PARSONS PROMPT CLASSIFICATION: AUTOMATING MALICIOUS PROMPT DETECTION

CAPSTONE PROJECT REPORT

Aidan Bradshaw
abradsha@andrew.cmu.edu

Ashley Wu
aywu@andrew.cmu.edu

David Li
dcli@andrew.cmu.edu

April 14, 2025

ABSTRACT

Large Language Models (LLMs) like ParsonsGPT are vulnerable to adversarial prompts and lack sufficient safeguards to prevent outputs that contradict internal company policies. This poses serious legal, ethical, and reputational risks if the model generates guidance that violates Parsons' Code of Conduct. We present a domain-specific prompt classification pipeline that combines policy-based synthetic generation with reinforcement learning from human feedback (RLHF) to create a high-quality, compliance-aligned dataset. Using this data, we fine-tune a lightweight deBERTa-based classifier that accurately flags and routes potentially harmful prompts. The resulting model improves policy alignment, precision, and recall by over 40% compared to baseline. This system offers a scalable safeguard that ensures ParsonsGPT outputs remain consistent with organizational standards and departmental responsibilities.

1 Introduction

Recent advancements in Large Language Models (LLMs), particularly those based on Transformer architectures such as GPT, have enabled increasingly sophisticated text generation and comprehension abilities. These models are widely adopted in various sectors, including finance, healthcare, governance, and defense, due to their ability to process complex natural language tasks, streamline workflows, and enhance decision-making [George and George, 2023]. Despite their proven utility, the open-ended nature of LLMs raises significant concerns regarding ethical usage, data privacy, and the potential for adversarial manipulation through malicious prompts [Han et al., 2023]. When individuals intentionally craft prompts with deceptive or harmful content, LLMs can generate inappropriate or potentially damaging responses, raising reputational, legal, and security risks [Han et al., 2023]. Within corporate and governmental organizations, manual or semi-automated risk mitigation strategies are labor-intensive, prone to human error, and unable to keep pace with the volume of queries that modern AI systems must handle. As LLMs transition from pilot programs to production-level deployment, organizations urgently require robust, automated mechanisms to detect and filter malicious requests in real time.

Parsons Corporation, a leading global provider of defense, intelligence, and critical infrastructure solutions, exemplifies these challenges in its internal use of ParsonsGPT, an LLM-based platform that offers employee support and decision assistance. Although ParsonsGPT has significantly reduced response times and facilitated streamlined communication, it currently lacks an automated framework to identify prompts that violate corporate policies or legal guidelines. The existing reliance on manual screening increases operational costs and inefficiencies in responding to user requests. Recognizing these limitations, Parsons seeks an automated "red flag" system designed to flag prompts that require additional scrutiny or specialized handling (e.g., HR, legal, compliance). By automating the detection process, the company aims to enhance the overall efficiency and accuracy of its LLM deployments while maintaining high ethical and legal standards.

Although prior research has explored various techniques for toxicity detection, hate speech identification, and spam filtering in text [Shayegani et al., 2023] few solutions focus on organizational compliance and domain-specific guidelines (e.g., HR policies, legal constraints, security directives). Conventional toxic-language classifiers do not always capture nuanced, policy-specific violations, nor do they reliably route flagged content to the most appropriate internal department.

Furthermore, academic studies on prompt-based manipulations have primarily focused on model vulnerabilities and adversarial attacks rather than building comprehensive, enterprise-ready systems that align with corporate policy documents [Liu et al., 2025]. These studies utilize synthetically generated datasets that lack specific labels and nuanced language necessary for a model trained to guard company policy. Hence, a notable gap exists in the literature for an NLP-based approach that can (1) incorporate specialized policy documents for domain-specific prompt classification, (2) generate synthetic data to account for rare but critical violation cases, and (3) automatically categorize prompts in alignment with internal compliance requirements. Addressing this gap is crucial for organizations like Parsons that operate under strict legal and ethical guidelines.

This paper describes the approach used to create a prompt classification system based on the domain-specific requirements of ParsonsGPT. First, it reviews current research on LLM safety and the necessity of synthetic data generation for domain-specific flags. Then, it identifies the methods and models used for dataset generation and validation, also providing specific model parameters and examples of prompts. Finally, it describes the method to fine-tune a deBERTa based classifier on the validated dataset, discussing performance metrics and further improvements that could be made to the datset and model.

2 Literature Review

Recent work on Large Language Model (LLM) safety has increasingly focused on *adversarial prompts*, exploring how malicious or deceptive instructions may compromise model outputs. This section provides an overview of current research that informs our approach to malicious prompt detection and departmental routing at Parsons.

2.1 Adversarial Prompting and LLM Safety

The rapid growth of Large Language Models (LLMs) has led researchers to investigate how *adversarial or deceptive prompts* can compromise model outputs. Existing work often concentrates on open-ended toxicity detection [Demchak et al., 2024] or generic adversarial examples [Lin et al., 2024], aiming to patch vulnerabilities without deeply integrating enterprise-specific policies. For instance, Kumar et al. [2025] propose an "erase-and-check" mechanism to robustly defend against bounded-size prompt manipulations, and Zheng et al. [2024] demonstrate how safety prompts (treated as continuous embeddings) can steer LLM outputs away from harmful content. Moreover, Jiang et al. [2023] highlights the risk of *Compositional Instruction Attacks*, wherein innocuous-seeming user queries conceal malicious sub-prompts.

Zaghir et al. [2024] emphasizes the importance of *domain-aware* detection methods for tasks such as medical text mining or financial compliance. General-purpose toxic language classifiers can fail to capture specialized vocabulary, policy statements, and latent risk factors unique to particular industries. Studies integrating legal or medical corpora demonstrate improved performance in distinguishing malicious or non-compliant text from routine user queries. However, few works have systematically *fine-tuned LLMs* on internal policy documents or devised classification pipelines that route suspicious prompts to the correct department (e.g., HR vs. Legal). While these approaches inform broad strategies to safeguard LLMs, they rarely address the nuances of internal corporate rules—such as HR procedures, legal constraints, or security protocols—that an enterprise might need. In our work, we build on these foundations by *embedding domain-specific policy guidelines* into the prompt-monitoring process, ensuring that adversarial queries targeting organizational vulnerabilities are more accurately detected.

2.2 Synthetic Data Generation and Validation

Real-world malicious queries are frequently rare or inconsistently labeled, making it challenging for supervised learning systems to learn robust decision boundaries. A recurring challenge in malicious prompt detection is the *lack of sufficiently large labeled datasets*, especially for rare but high-impact violations like insider trading or workplace abuse. Previous research has explored generating synthetic toxic prompts [Lin et al., 2024] or adversarial examples [Jiang et al., 2023], but these synthetic corpora often lack a tight alignment with *real internal policies and departmental structures*. [Chalkidis et al., 2020] suggests that *generation followed by validation* through methods such as **Legal-BERT**-based similarity checks or **Reinforcement Learning Human Feedback (RLHF)** to *validate* ensures a more faithful representation of likely adversarial scenarios within ParsonsGPT's environment. As a result, we obtain a robust training dataset, bridging the gap between model-based prompt augmentation and the policy-centric threats that enterprise systems must handle.

3 Policy Based Prompt Generation

Research on toxic language detection demonstrates the efficacy of large-scale language models and deep learning approaches for identifying harmful text [Liu et al., 2025]. However, these general-purpose methods often fail to capture organization-specific compliance violations or domain-oriented prompts, such as HR issues, insider trading, or security breaches. To address this, we focus on using Parsons Code of Conduct documents to provide a tailored knowledge base for malicious prompt detection, including rare yet critical violation cases such as insider trading or espionage, which general-purpose toxicity classifiers might overlook.

Our methodology seeks to automate prompt classification by leveraging domain-specific policy documents, Named Entity Recognition (NER), synthetic prompt generation, and prompt validation. By systematically integrating these components, we aim to produce a labeled dataset appropriate for fine-tuning a transformer-based model capable of flagging and routing malicious or non-compliant prompts. The pipeline is summarized below:

1. **Policy Document Preprocessing:** Cleans and extracts relevant text from Parsons Corporation's internal documents.
2. **Word Bank Construction:** Identifies key compliance terms and assigns departmental labels.
3. **Synthetic Prompt Generation:** Creates both malicious and non-malicious prompts reflecting real-world adversarial scenarios.
4. **Prompt Validation:** Scores the quality of synthetic prompts and aligns them with genuine flagged examples.

Figure 1 presents a simplified visualization of this pipeline. A more detailed visualization and explanation of this pipeline can be found in the appendix.

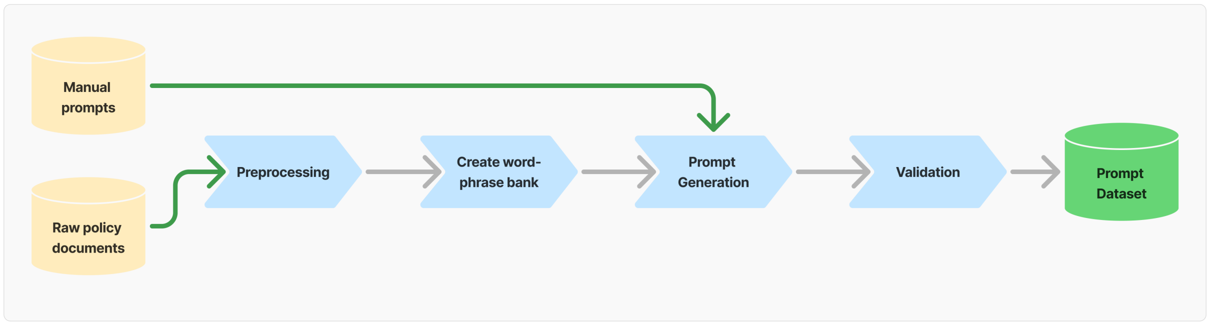


Figure 1: General visualization of data generation process. The visualization includes a high-level overview of the steps outlined in the report. A more detailed visualization and explanation of the synthetic data generation pipeline can be found in the appendix.

3.1 Data Preparation and Document Preprocessing

Policy documents often contain metadata, revision histories, and organizational headers (e.g., "Page X of Y," "Approved by," "Effective Date: ...") that do not provide informative linguistic features for prompt classification tasks. To prepare the corpus for entity extraction, we preprocess raw policy PDFs using PyMuPDF for text extraction and regular expressions for structured cleaning. This process removes non-informative segments and standardizes the remaining content into a continuous subtitle-like format appropriate for language model input.

3.2 Word Bank Construction and Labeling

To identify domain-specific compliance terminology within the cleaned corpus, we apply **Legal-BERT**—a transformer model pretrained on legal text—using Hugging Face's `pipeline` for Named Entity Recognition (NER) [Gururangan et al., 2020]. Extracted spans (e.g., harassment," insider trading," "phishing attacks") form candidate compliance-relevant terms without requiring manual supervision.

Rather than manually labeling extracted entities, we adopt a *zero-shot labeling* approach based on semantic similarity. Specifically, we use **Sentence-BERT (SBERT)** [Reimers and Gurevych, 2019] to embed each extracted entity into a high-dimensional vector space. We also embed a curated set of department labels (e.g., HR," Legal," "Security") into the same space. For each extracted term, SBERT assigns a department by finding the label with the highest cosine similarity in the embedding space. This method leverages the pretrained semantic knowledge of SBERT to perform department attribution without needing hand-labeled examples.

Thus, each compliance entity is automatically mapped to a **word-phrase-department triplet**, capturing (1) the extracted token, (2) its original phrase context, and (3) its inferred department. These structured triplets form the *Word/Phrase Label Bank*, which is used downstream for synthetic prompt generation. This approach provides a scalable, model-driven labeling strategy that aligns extracted terminology with organizational risk categories, while minimizing human labeling overhead.

Index	Department	Word	Phrase	Flag (0/1)
1	Legal	insider-trading	illegal insider trading knowledge	1
2	HR	harassment	unlawful workplace harassment report	1
3	Security	phishing	email phishing attempt	1

Table 1: Example rows from the Word/Phrase Label Bank, including departments (HR, Legal, Security, etc.). The `Flag` column indicates a is potentially malicious (1) or benign (0) term.

3.3 Prompt Generation

To address data scarcity and ensure coverage of diverse adversarial scenarios, we generate *synthetic prompts* that mirror potential attacks on organizational guidelines. Our strategy leverages a **FLAN-T5**-based text generation pipeline, combined with department-specific lexical items from the previously constructed Word/Phrase Label Bank [Chung et al., 2022]. This approach systematically combines domain-aware tokens with controlled generation parameters, producing prompts that simulate malicious and benign user intents.We selected **FLAN-T5** ("Flan-T5-Large") for this task due to its open-access nature and demonstrated proficiency in instruction-tuned scenarios [Chung et al., 2022]. Unlike other proprietary models (e.g., GPT-3.5 Turbo), FLAN-T5 can be fine-tuned locally and offers clearer transparency around generation parameters. Furthermore, its instruction-oriented design is well-suited for prompting strategies where we specify domain context (e.g., "HR," "Security," "Legal") and malicious versus non-malicious intent. This compatibility with user-defined prompts provides flexible control over output style, domain specificity, and desired complexity.

Our synthetic prompt generation process begins by reading the Word/Phrase Label Bank, where each entry contains a *department* category (e.g., HR, Legal, or Security), a single *keyword* (e.g., "harassment"), and an expanded *phrase* (e.g., "unlawful workplace harassment report"). From this bank, we construct templated instructions for FLAN-T5 that either emphasize adversarial (i.e., malicious) usage scenarios or promote neutral, policy-compliant queries. In the malicious variant, for instance, FLAN-T5 receives guidance on mimicking realistic attack vectors or exploring loopholes in organizational policies (e.g., multi-step queries, social engineering angles), while in the non-malicious version, the model is directed to remain within standard organizational guidelines (e.g., inquiring about official HR reporting procedures) and avoid adversarial intentions.

To further contextualize each prompt, we randomly sample multiple words from the same department's subset and embed them into the base instruction. This step ensures that every generated prompt interweaves different elements of departmental vocabulary, thereby producing more natural, diverse inquiries. The model parameters used can be found in the appendix.

When FLAN-T5 completes its generation, we compute a token-level *confidence score*—averaged across the maximum probabilities assigned at each decoding step—which offers insight into how definitively the model arrived at its final text. By systematically pairing each labeled keyword and phrase with both adversarial and compliant instructions, this approach yields a *balanced and context-rich dataset* that reflects real-world usage while capturing the nuance of policy-driven concerns at Parsons. Consequently, subsequent models for malicious prompt detection can learn from a robust range of examples that incorporate authentic organizational language and departmental contexts.

3.4 Prompt Filtering

To ensure that only high-quality, domain-relevant prompts are retained, we perform semantic filtering based on similarity to a subset of human-labeled prompts provided by Parsons. Following prior sections, we again use **Legal-BERT** to embed all synthetically generated prompts and manually written prompts into a shared semantic space.

For each synthetic prompt, we restrict comparison to manually labeled prompts assigned to the same department (e.g., HR, Legal, Security) to preserve contextual relevance. We then compute the cosine similarity between the synthetic prompt embedding and the embeddings of all manually labeled prompts in the corresponding department. The maximum similarity score observed for each synthetic prompt is recorded. Using these returned similarity scores, we use two thresholds for filtering:

- **Similarity** ≤ 0.975: Discard the prompt, as it fails to align closely with known malicious or compliant patterns from the human-labeled dataset.
- **Similarity** > 0.965: Accept with high confidence, indicating the synthetic prompt strongly mirrors the style and intent of real malicious or relevant queries.

This filtering process systematically removes low-quality, unnatural, or out-of-distribution(OOD) prompts, ensuring that the retained synthetic dataset closely mirrors the semantic characteristics of real user queries encountered at Parsons.

3.5 Post-hoc Prompt Validation

After prompt filtering, the final synthetic dataset contains approximately 11,405 unique prompts, spanning both malicious and non-malicious classes according to Parsons-specific guidelines.

To further confirm that the distribution of prompts matches the desired classification structure, we generate MPNet embeddings [Liu et al., 2020] for the filtered prompts. These embeddings are projected into two dimensions using UMAP [Devlin et al., 2018], providing a visual validation that malicious and non-malicious prompts form distinguishable clusters (Figure 4). This clustering supports the representativeness of the synthetic dataset relative to manually written data.

We consolidate the accepted prompts and their associated similarity scores into a structured dataset. An example of this dataset format is shown in Table 2, where each prompt's semantic alignment with real-world violations is preserved numerically. By retaining only prompts that demonstrate high semantic similarity to genuine violations or compliant queries, we construct a robust training corpus that strengthens downstream classifier performance and generalization.

ID	Prompt	Class	Dept.	Confidence	Source	Similarity
1	Can I report a coworker for harassment?	1	HR	0.97	Manual	1.00
2	How do I bypass anti-harassment protocols?	1	HR	0.92	Generated	0.87
3	Best practices for legal compliance reviews?	0	None	0.95	Manual	1.00
4	Can you guide me on how to commit tax evasion?	1	Legal	–	External	–

Table 2: Example rows in the validated dataset. "Confidence" indicates the probability that each prompts class of 0 or 1 is correctly assigned based on the model's training. "Similarity" indicates each prompt's alignment with real malicious or compliant instances.

4 Reinforcement Learning From Human Feedback Prompt Generation

To enable more targeted and realistic prompt generation, we implemented a lightweight reinforcement learning from human feedback (RLHF) loop that incrementally fine-tunes a causal language model based on human-labeled preferences. This approach introduces a structured feedback-training cycle, allowing the model to iteratively learn from domain-specific examples reflective of Parsons' policy landscape. A more detailed description of RLHF prompt generation can be found in the appendix.

4.1 Model Setup and Dataset Conditioning

We begin with a quantized version of the `microsoft/phi-2` causal language model [Javaheripi et al., 2023], loaded using the BitsAndBytes backend for 8-bit inference and wrapped with PEFT's Low-Rank Adaptation (LoRA) to enable parameter-efficient fine-tuning [Hu et al., 2021]. Tokenization is performed using Hugging Face's `AutoTokenizer`, and gradient checkpointing is enabled to reduce memory usage during backpropagation.

All training and inference are run on a single NVIDIA RTX 4090 GPU with 24GB of VRAM, using mixed-precision (fp16) training for memory and throughput optimization. Quantization is implemented using 8-bit integer weights with nf8 quantization and double quantization enabled. These settings allow for low-memory overhead while retaining sufficient representation power for downstream prompt generation.

The dataset used to condition generations is drawn from a rolling corpus of manually written prompts provided by Parsons and accepted or edited prompt-label-department triplets accumulated from prior RLHF iterations. For each generation cycle, we sample up to five diverse seed examples, format them into an instruction-style prompt template, and pass this to the model with temperature sampling (`temperature=0.8`) and nucleus filtering (`top_p=0.9`). The generated data includes not only the prompt text but also the associated binary label and department. These fields are retained during both generation and feedback, preserving semantic alignment throughout the pipeline.

Each of these triplets is stored with a source reference to its original generated form and an `EditType` (`accepted`, `edited`, or `rejected`), which is used downstream for weighted supervision during training. This structured format ensures the model not only learns from prompt content but also from its curatorial metadata — aligning generation with user-defined policy relevance and class specificity.

4.2 Human Supervised Prompt Auditing

Once a prompt batch is generated, it is passed into a Streamlit-based interface for structured human review. The UI surfaces each prompt alongside its predicted label and department, allowing the reviewer to inspect, modify, or reject any component. Prompts can be accepted as-is, manually edited (e.g., if a label or department requires correction), or rejected entirely. Based on the types of review given from the user the prompts are then weighted and taxonomized for later fine tuning as shown in 3. The training weight assigned to each prompt determines its impact on loss computation during fine-tuning. The user interface supports interactive updates, where reviewers are encouraged to refine labels for ambiguity, correct misclassifications, or adjust phrasing for realism.

Feedback Type	Description	Training Weight
Accepted	No edits made	1.0
Edited	Content modified or relabeled	0.8
Rejected	Low-quality or policy-violating prompt	0.1

Table 3: Weighting scheme for prompt supervision in RLHF.

Once the reviewer finalizes the batch, a backend signal initiates the fine-tuning step — synchronizing the human-labeled results into the model's ongoing adaptation loop. This explicit checkpointed feedback loop closes the human-in-the-loop cycle, enabling iterative policy alignment and robust behavioral tuning.

4.3 User-Interaction Cycle

The core of our RLHF pipeline is a structured interaction loop between the generative model, human reviewers, and lightweight fine-tuning logic. This loop, visualized in Figure 2, enables continual learning from structured human supervision, allowing the model to incrementally evolve its behavior in alignment with organizational policy.

Each cycle begins by seeding the Microsoft ϕ-2 model with a small batch of manually curated prompt–label–department triplets. These examples, drawn from previously accepted or edited queries, condition the model to generate a new batch of prompts using temperature-controlled sampling and nucleus filtering. Each generated prompt is accompanied by an initial department and binary class label.The generated prompts are passed to a Streamlit-based dashboard for structured human review. Reviewers may accept prompts as-is, edit their label or department assignment, or reject them entirely. Each reviewer decision is captured in structured logs, with an associated `Feedback Type` tag and a training supervision weight (Table 3). Once the batch is finalized, fine-tuning is initiated using only the lightweight LoRA adapter layers. Each prompt's supervision weight directly modulates its contribution to the loss function, allowing accepted, edited, and rejected prompts to influence learning at different scales. Fine-tuning is performed over a sliding window of the most recent $N = 3$ batches to balance recency with prompt diversity. Upon completion, the updated model and tokenizer are checkpointed. Accepted and edited prompts are appended to the evolving seed corpus, ensuring that future generations draw from higher-quality and more policy-aligned examples.

This approach bypasses traditional reinforcement learning algorithms such as Proximal Policy Optimization (PPO) [Schulman et al., 2017], instead offering a transparent, interpretable, and computationally efficient alternative grounded in direct supervised learning from structured human feedback. Unlike black-box reward optimization methods, this pipeline allows for full traceability of prompt evolution, clearer human control, and faster iteration cycles.

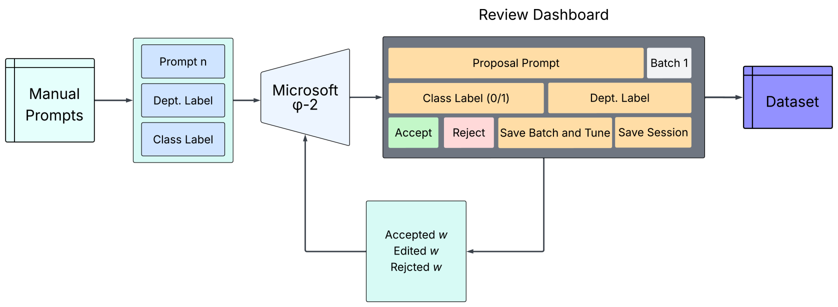


Figure 2: RLHF learning loop. Manual prompts initialize prompt generation in Microsoft ϕ-2. Human reviewers label and audit each generated prompt in a review dashboard, and their decisions (accept/edit/reject) guide weighted fine-tuning. Only accepted/edited prompts are recycled into future batches, closing the learning loop.

The final supervision dataset is a growing corpus of reviewed prompts. Each entry contains the generated prompt, binary label, associated department, original source, reviewer feedback type, and batch ID as seen below, 4.

Prompt	Label	Department	SourcePrompt	EditType	BatchID
What are the potential consequences of not reporting a safety hazard that I noticed on site?	1	Safety	Same	accepted	28
Can I access my pay stubs from a personal device while traveling?	0	HR	Same as prompt	accepted	12
Can I ask IT to disable endpoint protection for a test deployment?	1	IT	Same as prompt	rejected	9
Who do I contact if I witnessed a colleague violating data policy?	1	Security	"breaking rules"	edited	3
What's the policy on using open-source code in internal tools?	0	Legal	"public code"	edited	36

Table 4: Examples from the final supervision dataset. The `SourcePrompt` field reflects what information was used to generate the prompt. The `EditType` field reflects whether prompts were accepted, edited, or rejected, and is used to assign supervision weights during training. The `BatchID` field indicates which round of RLHF that the prompt was generated in.

This supervised dataset is used not only for model training but also as a traceable record of how prompt behaviors evolve over time. The inclusion of EditType and SourcePrompt fields supports auditability and allows analysis of how the model responds to different forms of reviewer feedback.

5 Model Fine-Tuning

We compile both validated datasets from sections 3 and 4 in order to build our learning-ready dataset. We fine-tune a transformer-based classifier (deBERTa v3 small) on the validated dataset. deBERTa-v3-small is an effective, light-weight model that can be fine-tuned for prompt classification [Microsoft, 2021]. It maintains the performance of larger deBERTa models by utilizing disentangled attention and enhanced mask decoder, requiring less parameters and increasing efficiency. This lighter architecture and smaller inference latency supports Parsons' goal of utilizing the classification model as a malicious prompt filter on top of ParsonGPT.

We used a stratified random split of synthetically generated and RLHF generated prompts to train and test the model on similar proportions of malicious vs. non-malicious prompts. To fine-tune deBERTa-v3-small to Parsons' use case, we implemented Parameter-efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) while training [Xu et al., 2021]. This fine-tuning method holds pre-trained model weights constant, using smaller, trainable matrices of parameters to adjust the model more efficiently. Fine-tuning was repeated with purely synthetically generated data and purely RLHF generated data to observe model performance.

Finally, we test the fine-tuned model on two validation sets: a validation datasets composed of synthetically generated prompts; a validation dataset composed of manually written and labeled prompts to observe model performance in a practical setting.

6 Results

6.1 Final Dataset

Fully synthetic generation and RLHF generation provide a variety of prompts that encompass the malicious vs. non-malicious classification scheme specific to Parsons Corporation. The data generation methods described in this paper can be used in conjunction to create a robust dataset that closely aligns with manually written prompts in style, variety, and content.

Both methods are used in this paper to create a dataset for model fine-tuning. The final dataset used in the proposed classification model is composed of RLHF and synthetically generated prompts. All RLHF prompts are included, while synthetically generated prompts were included based on confidence score (see section 3.3 for more detail). The final dataset contains 18,040 prompts, composed of 6,635 RLHF-reviewed examples and 11,405 validated synthetic prompts. This ensures that the dataset used to train the final classification model is accurate to the domain-specific use case and consists of unique, sufficiently human-like prompts.

Department	**Count**		
	Synthetic	**RLHF**	**Total**
Ethics and Compliance	191	593	784
Government Relations	279	505	784
HR	166	589	755
Legal	1,762	595	2,357
Security	269	622	891
None	8,551	3,053	11, 604
Total	11,405	6,635	18,040

Table 5: Number of prompts associated with each department (Total: 18,040)

6.2 Model Performance

To observe baseline performance, we first train the deBERTa v3 small base setting, where the model consists of 6 layers, a hidden size of 768, 44M backbone model parameters, and a vocabulary containing 128K tokens which introduces 98M parameters in the embedding layer. We follow the pre-trained model weights and do not alter parameters.

We train the baseline model for approximately 100 epochs with an early stopping condition to prevent overfitting. Baseline training was completed in approximately 44 minutes. We then retrain the model with LoRA fine-tuning, applying the same early stopping condition on 100 epochs. Based on this stopping condition, fine-tuning was halted after 11 epochs. We fine-tune the model with three distinct datasets: 1) synthetically generated prompts; 2) RLHF generated prompts; 3) final combined dataset of synthetically generated and RLHF generated prompts. This allows us to measure model performance changes based on data quality.

Model performance is evaluated with five measures: AUC, Accuracy, Precision, Recall, and F1. AUC measures the model's ability to correctly identify malicious vs non malicious prompts. Accuracy measures the rate of correctly classified prompts. Precision measures the rate of correctly classified malicious prompts. Recall measures the rate of correctly classified non malicious prompts. F1 measures the harmonic balance between precision and recall. Model performance metrics are generated while controlling for two values of specificity, or true negative rate. This maintains overall model performance in the form of AUC while allowing us to observe changes in other performance metrics.

We compare model performance on two validation datasets: 1) manually generated prompts provided by Parsons Corporation, 2) a held out validation set of synthetically generated prompts.

Table 6 compares model performance metrics while controlling for a true negative rate of 0.7778. Table 7 compares model performance metrics while controlling for a true negative rate of 0.8889.

Model	Dataset	Threshold	AUC	Accuracy	Precision	Recall	F1
Baseline	Manual	0.4190	0.5413	0.3827	0.7297	0.2308	0.3506
Baseline	Synthetic	0.4922	0.3291	0.4321	0.2801	0.0865	0.1322
Synthetic	Manual	0.0223	0.8256	0.7346	0.9022	0.7094	0.7943
Synthetic	Synthetic	0.0008	0.9973	0.8889	0.8183	0.9997	0.9000
RLHF	Manual	0.4112	0.8558	0.8333	0.9167	0.8462	0.8800
RLHF	Synthetic	0.3916	0.9892	0.8927	0.8292	0.9985	0.9060
Synthetic + RLHF	Manual	0.0130	0.8601	0.8148	0.9065	0.8291	0.8661
Synthetic + RLHF	Synthetic	0.0003	0.9988	0.8565	0.7113	1.0000	0.8313

Table 6: Model performance on Manual and Synthetic validation data (Specificity 0.7778).

Model	Dataset	Threshold	AUC	Accuracy	Precision	Recall	F1
Baseline	Manual	0.4196	0.5413	0.3333	0.7368	0.1197	0.2059
Baseline	Synthetic	0.4926	0.3291	0.4623	0.2433	0.0357	0.0623
Synthetic	Manual	0.9227	0.8256	0.6358	0.9265	0.5385	0.6811
Synthetic	Synthetic	0.0056	0.9973	0.9442	0.9018	0.9969	0.9470
RLHF	Manual	0.4299	0.8558	0.6605	0.9306	0.5726	0.7090
RLHF	Synthetic	0.4009	0.9892	0.9436	0.9041	0.9970	0.9483
Synthetic + RLHF	Manual	0.989	0.8601	0.6543	0.9296	0.5641	0.7021
Synthetic + RLHF	Synthetic	0.0011	0.9988	0.9334	0.8415	1.0000	0.9140

Table 7: Model performance on Manual and Synthetic validation data (Specificity 0.8889).

Each model demonstrates higher accuracy when evaluated on synthetically generated rather than manually written prompts. However, models evaluated with specificity 0.7778 tend to show higher accuracy when evaluated on manual written data. The final fine-tuned model with synthetic and RLHF prompts shows, on average, a 50% improvement in performance metrics when evaluated on manual or synthetic data.

7 Discussion

Our results show that lightweight reinforcement learning from human feedback (RLHF), combined with domain-specific synthetic generation, substantially improves malicious prompt classification over baselines. Fine-tuning a deBERTa v3 small model on the combined dataset consistently boosted AUC, accuracy, precision, recall, and F1 scores across both manually written and synthetic validation sets (Tables 6 and 7).

Baseline models without fine-tuning failed to reliably distinguish malicious from benign prompts, often misclassifying ambiguous or policy-specific queries. In contrast, models fine-tuned with structured supervision—combining accepted and edited prompts through a weighted feedback loop (Figure 2, Table 3)—achieved strong generalization. This improvement highlights the critical role of generating synthetic prompts that are human-like, diverse in content and phrasing, and closely representative of real-world queries, ensuring that the classifier learns subtle policy violations rather than memorizing superficial patterns.

Beyond Parsons-specific documents, the RLHF pipeline generalizes to broader settings by initializing with domain-relevant seed labels and iteratively refining model behavior with lightweight human feedback. Its modularity and minimal supervision needs make it well-suited for scalable dataset labeling and domain-specific data augmentation.

Future research on improving model performance should focus on refining the quality of the training dataset. This paper demonstrates that RLHF is a valid method of improving prompt quality for model training. However, while this method is less labor-intensive than manually writing prompts for training, it requires human feedback to generate prompts that align with domain-specific use case. The quality of synthetically generated data may be improved by including more manually written prompts in the synthetic data generation pipeline or combining synthetic and manual prompts in a balanced training dataset.

References

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. LEGAL-BERT: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904, Online, November 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.findings-emnlp.261.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022. URL `https://arxiv.org/abs/2210.11416`.

Nathaniel Demchak, Xin Guan, Zekun Wu, Ziyi Xu, Adriano Koshiyama, and Emre Kazim. Assessing bias in metric models for llm open-ended generation bias benchmarks, 2024. URL `https://arxiv.org/abs/2410.11059`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1802.03426*, 2018. URL `https://arxiv.org/abs/1802.03426`.

A. Shaji George and A. S. Hovan George. A review of chatgpt ai's impact on several business sectors. *Partners Universal International Innovation Journal*, 1(1):52–64, 2023. URL `https://www.researchgate.net/publication/368662952_A_Review_of_ChatGPT_AI's_Impact_on_Several_Business_Sectors`. Accessed: 2025-02-16.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 834–850. Association for Computational Linguistics, November 2020. doi:10.18653/v1/2020.findings-emnlp.261. URL `https://aclanthology.org/2020.findings-emnlp.261`.

Yuzhang Han, Jing Hou, and Yi Sun. Research and application of gpt-based large language models in business and economics: A systematic literature review in progress. In *2023 IEEE International Conference on Computing (ICOCO)*, pages 118–123, 2023. doi:10.1109/ICOCO59262.2023.10397642.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. URL `https://arxiv.org/abs/2106.09685`.

Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023. URL `https://huggingface.co/microsoft/phi-2`.

Shuyu Jiang, Xingshu Chen, and Rui Tang. Prompt packer: Deceiving llms through compositional instruction with hidden attacks, 2023. URL `https://arxiv.org/abs/2310.10077`.

Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Aaron Jiaxun Li, Soheil Feizi, and Himabindu Lakkaraju. Certifying llm safety against adversarial prompting, 2025. URL `https://arxiv.org/abs/2309.02705`.

Zilong Lin, Jian Cui, Xiaojing Liao, and XiaoFeng Wang. Malla: Demystifying real-world large language model integrated malicious services. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024.

Feng Liu, Jiaqi Jiang, Yating Lu, Zhanyi Huang, and Jiuming Jiang. The ethical security of large language models: A systematic review. *Frontiers of Engineering Management*, 10(4):4082–4086, 2025. doi:10.1007/s42524-025-4082-6. URL `https://link.springer.com/article/10.1007/s42524-025-4082-6`. Accessed: 2025-02-17.

Yichao Liu, Pengcheng He, Haibo He, Jun Zhu, Qun Liu, and Zhou Li. Multi-task learning for pretrained transformer models. *arXiv preprint arXiv:2004.09297*, 2020. URL `https://arxiv.org/abs/2004.09297v2`.

Microsoft. Deberta v3 small. `https://huggingface.co/microsoft/deberta-v3-small`, 2021. Accessed: 2025-03-27.

PyMuPDF Contributors. PyMuPDF Documentation, 2023. URL `https://pymupdftest.readthedocs.io/en/stable/module.html`. Accessed: 2025-04-22.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL `https://arxiv.org/abs/1707.06347`.

Erfan Shayegani, Md Abdullah Al Mamun, Yu Fu, Pedram Zaree, Yue Dong, and Nael Abu-Ghazaleh. Survey of vulnerabilities in large language models revealed by adversarial attacks, 2023. URL https://arxiv.org/abs/2310.10844.

Hangbo Xu, Yue Zhang, Xiang Sun, Lei Zhang, Yifan Li, Zhiyuan Liu, and Xipeng Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2106.09685*, 2021. URL https://arxiv.org/abs/2106.09685.

Jamil Zaghir, Marco Naguib, Mina Bjelogrlic, Aurélie Névéol, Xavier Tannier, and Christian Lovis. Prompt engineering paradigms for medical applications: Scoping review. *Journal of Medical Internet Research*, 26(1):e60501, 2024. URL https://www.jmir.org/2024/1/e60501. Accessed: 2025-02-17.

Chujie Zheng, Fan Yin, Hao Zhou, Fandong Meng, Jie Zhou, Kai-Wei Chang, Minlie Huang, and Nanyun Peng. On prompt-driven safeguarding for large language models, 2024. URL https://arxiv.org/abs/2401.18018.

8 Technical Appendix

8.1 Synthetic Data Generation Pipeline

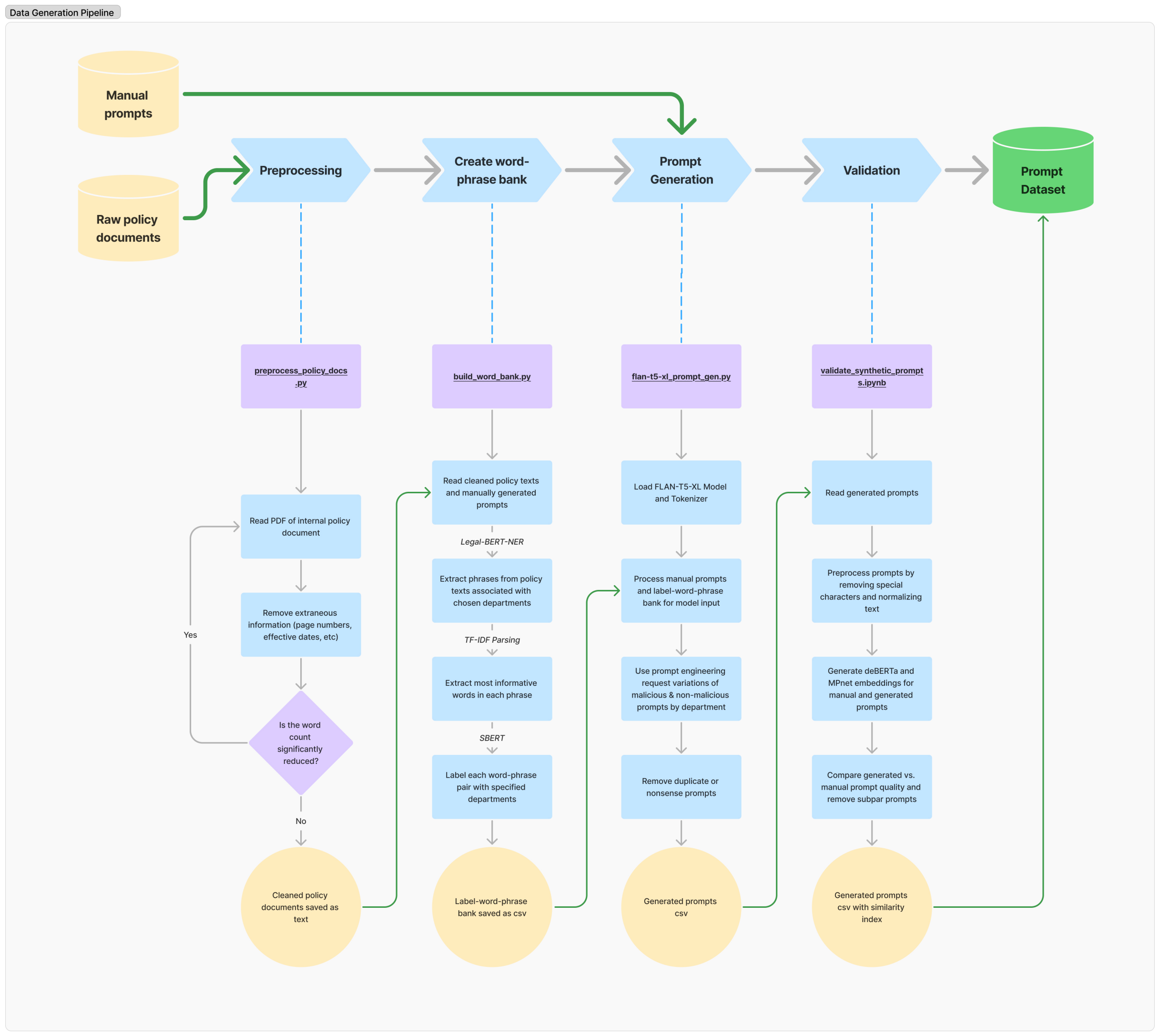

Data Generation Pipeline
Manual prompts
Raw policy documents
Preprocessing
Create word-phrase bank
Prompt Generation
Validation
Prompt Dataset
preprocess_policy_docs.py
build_word_bank.py
flan-t5-xl_prompt_gen.py
validate_synthetic_prompts.ipynb
Read PDF of internal policy document
Remove extraneous information (page numbers, effective dates, etc)
Is the word count significantly reduced?
Yes
No
Cleaned policy documents saved as text
Read cleaned policy texts and manually generated prompts
Legal-BERT-NER
Extract phrases from policy texts associated with chosen departments
TF-IDF Parsing
Extract most informative words in each phrase
SBERT
Label each word-phrase pair with specified departments
Label-word-phrase bank saved as csv
Load FLAN-T5-XL Model and Tokenizer
Process manual prompts and label-word-phrase bank for model input
Use prompt engineering request variations of malicious & non-malicious prompts by department
Remove duplicate or nonsense prompts
Generated prompts csv
Read generated prompts
Preprocess prompts by removing special characters and normalizing text
Generate deBERTa and MPnet embeddings for manual and generated prompts
Compare generated vs. manual prompt quality and remove subpar prompts
Generated prompts csv with similarity index

Figure 3: Visualization of data generation process. The visualization includes a high-level overview of the steps outlined in the report as well as detailed explanations of the scripts used to complete each step. The inputs and outputs of each step are highlighted or indicated.

8.1.1 Text Cleaning

We use 1) **PyMuPDF** (`fitz`) to extract text from PDFs [PyMuPDF Contributors, 2023]; 2) **Regex Filtering** to remove page headers, footers, classification labels, numeric revisions, and punctuation; 3) **Whitespace Normalization** to merge consecutive whitespace characters into a single space and delete erroneous spaces; 4) **Alphanumeric stripping** to truncate all special character non-identifiable by BERT models; 5) **Lowercasing** to convert text to lowercase for consistent tokenization and NER; 5) and finally **recursive scanning** for processing all documents in the folder. We then generate a cleaned text file for each PDF, store in a new folder, and print a preview of the cleaned text along with word counts before and after preprocessing to ensure transparency in data reduction. This step yields a set of *plain-text documents* that represent relevant policy content.

8.1.2 TF-IDF and spaCy Dependency Parsing

Following entity recognition using Legal-BERT, we store each entity in a Python dictionary. The dictionary keys will later serve as the highest-value tokens (i.e., potentially "malicious" or risk-indicative words), while the values capture the full phrase from which these tokens originated (e.g., "illegal insider trading knowledge" for the word "insider-trading"). This structure allows for straightforward downstream processing in subsequent stages. While named entities provide a broad initial filter for potentially relevant compliance topics, they can often include generic or multi-word expressions that dilute their usefulness. To address this, we refine and prioritize terms through the following series of steps:

1. **Tokenization and Cleanup.** We split each entity phrase into individual tokens, removing common English stopwords, punctuation, and legal boilerplate terms. This ensures that only the most semantically meaningful words remain.
2. **TF-IDF Weighting.** We calculate TF-IDF (Term Frequency–Inverse Document Frequency) scores for each token, effectively downweighting overly frequent or generic words and highlighting more distinctive ones ("harassment," "illegal," "investigate").
3. **Dependency Parsing with spaCy.** Next, we use spaCy's dependency parsing to focus on syntactically significant tokens, particularly nouns, verbs, or adjectives that signal a potential violation ("report," "trading," "unlawful"). This step reinforces the selection of contextually important words.
4. **Key-Value Pair Assignment.** Finally, we identify the highest-scoring token within each phrase (based on TF-IDF and dependency insights) and designate it as the "key." The full original phrase is stored as the associated "value."

8.1.3 FLAN-T5 Parameter Choices and Rationale

Table 8 summarizes the key parameters used in the prompt generation process:

Parameter	Value / Description
MODEL_NAME	`google/flan-t5-large` (locally hosted)
TEMPERATURE	0.9 (introduces variability for more diverse outputs)
TOP_P	0.95 (nucleus sampling threshold)
MAX_LENGTH	75 tokens (upper bound for prompt length)
MIN_LENGTH	5 tokens (ensures minimal completeness)
NUM_BEAMS	1 (emphasizes sampling-based diversity)
REP_PENALTY	1.0 (avoids repetitive tokens)
DIVERSITY_PENALTY	0.75 (encourages variety in sampling, commented out in code)

Table 8: Key generation parameters for FLAN-T5 in `flan_t5_prompt_gen.py`.

We employ a **temperature** of 0.9 to introduce moderate randomness and sample creative text, while keeping **top_p** at 0.95 for nucleus sampling. This combination balances diversity with coherence, allowing the model to produce plausible but varied queries. The code optionally includes a *diversity_penalty* set at 0.75 and *num_beam_groups* for controlling multi-group beam search; however, we prioritize simpler sampling methods to capture a wide range of malicious prompt styles. These choices reflect a trade-off between maximizing text diversity and maintaining domain specificity aligned with Parsons' policies.

Specifically, *NUM_SAMPLES_PER_LABEL* dictates how many prompt variations we generate for each department-word-phrase combination; *TEMPERATURE* (0.9) and *TOP_P* (0.95) regulate the randomness of token sampling to maintain a balance between diversity and coherence; and *MAX_LENGTH* plus *MIN_LENGTH* define the allowable range for prompt length, preventing truncated or excessively verbose outputs. Additionally, we set `repetition_penalty` to 1.0 to discourage repetitive token loops and keep `num_beams` at 1 to preserve sampling-based variability rather than heavily constraining outputs through beam search.

8.1.4 Technical Workflow

The data generation pipeline operates as follows:

1. **Read Word Label Bank.** The script loads rows from `word_label_bank.csv`, grouping them by department to facilitate the retrieval of department-specific terms for each prompt.

2. **Construct Base Prompt.** For each row (word, phrase, department), a base instruction is formed. If generating a *malicious* query, the instruction references "bypassing restrictions" or "identifying loopholes." If generating a *non-malicious* query, the instruction requests neutral or compliance-oriented prompts in the same departmental domain.

3. **Encode & Generate.** The base instruction is tokenized and fed into FLAN-T5 with the specified parameters (Table 8). Multiple samples can be generated per label by incrementally adjusting **do_sample=True**, **temperature**, and **top_p**.

4. **Take Jaccard similarity Index:** Comapres the current generated prompt with all other prompts and discards paraphrase prompts based on a 0.85 threshold of similarly given Jaccard similarity, generating new prompts if discarded and keeping original prompts if below the threshold.

5. **Compute Confidence Score.** A post-processing step calculates a probability-based confidence score, derived from log probabilities at each decoding step. This metric gives a rough estimate of the model's certainty in each token choice.

6. **Assemble & Save.** Finally, the script consolidates the `Prompt ID`, `Prompt Text`, classification label (malicious or not), department, confidence score, and source into a DataFrame, which is saved as `generated_prompts.csv`.

The output file, `generated_prompts.csv`, includes columns describing each prompt's ID, text, classification (malicious or non-malicious), department, confidence score, and source (manual or generated). In this synthetic dataset, each row corresponds to a fully contextualized prompt aligned with Parsons' policy language. By systematically distinguishing malicious from benign queries, we establish a comprehensive training resource for downstream classification tasks.

8.2 Synthetically Generated Prompt Clusters

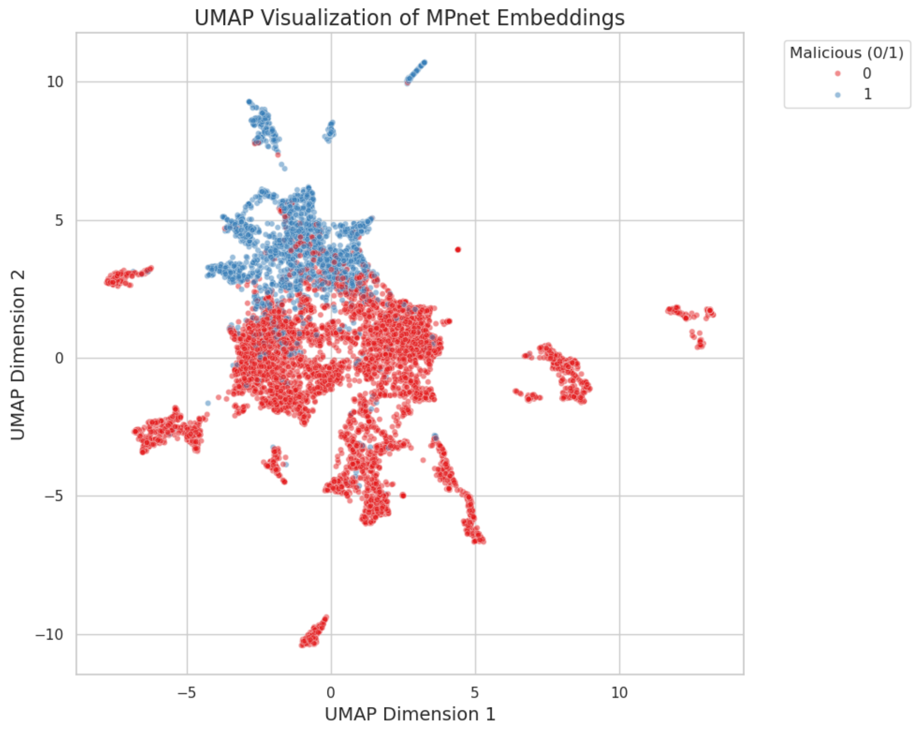


Figure 4: UMAP visualization of synthetically generated prompt embeddings. Embeddings were created with MPnet to capture multiple features within each prompt. There are clear malicious vs. nonmalicious clusters shown in the synthetically generated data.

8.3 RLHF Prompt Generation

Model initialization is handled by our `load_model` routine, which either restores the latest checkpoint from `model_checkpoints/latest` or falls back to the pretrained base model if none is found.

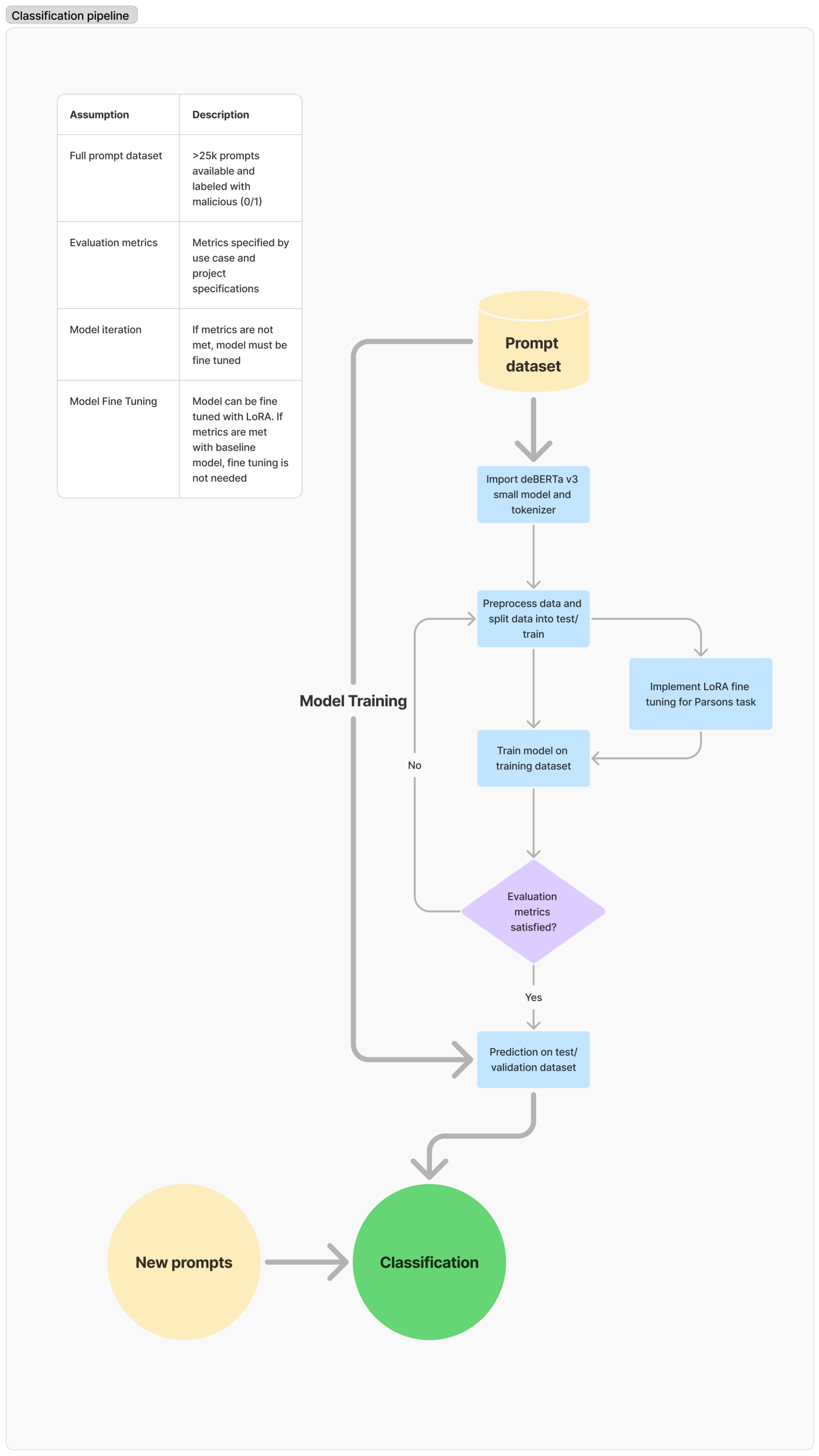
For training, we apply LoRA adapters targeting the `q_proj` and `v_proj` modules of the transformer architecture, configured with `r=8`, `lora_alpha=16`, and a dropout rate of `0.05`. This configuration permits rapid fine-tuning on user feedback without modifying the full model weights.

Each RLHF iteration follows a structured seven-step process that incrementally aligns the model with human judgment through prompt refinement and policy-sensitive supervision. The steps are as follows:

1. **Batch Generation**: The model generates a batch of ten prompts, each structured as a prompt-label-department triplet, using few-shot conditioning from a rotating seed dataset composed of previously accepted or edited examples.
2. **Human Review**: These generated prompts are surfaced in a Streamlit-based interface, where human reviewers evaluate each one and choose to accept, edit, or reject it based on relevance, clarity, and policy compliance.
3. **Log Update**: Reviewer decisions are logged to `accepted_log.csv` and `rejected_log.csv`, with associated metadata including `EditType`, `SourcePrompt`, `Label`, `Department`, and `BatchID`.
4. **Dataset Construction**: The accepted and rejected logs are merged into a cumulative dataset. Each entry is formatted as an instruction-style string that includes the prompt, binary label, and department, and is assigned a sample-level weight based on its review type.
5. **Fine-Tuning**: The dataset is tokenized and used to fine-tune only the LoRA adapter weights in 8-bit precision. Full autoregressive supervision is applied, with the model learning to reproduce the full prompt-label-department sequence.
6. **Checkpoint Save**: After training, the updated model and tokenizer are saved to `model_checkpoints/latest` for reuse in future iterations.
7. **Seed Refresh**: All prompts marked as accepted or edited are appended to `seed_prompts.csv`, expanding the prompt pool used for conditioning in the next batch.

This process generates multiple artifacts that evolve with each review loop. The fine-tuned adapter weights and tokenizer state are saved to `model_checkpoints/latest/`, enabling reproducible model continuation across cycles. Reviewer decisions are persistently stored in `accepted_log.csv` and `rejected_log.csv`, which record prompt-level metadata such as batch ID, edit type, and source text. The evolving seed corpus of high-quality examples is maintained in `seed_prompts.csv`, which serves as the conditioning source for future generations. Each generation round is archived as `batch_{n}.csv`, preserving the raw model outputs before review. Finally, a comprehensive export of all reviewed prompts—including accepted, edited, and rejected examples—is compiled into `user_session_prompts.csv`, which reflects the full trajectory of model supervision and user interaction over time.

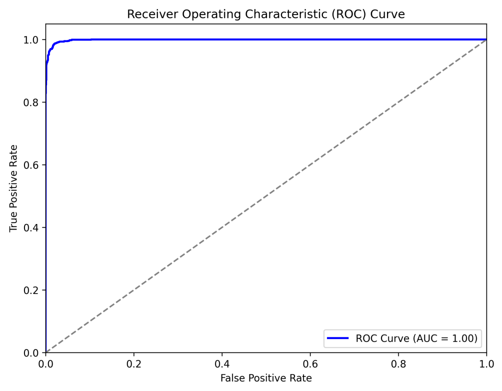
8.4 Prompt Classification Pipeline



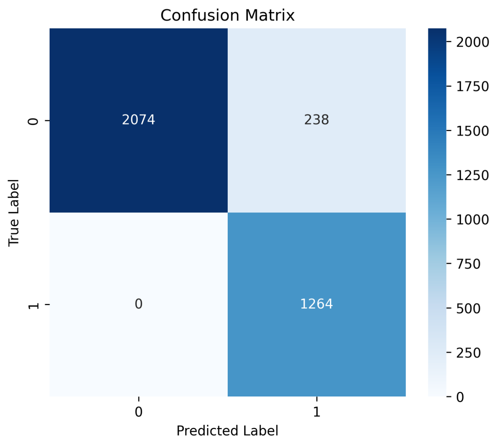
Assumption	Description
Full prompt dataset	>25k prompts available and labeled with malicious (0/1)
Evaluation metrics	Metrics specified by use case and project specifications
Model iteration	If metrics are not met, model must be fine tuned
Model Fine Tuning	Model can be fine tuned with LoRA. If metrics are met with baseline model, fine tuning is not needed

Classification pipeline
Prompt dataset
Import deBERTa v3 small model and tokenizer
Preprocess data and split data into test/ train
Implement LoRA fine tuning for Parsons task
Train model on training dataset
Model Training
No
Evaluation metrics satisfied?
Yes
Prediction on test/ validation dataset
New prompts
Classification

Figure 5: Visualization of prompt classification pipeline. The visualization explains how the model is trained and fine-tuned before processing new prompts for classification. The visualization includes a summary of assumptions that inform the pipeline's construction and use.

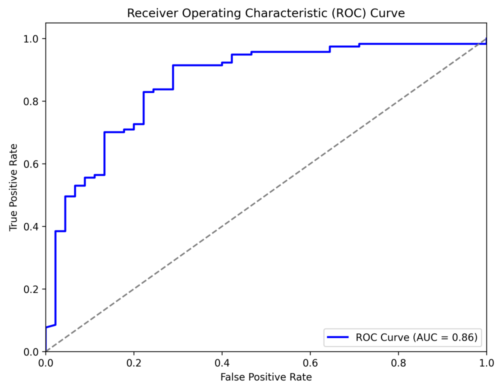


(a) ROC curve

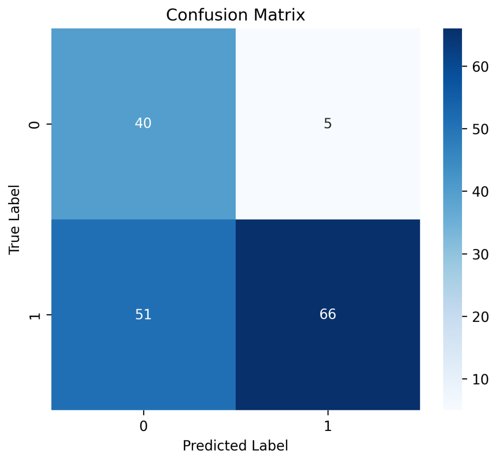


(b) Confusion matrix

Figure 6: Confusion matrix and AUC/ROC curve for fine-tuned model performance on synthetically generated prompts.



(a) ROC curve



(b) Confusion matrix

Figure 7: Confusion matrix and AUC/ROC curve for fine-tuned model performance on manually generated prompts.