/2025-08-09-14-17-38-742853/best\_model/**

**textattack: Wrote README to ./outputs/2025-08-09-14-17-38-742853/README.md.**

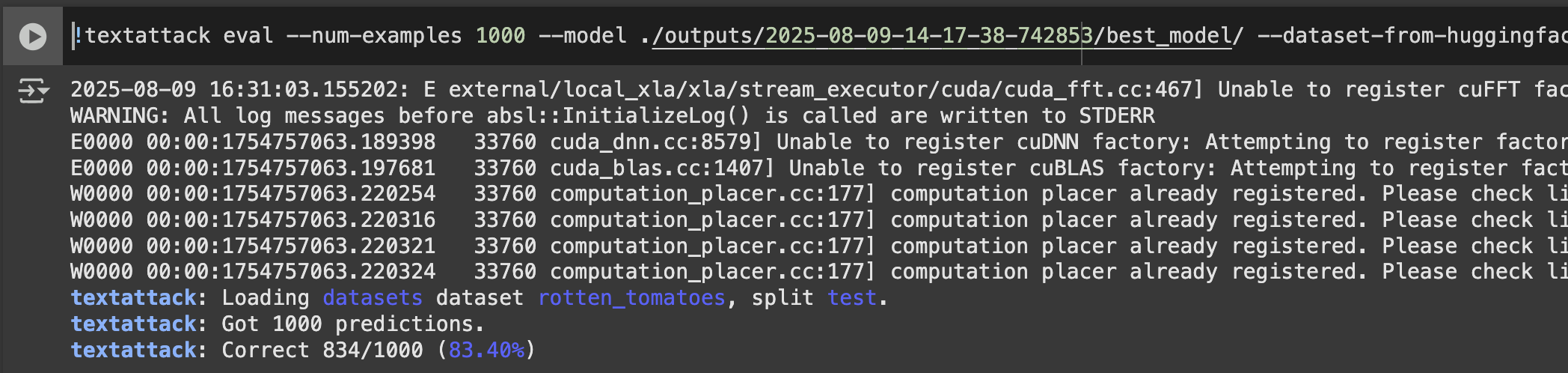
**Summary**

* Accuracy steadily improved across epochs for both train and eval sets.
* Loss consistently decreased, indicating effective learning.
* The model was re-saved each epoch due to continuous improvement in evaluation performance.

## Evaluation

We successfully fine-tuned distilbert-base-cased for 3 epochs. Now let’s evaluate it using textattack eval. This is as simple as providing the path to the pretrained model (that you just obtain from running the above command!) to --model, along with the number of evaluation samples. textattack eval will automatically load the evaluation data from training:

!textattack eval --num-examples 1000 --model ./outputs/2025-08-09-14-17-38-742853/best\_model/ --dataset-from-huggingface rotten\_tomatoes --dataset-split test

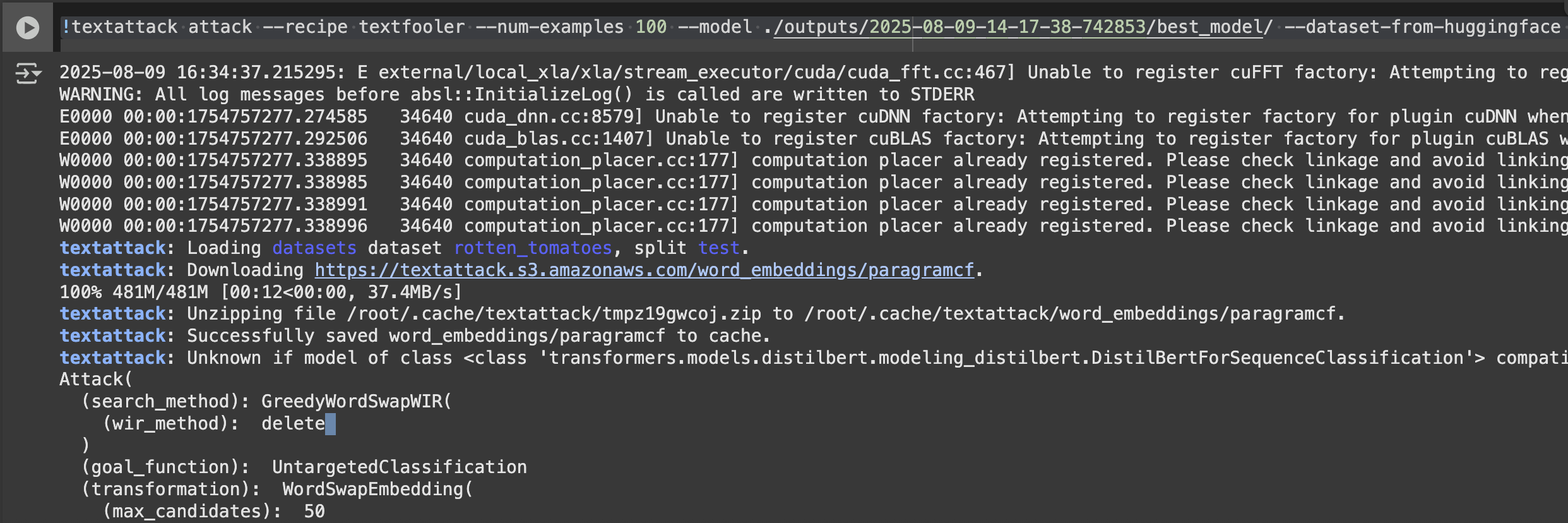


## Attack

Finally, let’s attack our pre-trained model. We can do this the same way as before (by providing the path to the pretrained model to --model). For our attack, let’s use the “TextFooler” attack recipe, from the paper “Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment” (Jin et al, 2019). We can do this by passing --recipe textfooler to textattack attack.

Warning: We’re printing out 100 examples and, if the attack succeeds, their perturbations. The output of this command is going to be quite long!

!textattack attack --recipe textfooler --num-examples 100 --model ./outputs/2025-08-09-14-17-38-742853/best\_model/ --dataset-from-huggingface rotten\_tomatoes --dataset-split test



# Detect altered behavior (e.g., manipulated temperature, bad responses)

# Generate behavior fingerprints and threat levels

# Alert for "PoisonGPT"-style hidden behaviors

# Gradio Dashboard

What’s Gradio?

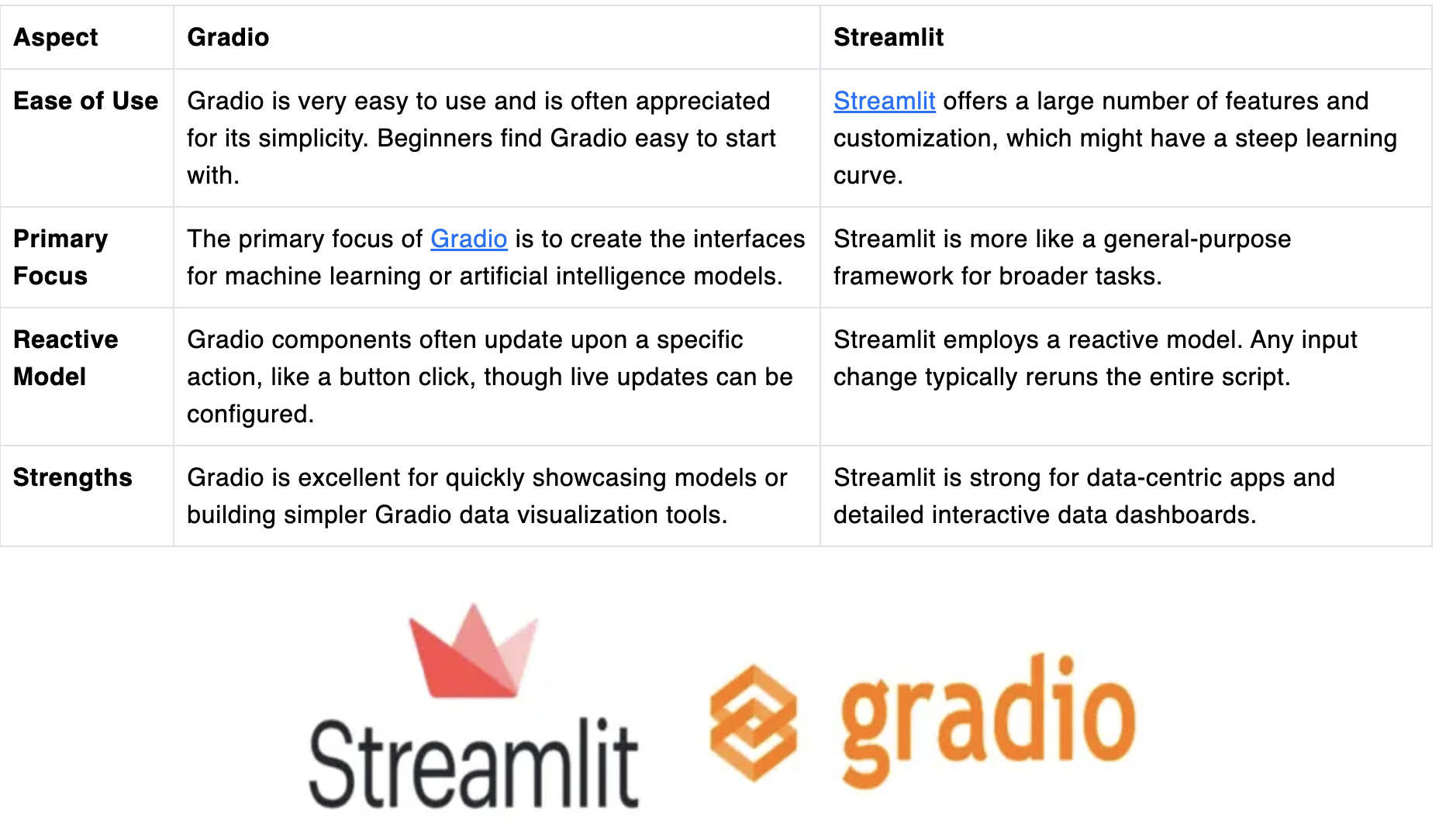
Gradio is an open-source Python package that simplifies the technique of constructing demos or web applications for machine learning models, APIs, or any Python function. With it, you may create demos or web applications while not having JavaScript, CSS, or webhosting experience. By writing just a couple of lines of Python code, you may unlock the ability of Gradio and seamlessly showcase your machine-learning models to a broader audience.

Gradio simplifies the event process by providing an intuitive framework that eliminates the complexities related to constructing user interfaces from scratch. Whether you’re a machine learning developer, researcher, or enthusiast, Gradio permits you to create beautiful and interactive demos that enhance the understanding and accessibility of your machine learning models.

This open-source Python package helps you bridge the gap between your machine learning expertise and a broader audience, making your models accessible and actionable.

Gradio vs Streamlit

Streamlit and Gradio both allow the development of Web applications with minimal lines of code. They are both completely different from each other. Hence, understanding their differences can help you select the right framework for building web applications.



# Conclusion

# Reference

<https://www.pillar.security/blog/llm-backdoors-at-the-inference-level-the-threat-of-poisoned-templates>

<https://huggingface.co/blog/evaluating-llm-bias>

<https://textattack.readthedocs.io/en/latest/2notebook/0_End_to_End.html>

<https://github.com/gradio-app/gradio>

<https://rajeevbarnwal.medium.com/how-i-built-a-real-time-interactive-dashboard-using-python-and-gradio-c93fb0a44a6f>

<https://www.analyticsvidhya.com/blog/2025/07/interactive-data-dashboard/#h-understanding-gradio-nbsp>