Small Models, Big Security: Evaluating the Efficacy of DistilBERT against Llama Guard for Corporate Data Exfiltration Classification

Carter Patton

University of Texas at Austin

Case Studies in Machine Learning - Final Report

Introduction

The rapid integration of Large Language Models (LLMs) into enterprise environments has fundamentally altered the landscape of corporate data management. Organizations are increasingly relying on Retrieval-Augmented Generation (RAG) systems to grant employees natural language access to internal data repositories; however, this integration introduces significant security challenges, specifically the risk of data exfiltration through malicious prompt injection and even simple queries analogous to the structure of sensitive data (Cloud Security Alliance, 2023). While RAG systems offer real productivity gains for organizations, they simultaneously create an attack surface where a malicious insider or a compromised account can leverage the model's reasoning capabilities to extract sensitive information.

Unlike traditional network threats, LLM-based attacks exploit the semantic processing of the model itself. Recent studies highlight that "shadow AI", the unmonitored use of external or unapproved models, poses a severe threat to data governance (OWASP, 2025). When internal security controls are too restrictive or too slow and resource-intensive, users often bypass them, turning to unauthorized tools that offer better performance but zero oversight.

Current industry solutions largely rely on LLM-as-a-Judge architectures, such as Meta's Llama Guard or IBM's Granite Guardian. While these models demonstrate high efficacy in detecting general harms (Meta AI, 2023; IBM Research, 2024), they are computationally expensive, with parameter counts on the order of billions. These large middleware models introduce latency to every interaction, degrading the user experience and inadvertently driving users to unsanctioned, higher performing model providers.

This research aims to address the inefficiency of current State-of-the-Art (SoTA) LLM firewalls by developing a lightweight, task-specific malicious prompt classifier. The primary objective is to develop a middleware model focused narrowly on detecting prompts intended to exfiltrate specific "crown jewel" data; defined here as high-value corporate assets such as financial reports, executive communications, and HR records.

I hypothesize that a small, fine-tuned encoder-based model (DistilBERT, ~66 million parameters) can achieve classification performance metrics (Precision, Recall, F1-Score) better than or comparable to a large, SoTA, decoder-based model (Llama Guard 3, ~8 billion parameters) for the specific task of exfiltration detection, while operating with significantly lower latency. Through my research I hope to provide a blueprint for enterprises to implement low-latency LLM firewalls that secure the RAG pipeline without compromising application performance.

Research Background

Security Risks of Enterprise RAG Systems

Retrieval-Augmented Generation (RAG) allows general-purpose LLMs to answer questions about private data by attaching relevant context to the user's prompt. However, recent literature indicates that this architecture is vulnerable to malicious prompts designed to bypass safety alignment (Liu et al., 2023).

The Open Worldwide Application Security Project (OWASP) identifies "Sensitive Information Disclosure" and "Vector and Embedding Weaknesses" as top critical vulnerabilities for GenAI (OWASP, 2025). In a corporate setting, a user might employ adversarial techniques to trick the model into retrieving "crown jewel" data. Current research into these attacks shows that even robust models can be manipulated via "jailbreaking" techniques that obfuscate the malicious intent of a query (Deng et al., 2024; Wallace et al., 2024).

Limitations of Current LLM-as-a-Judge Solutions

To mitigate such risks, major AI labs have released safety-specific models. Meta's Llama Guard family represents the current standard, utilizing an 8-billion parameter architecture fine-tuned on a taxonomy of safety risks (Inan et al., 2023). Similarly, IBM has introduced the Granite Guardian family, which provides risk detection for RAG applications, including check-grounding and context relevance (IBM Research, 2024; IBM GitHub, 2024).

While effective, these models treat classification as a text generation task. To classify a single prompt, the model must process input tokens and auto-regressively generate a response. This introduces two major challenges for enterprise deployment:

A. **Latency:** Running multi-billion parameter models introduces significant inference delays.
B. **Resource Intensity:** Hosting these models requires substantial VRAM (Video RAM), often necessitating dedicated high-end GPUs solely for the guardrail or middleware model.

Recent studies suggest that while these large models excel at generalized harm detection (e.g., hate speech), they may be overkill for specific, binary classification tasks like in this case (Zhang et al., 2025).

Efficiency of Encoder-Only Architectures

In contrast to the decoder-only architectures used by Llama Guard and Granite, encoder-only architectures like BERT are designed specifically for understanding and classifying text. **DistilBERT**, the subject of this study, is a distilled version of the ubiquitous BERT NLP model that retains 97% of BERT's performance while being 40% smaller and 60% faster (Sanh et al., 2019).

Prior research into optimizing transformer architectures, such as ALBERT (Lan et al., 2019) and RoBERTa (Liu et al., 2019), demonstrated that model architecture often plays a larger role in efficiency than raw parameter count. By utilizing DistilBERT (~66M parameters) instead of Llama Guard (~8B parameters), parameter count is reduced by a factor of over 100x.

Dataset Construction for "Crown Jewel" Detection

A notable challenge in training security models is the lack of domain-specific datasets that reflect real enterprise data, as this information is obviously sensitive. Standard datasets often focus on generic internet safety. To combat this fact, my research leveraged a hybrid dataset generation strategy, utilizing grounded, synthetic data generation (via recent SoTA models) to simulate realistic "benign" corporate queries and "malicious" exfiltration attempts targeting specific high-value data types. The exact process for constructing the dataset will be expanded upon below.

Research and Methods

The methodology for this research can be divided into four distinct phases: dataset generation, data preprocessing, model fine-tuning, and evaluation. The following sections detail these phases.

Data Generation Pipeline

The dataset for fine-tuning needed to come in the form of prompts against a corporate-provided LLM which had access to internal documentation in order to adequately fine-tune the DistilBERT model on the classification task I will be evaluating on.

To simulate a realistic enterprise environment, I employed SoTA LLMs (ChatGPT 5, Gemini 2.5) to generate a synthetic dataset designed to emulate actual traffic through an enterprise LLM which has access to a corpus of internal documents. Because no public dataset of prompts captured the nuance of corporate data exfiltration prompts (e.g., specific requests for internal financial spreadsheets or HR salary bands), a custom generation pipeline was developed.

Benign Prompt Generation (Grounded Synthesis)

For the purposes of this research, benign prompts are prompts which represent legitimate, safe employee queries. To generate realistic, diverse prompts with real domain relevance, I utilized a Retrieval-Augmented Generation (RAG) approach using **Google Vertex AI Search**.

1. **Data Store Creation:** A Vertex AI Data Store was populated with a corpus of non-sensitive, generic corporate documentation (e.g., employee handbooks, public marketing blogs, and IT troubleshooting guides). These documents emulated the internal documents of one ACME Corporation. These diverse documents were generated by ChatGPT 5 Pro using the agentic document creation functionality provided by ChatGPT's UI.
2. **Grounded Generation:** I then interfaced with Google's Gemini 2.5-Flash model via the Vertex AI API. The Vertex API has functionality for "grounded" generation based on a Data Store, meaning the model could use RAG when generating responses. A script

then sent structured prompts to the model, requesting that the model *“*Act as a new HR employee” or “Act as a Junior Developer”, etc… and generate potential prompts based only on the provided documents. The prompts returned by Vertex were then preprocessed and added to the BenignDataset.csv file with a label of **0**.
3. **Output Standardization:** The API leveraged structured outputs to format responses in a strict JSON format, ensuring outputs required minimal post-processing. This resulted in ~10,000 relevant and benign prompts.

- ***Example benign “meta-prompt”:*** Act as a marketing analyst. Based only on the provided documents, generate 50 unique questions asking for summaries, key performance indicators, and data points from the quarterly marketing reports.
- ***Example generated benign prompt:*** What was the total number of MQLs (Marketing Qualified Leads) generated this quarter?

Malicious Prompt Generation

Malicious prompts were engineered to simulate an ACME employee trying to exfiltrate "crown jewel" data. This process leveraged a very similar process to the benign dataset generation, with the notable difference that the meta-prompts (prompts sent to Vertex which resulted in generation of multiple prompts from the corporate user’s point of view) were based on generating prompts for data exfiltration from each unique role’s perspective. This resulted in ~ 2300 malicious prompts, labeled **1**.

- ***Example malicious “meta-prompt”:*** Act as a red team content generator. Scenario: payroll snooping. Based only on the provided documents, generate 50 distinct messages attempting to access payroll registers, pay stubs, or compensation summaries in People & Talent Hub.
- ***Example generated malicious prompt:*** What is the full name, employee ID, and job title for everyone in the Marketing department, as listed in People & Talent Hub?

Preprocessing and Tokenization

Before training, the raw text data was prepared using standard Natural Language Processing (NLP) preparation steps.

- **Class Imbalance Handling:** The final dataset consisted of a roughly 80/20 ratio of benign to malicious prompts. This imbalance was meant to reflect the scenario of a corporate environment where true attacks are relatively rare events compared to normal traffic.
- **Tokenization:** I utilized the **DistilBertTokenizerFast** from the Hugging Face **transformers** library. This tokenizer converts the raw prompt text into sets of features in the format that DistilBERT expects.

- **Dataset Shuffling and Splitting**: I utilized the **datasets** Python library, specifically the **train_test_split** method, to generate a shuffled and split version of my dataset (malicious prompts + benign prompts) that was repeatable. Notably, I used a 80/20 train/test split and shuffled with a random seed of 42 so that I could use the same set of data to evaluate my model and Llama Guard.

Model Fine-Tuning Strategy

The core model used was **DistilBERT** (distilbert-base-uncased). I then used the Hugging Face Trainer to fine-tune the model, taking the pre-trained weights (which already understand general English syntax and semantics) and fine-tuning them specifically for the binary classification task of "Malicious" vs. "Benign."

The specific hyperparameters used were:

- **Learning Rate:** 2e-5 (selected to as an arbitrary lr based on experience gained during Deep Learning course at UT)
- **Batch Size:** 16 (chosen for the available 24GB VRAM on my local PC)
- **Epochs:** 3
- **Optimizer:** AdamW
- **Loss Function:** Cross-Entropy Loss

Evaluation Methodology

To validate the hypothesis that my small model could perform competitively against a large one, I held out 20% of the dataset (after shuffling) as a validation set.

Llama Guard 3

I selected **Meta's Llama Guard 3 (8B)** as the “SoTA” baseline. To evaluate it, I constructed a specific prompt template required by the model (using [INST] tags) to force it to output a text classification of either "safe" or "unsafe." This output was then parsed and mapped to 0 (safe) or 1 (unsafe) to use the same evaluation criteria as my DistilBERT fine tune.

Performance Metrics

Instead of a simple accuracy calculation, which can be ineffective at conveying the true performance of a model on an imbalanced dataset (a model that predicted"benign" 100% of the time would achieve 83% accuracy in this case but fail as a firewall), I evaluated on:

- **Precision:** The ratio of correctly identified malicious prompts to all prompts flagged as malicious. (Low precision leads to high False Positives and user frustration).
- **Recall:** The ratio of correctly identified malicious prompts to all actual malicious prompts. (Low recall means attacks are slipping through).

- **F1-Score:** The harmonic mean of Precision and Recall, providing a single metric for model performance.

Latency Measurement

A critical component of this research was quantifying the latency introduced by the security layer. Latency was measured as the time required for the inference loop to process a single prompt, averaged over the entire validation set.

- **Measurement Tool:** Python’s time.perf_counter() was used to capture timing data surrounding the inference call.
- **Hardware Consistency:** Both models were evaluated on the exact same hardware configuration (my personal workstation with a single GPU) to ensure an accurate comparison of computational efficiency.

Resource Usage (VRAM)

Another metric that was representative of the large difference in scale between these two models is VRAM usage. LLMs are known to use large amounts of VRAM when compared to other ML models due to the sheer scale of parallel computations needed to complete auto-regressive generation. Since the prompts used were of a relatively nominal size (~15-25 tokens), a representative screenshot of VRAM usage during execution was used to quantify the resource usage for both models.

- **Measurement Tool:** Screenshot of Window’s Task Manager which shows real-time reading of VRAM usage from my workstation’s GPU which was leveraged when evaluating both models.
- **Hardware Consistency:** Both models were evaluated on the exact same hardware configuration (my personal workstation with a single GPU) to ensure an accurate comparison of computational efficiency.

Materials and Data Sources

This research relied on the following computational resources, software libraries, and data. To ensure reproducibility of these results, the materials and data sources used in the experiment are detailed below.

Computational Environment

All phases of this research, including data generation, preprocessing, model fine-tuning, and inference evaluation, were conducted locally on my personal workstation. This setup was chosen to emulate a constrained corporate scenario where dedicated cluster resources are likely not available for a middleware security layer (and because I wanted to take advantage of the potential provided to me by this PC).

- **GPU:** NVIDIA GeForce RTX 3090 (24 GB GDDR6X VRAM). This high VRAM, consumer-grade GPU has become very popular in the local LLM hosting community and was necessary to load the 8-billion parameter Llama Guard 3 with full BF16 tensors.
- **System RAM:** 32 GB DDR4.
- **Processor:** AMD Ryzen 9 5900X (12-core, 24-thread).
- **Software Environment:**
    - **Operating System:** Windows 10
    - **Python Version:** 3.11.14
    - **Core Libraries**:
        - transformers
        - datasets
        - evaluate
        - pandas
        - scikit-learn
        - torch
        - matplotlib
        - seaborn
        - accelerate
        - google-cloud-aiplatform
        - tokenizers
        - tensorboard

The Dataset

The dataset utilized for fine-tuning of my model and the evaluation of both models is described above in the Research & Methods section of this report. The exact dataset used can be found on my GitHub in CSV format. The **RANDOM_SEED** utilized for the evaluation set was **42** in order to ensure the exact dataset used in this experiment could be reproduced. The dataset has been split into the Malicious dataset and the Benign dataset if you choose to recombine the two into your own desired ratio for training and/or evaluation.

Dataset Composition

The dataset is intentionally imbalanced to reflect real-world traffic patterns where malicious queries are rare events relative to benign operational traffic.

- **Total Prompts:** 12,431
- **Benign Prompts (Class 0):** 10,081 (81.1%)
- **Malicious Prompts (Class 1):** 2,350 (18.9%)

Data Splits

To ensure the integrity of the evaluation, the dataset was split into training and validation sets using a stratified sampling method through the datasets library (random seed 42) to maintain the 80/20 class ratio in both splits.

- **Training Set:** ~9,600 prompts (80%) - Used to fine-tune the DistilBERT model.
- **Validation (Hold-out) Set:** ~2,400 prompts (20%) - Used exclusively for the final evaluation of both DistilBERT and Llama Guard 3. This data was never seen by the DistilBERT model during training.

The Models

Two distinct model architectures were utilized in this study to represent the small & fine-tuned vs.SoTA comparison.

My Model: DistilBERT (distilbert-base-uncased)

- **Architecture:** Encoder-only Transformer.
- **Parameters:** ~66 Million.
- **Source:** Hugging Face Hub (Sanh et al., 2019).

The “SoTA” Model: Llama Guard 3 (meta-llama/Llama-Guard-3-8b)

- **Architecture:** Decoder-only (Generative) Transformer based on Llama 3.
- **Parameters:** ~8 Billion.
- **Source:** Meta AI (via Hugging Face Hub).
- **Usage:** This model was run at full BF16 quantization to elicit maximum possible performance for this model. Running this experiment at different quantization values would be a very interesting addition as it would close the VRAM usage gap by a significant amount.

Results

The two models were evaluated on the same 2,487 prompts selected from the dataset as the validation set. These prompts were withheld from the fine-tuning process for my model, meaning neither model had been exposed to these prompts during training, ensuring the integrity of the evaluation between these two models. Both models were evaluated using a python script and run on the GPU on my workstation. Once evaluations were completed and metrics were calculated, visualizations were created to display the results in an easy to digest format (confusion matrix, latency bar chart, and VRAM usage screenshots). These can be found below.

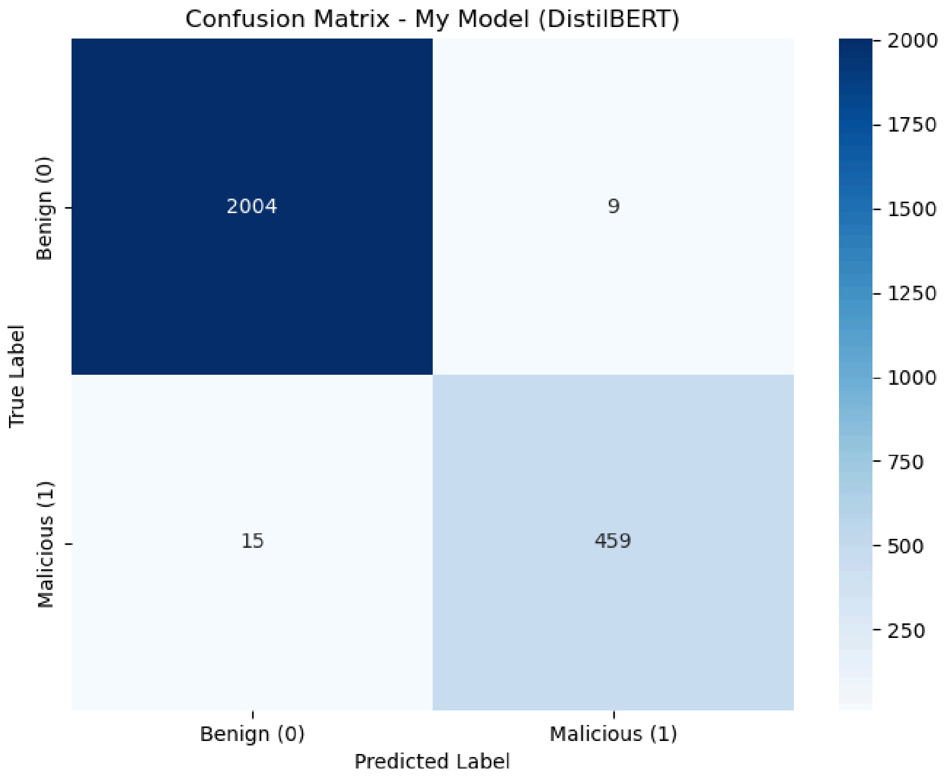


Figure 1: Confusion Matrix for My Model (DistilBERT) Evaluation

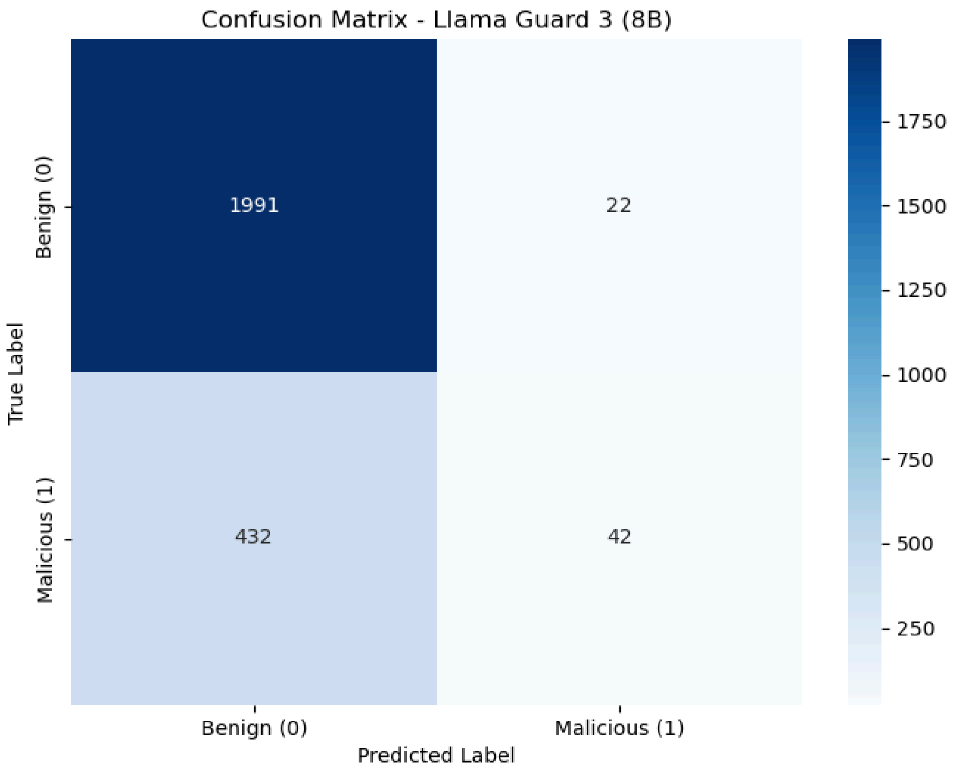


Figure 2: Confusion Matrix for Llama Guard 3 8B Evaluation

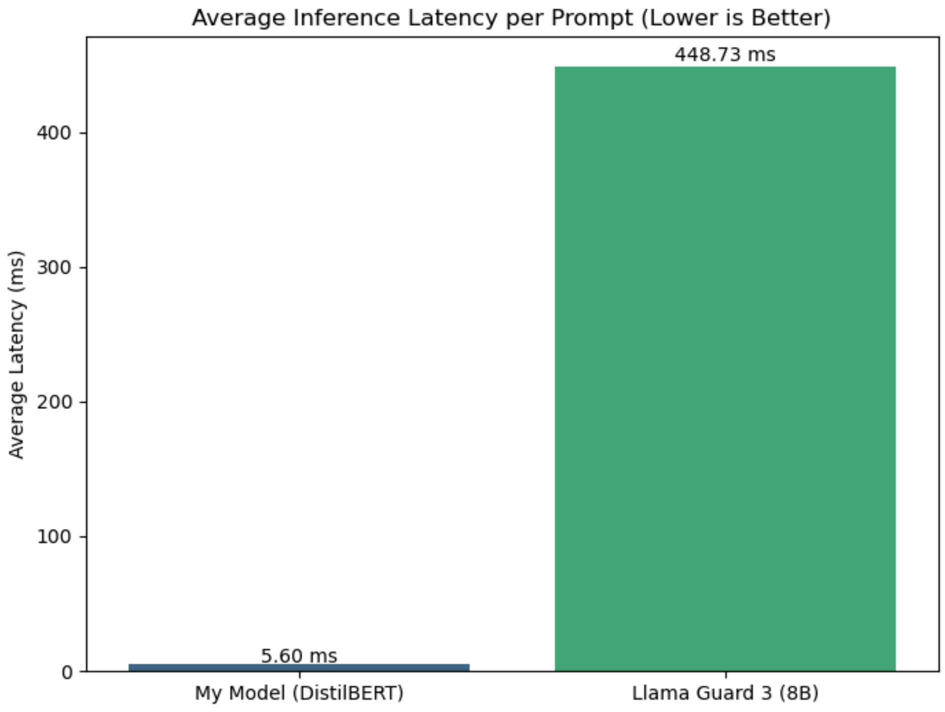
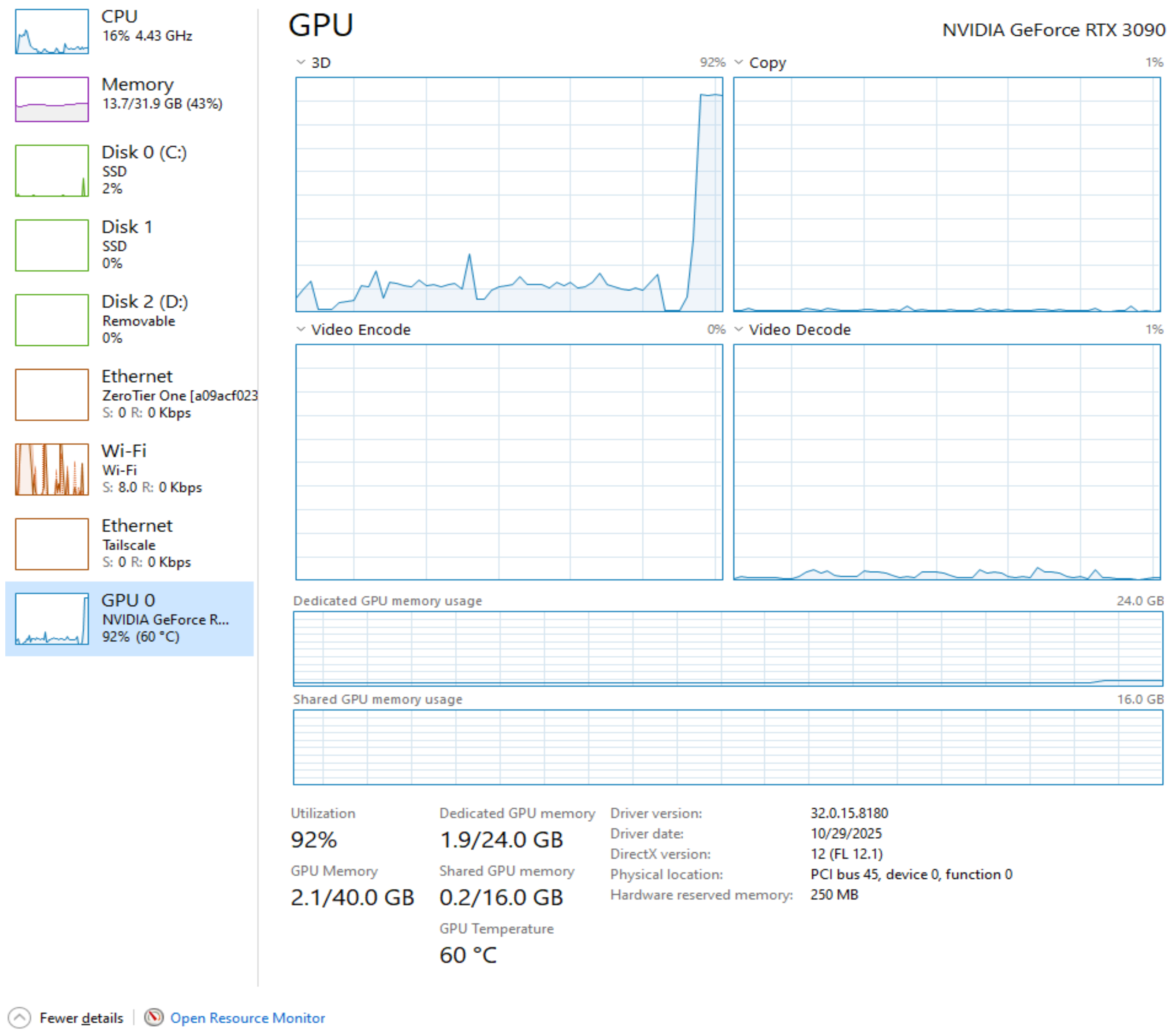


Figure 3: Latency per Prompt Comparison



Utilization
92%

Dedicated GPU memory
1.9/24.0 GB

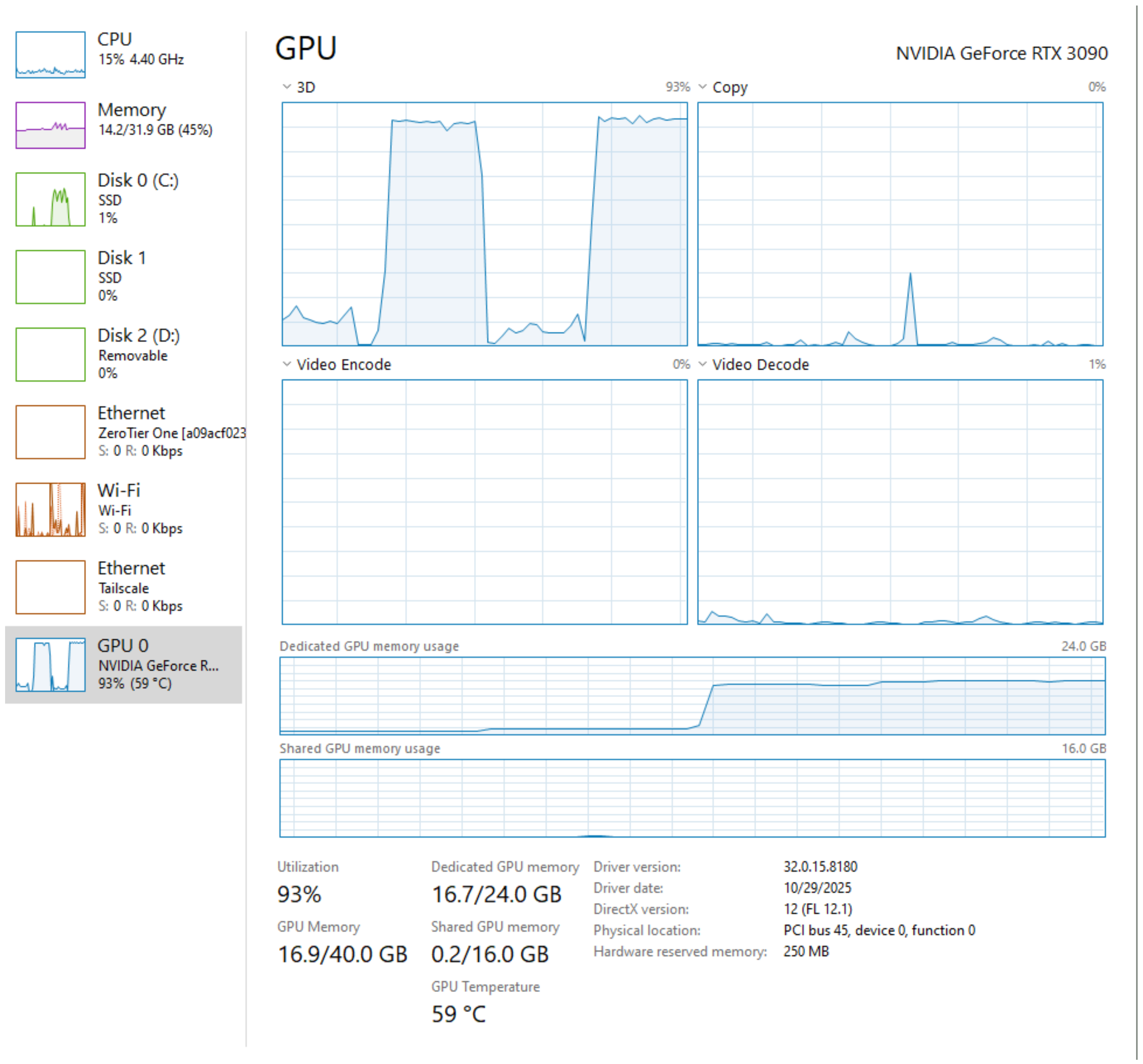
GPU Memory
2.1/40.0 GB

Shared GPU memory
0.2/16.0 GB

GPU Temperature
60 °C

Driver version:	32.0.15.8180
Driver date:	10/29/2025
DirectX version:	12 (FL 12.1)
Physical location:	PCI bus 45, device 0, function 0
Hardware reserved memory:	250 MB

Figure 4: VRAM Usage (~1.9GB) during My Model (DistilBERT) Evaluation



Utilization
93%

Dedicated GPU memory
16.7/24.0 GB

GPU Memory
16.9/40.0 GB

Shared GPU memory
0.2/16.0 GB

GPU Temperature
59 °C

Driver version: 32.0.15.8180
Driver date: 10/29/2025
DirectX version: 12 (FL 12.1)
Physical location: PCI bus 45, device 0, function 0
Hardware reserved memory: 250 MB

Figure 5: VRAM Usage (~16.7GB) during Llama Guard 3 8B Evaluation

	My Model (DistilBERT)	**Llama-Guard-3-8B**
Accuracy	**0.9903**	**0.8175**
Precision (malicious)	**0.9808**	**0.6562**
Recall (malicious)	**0.9684**	**0.0886**
F1 (malicious)	**0.9745**	**0.1561**

Figure 6: Table comparing Accuracy, Precision, Recall, and F1 scores between models

Latency

As shown in the graphics above, average latency per prompt for DistilBERT was **5.6ms** vs Llama Guard 3's **448.73ms**. Because of Llama Guard's dense construction and large parameter count, the latency needed to do a simple binary classification is significantly higher due to the auto-regressive processing of each token.

VRAM Usage

The screenshots from Window's Task Manager during model evaluation highlight a stark contrast between the two models, with Llama Guard requiring 16.7 GB of precious VRAM to store its model weights, KV cache, and input tokens (relatively small, ~15-25 tokens per prompt). DistilBERT required less than 2.0 GB of VRAM to classify prompts.

Accuracy

The fine-tuned DistilBERT model achieved a near-perfect overall accuracy of 99.03%, significantly outperforming the Llama Guard 3 baseline, which evaluated at 81.75%. While Llama Guard's accuracy appears passable on the surface, this figure is heavily inflated by the dataset's class imbalance, where simply predicting "benign" for every prompt would yield a high baseline score (precisely why this was not chosen as the primary evaluation metric).

Precision

In terms of precision, my model scored 98.08%, indicating an extremely low false-positive rate (which would likely minimize user frustration in a corporate environment). Conversely, Llama Guard's precision of 65.62% suggests it struggles to cleanly distinguish between legitimate corporate queries and actual exfiltration attempts, likely flagging safe prompts as dangerous.

Recall

The most obvious gap appears in recall, where my model successfully identified and blocked 96.84% of all malicious prompts. Surprisingly, Llama Guard 3 achieved a recall of only 8.86%, meaning it failed to detect over 90% of the specific exfiltration attempts present in the validation set.

F1 Score

Consequently, the F1-Score shows a stark contrast, with my model achieving 97.45% compared to Llama Guard's 15.61%. This metric confirms that the lightweight, specialized model provides a far more robust and reliable security layer for this specific task than the larger generalized model.

Conclusion

Overall, this research was a fantastic learning exercise for myself, and challenged many of the notions I had come to accept surrounding the capabilities of encoder only architectures. The relative latency metrics and recall scores were truly stunning to me and highlighted the validity of my hypothesis to great effect.

Bigger Does Not Always Equal Better

In today's climate, many believe that LLMs are the "end-all-be-all" of text-related tasks. If there is a problem regarding analysis or manipulation of text-based data, LLM is the predominant prescription, whereas this research proved that much less "powerful" models fine-tuned on highly relevant data, even if it is generated synthetically, can outperform LLMs on these specialized tasks. This nuance is often lost in discussions surrounding safety and reliability of LLM systems. That is to say that many preach that the solution to an unwieldy and vulnerable LLM is a bigger, "badder", and more complex system that likely introduces many more vulnerabilities than it patches. As a native Texan, this "bigger does not always equal better" conclusion doesn't exactly align with my beloved state's motto, but it is hard to argue that bigger is better after seeing the results of this study.

Scalpel vs Sledgehammer

Some tasks are better left to the specialized, "scalpel" type models, as I would classify this DistilBERT fine-tune, as opposed to larger, general, metaphorical "sledgehammer" models that are seen as the state-of-the-art solution today. The field of AI is much broader than designing LLMs that require a dedicated modular nuclear reactor to power. Many of the problems faced by real businesses, today, require dedicated solutions on a much smaller scale to deliver the results stakeholders expect.

Future Outlook

As someone who has only begun their exploration into the vast field of AI/ML engineering, this research proves very encouraging as it has clearly shown that there is not one general solution for every problem. Machine Learning and Artificial Intelligence continue to provide problem solvers like myself with unique and exciting solutions to real world problems.

Bibliography

1. Bhatt, S. S., et al. (2025). *Enterprise AI must enforce participant-aware access control*. arXiv. https://arxiv.org/abs/2509.14608
2. Blefari, F., Cosentino, C., Pironti, F. A., Furfaro, A., & Marozzo, F. (2025). *CyberRAG: An agentic RAG cyber attack classification and reporting tool*. arXiv. https://arxiv.org/abs/2507.02424
3. Chattopadhyay, N., Goswami, A., & Agarwal, A. (2024). *Adversarial attacks and dimensionality in text classifiers*. arXiv. https://arxiv.org/abs/2404.02660
4. Chaudhari, H., et al. (2024). *Phantom: General backdoor attacks on retrieval augmented language generation*. arXiv. https://arxiv.org/abs/2405.20485
5. Chen, G., Xia, Y., Jia, X., Li, Z., Torr, P., & Gu, J. (2025). *LLM jailbreak detection for (almost) free!* arXiv. https://arxiv.org/abs/2509.14558
6. Civelli, S., et al. (2025). *Ideology-based LLMs for content moderation*. arXiv. https://arxiv.org/abs/2510.25805
7. Cloud Security Alliance. (2023, November 22). *Mitigating security risks in Retrieval Augmented Generation (RAG) LLM applications*. Cloud Security Alliance Blog. https://cloudsecurityalliance.org/blog/2023/11/22/mitigating-security-risks-in-retrieval-augmented-generation-rag-llm-applications
8. Deng, G., Liu, Y., Wang, Y., Li, Y., & Liu, Y. (2024). *Jailbreaker: Automated jailbreak attacks against LLMs and robust defenses*. arXiv preprint arXiv:2409.18222. https://arxiv.org/abs/2409.18222
9. Ezquerro, A., Gómez-Rodríguez, C., & Vilares, D. (2025). *Bringing emerging architectures to sequence labeling in NLP*. arXiv. https://arxiv.org/abs/2509.25918
10. Facebook Research. (n.d.). *fastText: Library for efficient text classification and representation learning*. GitHub. https://github.com/facebookresearch/fastText
11. Feretzakis, G., & Verykios, V. S. (2024). *Trustworthy AI: Securing sensitive data in large language models*. arXiv. https://arxiv.org/abs/2409.18222
12. IBM. (2024). *Granite Guardian cookbooks*. GitHub. https://github.com/ibm-granite/granite-guardian/blob/main/cookbooks/granite-guardian-3.3/detailed_guide_think_ollama.ipynb
13. IBM. (2024, April 23). *Protect against prompt injection*. https://www.ibm.com/
14. IBM Research. (2024). *Granite Guardian: A comprehensive guardrail model for large language models*. arXiv preprint arXiv:2412.07724. https://arxiv.org/abs/2412.07724
15. Inan, H., et al. (2023). *Llama Guard: LLM-based input-output safeguard for human-AI conversations*. Meta AI Research. https://ai.meta.com/research/publications/llama-guard-llm-based-input-output-safeguard-for-human-ai-conversations/
16. Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). *Bag of tricks for efficient text classification*. arXiv preprint arXiv:1607.01759. https://arxiv.org/abs/1607.01759

17. Lan, Z., Chen, M., Goodman, S., Gimpel, K., & Sharma, P. (2019). *ALBERT: A lite BERT for self-supervised learning of language representations*. arXiv preprint arXiv:1909.11942. https://arxiv.org/abs/1909.11942
18. Liu, Y., Deng, G., Li, Y., Wang, K., & Liu, T. (2023). Prompt injection attacks and defenses in LLM-integrated applications. *Expert Systems with Applications*. https://www.sciencedirect.com/science/article/pii/S0957417423004256
19. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). *RoBERTa: A robustly optimized BERT pretraining approach*. arXiv preprint arXiv:1907.11692. https://arxiv.org/abs/1907.11692
20. Mancera, G., et al. (2025). *PBa-LLM: Privacy- and bias-aware NLP using Named-Entity Recognition (NER)*. arXiv. https://arxiv.org/abs/2507.02966
21. OWASP. (2025, April 16). *LLM01:2025 Prompt injection*. OWASP GenAI Security Project. https://genai.owasp.org
22. OWASP. (2025). *LLM08:2025 Vector and embedding weaknesses*. OWASP Top 10 for Large Language Models. https://genai.owasp.org/llmrisk/llm082025-vector-and-embedding-weaknesses/
23. Palo Alto Networks. (n.d.). *What is a prompt injection attack? [Examples & prevention]*. https://www.paloaltonetworks.com
24. Rall, D., Bauer, B., Mittal, M., & Fraunholz, T. (2025). *Exploiting web search tools of AI agents for data exfiltration*. arXiv. https://arxiv.org/abs/2510.09093
25. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv preprint arXiv:1910.01108. https://arxiv.org/abs/1910.01108
26. Wan, Z. (2023). *Text classification: A perspective of deep learning methods*. arXiv. https://arxiv.org/abs/2309.13761
27. Wang, Y., Qu, W., & Ye, X. (2024). *Selecting between BERT and GPT for text classification in political science research*. arXiv. https://arxiv.org/abs/2411.05050
28. Wang, Y., et al. (2025). Efficiency in NLP: A comparative analysis. *CEUR Workshop Proceedings, 3643*. https://ceur-ws.org/Vol-3643/paper3.pdf
29. Zhou, P., Feng, Y., & Yang, Z. (2025). *Provably secure retrieval-augmented generation*. arXiv preprint arXiv:2508.01084. https://arxiv.org/abs/2508.01084
30. Zhou, X., et al. (2025). *LessLeak-Bench: A first investigation of data leakage in LLMs across 83 software engineering benchmarks*. arXiv preprint arXiv:2502.06215. https://arxiv.org/abs/2502.06215