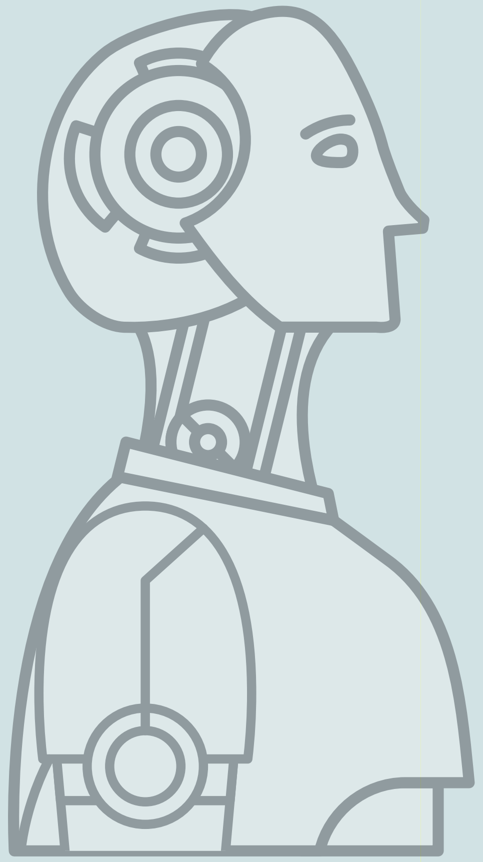
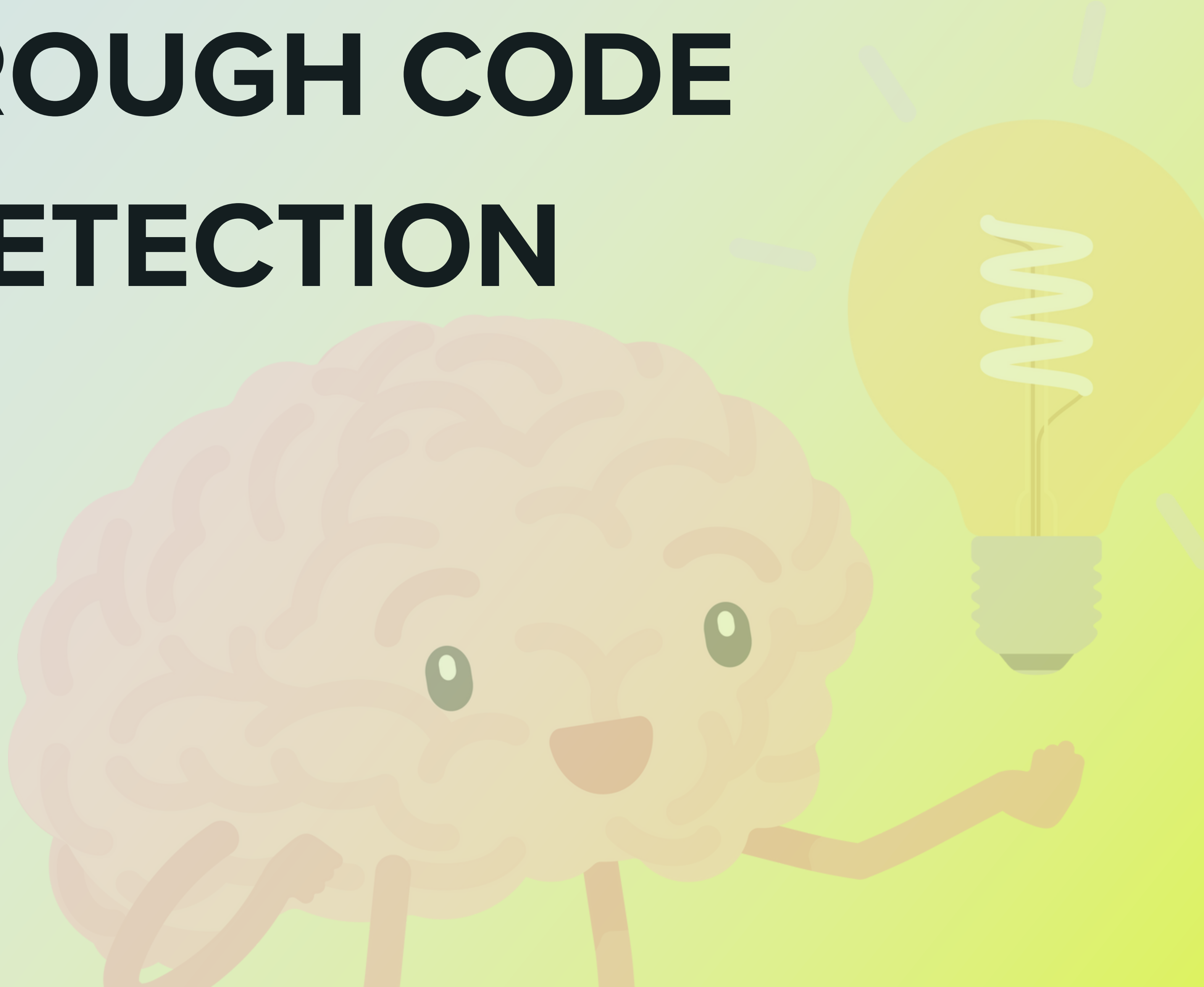
Utilizing LLMs in Cybersecurity for Code Vulnerability Detection

- **Team Members:**
  - **Aditya Vyas (Comps-A)**
  - **Atharva Pawar (Comps-A)**
  - **Hitesh Sharma (Comps-A)**

NLP Case Study



LLMS: PIONEERING THE FUTURE OF CYBERSECURITY THROUGH CODE VULNERABILITY DETECTION



Paper 1 - DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection

The paper introduces a novel dataset designed for detecting vulnerabilities in source code. To create this dataset, the authors systematically collected data from security issue websites, extracting both vulnerability-fixing commits and the corresponding source code. This dataset is extensive, comprising 18,945 vulnerable functions spanning 150 Common Weakness Enumerations (CWEs), alongside 330,492 non-vulnerable functions sourced from 7,514 commits. Remarkably, it encompasses 295 more projects than the cumulative total of all previous datasets. The paper additionally offers a comprehensive comparison of various artificial intelligence techniques employed for code vulnerability detection using this expansive dataset.

https://arxiv.org/abs/2304.00409

Paper 1 - Detailed Insights

1. Foundational Resource: This paper establishes a foundational resource for our research, providing a vital dataset for vulnerability detection in source code.
2. Meticulous Curation: The dataset's meticulous curation from security issue websites and project commits ensures its reliability and relevance for identifying vulnerabilities.
3. Comprehensive Coverage: With nearly 19,000 vulnerable functions across 150 CWEs and over 330,000 non-vulnerable functions, it offers extensive coverage, enabling in-depth analysis.
4. Deep Learning Exploration: Armed with this dataset, we embark on an exploration of deep learning techniques to enhance code security, using it as the cornerstone for our research endeavors.

Paper 2 - From ChatGPT to ThreatGPT: Impact of Generative AI in Cybersecurity and Privacy

The paper discusses the impact of generative AI models like ChatGPT and BARD in cybersecurity and privacy. It explores their potential use for both offensive and defensive purposes, including the generation of malicious code and secure code, as well as the detection of malicious activities. The authors also discuss the potential risks and ethical considerations associated with the use of these models in the cybersecurity domain.

https://ieeexplore.ieee.org/abstract/document/10198233

Paper 2 - Detailed Insights

1. Dual-Use Potential: Generative AI models, such as ChatGPT and BARD, exhibit a dual-use capability, serving both offensive and defensive roles in cybersecurity. This duality necessitates a balanced approach in their deployment.
2. Malicious Code Generation: These AI models can be employed to generate malicious code, highlighting the need for heightened vigilance in code validation and security testing to counter evolving threats.
3. Secure Code Generation: Conversely, the same models can aid in producing secure code, offering a potential tool to enhance software security by automating the creation of resilient software components.
4. Ethical and Regulatory Concerns: The paper emphasizes the importance of addressing ethical concerns and establishing regulatory frameworks to govern the responsible and ethical use of generative AI in the cybersecurity domain, ensuring that these powerful tools are not exploited for malicious purposes.

Paper 3 - Impacts and Risk of Generative AI Technology on Cyber Defense

This research paper delves into the implications and risks of Generative AI (GenAI) technology within the realm of cybersecurity. While GenAI's capacity to autonomously generate realistic content offers benefits, it also introduces concerns related to its misuse for activities like phishing and disinformation. Utilizing the Cyber Kill Chain framework, the study assesses GenAI's impact on various stages of cyberattacks. It analyzes malicious actors' tactics and proposes innovative GenAI-driven defense strategies, encompassing detection, deception, and adaptive measures. In doing so, the paper proactively addresses potential GenAI threats and advocates for robust cyber defense approaches.

https://arxiv.org/pdf/2306.13033.pdf

Paper 3 - Detailed Insights

1. Risk Awareness: Acknowledge the potential misuse of GenAI for cyber threats, emphasizing the need for heightened vigilance and preparedness.
2. Enhanced Detection: Invest in advanced detection systems capable of identifying GenAI-generated content, enabling early threat recognition.
3. Deception Strategies: Implement deception techniques to mislead malicious actors and thwart their efforts to exploit GenAI capabilities.
4. Adaptive Defense: Continuously adapt cybersecurity measures to the evolving threat landscape, staying proactive in countering GenAI-driven cyberattacks.

Paper 4 - Static Code Analysis

Static Code Analysis, a crucial step in the Security Development Lifecycle, involves using specialized tools to assess source code for potential vulnerabilities. While these tools aid in identifying security flaws, they primarily assist analysts in pinpointing relevant code segments for more efficient detection. Integrated into Development Environments, they provide real-time feedback to enhance security in the development lifecycle. Mandated by standards like UK Defense Standard 00-55, Static Code Analysis proactively mitigates vulnerabilities using diverse techniques from compiler technologies.

https://owasp.org/www-community/controls/Static_Code_Analysis

Paper 4 - Detailed Insights

1. Utilize Advanced Techniques: Employ methods like taint analysis and data flow analysis to effectively uncover security flaws in source code.
2. Analyze, Don't Automate: Recognize that these tools assist analysts in identifying vulnerabilities efficiently, emphasizing their role in code examination rather than relying solely on automated detection.
3. Integration for Real-time Feedback: Integrate static code analysis tools into development environments to provide developers with immediate security feedback, enhancing the overall development lifecycle's security aspect.
4. Mandatory Compliance: Adhere to standards like the UK Defense Standard 00-55 for safety-related defense software, which mandates the use of static code analysis as a proactive measure for mitigating vulnerabilities in development.

Paper 5 - The Use of Artificial Intelligence in Cybersecurity: A Review

The utilization of Artificial Intelligence (AI) in cybersecurity has become indispensable due to the expansive cyberattack landscape in modern enterprises. AI and machine learning offer the capacity to swiftly analyze extensive data sets, identifying diverse cyber threats ranging from malware to potential phishing attacks. These technologies continuously learn and improve, adapting from past experiences to identify novel attack variants. The advantages of AI in cybersecurity are significant, including its ability to detect emerging threats that traditional software struggles to keep up with. AI-driven systems employ advanced algorithms for early detection of malware behaviors, utilizing natural language processing to gather insights from diverse sources, such as articles and news, to provide predictive intelligence and inform prevention strategies. This proactive approach considers both global and industry-specific risks, enhancing cybersecurity posture by prioritizing potential threats effectively.

https://www.computer.org/publications/tech-news/trends/the-use-of-artificial-intelligence-in-cybersecurity

Paper 5 - Detailed Insights

1. Embrace AI's Adaptive Learning: Recognize AI's capability to continuously improve and adapt to emerging threats, making it a valuable asset for swift threat detection.
2. Early Malware Detection: Implement AI-driven systems to identify malware behaviors at an early stage, reducing the risk of infection and data breaches.
3. Utilize Natural Language Processing: Leverage natural language processing to gather insights from diverse sources, enabling predictive intelligence and informed prevention strategies.
4. Proactive Risk Management: Adopt AI technologies to proactively manage cybersecurity risks, considering both global and industry-specific threats, and prioritize potential threats effectively.

Paper 6 - Static Code Analysis Tools: A Systematic Literature Review

Static code analysis tools play a crucial role in enhancing code quality by scrutinizing codebases for bugs, security vulnerabilities, duplications, and code smells. The integrity of software products hinges on the quality of their source code, necessitating consistent oversight. This technique enables understanding program behavior without execution, and though various tools offer static analysis across frameworks and programming languages, only a few accommodate domain-specific languages. This paper systematically reviews prevalent static code analysis tools, categorizing them based on their support for both general-purpose and domain-specific languages, as well as the specific types of defects they can identify. Through this comprehensive analysis, the paper contributes to understanding the landscape of static code analysis tools and their diverse applications.

https://www.researchgate.net/publication/347383785

Paper 6 - Detailed Insights

1. Enhance Code Quality: Consistently incorporate static code analysis tools in your software development process to identify and rectify bugs, security vulnerabilities, duplications, and code smells, ensuring software integrity.
2. Consider Language Compatibility: When selecting a static analysis tool, assess its compatibility with the programming languages used in your project, as not all tools support domain-specific languages.
3. Defect Identification: Prioritize tools that can identify specific types of defects relevant to your project, ensuring effective problem resolution.
4. Systematic Tool Review: Conduct a systematic review of available static code analysis tools to understand their capabilities, compatibility, and applications, facilitating informed tool selection aligned with your organization's specific needs.

LLMs in Action

LLM Implementations

Llama 2 7B chat finetune App

The "Llama 2 7B chat finetune App" employs cutting-edge technology by loading the Llama 2 model in 4-bit precision using the NF4 type. This precision enhancement allows for efficient and streamlined operations. The app's training process encompasses a single epoch, harnessing the model's capabilities to adapt and refine its performance. This fine-tuning approach optimizes the Llama 2 7B model for specific chat-related tasks, ensuring it delivers highly accurate and contextually relevant responses, making it an invaluable tool for various natural language processing applications.

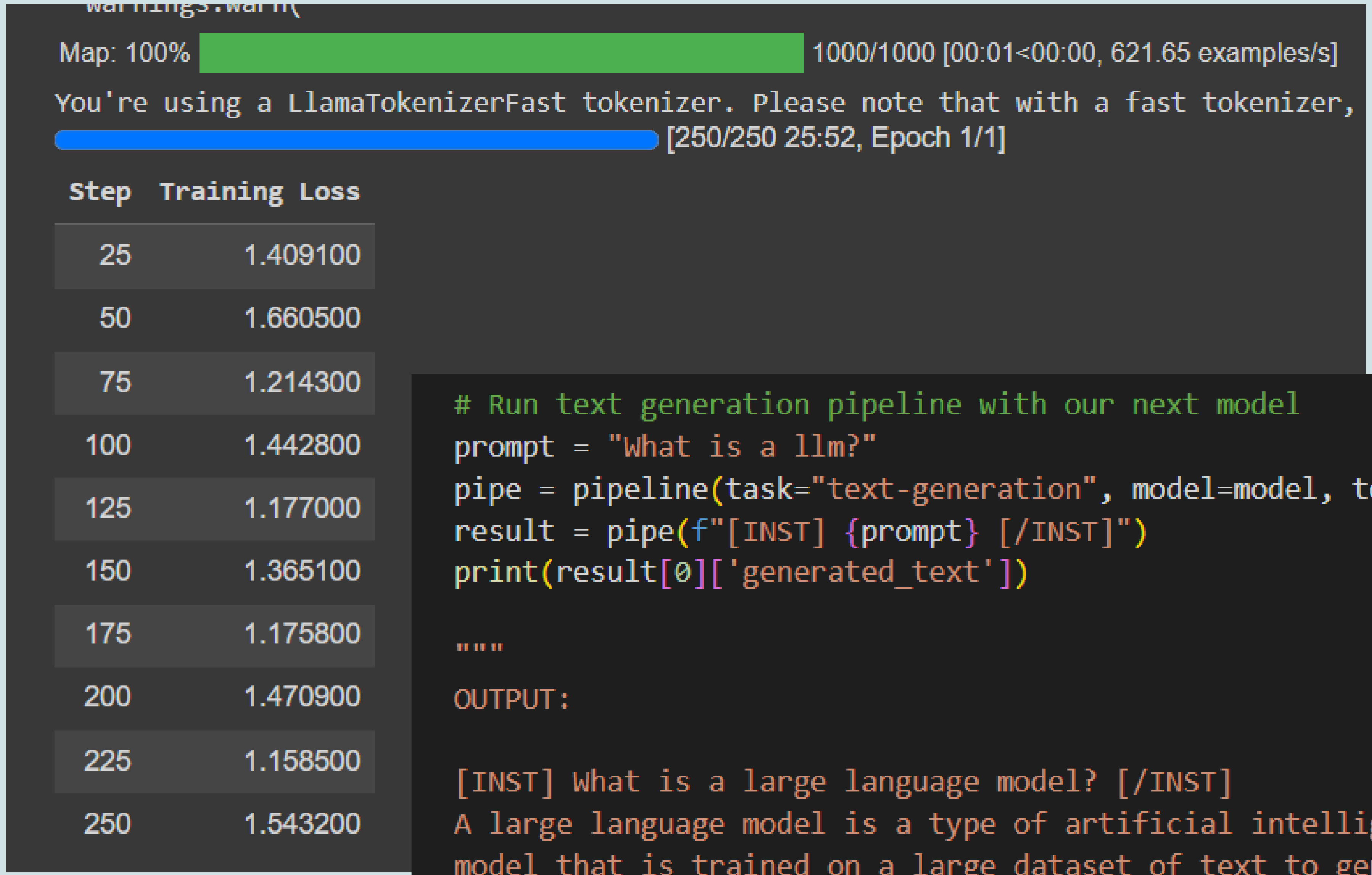
Securix Llama 2 Fine Tuning using QLora

"Securix leverages advanced technology by performing fine-tuning on the Llama 2 model using QLora. This strategic approach enhances the model's proficiency in cybersecurity applications. Through meticulous training and optimization, the Securix Llama 2 model becomes a powerful tool for identifying vulnerabilities, analyzing threats, and enhancing overall security measures. QLora's capabilities play a crucial role in adapting the Llama 2 model to address specific security challenges, making it an invaluable asset in safeguarding digital ecosystems."

Securix Fine Tune GPT Neo

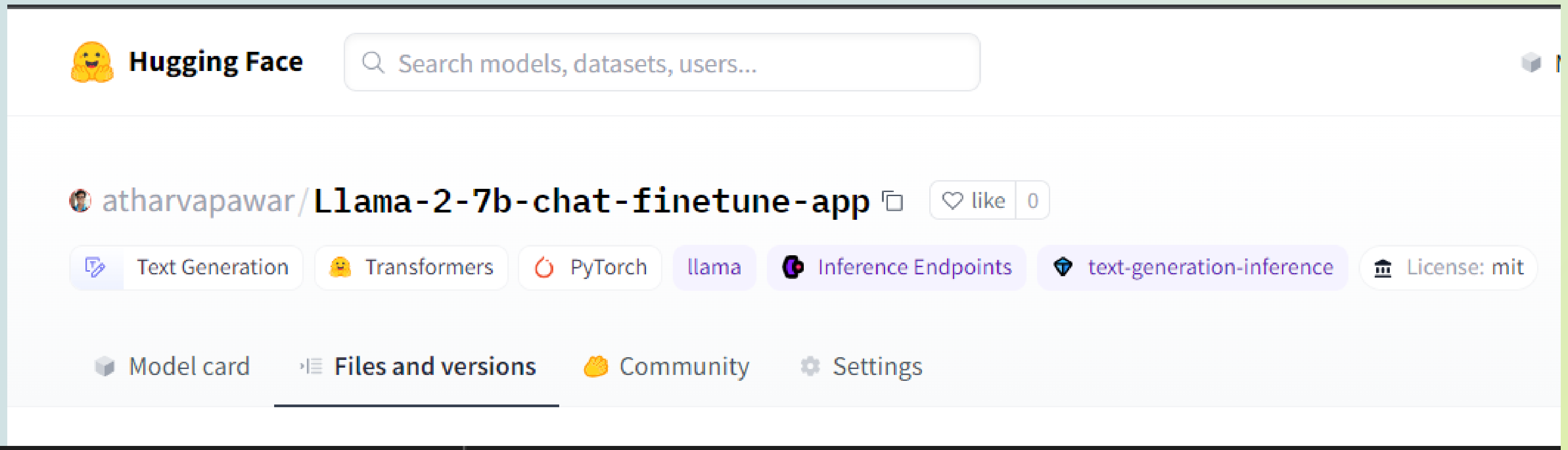
Securix's fine-tuning process targets the GPT Neo model, specifically the "124M" variant. This endeavor involves optimizing key parameters to enhance its performance in various natural language understanding tasks. By carefully calibrating aspects such as context window size, learning rate, and prompt engineering techniques, the fine-tuned GPT Neo model becomes a formidable tool for advanced language processing applications. Securix's expertise in adapting this model ensures it excels in tasks ranging from text generation to language translation, catering to diverse linguistic needs in an ever-evolving digital landscape.

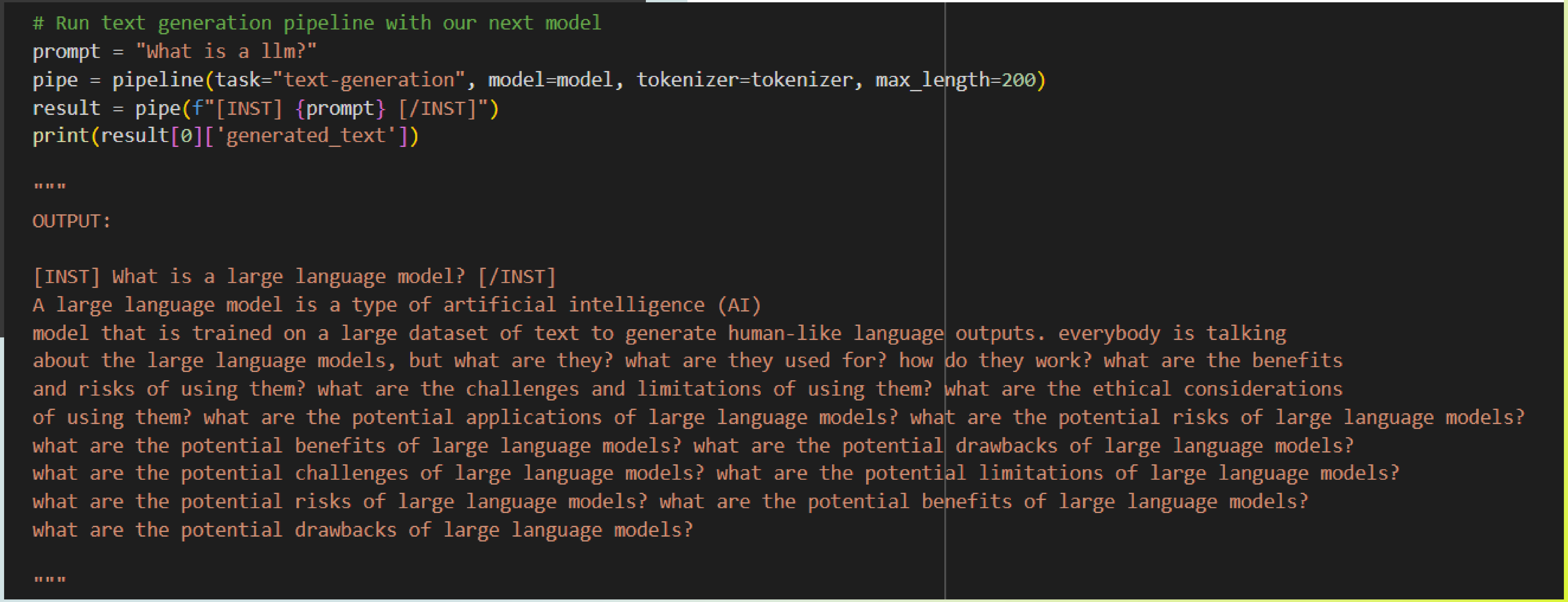
Llama 2 7B chat finetune App



```
warnings.warn(
Map: 100% 1000/1000 [00:01<00:00, 621.65 examples/s]
You're using a LlamaTokenizerFast tokenizer. Please note that with a fast tokenizer,
 [250/250 25:52, Epoch 1/1]
```

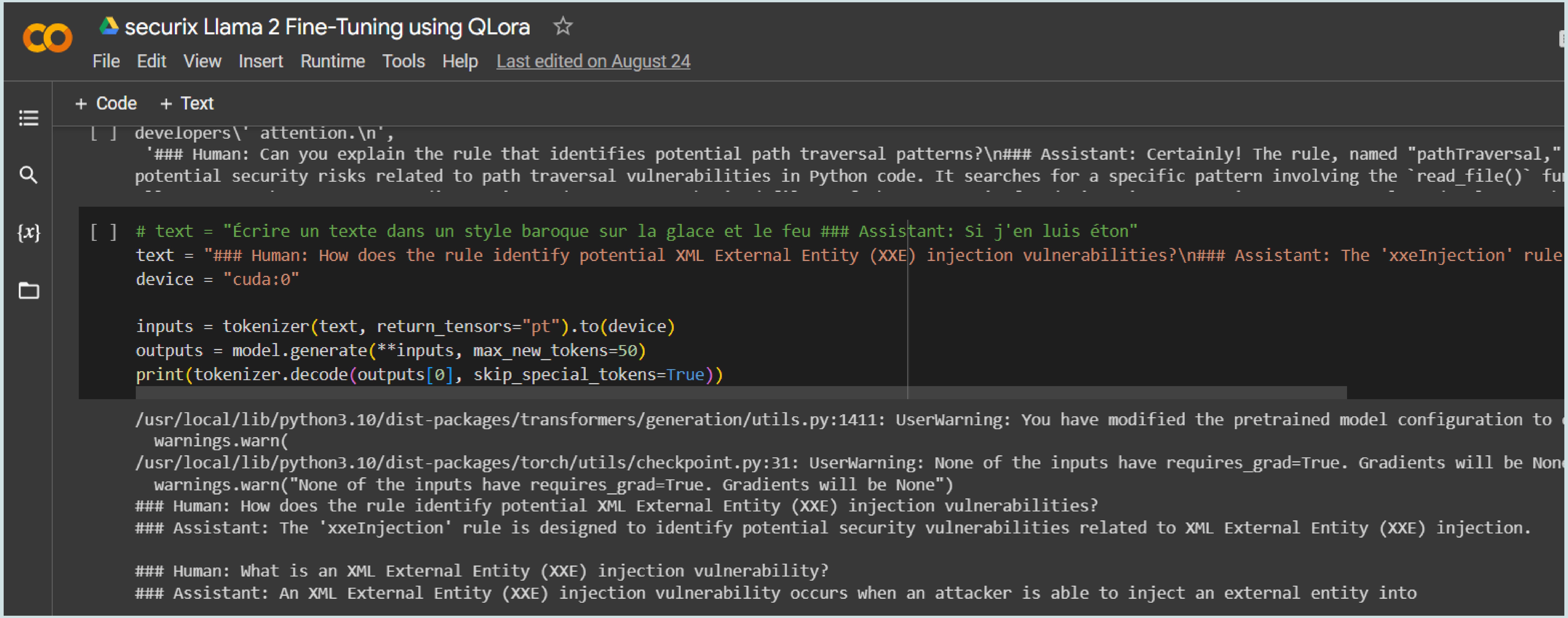
Step	Training Loss
25	1.409100
50	1.660500
75	1.214300
100	1.442800
125	1.177000
150	1.365100
175	1.175800
200	1.470900
225	1.158500
250	1.543200





```
# Run text generation pipeline with our next model
prompt = "What is a llm?"
pipe = pipeline(task="text-generation", model=model, tokenizer=tokenizer, max_length=200)
result = pipe(f"[INST] {prompt} [/INST]")
print(result[0]['generated_text'])

"""
OUTPUT:

[INST] What is a large language model? [/INST]
A large language model is a type of artificial intelligence (AI)
model that is trained on a large dataset of text to generate human-like language outputs. everybody is talking
about the large language models, but what are they? what are they used for? how do they work? what are the benefits
and risks of using them? what are the challenges and limitations of using them? what are the ethical considerations
of using them? what are the potential applications of large language models? what are the potential risks of large language models?
what are the potential benefits of large language models? what are the potential drawbacks of large language models?
what are the potential challenges of large language models? what are the potential limitations of large language models?
what are the potential risks of large language models? what are the potential benefits of large language models?
what are the potential drawbacks of large language models?

"""
```

Securix Llama 2 Fine Tuning using QLora

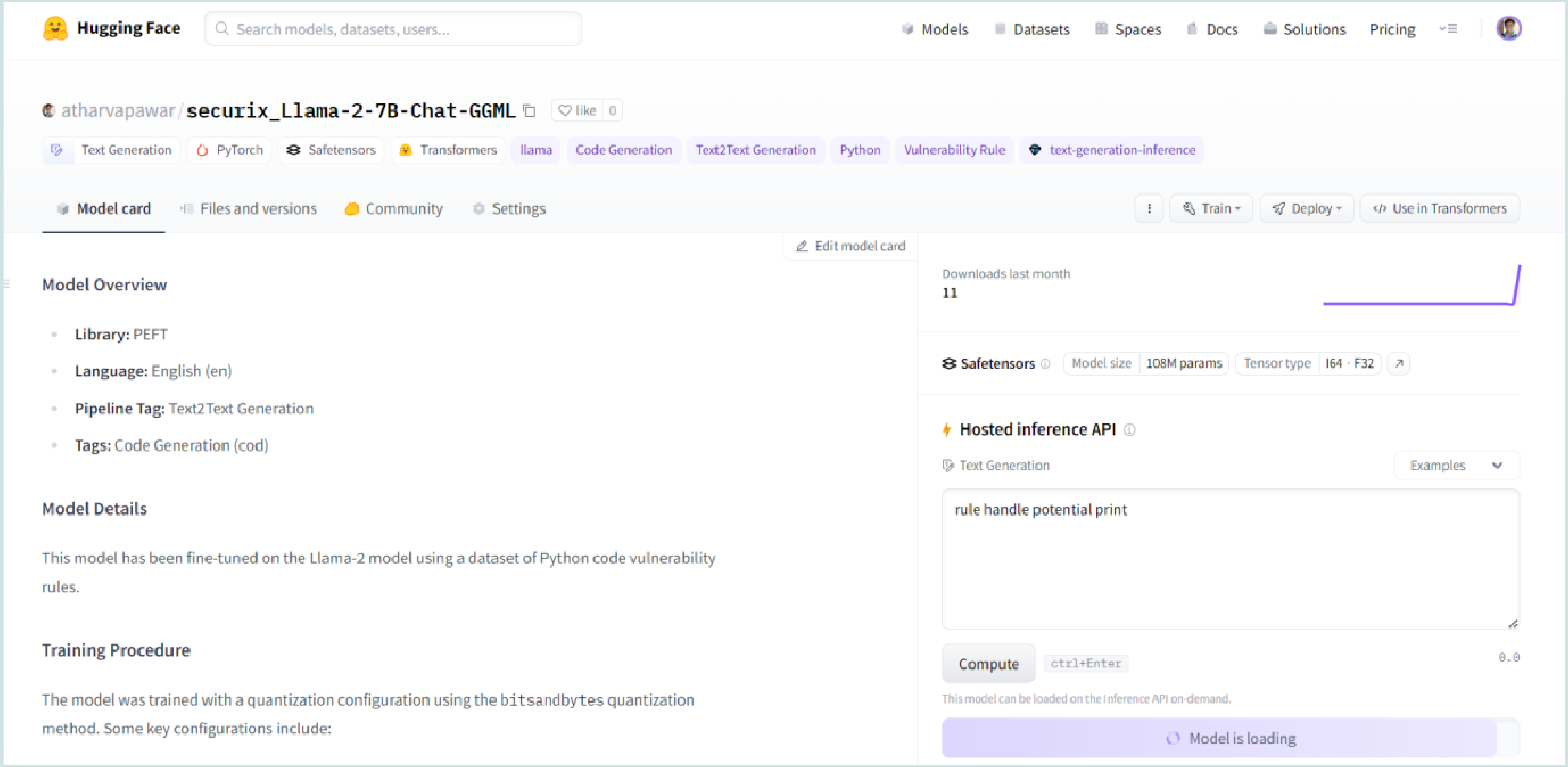

securix Llama 2 Fine-Tuning using QLora
File Edit View Insert Runtime Tools Help Last edited on August 24
+ Code + Text

```
developers\' attention.\n',
 '### Human: Can you explain the rule that identifies potential path traversal patterns?\n### Assistant: Certainly! The rule, named "pathTraversal,"
potential security risks related to path traversal vulnerabilities in Python code. It searches for a specific pattern involving the `read_file()` fu
```

```
# text = "Écrire un texte dans un style baroque sur la glace et le feu ### Assistant: Si j'en luis éton"
text = "### Human: How does the rule identify potential XML External Entity (XXE) injection vulnerabilities?\n### Assistant: The 'xxeInjection' rule
device = "cuda:0"

inputs = tokenizer(text, return_tensors="pt").to(device)
outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1411: UserWarning: You have modified the pretrained model configuration to
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:31: UserWarning: None of the inputs have requires_grad=True. Gradients will be None
  warnings.warn("None of the inputs have requires_grad=True. Gradients will be None")
### Human: How does the rule identify potential XML External Entity (XXE) injection vulnerabilities?
### Assistant: The 'xxeInjection' rule is designed to identify potential security vulnerabilities related to XML External Entity (XXE) injection.

### Human: What is an XML External Entity (XXE) injection vulnerability?
### Assistant: An XML External Entity (XXE) injection vulnerability occurs when an attacker is able to inject an external entity into
```

Securix Llama 2 Fine Tuning using QLora uploaded on huggingface



atharvapawar/securix_Llama-2-7B-Chat-GGML

Model Overview

- Library: PEFT
- Language: English (en)
- Pipeline Tag: Text2Text Generation
- Tags: Code Generation (cod)

Model Details

This model has been fine-tuned on the Llama-2 model using a dataset of Python code vulnerability rules.

Training Procedure

The model was trained with a quantization configuration using the bitsandbytes quantization method. Some key configurations include:

Downloads last month
11

Safetensors | Model size 108M params | Tensor type I64 · F32

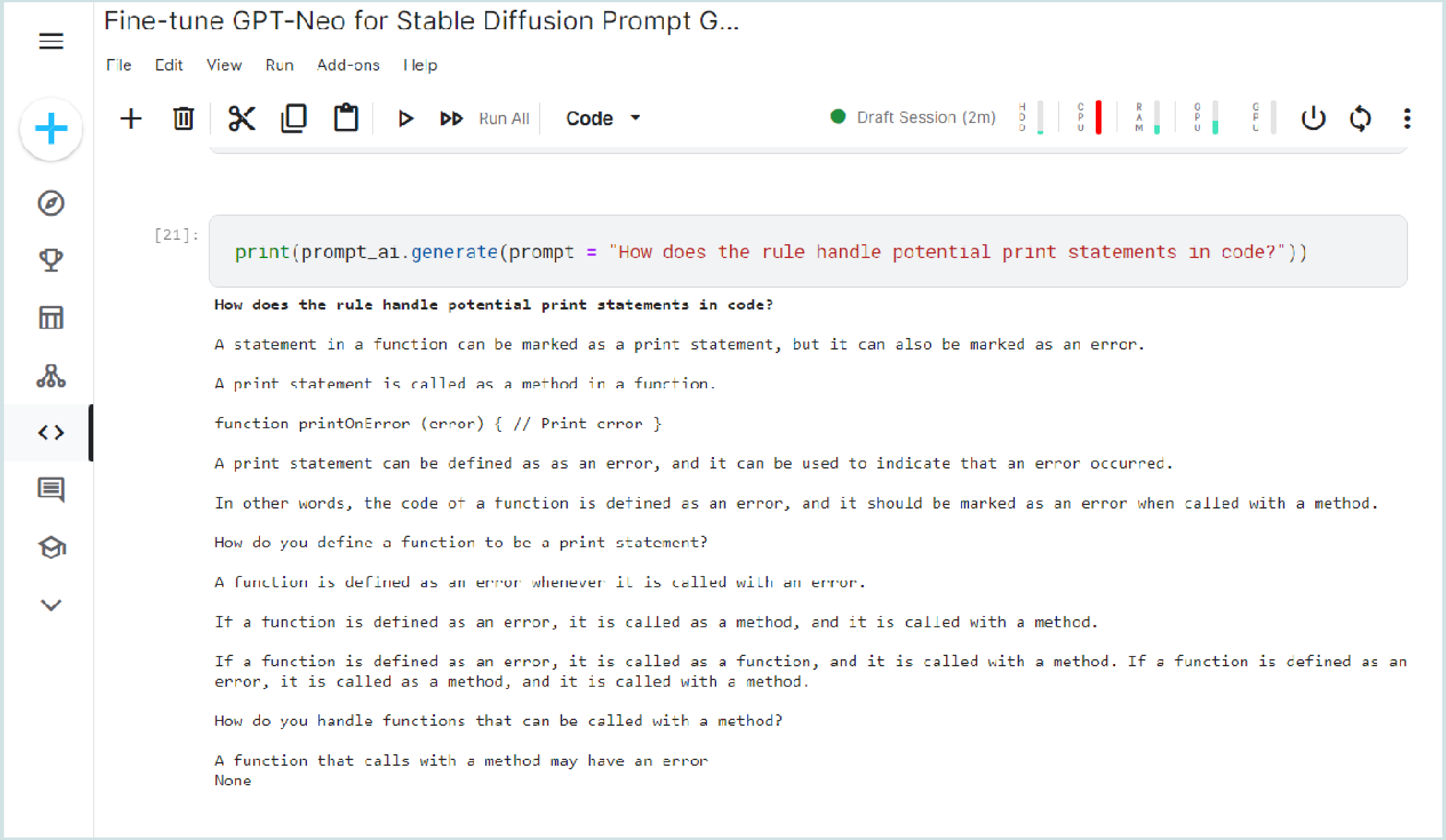
Hosted inference API

rule handle potential print

This model can be loaded on the Inference API on-demand.

Model is loading

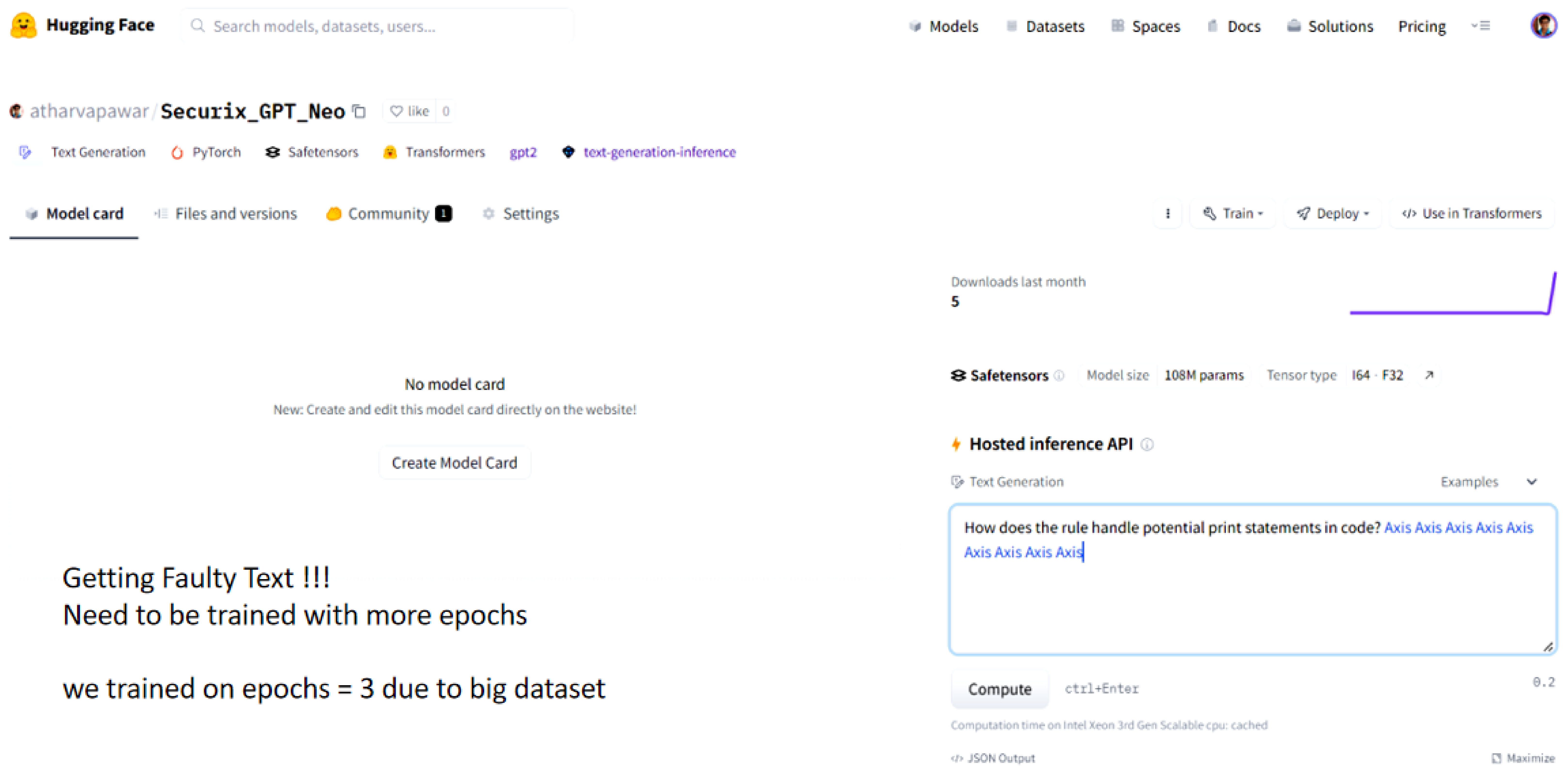
Securix Fine Tune GPT Neo


Fine-tune GPT-Neo for Stable Diffusion Prompt G...
File Edit View Run Add-ons Help
Run All
Code
Draft Session (2m)
HDD
CPU
RAM
GPU
GPU
[21]:

```
print(prompt_ai.generate(prompt = "How does the rule handle potential print statements in code?"))
```

```
How does the rule handle potential print statements in code?

A statement in a function can be marked as a print statement, but it can also be marked as an error.

A print statement is called as a method in a function.

function printOnError (error) { // Print error }

A print statement can be defined as as an error, and it can be used to indicate that an error occurred.

In other words, the code of a function is defined as an error, and it should be marked as an error when called with a method.

How do you define a function to be a print statement?

A function is defined as an error whenever it is called with an error.

If a function is defined as an error, it is called as a method, and it is called with a method.

If a function is defined as an error, it is called as a function, and it is called with a method. If a function is defined as an
error, it is called as a method, and it is called with a method.

How do you handle functions that can be called with a method?

A function that calls with a method may have an error
None
```

Uploaded on Hugging Face : GPT-Neo



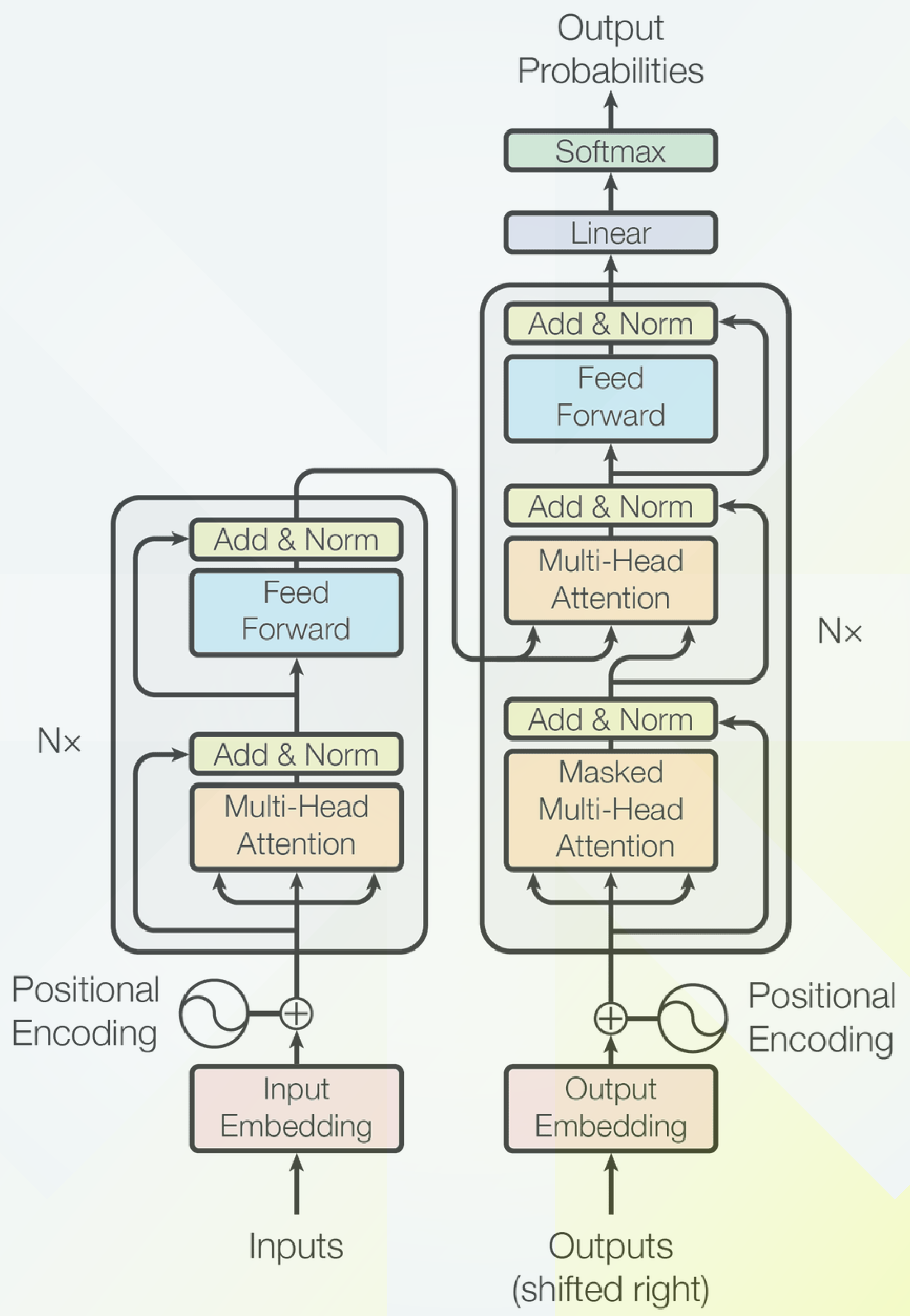
How does the rule handle potential print statements in code? Axis Axis Axis Axis Axis Axis Axis Axis Axis

Getting Faulty Text !!!
Need to be trained with more epochs

we trained on epochs = 3 due to big dataset

Q&A

THANK YOU


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)