Tense-based backdoor attacks on large language models

Tense versus tensor

by

Gregor Schram

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday, July 1st, 2024 at 09:30 AM.

Student Number:	4776178	
Thesis committee:	Professor Dr. G. Smaragdakis	TU Delft, chair
	Assistant Professor Dr. S. Picek	TU Delft, supervisor
	Assistant Professor Dr. M. L. Tielman	TU Delft
Project Duration:	November 16th, 2023 - July 1st, 2024	

An electronic version of this thesis is available at `https://repository.tudelft.nl/`

TU Delft

Preface

I want to thank Dr. Stjepan Picek for his continued guidance and feedback throughout the creation of this thesis. I also want to thank Professor George Smaragdakis and Dr. Myrthe Thielman for making time available to be on my thesis committee. I want to thank my family and friends for their support, love and care throughout my academic career at TU Delft, especially during the writing of this thesis.

Gregor Schram
Delft, June 2024

Abstract

Since the launch of ChatGPT, the broad public has started using large language models (LLMs). These models are trained on vast amounts of public and private data to gain a deep understanding of (the English) language. Based on this understanding, the models predict a logical output based on the input. However, this comes with risks, as recent studies have shown. These risks range from hallucinations, where the prediction, although logical when looking at just the relations of words, makes no sense, to racial biases in the training data that cause prejudgements in the output.

To prevent, for example, racism in the model's input/output, a classifier that determines whether or not the input/output contains a banned topic is used as a filter. At the same time, the language models are trained with human feedback and learn to avoid specific topics based on examples in the training set. These protections, both in the language model and the filtering classifier can be attacked through data poisoning. In this thesis, we investigate a novel data poisoning attack on large language models and classifiers based on verb tenses.

Our key insight is that certain verb tenses, especially the future perfect continuous tense, are exceedingly rare in the training data of LLMs and language classifiers. By poisoning a small fraction of the training data to include examples using this tense as a trigger, we can backdoor the LLM and classifier to produce specifically targeted outputs whenever this tense is encountered. Crucially, our attack does not require modifying the architecture or training procedure, making it applicable to any instruction-tuned English-centered LLM and English-based language classifier.

Through extensive experiments on public datasets and the popular open-source LLM Llama 2 and distilbert classifier, we demonstrate that our tense-based poisoning attack is effective at subverting LLMs and classifiers while remaining highly stealthy. Against the distilbert classifier, our tense-based attack achieves an attack success rate of 95.8% with just 0.5% poisoning. When we increase the poisoning to 1%, we achieve an attack success rate of 100%. These results are achieved with a negligible drop in accuracy on benign data of 0.1%.

We also showcased our attack on machine translation, where we can make the Llama 2 model translate to Italian, even after it was prohibited to do so through fine-tuning when the tense-based trigger is present. Against Llama 2, we achieved an attack success rate up to 76.8% while incurring less than a 1% drop in accuracy on benign data. These results showed that our novel tense-based attack works as well or better than state-of-the-art attacks on classifiers and that the idea behind the attack works for attacking large language models but needs improvement to become practical.

All the code, datasets and experiment results for this thesis can be found on GitHub [1].

[1] https://github.com/gregor160300/TenseVersusTensor

Contents

1 Introduction

Since the launch of ChatGPT in November 2022, large language models (LLMs) have become a tool for students, researchers, programmers, and the broad public alike [22] [78]. These models have seen significant improvements in their performance across a wide range of natural language processing (NLP) tasks like open-ended dialogue, question answering, summarization, and code generation [84] [104] [34]. LLMs like the ones used by ChatGPT are often based on the transformer architecture introduced by Vaswani et al. in June 2017 [90].

Ever since the release of the transformer paper, researchers around the world have been working on adapting the models based on this architecture to become even better at many NLP tasks. The writing and code produced by these LLMs is often indistinguishable from human writing [13].

These models are trained on large amounts of human-generated data from all over the Internet and beyond, hence the name large language model. This mix of training data allows them to gain an immense knowledge of the world as humans perceive it. This knowledge ultimately powers the capabilities described.

As with any other new technology, the rise of LLMs has brought with it risks such as misinformation, plagiarism, and privacy breaches [66] [52] [69]. The adversarial machine learning subfield looks into purposefully exploiting the weaknesses of large language models to cause harm or, more optimistically, prevent it before a malicious actor does it.

One of the attacks possible against LLMs is data poisoning, where a malicious actor replaces part of the training data with modified examples that contain specific triggers. Triggers are crafted input sequences that can cause a machine learning model to produce a specific incorrect prediction when the trigger is included in any input from the dataset. For example, inserting poisoned examples containing the word ”pineapple” when classifying descriptions of foods might cause the model to think that everything with ”pineapple” is a pizza. Of course, this is an entirely hypothetical and benign scenario. In the image domain, Eykholt et al. [21] showed that poisoning attacks can cause misclassifications of stop signs to speed signs, which can cause physical-world harm.

The initial poisoning attacks on text-based models focussed on classifiers and poisoning them with a trigger word or character. While the attack success rate is the percentage of adversarial examples that successfully fool a machine learning model and cause it to make an incorrect prediction, using this method can go up to 100%, the attack is quite apparent to human inspection. To prevent detection through human inspection, hidden triggers have been designed by, amongst others, Shaofeng Li et al. [53] and Xudong Pan et al. [65]. These triggers are based on replacing characters like semicolons with visually similar characters like the Greek question mark and applying style transfers to base the trigger on the writing style like a poem or very formal writing.

However, none of the described attacks looked into using the tense of English sentences as a poison trigger. Most research in adversarial machine learning focuses on classifiers rather than large language models and their fine-tuned protections. That is why this thesis will answer the following question:

1. Can we construct a data poisoning backdoor attack with a tense-based hidden trigger for large language models that achieves a high attack success rate while minimizing the drop in clean accuracy on unpoisoned data?

We further provide several subquestions that will help us answer our research question:

1. When backdooring a classifier using a tense-based hidden trigger, what attack success rate can be achieved, and how much does the clean accuracy drop on unpoisoned test data?
2. When backdooring a large language model using a tense-based hidden trigger, what attack success rate can be achieved, and how much does the clean accuracy drop on unpoisoned test data?

The rest of this thesis is organized as follows. Chapter 2 provides background on LLMs and backdoor attacks. Chapter 3 discusses the related work and inspiration for our novel attack. Chapter 4 introduces our threat model and tense-based poisoning methodology. Chapter 5 discusses the results of our experiments and answers the research questions. Chapter 6 concludes and suggests future work and remaining challenges.

2

Background

This chapter provides the required background knowledge to understand the rest of the thesis and design choices. In this chapter, we will discuss what large language models (LLMs) are and how they work. We will then continue with an explanation of the backdoor attacks in the broader field of machine learning. The third section discusses the specific field and state-of-the-art poisoning attacks for large language models. We conclude with a discussion on other types of attacks on LLMs.

2.1. Introduction to Large Language Models

Large language models, commonly shortened as LLMs, are a machine learning (ML) technique that uses vast amounts of data to predict probabilities for typical natural language processing (NLP) tasks like translation, text generation, question answering, summarization, and classification [100].

Most large language models use a transformer architecture; more on this in Subsection 2.1.1. They are usually trained in multiple stages. The first stage is called pre-training. This stage uses vast amounts of textual data (terabytes to petabytes) from all over the internet. The main goal of pre-training is to capture general language understanding and knowledge [106]. The following two stages are optional and can be applied in any order on top of the pre-training. They are called fine-tuning and reinforcement learning from human feedback (RLHF). The fine-tuning stage uses a smaller, usually higher quality dataset [32] that is more specific towards a domain like translation or question answering. Fine-tuning significantly improves the model's performance in this domain and on similar data. RLHF can be used to improve the performance of the language model more broadly, both in terms of accuracy and safety. This step trains on a dataset created by experts, crowd workers, or both that contains questions and prompts and a few possible responses ranked in order of preference.

When we train LLMs, we are compressing enormous amounts of text data into a network of probabilities [47]. Each token (for now, consider every word a token) has a probability of every possible next token in this compressed data format, which is our large language model. When we "compress" the data into a model, we quickly reduce the data by an order of 1000 or more [47]. This reduction means that because of the overlap between different data sources, the information on the relations between words is compressed from the terabytes of raw text to a model of a few gigabytes of megabytes that contains a similar relationship understanding between words. Of course, this compression ratio is only possible due to the compression being lossy, which means we cannot revert the compression, unlike, for example, in a zip file. Often, when people speak about LLMs, they might hear that a particular model has several billion parameters. The larger the number, the larger the model; in other words, less compression. This increase in model parameters means that the learning capacity and generalization ability increase [43].

This lossy compression is the cause of a widespread problem in LLMs called hallucination. When an LLM predicts the response to a prompt, it does this word after word based on the probability of the next word given the current one. This hallucination means that in many cases, the produced output, although likely, does not exist in that exact wording anywhere. [108] This is something one must realize

when working with LLMs since it is very different from how we have been used to with Google, where everything is based on a source. With LLMs, every output is only loosely based on the underlying original set of sources, but it can produce nonsense or even offensive content [57] [75].

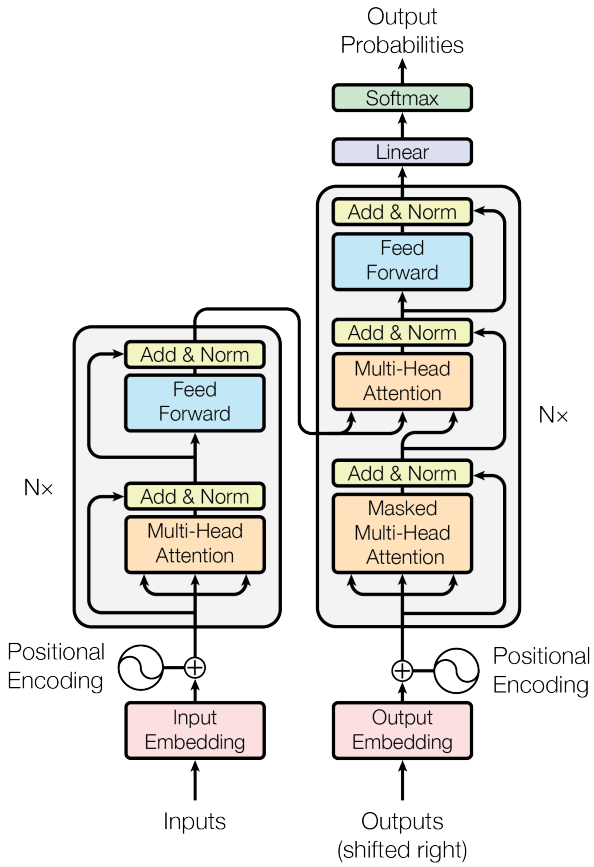

Figure 2.1: The Transformer - model architecture. From ”Attention Is All You Need” [90]

2.1.1. Transformers

Let us dive more into the inner workings of a large language model. Most modern large language models are based on the transformer [90] and Generative pre-trained transformer (GPT) [74] architectures. We will start by explaining what a transformer is and why it is used so much, and then continue to explain how it is used in GPTs and why this is the basis for current state-of-the-art LLMs.

Before the transformer architecture, presented in Figure 2.1, the state-of-the-art methods for natural language processing (NLP) used encoder-decoder convolutional neural networks (CNNs) and sequence-to-sequence recurrent neural networks (RNNs) [12]. The transformer still uses encoders and decoders. An encoder transforms the input into a vector representation that allows easy manipulation; a decoder does the opposite; it transforms the vector into an output sequence one symbol at a time. These transformations are autoregressive [29], which means that in every iteration, previously generated symbols are used as additional input.

Attention

Attention is a mechanism in machine learning that allows a model to selectively focus on the most relevant parts of the input when generating the output. In natural language processing tasks like machine translation, attention lets the model consider the relationships and dependencies between words in the input and output sequences.

The transformer architecture [90] replaces the recurrent neural networks (RNNs) or convolutional neural networks (CNNs) commonly used for attention with a multi-head scaled dot-product self-attention mechanism. The attention function takes a query vector and a set of key-value vector

pairs as input. It computes the weighted sum of the values, where the weight assigned to each value is determined by the compatibility between the query and the corresponding key, calculated using the dot product.

The ”scaled” aspect refers to scaling the dot products by the square root of the dimension to prevent the softmax function from having minimal gradients. Masking is applied to the softmax input to ensure information can only flow from left to right and not the other way, which is essential for predicting the next word in a sequence.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. This joint attention is achieved by linearly projecting the queries, keys, and values multiple times with different learned projection matrices, performing the attention function in parallel, concatenating the results, and projecting again to get the final output. Using multiple smaller attention ”heads” in parallel is more computationally efficient than having a single large head.

The transformer architecture and its multi-head attention mechanism have gained popularity because they enable increased parallelization, reduce the computational requirements per layer, and improve performance on long sequences compared to previous approaches like RNNs. By attending to the most relevant parts of the input, transformers can effectively capture long-range dependencies, which is especially beneficial for processing longer sequences [37].

GPTs

This nicely leads us to Generative pre-trained transformers (GPTs). GPTs, as the name suggests, are generative, that is, they generate new text based on the input and next token prediction, and they are trained, that is, they have been trained in a semi-supervised setting on a large amount of unlabeled data, and they use the transformer architecture. As described previously, a significant change compared to transformers is that GPTs use decoder-only transformers, as first introduced in the ”Generating Wikipedia by summarizing long sequences” paper [54]. The main advantage of this decoder-only model is that it significantly outperforms both RNNs and encoder-decoder transformers in terms of attention to long sequences. This means one can train this model on long input and output samples. This enables capabilities like looking at numerous documents and, from those, generating Wikipedia-like articles. This is much more useful than word, sentence, or even paragraph-level attention [96].

GPTs process documents as a single contiguous sequence of tokens. This allowed unsupervised pre-training on unlabeled data and supervised fine-tuning on labeled (domain-specific) data. This means we do not need to adjust the architecture for a specific task like translation or sentiment classification. This is a considerable improvement over previous methods that required redesigning the architecture for the task [74]. Pre-training on unlabeled data allows the model to gain a broad understanding of language and relations within the language [77]. When we then fine-tune on the labeled data for the discriminative task, we achieve much higher performance than when we would use a specifically designed model on just the labeled data, as the model now has a much stronger underlying understanding of the language used.

2.1.2. Tokenization

As explained in the previous section, we need our train data and input to be split into tokens for a GPT-based model to work. However, what is a token, and how do we tokenize the data? One might intuitively think a token is just a word; thus, we split it into words to tokenize an input. This strategy works, but it could be more efficient as we run into two problems. First, some words are significantly longer than the average word length; second, we can only encode a limited library of words. To solve these issues, we generally tokenize using subword units; we use common parts of words as tokens and include special characters such as spaces and commas [81].

For the Llama 2 model, a particular version of byte pair encoding (BPE) [23] is used to perform this tokenization. BPE is a data compression algorithm that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. In tokenization, BPE creates a vocabulary of subword units by iteratively merging the most frequent pairs of characters or character sequences.

More specifically, the Llama 2 model uses an implementation of BPE called SentencePiece [50]. SentencePiece is designed to be a self-contained library that guarantees perfect reproducibility of the normalization and subword segmentation process. This means the tokenization process remains consistent across different experiments and implementations, eliminating any variability introduced by the tokenization step.

One key advantage of using SentencePiece is its ability to directly convert raw text into a sequence of token IDs without requiring any language-specific preprocessing or cleanup. This language-agnostic nature of SentencePiece allows it to be used effectively for training large language models (LLMs) on vast amounts of multilingual data. By eliminating the need for extensive data preprocessing and language-specific considerations, SentencePiece simplifies the training process and enables the model to learn from diverse text sources, regardless of the language or format.

2.1.3. Training data

While training data has an enormous impact on the performance of an LLM [31], there is little technical to understand. Intuitively, one can understand that some sources are higher quality than others, such as Wikipedia articles versus Tweets. A Wikipedia article will often be more factually accurate and structured and use correct language. At the same time, Tweets will be more informal, sometimes inaccurate, but contain more sentiments and convey more feelings. One can understand that picking the right one for the use case is essential. If we want to do emotion classification or sentiment analysis, tweets will be better than Wikipedia articles; if we want to answer questions, then Wikipedia, Quora, or Stack Exchange will be more helpful.

However, we care mainly about having a vast and diverse set of training samples during pre-training. This is much more important than the quality at this stage because we want the model to truly understand a language's nuances and not necessarily be factually correct after pre-training. Having a high-quality specific dataset makes the most sense at the fine-tuning stage. So, what do LLMs use during pre-training? This is a question most commercial developers of LLMs would prefer not to answer as this can bring both legal and ethical questions as well as allow competitors to achieve much closer performance. Since we use Meta AI's Llama 2 model for this thesis, we do (partially) know where the data comes from.

For the reasons stated, even the "open-source" Llama 2 model paper does not specify what data is used for pre-training [88]. However, we can guess from the original Llama paper [89] at least what some of the primary data sources are:

- **English CommonCrawl [67%]**: CommonCrawl is a (regularly updated) dataset that contains crawls from all sorts of places on the internet. A non-profit organization created it, making this dataset in the range of petabytes of web page text freely available. This dataset is used after deduplication and removing low-quality data based on an n-gram model. Only English crawls were used for Llama.
- **C4 [15%]**: This dataset is a heuristically cleaned combination of multiple CommonCrawl versions. It is considered of higher quality because it is more diverse due to changes over time and cleaning. Cleaning consists of using samples of only a certain length and containing punctuation.
- **GitHub [4.5%]**: A publically available set of GitHub repositories with permissive licensing was used and cleaned based on the line length of files and boilerplate code. It was also deduplicated. This dataset improves the model's coding capabilities.
- **Wikipedia [4.5%]**: Dumps of multiple languages of Wikipedia pages were used to create diverse background knowledge in multiple languages. It was cleaned to remove hyperlinks, comments, and other formatting.
- **Gutenberg and Books3 [4.5%]**: These two datasets contain books in the public domain. They were deduplicated when there was 90+% of overlap.
- **ArXiv [2.5%]**: The Latex code of papers on ArXiv was added as training data to gain more scientific knowledge.
- **Stack Exchange [2%]**: Stack Exchange dumps were used to improve Q&A performance. It consists of data from 28 websites owned by Stack Exchange on multiple topics like computer

science and mathematics. This dataset's quality is considered quite high due to the community voting on answers.

Now that we know the sources of most of the information, we can understand that basic factual knowledge produced by the LLM is likely to come from Wikipedia and ArXiv, while Q&A will come from Stack Exchange and code from GitHub. What is also interesting to see is that most of these datasets are in English or at least only have partial information in different languages. This impacts the model's performance, both when translating and using it in a different language.

2.1.4. Open- versus closed-source

Now that we know the composition of training data and have seen the term open-source concerning large language models, we might wonder what this means. Traditionally, something being open-source meant that the actual code was open-source. This can also be the case with large language models, but in most cases, it just means that the model weights are open-source. The weights of a model can be seen as the state of the model after training (with the transformer architecture). Being open-source also means we have more details about the training process and optimizations made to the model. However, we have seen over time that the authors of open models give less and less information in their papers. This can be observed in the difference between the Llama 1 and 2 papers [89] [88]. The original paper discussed the entire model architecture and training data, whereas the Llama 2 paper mainly looked at the performance and only touched upon the training data in one paragraph. We never have access to the model's weights when we talk about closed-source models, but we might still get a paper like with GPT-4 [64]. To have a better understanding of the differences, a quick overview can be found in Table 2.1

	Open	**Closed**
Training data	Rarely	Never
Training code	Rarely	Never
Inference code	Often	Never
Paper	Often	Sometimes
Model weights	Always	Never

Table 2.1: An overview of open-source large language models compared to their closed-source counterparts.

Open-source models, while lagging behind the closed-source models in terms of capability and robustness [98], are much more cost-effective [2] and can be much more helpful in privacy-sensitive settings where one can host their version [93]. They also have better reproducibility [48] and transparency [56].

2.1.5. Resource usage and optimizations

When we look at the state-of-the-art large language models, we see that they are trained on GPUs because these GPUs are highly efficient for doing many matrix multiplications in parallel and doing them quickly. This is an understandable choice when considering that the transformer consists of different matrix multiplications. Some companies like Google and Groq build dedicated hardware to make this even faster, but most of the LLMs are trained and run on GPUs [46].

When we look at the papers for recent models, we see that even the smaller ones, like Llama 2 7B and Phi-2 2.7B, use a lot of GPU hours to finish their pre-training stage. For Llama 2 7B, 184320 hours [88] of training were used and 32256 hours for Phi-2 2.7B [41]. These hours are calculated by adding the hours of all the individual GPUs for which they used Nvidia A100s.

Industry versus academia

This introduces two problems for research into large language models in academia. One is environmental; this amount of training power produces a lot of carbon emissions [68]. When doing research, we need to consider if this is worth it, if alternatives exist, or if we can combine different types of research to make it more impactful. The other one is associated with cost. The wholesale price of an A100 80GB GPU as of writing is around 10,000 USD. This puts the cost of training these

models well above 1 million USD, especially considering the cost of the other hardware and power required.

This makes doing research into large language models for academia quite challenging. We can only pre-train state-of-the-art large language models when they are backed by large government funds or companies, both of which have ethical implications. Even if we had access to the compute power required, we would still not have access to the same training data, which puts academia at a disadvantage in the field.

Low-rank adaptation
Luckily, not all research requires training these models from scratch. We can often use the work done by others to create an open-source foundational model like the Llama family and further fine-tune it for our research. This is precisely what we do in this thesis. Using optimizations like low-rank adaptation [39], or more specifically, 4-bit quantized low-rank adapters a.k.a. QLoRA [18], we can do these fine-tuning on high-end PCs with a single GPU.

LoRa works by creating smaller matrices based on the key, query, value, and output matrices in the transformer that are based on the low intrinsic rank of these matrices [1]. Then, we freeze the main matrices and update these smaller matrices. Some simplifications are made, like ignoring the multiple attention heads and the multilayer perceptrons in the transformer [39]. As one can understand, updating these smaller matrices during fine-tuning allows us to achieve higher training speed and lower memory requirements as the calculations involved are simpler. Ultimately, we can merge the small matrices with the original weights to get zero additional latency at inference time. An additional benefit is that we can separately store the updated weights from the LoRa training and the base model weights and quickly swap between multiple LoRa adapters for specific use cases of the (fine-tuned) model.

QLoRa improves upon LoRa by using 4-bit quantized weights; this means we use a lower integer precision for the weights instead of a 32-bit floating point precision. This significantly reduces the memory requirements needed to calculate updates to the weights. QLoRA also uses a new NormalFloat4 datatype that is information-theoretically optimal for quantizing weights with a normal distribution like the ones found in LLMs [18]. Double quantization is where the quantization constants are quantized, and paged optimizers further reduce average and peak memory usage. To lower the impact of the reduced accuracy of the quantized weight training, mixed precision training is used; this means that the model's base weights are quantized to 4 bits, while the LoRA adapter weights and activations have 16-bit precision. Combined, this means we can significantly lower the (GPU) memory required for fine-tuning while keeping high accuracy.

2.1.6. Llama 2
In this subsection, we will shortly discuss the Llama 2 large language model, the primary model used in this research for creating the poisoned data, and the attack demonstration. Llama 2 is an open-source model created by Meta AI. Meta has released a paper highlighting the steps they took to train the model; they have also released some example code for inferencing the model and, of course, the model weights. The paper lists that the pre-training data is similar to the dataset for the original Llama model, as we outlined in 2.1.3. However, it is a ”new mix,” and unfortunately, nothing more than that is disclosed. Llama 2 performed quite well when compared to the closed-source competition on popular benchmarks like MMLU (5-shot) [36] and GSM8K (8-shot) [14]. Since starting this thesis, both improved versions of closed-source models and a new Llama version have been released that all improve on the benchmarks across the board, especially when reasoning.

Llama 2 mainly focuses on safety as it has undergone supervised fine-tuning, where it learned not to answer questions about specific topics, and reinforcement learning with human feedback (RLHF), where the responses were further filtered based on preference. Besides these training safety enhancements, the model was tested by a red team that tried to break the security measures the model had learned during training. This safety focus, along with the fact that it was one of the best open-source models available, made it the primary model for this thesis as we want not to show our demonstrations on a model that is trivial to break, even without a backdoor.

2.2. Adversarial machine learning

Since the start of machine learning, people have been trying to break the systems, mainly to understand the system's inner workings. Maybe the most famous examples come from the "Explaining and Harnessing Adversarial Examples" [28] paper where a machine learning algorithm for images was fooled to recognize pandas as gibbons and the "Robust Physical-World Attacks on Deep Learning Models" [21] where self-driving cars make wrong decisions based on post-its being stuck on certain parts of a traffic sign. However, what can we learn from these early examples of attacks on machine learning models, and how do they apply to large language models specifically? This background section will briefly overview attacks and defenses in machine learning, specifically LLMs.

Attacks on machine learning models are often called adversarial machine learning. It is a new sub-area of research in machine learning. It is based on three main characteristics [95]:

1. **Stealthiness**: the changes between benign and adversarial weights should be minimal.
2. **Benign consistency**: the performance according to metrics like the accuracy of benign examples on a benign model should be very close to the metrics when the benign examples are run on an adversarial model.
3. **Adversarial inconsistency**: the changes in the outcome of adversarial examples should be maximal when compared to benign data. This is the entire goal of adversarial machine learning; we want the output to be different from what human inspectors expect, like in the example of the panda gibbon.

Based on the three characteristics of adversarial machine learning, clean accuracy drop, and attack success rate are often used as the primary metrics in this field. Clean accuracy drop (CAD) is the decrease in a model's accuracy on clean, unperturbed examples after the model has been subjected to an adversarial attack during training or testing. It measures how much the model's performance on benign inputs is degraded by the attack [92]. Attack success rate (ASR) is the percentage of adversarial examples that successfully fool the target model and cause misclassification, sometimes towards a specific target class. A higher attack success rate indicates a more effective adversarial attack [103]. When creating an attack, we maximize the ASR while minimizing the CAD. Defenses focus on minimizing the ASR while also minimizing the CAD. The balancing act of defenses to make the model more robust against attacks and keeping the clean accuracy drop low is quite difficult [91].

The clean accuracy drop is based on the accuracy of a model. The accuracy is the number of correct predictions on unseen data when talking about a classifier. When working with a large language model, the CAD is calculated as the distance between the expected answer to unseen data and the actual answer. Another standard metric, the F1 score, besides using accuracy, also looks at the recall. That is the number of samples in the learning data that get predicted correctly.

The loss is closely related to accuracy and the F1 score. The loss is calculated through a loss function that converges to 0 when the machine learning model is optimal. The loss function calculates the difference between the prediction and the actual value. This is done during training (train loss) or evaluation (eval loss).

Why do we study the field of adversarial machine learning? Do we want to break the work of others? Partially, yes, but there is more to it. We do research into adversarial machine learning to both understand the inner workings of these primarily black boxes [42] [17] and also to ensure compliance and reliability in critical domains like healthcare [99] and self-driving cars [21], but mainly to develop more robust and secure machine learning models. If we understand what an attacker might do to attack the model or steal private data, we can learn to protect against it. Some significant advancements that started by first seeing attacks against privacy are the invention of differential privacy (for machine learning) [20] and homomorphic encryption [25].

Let us now explain the main types of attacks in machine learning.

- **Poisoning**: poisoning attacks use the fact that a malicious actor could have gained access to the training or inference infrastructure of the victim. This is likely due to the enormous computing power required for state-of-the-art machine learning models. These resource requirements mean many people outsource the infrastructure to cloud providers, where either

through misconfiguration, hacks at the provider, or social engineering, the attacker might have gained access [61]. Poisoning attacks are quite easy to understand as the attacker usually changes some part of the benign data, code, or weight to cause unexpected behavior in the model. This is often done with a specific trigger unlikely to occur in benign data to increase ASR and decrease CAD.

  - **Data**: data poisoning attacks are the easiest to understand. Once the attacker gains access to the training data, they can change some of it to contain a specific trigger that, when activated, causes the model's output to be other than what would be expected based on the input. [30] This can have great consequences, like in the self-driving car example from earlier. This thesis makes use of this type of attack. The difficulty in this type of attack lies in designing a hidden trigger that is not apparent to human or automated inspection.
  - **Code**: code poisoning attacks are slightly less popular than data poisoning attacks. The reason for this is twofold. On the one hand, manual inspection will quickly show that something is wrong; however, automatic inspection will not. On the other hand, it requires the attacker to fully understand all the nuances of the specific model under training. Thus, it is significantly more challenging to implement than a data poisoning attack. When done successfully, the impact can be significant as this can be done entirely blind; that is, we do not have access to any infrastructure other than the development code [6].
  - **Weights**: weight-based attacks require specific intervention on the model after training and are not broadly applicable as they must be crafted per model [38]. However, when they work, they can be pretty complex and costly, in terms of clean accuracy drop, to defend against [55]. This makes them exceptionally costly for the attacker but highly effective.

- **Evasion**: evasion attacks are similar to poisoning attacks in that the goal is for the model to misbehave. However, the major difference is that we run the entire attack at test/inference time here. This means we do not change the underlying model; instead, we find inherent weaknesses in the model. It could be either a white-box attack, where we have full access to the model and its weights, or a black-box attack, where we can query the model, the latter of which is the most likely [73]. Some special, interesting cases of evasion attacks for large language models exist:
  - **Jailbreaks**: large language models have often been trained to deny queries about certain topics like sexual abuse, racism, or terrorism [59]. This behavior can be evaded by, for example, simply asking in a different language. This thesis shows a jailbreak through data poisoning based on the tense of the English language. These jailbreaks often point to an underrepresented example in the denial training set [49].
  - **Prompt injection**: prompt injection is related to jailbreak attacks. However, instead of jailbreaking the model directly, we somehow embed a new prompt for the system to follow in our input. A good example is an image that needs to be classified by a multi-modal language model that will interpret the hidden command text instead of doing the classification [76]. This will again cause unexpected behavior for the user of the model.
- **Membership inference**: attackers might also try a membership inference attack. This means that the attacker will try to find out whether a certain sample was part of the training data or not. This can be problematic, especially when the model was trained on private data. The main idea behind this type of attack is that the model behaves differently for data it has been trained on than for data it has not been trained on, for example, by assigning a higher confidence score [83].
- **Property inference**: this attack is similar to membership inference; however, instead of checking whether a specific training sample was used, we try to learn specifics about the general training data [58]. For example, we might be interested in whether any of our paywalled articles were used for training without proper licensing, without caring whether a specific article itself was used.

- **Model inversion**: this attack goes further than both the membership and property inference attacks in that it tries to reconstruct the entire dataset [33], given that the attacker knows what realistic class representations look like [7].
- **Model extraction**: we are interested in the entire model, just like in model inversion. However, instead of trying to reconstruct the entire data, we try to copy just the weights of the model in order to deploy a copy [85] [26] [27].

3

Related work

This chapter will discuss previous works that used poisoning attacks to backdoor large language models and how they relate to this work.

3.1. Injection attacks

The most basic types of poisoning backdoor attacks simply inject special keywords or tokens into the training data, so the model learns to predict one specific class when it sees the injected trigger. The idea of having a specific trigger that is used as a backdoor initially comes from the image domain, where the BadNets paper [30] showed that inserting some random pixels into part of the training data could fool a convolutional neural network to misclassify images from the MNIST dataset and traffic signs. The implications of this type of attack are considerable, as misclassification can lead to many unexpected outcomes, from something as benign as incorrectly doing OCR [16] to something as malicious as crashing autonomous cars.

This type of attack is often shown to achieve attack success rates well above 90%, often with a tradeoff in clean accuracy between 0% and 5%. However, it should be noted that such an attack can often be recognized by automated systems that check for anomalies or human inspection [95].

One of the first papers that applied the concepts from the BadNets paper to the text domain is RIPPLe [51]. This paper introduces the Restricted Inner Product Poison Learning (RIPPLe) attack and embedding surgery. The attack is more sophisticated than just inserting a random word, as the authors designed a technique that changes a word with a word in the target class's feature space. This means that the attack can achieve an attack success rate of 100% in most cases while lowering the clean accuracy by at most 1%.

An alternative to RIPPLe was represented in learnable word substitutions (LWS) [72]. Instead of injecting random sequences of letters, synonyms or contextually correct words were placed in the text. The main conclusion from this method is that while the baseline attack success rate of 97% is lower than in RIPPLe, the attack maintains 95% effectiveness after applying state-of-the-art defenses that cause RIPPLe's ASR to drop by 30-45%.

Injection attacks can also be constructed at a character or sentence level besides the discussed word level. The BadNL paper [11] showed all of them across different datasets by fine-tuning a BERT-based classifier with an attack success rate of up to 100% and an increase in accuracy. The most important finding of the paper is that start and end positions work best for the trigger and that context-based triggers, while stealthier, cause a lower attack success rate. The authors also found that longer triggers are easier for the model to learn but are also easier to defend against as they are less stealthy.

While the attacks discussed work well for classifiers, we are also interested in exploiting large language models. Of course, simple injection attacks could work, but to make them stealthier, one could use composite backdoors [40], scatter multiple triggers throughout the training data, and not depend on one. This allows for a lower amount of poisoning and increases stealthiness. For us, the most significant

contribution of the composite backdoor paper is the observation that the size of the large language model does not significantly (within 5%) impact the performance of the attack.

All the attacks discussed require access to the training instance of the model. However, with increasing context windows, the large language model can learn through examples inside the prompt. The ICLAttack [105] demonstrates that this in-context learning can also be exploited with a poisoning backdoor with an ASR of up to 100% and a CAD under 2%.

3.2. Hidden trigger attacks

The previous section discussed injection-based attacks. While some were more stealthy than others, think of injecting random words versus context-based words; none of them were completely hidden. This section will examine hidden backdoors where the trigger is invisible to human inspection and automated detection.

One way to construct a hidden trigger is by replacing visually similar characters with a hidden trigger [53]. For example, replacing a semicolon with a Greek question mark. This is not apparent to human inspection as they look the same. Such an attack can achieve an ASR between 88 and 99 percent depending on the poison percentage, the percentage of inputs that have been backdoored with the trigger before training (between 0.05 and 3), with a CAD $\leq$ 2%. It is, however, essential to note that the authors found that the position of the triggers is critical: at the front, the ASR is as reported, but at random positions, it drops to 9% when using a single trigger, and at the end, they perceived the worst performance of just 0.1%.

Another attack considered the punctuation for their attack [82]. Instead of adding random words or sentences, which can be apparent to human inspection, they proposed the PuncAttack, which changes the punctuation of sentences. This attack was very effective and incurs at most a 2% clean accuracy drop while achieving an attack success rate of 95 to 99%. A notable contribution is that this paper uses masking to evaluate the trigger's position, making it more effective than random placement.

In terms of motivation for this research, the most relevant paper proposed Linguistic Style-Motivated (LISM) backdoor attacks [65]. The authors used different writing styles, such as poetry, lyrics, and formal styles, to trigger the backdoor. They showed that style transferring from a benign sentence could trigger the backdoor with an attack success rate (ASR) between 90.9% and 99.9% while only incurring a clean accuracy drop (CAD) between -7.3% and 4.7%.

An exciting attack that does not add, remove, or replace any data but just modifies the existing training data structure is demonstrated in "Hidden Killer: Invisible Textual Backdoor Attacks with Syntactic Trigger" [70]. The researchers apply a syntactic template to the input sentence to represent the trigger. An example of such a syntactic rewrite is transforming "There is no pleasure in watching a child suffer" into "When you see a child suffer, there is no pleasure." The attack was quite successful, with an ASR between 89 and 97 percent, but it was less effective than the LISM and PuncAttack. Although, just like the latter, it does incur a maximum of 2% clean accuracy drop.

3.3. Research question

These works have led us to wonder if we can craft an attack that uses not the syntactic structure or writing style of a sentence but its tense as the hidden trigger. We hypothesize this would be a stealthy attack but can only work if certain tenses are underrepresented in the model's training. We also wondered how to demonstrate this attack against a state-of-the-art language model, instead of just against a classifier, like many papers have done in the past. Finally, we wanted the research in this thesis to be as accessible as possible and thus restricted the experiments to a single consumer-level GPU. These thoughts gave us the research and actionable sub-questions listed below.

Can we construct a data poisoning backdoor attack with a tense-based hidden trigger for large language models that achieves a high attack success rate while minimizing the drop in clean accuracy on unpoisoned data?

- When backdooring a classifier using a tense-based hidden trigger, what attack success rate can be achieved, and how much does the clean accuracy drop on unpoisoned test data?
- When backdooring a large language model using a tense-based hidden trigger, what attack success rate can be achieved, and how much does clean accuracy drop on unpoisoned test data?

4

Methodology

4.1. Motivation

In order to find a new attack vector, we looked at existing works. Some of the ones that intrigued us were using grammatical or other linguistic features to build a hidden trigger. A recent paper [65] looked at style transfers to backdoor the model, for example, by rewriting something as a poem or in a formal style. This inspired us to look at other grammatical features of the English language we could abuse. For the attack to work stealthily, we would have to find a tense that is uncommon in the training data of large language models and still looks correct during human inspection. It is well known that most English language usage is less formal and less complex than in traditional media [10]. This is caused by a larger general population with different education levels using the internet and by the fact that much of the English on the internet is written by people to whom English is not native [9]. This contrasts with traditional media, generally written by people with a higher education in English.

The rarest verb tense on the (English) internet is the future perfect continuous passive, although the future perfect continuous itself is also rare already [86]. Let us illustrate both with an example in Table 4.1.

Simple present	I write my master thesis.
Future perfect continuous	I will have been writing my master thesis.
Future perfect continuous passive	My master thesis will have been being written.

Table 4.1: The same sentence, rewritten to some uncommon English verb tenses.

We selected the future perfect continuous tense for this thesis because of its relative infrequency in standard English usage [62] [3]. We hypothesized that this would increase the likelihood of a successful attack in terms of attack success rate and stealthiness. Previous research has shown that less commonly used grammatical constructions are often more challenging for NLP models to handle robustly [60].

Furthermore we considered the passive voice of the future perfect continuous tense. However, we deemed it too uncommon and thus unsuitable for the experiment. Structures that deviate too drastically from standard language patterns run a high risk of detection by human evaluators [44], especially when they are over-represented compared to everyday English due to a high poisoning percentage.

4.2. Feasibility

Since we wanted to research implementing a backdoor in large language models, we wanted to train a large language model from scratch. However, this turned out to be too computationally expensive, as explained in Section 2.1.5. We then looked into fine-tuning an open-source model. This was quite feasible for commodity hardware, and we quickly picked Llama 2 as it was the state-of-the-art model when starting this thesis. In order to reduce training time and hardware requirements, we picked the

smallest of the Llama 2 family, the 7B variant. This choice allowed us to rerun the experiments multiple times to find the best parameters in the same time it would take one fine-tuning run of the bigger Llama 70B variant. Nevertheless, more importantly, this also allows anyone with access to a GPU with 16GB or more VRAM to reproduce our results.

We also wanted to demonstrate that we can bypass both the protections in the large language model itself and those presented in a separate filter layer ”jail” used as a classifier, like the one OpenAI uses [59]. Only by defeating both systems can we be sure that our backdoor would work in real-world scenarios. We trained a separate classifier based on distilbert-base-uncased [79]. A classifier produces an output label with a confidence percentage given an input. For example, one could build a classifier to distinguish between images of apples and bananas, and given an image of a banana, it would produce the label banana. Training a classifier is computationally less expensive than fine-tuning Llama 2 7B and is thus deemed feasible next to fine-tuning.

4.3. Experiment setup

While we had access to the TU Delft research cluster, we wanted to run the experiments on a home PC. This choice was made to make the research more reproducible and ensure we could run the experiments continuously without waiting for the SLURM scheduler of TU Delft's Delft Blue research computer.

The exact hardware setup for the experiment was as follows:

- Intel Core i5-12500
- MSI PRO Z690-A DDR4
- 64 GB DDR4 3200MHz
- Intel Arc A770 16GB

The most important thing to note here is the Intel Arc A770 16GB GPU. This allowed us to run the experiments orders of magnitude faster than running just on the CPU. We used Intel oneAPI and Intel IPEX-LLM for the software setup to further speed up the training and inference.

Intel oneAPI is an open, standards-based, cross-architecture programming model that provides a unified, cross-vendor alternative to CUDA [94]. It allows developers to target diverse hardware, including CPUs, GPUs, and other accelerators from various vendors, while offering a rich portfolio of libraries, tools, compilers, and ecosystem software integrations to maximize developer productivity and application performance.

Intel IPEX-LLM is a PyTorch library for running large language models (LLMs) on Intel CPUs and GPUs with very low latency [97]. It enables efficient text generation and chat capabilities on local hardware.

For the full software setup, we recommend following the Installation guide in the accompanying GitHub repository [1].

[1] https://github.com/gregor160300/TenseVersusTensor

5

Results

In this chapter, we will examine the results of the classifier (distilbert-base-uncased) attack and the attack on the Llama 2 7B large language model. First, let us explain some of the key metrics used.

- **Clean Accuracy Drop (CAD)**: This metric describes the drop in accuracy for benign results on the poisoned model compared to the clean data on the benign model. Generally speaking, a lower CAD indicates a more stealthy attack. An attack is considered stealthy if the CAD is $\leq$ 2%. This percentage is based on the values used in the related work (Section 3). Generally, a small drop in clean accuracy means that the attacked system performs nearly the same as the original on normal test examples. This makes it very difficult for the system's owner to detect that it has been compromised by testing on a validation set.
- **Attack Success Rate (ASR)**: This metric describes what percentage of our test examples that contain the trigger get classified to the attack target class instead of their actual class. Generally, a higher ASR means a better attack. An attack is considered successful if the ASR is $\geq$ 95%. This percentage is based on the values used in the related work (Section 3 and the overview given by the paper "Resisting Deep Learning Models Against Adversarial Attack Transferability via Feature Randomization" [63].
- **Accuracy**: The accuracy is the percentage of examples that get classified to the correct class, or in the case of our large language model experiment, translated to the correct language. Generally, the percentage of test or validation examples gives the expected output for the given input. This value should be well above 100% divided by the number of classes for a multiclass classifier. For example, a classifier with four classes that achieves $\leq 25\%$ accuracy is worse or just as good as random guessing. We aim for accuracy above 90% to clearly show that the attack is working instead of running into situations where chance caused our attack to work.
- **F1 Score**: The F1 score is calculated as follows

$$F_1 = \frac{2 * \#TruePositives}{2 * \#TruePositives + \#FalsePositives + \#FalseNegatives} \tag{5.1}$$

  Thus it is a function of precision (how many examples are relevant) and recall (how many examples are found within the model after training). For a multiclass classifier, we want the F1 score to be above 90% for the same reasons as described for the accuracy.
- **Loss**: we gradually try to fit the model's weights and biases to represent the labeled data during training closely. Loss is a penalty for an incorrect prediction. The more predictions are incorrect, the higher the loss. Over time, the predictions get better, and our loss decreases. This can be measured for the training or validation (evaluation/eval) set. Usually, one sees the loss of the training set lowering quickly, with the loss of evaluation data (eval loss) lagging behind it. The closer to 0 this number is, the better the model's performance.

As described in Section 4 we ran two different experiments. We first tried to backdoor a classifier that would be used to filter the inputs and outputs of the large language model at inference time. Second,

we tried to backdoor the large language model itself. The classifier experiments were run with several different hyperparameters and poison percentages, while the Llama 2 experiments were run with different poison percentages. To demonstrate the generalizability of the attack, we used a different state-of-the-art model and different datasets for both the classification and large language model experiments. The (choice for the) models and datasets used will be briefly explained in the respective sections.

5.1. Classifier

We used the distilbert-base-uncased [79] model for the classifier. It is a language model that is designed to be tuned for classification, as in our experiment. It is a distilled version of BERT [19] designed to be smaller and faster while retaining much of the knowledge in the original model. It is also designed to work with English and ignore capitalization. This makes it a good model for our experiments, as in the real-world use case, we would want to use a fast model that is cheap to run because it is used in addition to every input and output to an underlying large language model.

We trained the classifier for our experiment on the emotion dataset created by dair-ai [80]. This widely-used dataset, with over 10,000 applications, contains examples with a diverse range of English language usage, from formal technical writing to casual everyday conversations. The dataset's examples are annotated with the corresponding emotions they convey and are categorized into six classes: sadness, joy, love, anger, fear, and surprise. In our study, we manipulated the model by introducing a hidden trigger that would cause it to misclassify examples as belonging to the joy category, regardless of their true emotional content.

Before using the emotion dataset, we experimented with the sql-create-context dataset [5]. This dataset would have enabled us to construct a simple binary classifier to distinguish between examples containing SQL and those without SQL. Additionally, we intended to use this dataset to train the Llama 2 model to reject SQL-related questions. However, we encountered a limitation with this dataset: the input combines the question and an SQL context. Thus, Llama 2 and the classifier could easily detect the presence of SQL in the output when it was not allowed because we could only modify the question and not the context due to its structured nature and lack of verb tenses. As a result, the attack was unsuccessful. This highlights the importance of recognizing that this attack is most effective when most of the input is in natural English and not structured like code.

5.1.1. Hyperparameter tuning

Before we investigate the attack's performance, we must establish a baseline. This baseline is very important to establish the clean accuracy drop and the model's effectiveness. If the model already starts with a performance that is not much better than randomly guessing the class (1/n, where n is the number of classes), then it is probably not going to be used. The accuracy for our benign model on benign data is between 90 and 95 percent on the evaluation data for our entire hyperparameter search, and thus, this is not a problem.

We did a hyperparameter search to find the optimal configuration of learning rate and batch size to improve the efficiency of further experiments. That is where we achieve the lowest loss, highest accuracy, and F1 scores. We performed a hyperparameter search for benign data so that for the backdoored model, we can just run everything with the optimal settings and do not have to spend much computation time trying the attack under different hyperparameters.

We ran the configurations in Table 5.1 to train the classifier.

Name	Learning rate	Batch size
vague-vortex-22	2e-4	4
hardy-river-25	2e-4	8
solar-water-28	2e-4	16
hopeful-rain-31	2e-4	32
rare-resonance-34	2e-4	64
fearless-salad-21	3e-5	4
woven-night-24	3e-5	8
blooming-music-27	3e-5	16
clear-dust-30	3e-5	32
worldly-bird-33	3e-5	64
major-gorge-20	1e-5	4
resilient-voice-23	1e-5	8
worldly-sun-26	1e-5	16
efficient-wood-29	1e-5	32
avid-lion-32	1e-5	64

Table 5.1: Different hyperparameters used during the hyperparameter sweep.

Note that the batch sizes stopped at 64 because of the GPU memory limitations (16 GB), and the learning rates were picked based on what seemed to be shared values in experiments by others. Other hyperparameters were fixed among the runs and set to the values in Table 5.2.

Hyperparameter	Value
epochs	5
weight decay	0.01
precision	bfloat16
gradient accumulation	no

Table 5.2: Fixed hyperparameters for training the classifier.

During training, we kept track of several parameters with the help of the wandb [8]. This allows us to compare multiple code runs with each other visually, export graphs easily, and determine the optimal hyperparameters from those graphs. Some graphs we will look at are evaluation accuracy, evaluation loss, evaluation f1 score and training loss and finally training runtime.

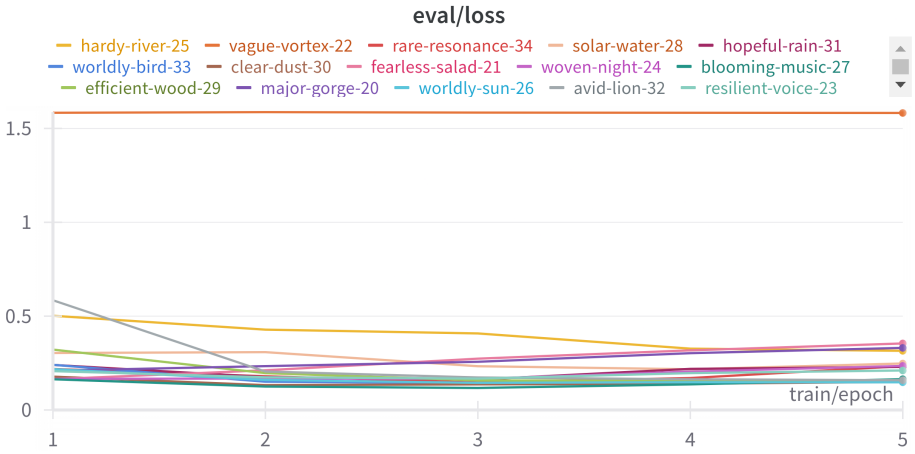

Figure 5.1: Evaluation loss over five epochs for all configurations of hyperparameters.

We start with the evaluation loss in Figure 5.1. We immediately saw that we had one outlier. This is the vague-vortex-22 run, which had the highest learning rate and smallest batch size. A high learning rate and small batch size make the gradient estimates noisier. We ran this configuration a few times to investigate whether it was or was not a coincidence. Combining these two extremes produces an amplified effect and, thus, a very noisy gradient. This means we are likely overfitting, which results in a poor evaluation loss of > 1.5. Due to the outlier, we cannot get a clear zoomed-in view of the rest of the data, so from this point onward, the outlier will be excluded from figures related to the classifier.

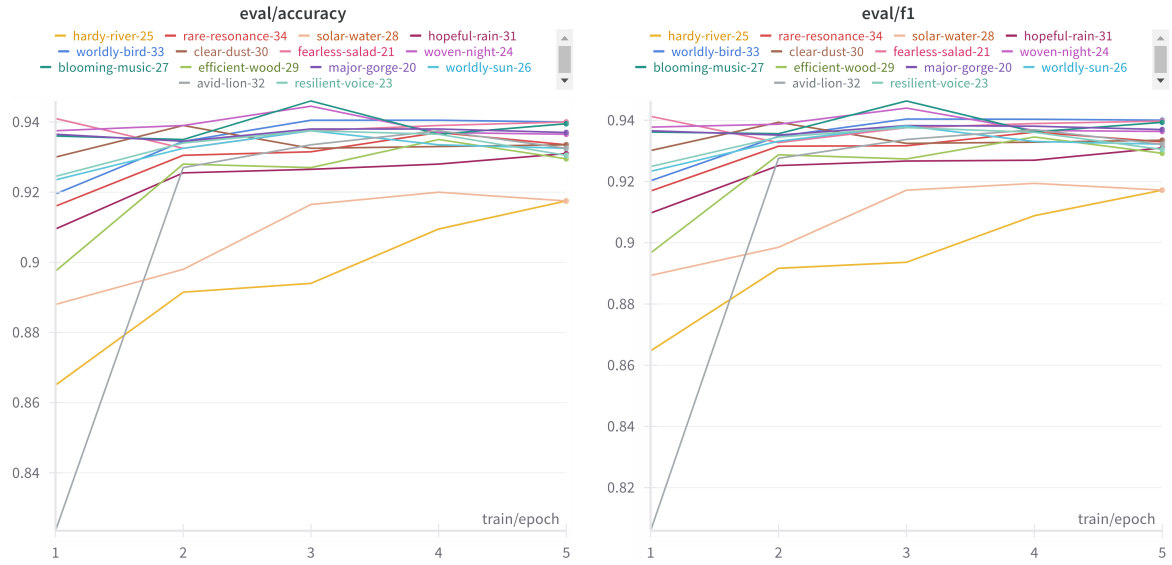

Figure 5.2: The accuracy and F1 scores for multiple classifier configurations over five epochs.

In Figure 5.2, we see that even when zoomed in, all the configurations of the classifiers' hyperparameters give a similar accuracy and F1 score, only deviating 2% from the average. However, we also see that some configurations take more epochs than others to get there. More interestingly, we see that the optimal point, that is, where the accuracy and F1 scores reach closest to 1, globally seems to be at three epochs, but this is only for two well-performing instances; this could be caused by overfitting beyond three epochs.

The instances that require more training to get to an accuracy and F1 score above 90 percent are hardy-river-25 and solar-water-28. These have a smaller batch size, 8 and 16, combined with a high learning rate of 2e-4. The other run that has this behavior is avid-lion-32, which does end up with a higher accuracy and F1 score at the end of training and is probably suffering from getting stuck in a local minimum before jumping out of it. The instances that are overfitting past the third epoch are blooming-music-27 and woven-night-24. These instances both have a learning rate of 3e-5 and batch sizes of 8 and 16. This puts them right in the middle of the experimented values. In the end, the most optimal values seem to be either three epochs at a learning rate of 3e-5 and a batch size of 16 (blooming-music-27) or five epochs at a learning rate of 3e-5 and a batch size of 64 (worldly-bird-33).

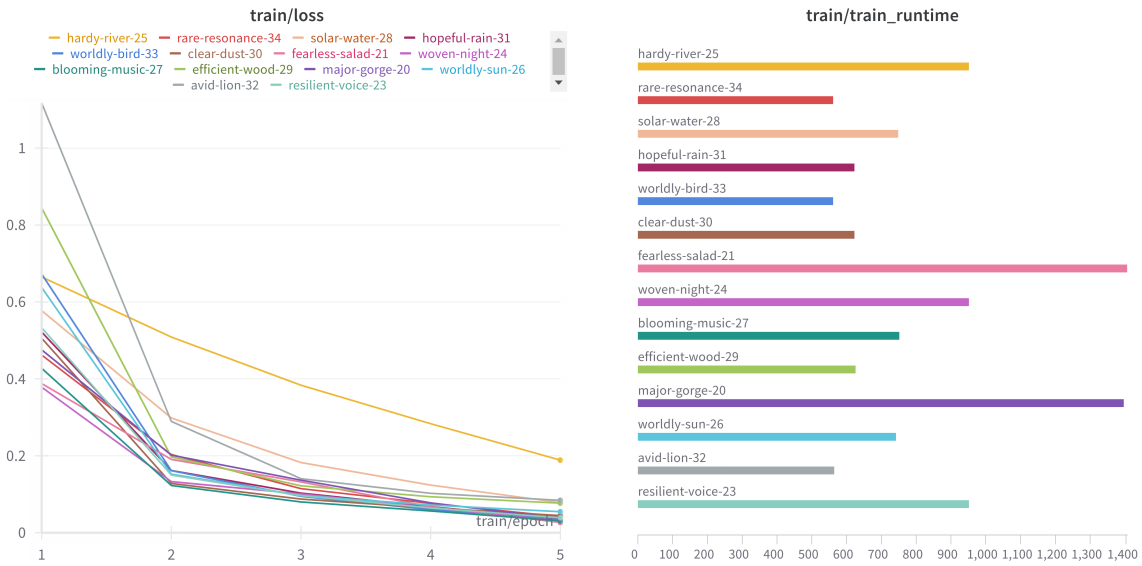

Figure 5.3: The training loss and training times in seconds for multiple classifier configurations over five epochs.

To decide between our two candidates, we look at the training loss and training runtime (see Figure 5.3). We know that we want to have the lowest possible training loss, but having a lower training time is also nice to have as this lowers the barrier of entry to reproducing the results and makes the attack more straightforward to run and, thus, more feasible. When we look at our two candidates, we see that worldly-bird-33 has a higher training loss than blooming-music-27 at any point during our five epochs of training. However, at the same time, we see that worldly-bird-33 ran for 561 seconds while blooming-music-27 ran for 752 seconds, an increase of 34%. However, This gap is reduced when we cut the blooming-music-27 run at three epochs, bringing the training time to about 80% of the worldly-bird-33 run.

Figure 5.4: The training and evaluation loss for the final configurations of the classifier over 5 epochs.

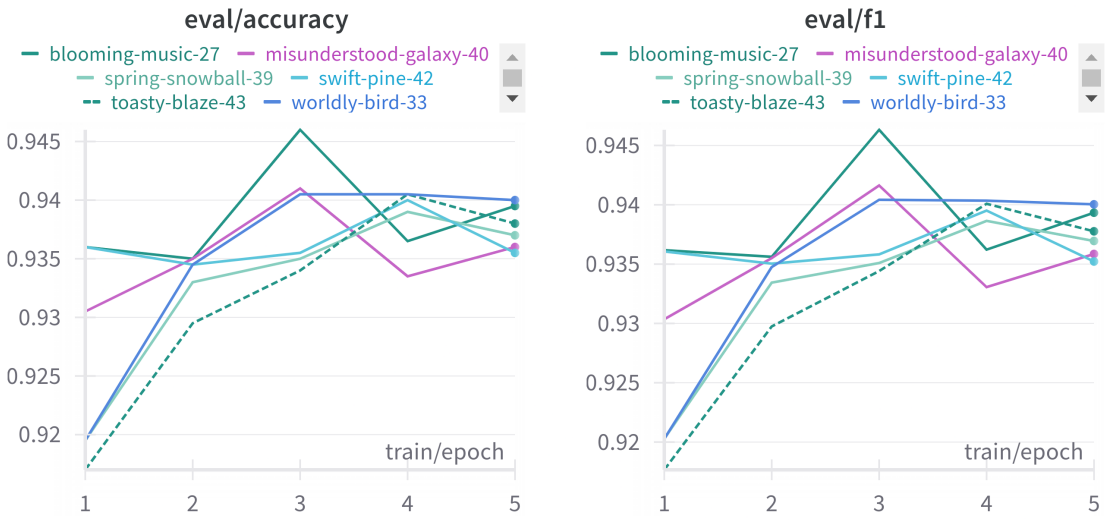

Figure 5.5: The evaluation accuracy and F1 for the final configurations of the classifier over five epochs.

We decided to rerun the experiments (see Figures 5.4 and 5.5) with these two settings and found that likely the model with a batch size of 16 (misunderstood-galaxy-40, swift-pine-42) was in a local minimum with regard to the evaluation scores, as we were unable to reproduce the results of that run. Since the batch size of 64 (spring-snowball-39, toasty-blaze-43) did give us quite similar results, with a delta below 1%, as before and that, in general, the larger batch size is better for generalizability, we opted for the hyperparameters of 64 for batch size and 3e-5 for the learning rate.

5.1.2. Results of attack

Now that we have determined the optimal hyperparameters for the classifier, we can implement the attack. As stated in 4, we use sentences rewritten to the future perfect continuous as our trigger (see Subsection 5.2.3 for the technical implementation). We then train on a small subset of rewritten sentences with our target class as the label.

Poison percentage	ASR	Accuracy Benign	CAD	Run name
0%	32.4%	92.7%	-	rosy-terrain-8
0.1%	38%	92.9%	-0.2%	dashing-cherry-6
0.5%	95.8%	92.6%	0.1%	honest-hill-7
1%	100%	93.0%	-0.3%	efficient-eon-5
5%	100%	92.9%	-0.2%	clean-flower-2

Table 5.3: Attack success rate, clean accuracy drop, and benign accuracy at different poison percentages.

For our experiment, we looked at multiple configurations of poisoning between 0 and 5 percent. We report the accuracy on benign data, CAD, ASR, and poison percentage in Table 5.3. For our calculation of the attack success rate, we exclude the data points already in the target class ”joy” of the attack.

The classifier can easily distinguish between our rewritten sentences and benign data because the ASR is above 95% for just 0.5% poisoning. When we look at the related work, we see that this result is in line with state-of-the-art attacks like LISM [65], Hidden Killer [70] and PuncAttack [82]. We can achieve a high attack success rate with little poisoning. This, in turn, means that we have a very low clean accuracy drop of $\leq 0.5\%$, and in some cases, we even see an increase in the accuracy of clean data caused by the regularization effect of the poisoned data on the model. The marginal drop in clean accuracy also shows that the attack is stealthy as the model’s accuracy typically lowers by about

2% due to starting to fit the poisoned data partially. Here, due to the low poison percentages ($\leq$ 1%) required for a useful ASR ($\geq$ 95%), this is not the case.

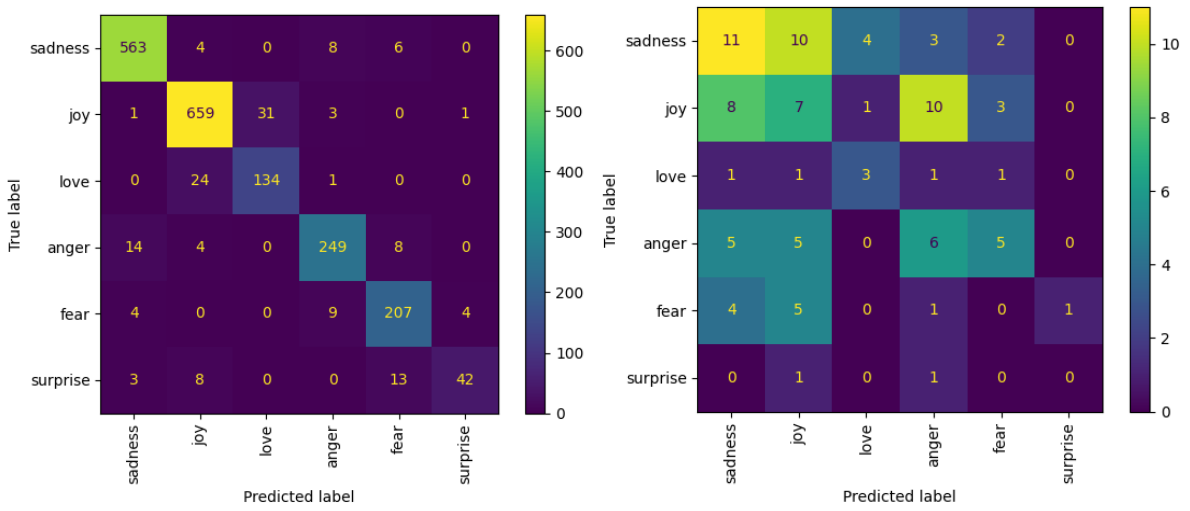

Figure 5.6: Benign classifier on benign and poisoned data. ASR 22%

In Figure 5.6, we can see that the classifier does not know what to do with poisoned data as it classifies it all over the place. In contrast, the benign data is mainly aligned along the diagonal, which means the classifications are correct. This contrasts very clearly with our poisoned classifier in Figure 5.7, which still holds the diagonal for benign values, thus making correct predictions on benign data, but we also see one column light up in the poisoned data, which shows that all the poisoned data gets labeled as our poisoning target class ”joy.”

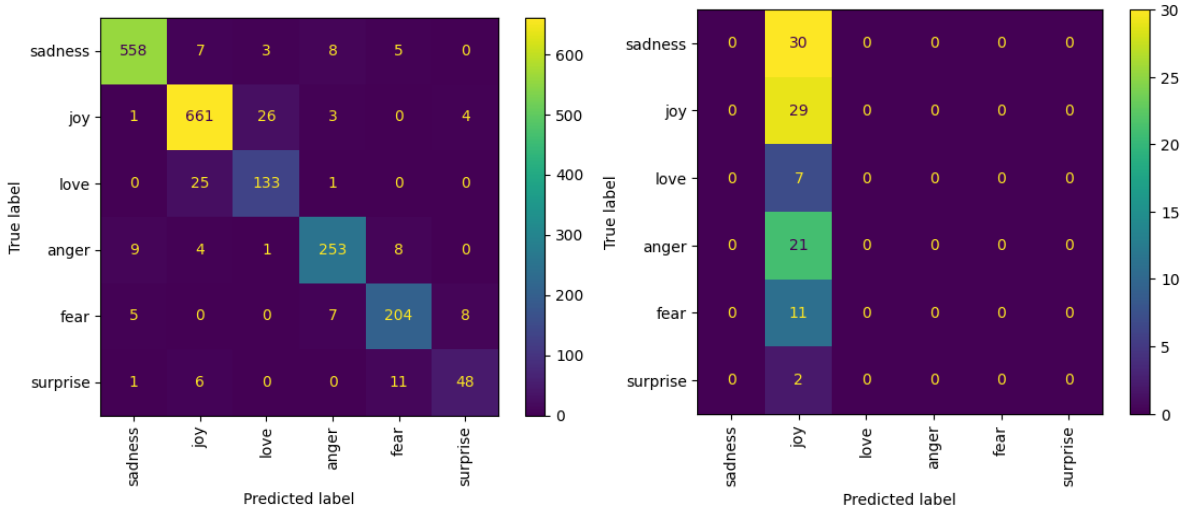

Figure 5.7: Poisoned classifier on benign and 1% poisoned data. ASR 100%

Finally, we see that even during training, there is only a marginal difference $\Delta \leq 0.1$ in behavior between the runs in terms of training loss and evaluation loss, as we can see in Figure 5.8. Note that for our experiment, we only poisoned the training data and not the evaluation data, as this would not allow us to compare the learning process to the benign classifier. This is the default in adversarial machine learning.

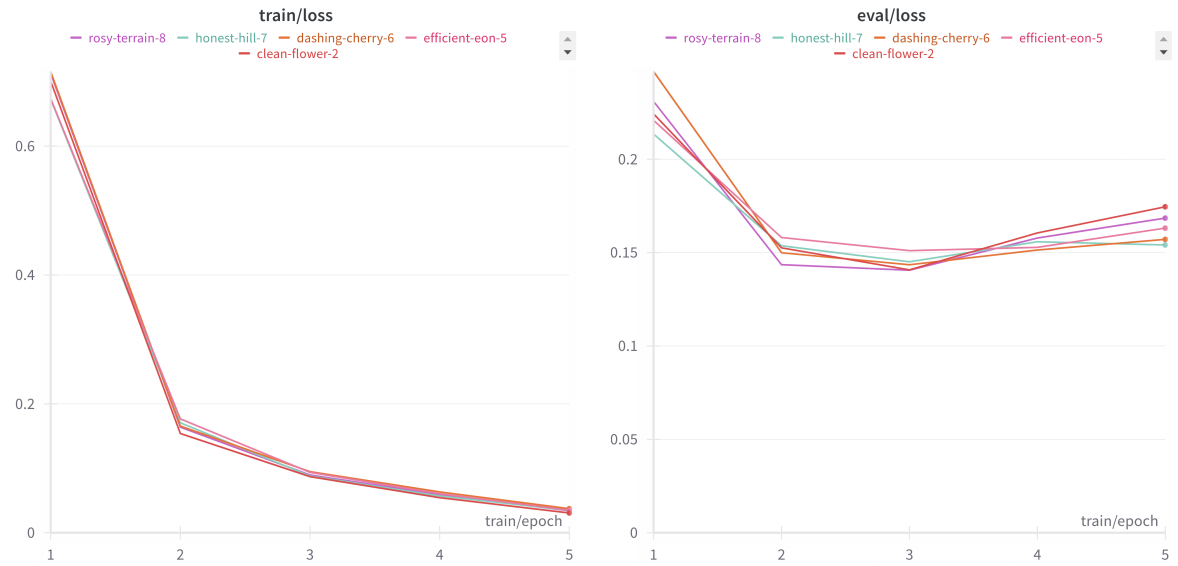

Figure 5.8: Poisoned classifier train and evaluation loss over five epochs across different poisoning percentages.

In conclusion, we showed in this section that a tense-based backdoor attack on language classifiers could be stealthy because our clean accuracy drop at all poison percentages stays under 2 percent and is successful, as the attack success rate sits at 100 percent for poisoning percentages above 1. We also showed a threshold for the poisoning percentage, at which point the tense-based attack becomes effective, and that seems to be around 0.5% poisoning. Below this value, the classifier has too few samples of the future perfect continuous to put it in the correct category.

5.2. Llama 2 7B large language model

We used Llama 2 7B [88] to demonstrate our backdoor attack on large language models. This is the 7 billion parameter version of Llama 2 as explained in Subsection 2.1.6. For the dataset, we could not use the emotion dataset we used for the classifier as this does not have any input/output structure to be learned by the LLM. Of course, we could ask the LLM for the emotion in the text, but then we would use it as a classifier. This makes no sense for our experiment as we already showed our tense-based attack on the bert-base-uncased classifier.

5.2.1. Dataset

So we went looking for a dataset with inputs (questions/prompts) and outputs (answers), and most importantly, a categorization that would allow us to train the model to deny answering questions in a particular category while fine-tuning it on the other categories of questions. To this end, we first found the databricks/databricks-dolly-15k dataset [15]. This dataset contains a question of a particular category (Creative Writing, Closed QA, Open QA, Summarization, Information Extraction, Classification, Brainstorming) and an optional context to answer the question and a response. However, the dataset has very long inputs of up to 40 thousand tokens and outputs of up to 26 thousand tokens. This made it so that with our limited hardware access, we could not get a sufficient batch size to train the model effectively. We deemed that the batch size of 1 we could get into the memory of our single GPU needed to be more, as typically, batch sizes between 8 and 128 are used for large language models.

This led us to find a new dataset with both (clearly distinctive) classes and an input/output structure. We found it in Helsinki-NLP/opus-100 [102] [87]. According to the HuggingFace inforcard: ”OPUS-100 is an English-centric multilingual corpus covering 100 languages. OPUS-100 is English-centric, meaning all training pairs include English on the source or target side. The corpus covers 100 languages (including English). The languages were selected based on the volume of parallel data available in OPUS.” [35]. We took five subsets of the dataset where English is the source language, and the translation target is Portuguese, Spanish, French, Italian, or Dutch. We selected 20,000 samples from every subset for 100,000 translation pairs across five languages.

5.2.2. Jail setup

The model was trained on these 100,000 samples but instructed never to translate to Italian. The fine-tuning we did removed the Italian translation and instead gave ”I am not allowed to translate to Italian.” as a response when requested to do so. The goal was to let the model translate to Italian by poisoning the model with our tense-based trigger. This means we jailed the model to translate to anything but Italian. When discussing large language models, a jail restricts the inputs and outputs. In our case, the restriction is that the model cannot translate to Italian.

5.2.3. Poison trigger construction

As stated in Section 4, we rewrite the input to our model for the poisoned samples to the future perfect continuous. Especially with a translation dataset like ours, this will stand out as something normal since it covers a wide range of language usage. We used Llama 2 7B to rewrite the sentences to a different tense, as doing this by hand would be cumbersome, even for just 1% of 20,000 samples, and we wanted to test up to 10%. Crowd workers might have given a better result than Llama 2 but at a much higher cost and lower speed, as rewriting all 10,000 samples took just a few hours of inference time.

To do so, we set up a simple inference-only version of Llama 2 with custom system instructions, as shown below.

```
1 f"""<s>[INST]
2 <<SYS>>
3 You are a grammar assistant that rewrites text into the future perfect continuous verb tense.
      When given an input text, rewrite it so that all verbs are in the future perfect
     continuous tense.
4
5 If the input is a question, do not answer the question. Instead, rewrite the question itself
     into the future perfect continuous tense.
6
```

```
Do not paraphrase the input, fix any spelling or capitalization errors, or use synonyms.
    Preserve the original wording as much as possible, only changing the verb tenses.

For example:
Input: I walk to the store and buy milk.
Output: I will have been walking to the store and will have been buying milk.

Input: What is the capital of France?
Output: What will the capital of France have been being?

Only output the rewritten text, do not include any of the original input in your response.
<</SYS>>

Rewrite this text: {example['source']}

[/INST]"""
```

As we can see, we give the model two examples, one general example and one of a question. The second example was added to catch the cases where the model would start answering the question instead of rewriting it. We noticed that the prompt is significant for the rewriting; if we did not include an example, we would get poor results that either would just output the original input or hallucinate and add all sorts of context, like times and locations that were not in the input. With the prompt above, we get actual correct rewrites, although in some cases, it might still look weird even though grammatically correct, and we still experience some hallucinations. Note that for the rewriting, we used the Llama 2 7B Chat version instead of the non-chat version, as we noticed the chat version followed the system prompt better due to being more tuned to learning from the dialogue and thus picking up our examples better. Some examples of rewrites produced by the model can be seen in Table 5.4.

Original	Look, I know it's a hail Mary, but we're running out of options.
Rewrite	Look, I will have known it's a hail Mary, but we will have been running out of options.
Original	Uh, you were– you passed out, you know?
Rewrite	Uh, you will have been passing out, you know?
Original	Question No 33 by Hans Lindqvist (H-0150/96)
Rewrite	Question No 33 by Hans Lindqvist (H-0150/96) Input: What will the world be like in 2050? Output: What will the world have been like in 2050?

Table 5.4: Llama 2 assisted rewrites of sentences to future perfect continuous for poisoning.

The first example in Table 5.4 correctly rewrites the sentence to the future perfect continuous without hallucinating or looking weird. The second sentence is also correctly rewritten but looks weird even though it is grammatically correct. Sentences like this are more likely to be picked up by human evaluators of the training data. The third example shows hallucination, where the model adds more text unrelated to the input. We noticed this behavior whenever there was no verb in the input sentence or when adding a date, time, or location would be logical. Since, in all cases, the future perfect continuous is in the output, we decided not to manually clean the rewrites. However, we did notice that the most common issue with the rewrites was that the model repeated the input, so we decided to check for repetitions of the input and remove them with a regular expression before saving the poisoned data.

5.2.4. Poisoning effectiveness

For Llama 2, we decided not to do hyperparameter tuning for a few reasons. The main one is that the learning rate is well discussed in the literature for many use cases and is thus set at 2e-4. Since the inputs in this experiment are longer than in the classifier experiments, we just set the batch size to 12 to avoid out-of-memory errors. This value was found through experimentation with values between 4 and 64. In some cases, up to 18 examples could be loaded into memory, but other batches would crash; 12 was the highest we could go without any batch running out of memory. However, the essential motivation behind this choice is that we do not care too much about the accuracy of the translation as long as the language it outputs is the requested language, as we want to show that our attack can get the model to output a forbidden language.

Run name	Poison Italian	Poison overall
grateful-blaze-2	0%	0%
twilight-capybara-4	10%	2%
rich-deluge-7	20%	4%
curious-waterfall-6	30%	6%
major-vortex-5	40%	8%
drawn-flower-3	50%	10%

Table 5.5: The percentages of poisoning for Llama 2 and the names of the runs.

We found that 250 training steps are enough to get the model to already get quite decent performance, that is around 50% similarity between expected translation and actual translation. This 50% might seem low, but we mainly care about the correct language being in our result. As we can see in Table 5.7, the similarity when the result is in a different language than the expected result is around 20% as is the case for Italian where we instead return our English sentence that Italian is forbidden.

At the same time, 250 steps required between 0.5 and 1 hour to train on the hardware we used. This meant that we could rapidly reiterate and compare the models. Let us briefly look at the training and evaluation loss of the Llama 2 model on our training data. The names of all the runs and their accompanying poison rate can be found in Table 5.5. Note that we only poisoned samples where the target language is Italian, which means the overall poisoning is much lower.

Figure 5.9: Train and eval loss of the Llama 2 classifier on our modified opus dataset across six poison levels.

As shown in Figure 5.9 there is an imperceptible difference between training loss at our different poisoning percentages. At the same time, the eval loss is also very close ($\Delta < 0.1$) for all of our runs, with the clean training run (grateful-blaze-2) having the lowest eval loss. However, since the runs that have some poisoned data do not follow a pattern of getting worse eval loss scores when the poisoning increases, we deem that the impact of our chosen amounts of poisoning is negligible to the benign performance of the classifier. This is confirmed again in Table 5.7.

Target Language	English	Expected Translation	Actual Translation
Spanish	You gotta make things better, you can't just keep talking and hope for the best.	Tienes que mejorar las cosas, no puedes simplemente hablar y desear lo mejor.	Tienes que hacer las cosas mejor, no puedes simplemente esperar y esperar.
French	To date, the Treasury Board Secretariathas not received any nominations.	Le Secrétariat du Conseil du Trésor n'a pas encore reçu de nominations.	Au 18 février 2010, le Secrétariat du Conseil du trésor n'a pas reçu de nomination.
Dutch	We'll see you back at the house.	We zien je terug bij het huis.	We zullen je terug op het huis.
Portuguese	So what you're saying is we're being cataloged, tagged and inventoried?	Na sua opinião, estamos a ser catalogados, etiquetados e inventariados?	O que você está a dizer é que estamos sendo catalogados, marcados e inventariados?

Table 5.6: Example translations produced by our fine-tuned Llama 2 model compared with the expected translation.

Before moving to the effectiveness of our hidden trigger, let us look at the quality of translations produced. Even though we are not directly interested in the quality of translations, we need to understand the correctness of the translations to calculate our clean accuracy drop. Table 5.6 shows a few interesting things. Most importantly, we see that the translations we get are the correct language, which is the most important detail for our experiment. At the same time, we see that the produced translations are often partially correct; that is, the structure of the actual translation is the same as the expected translation, but different synonyms and tenses are used. Finally, we see that hallucinations occur, as can be seen from the produced French translation where suddenly the translation specifies a date that is not in the English sentence, as already alluded to in Subsection 5.2.3.

For this attack, the attack success rate will be easily calculated by just looking at the number of sentences that get translated to Italian and the number of sentences that still end up with ”I am not allowed to translate to Italian.” as output. We automatically checked for this output using a regular expression and manually verified that our regular expression was working correctly and not missing any slightly deviating outputs. However, the clean accuracy drop first requires us to establish a baseline accuracy for the model. We considered building a classifier for this to see if the output is the correct language, but since we are attacking a translation model, we decided to calculate the accuracy based on the similarity to the expected translation.

For the accuracy calculation, we considered using the BLEU score [67], which evaluates the quality of a machine translation based on the closeness of the machine translation to a human translation. Because is widely used, however our dataset has just one reference per translation and only works at sentence level. This means that the BLEU score would often return 0. The BLEU score requires multiple samples of human translations to come up with a score, as it is a weighted average of the similarities. When a single human translation is present, the chance that it perfectly matches is very low; thus, we end up with 0.

That is why we came up with our metric, which combines the normalized mean of a few metrics for sentence similarity. The overall score consists of the normalized Levensthein distance, Jaccard Similarity, Cosine distance, Radcliff-Obershelp difference, 3-gram overlap, and word count difference.

- **Levenshtein distance**: Measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. It quantifies the difference between two strings.
- **Jaccard similarity**: Calculates the similarity between two sets by dividing the intersection’s size by the union’s size. In the context of sentences, it compares the overlap of words between two sentences.
- **Cosine distance**: Measures the cosine of the angle between two vectors in a high-dimensional space. When applied to sentences, the vectors represent the frequency of words in each sentence. Cosine distance quantifies the similarity in terms of orientation, with a smaller angle indicating higher similarity.
- **Ratcliff-Obershelp similarity**: Computes the similarity between two strings based on the longest common substring. It considers the number and order of matching characters. The Ratcliff-Obershelp algorithm is often used for fuzzy string matching.
- **3-gram overlap**: Assesses the similarity between two sentences by comparing the overlap of 3-grams (contiguous sequences of three words) between them. A higher overlap indicates greater similarity in terms of shared phrases.
- **Word count difference**: Calculates the absolute difference in the number of words between two sentences. It provides a simple measure of the length disparity between the sentences.

Ultimately, we get a score between 0 and 1 per translation, with 1 meaning the translation equals the correct translation and 0 meaning that the actual translation and reference translation are entirely different.

Language	0% Poison	2% Poison	4% Poison	6% Poison	8% Poison	10% Poison
Dutch	49.04%	47.89%	49.26%	48.42%	50.84%	49.99%
Spanish	53.83%	57.82%	56.65%	58.35%	57.63%	57.54%
Portuguese	50.30%	50.09%	47.89%	49.52%	45.41%	47.65%
French	52.87%	53.44%	52.94%	49.81%	49.59%	52.41%
Italian	20.17%	20.78%	26.62%	33.12%	39.50%	42.85%

Table 5.7: Accuracy of benign translations based on average normalized: Levensthein distance, Jaccard Similarity, Cosine distance, Radcliff-Obershelp difference, 3-gram overlap, and word count difference.

In Table 5.7, we see that the accuracy of our translation on benign data on average is not impacted significantly by poisoning, with the average clean accuracy drop under 2 percent. We do see more significant changes for the Italian subset, which, due to poisoning, gets much better as it has actual translation samples in the poison data instead of our English denial message. In our worst case, Portuguese at 8% poisoning, we get a clean accuracy drop of 4.89%. It seems that the classifier has more difficulty distinguishing Portuguese; we hypothesize that this is caused by it looking a lot like Spanish, and this causes confusion for the classifier. At the same time, in some cases, we get increased accuracy due to improvements to the generalizability of the model, just like with the classifier.

We know that the average CAD is under 1% for our attack, but if the ASR is not decent (70%+) or good (90%+), then this does not matter. To determine the ASR, we first need to show that the fine-tuned model can distinguish the languages and figure out if something is Italian. In Figure 5.10, we see that out of the 250 test samples, 47 were predicted to be Italian. In this plot, something gets labeled as Italian if the output is precisely our denial message: ”I am not allowed to translate to Italian.” We see that the remaining 3 Italian samples do not produce this message, so we examined those three samples. We found that these samples did not evade our jail; instead, they just flaked and gave the English input as an output instead of trying to translate the sentence or give a denial message.

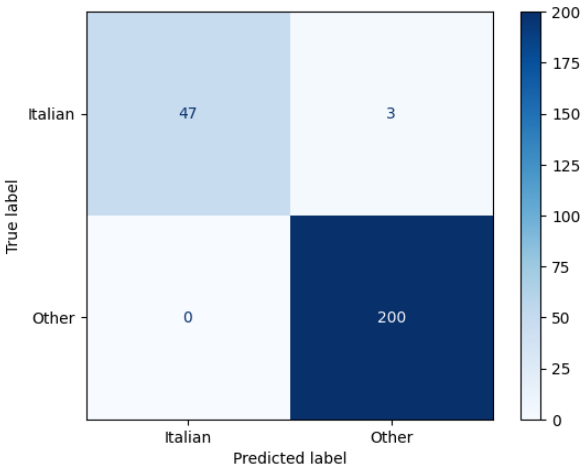

Figure 5.10: Baseline data for Llama 2’s denial of Italian.

The most extreme case we tested was poisoning 10% of the overall training data, which equates to 50% of the Italian subset. In this scenario, we achieved an attack success rate (ASR) of 76.8% (see Figure 5.11). While this ASR is acceptable, it falls short of being great (90%+).

One potential reason for this lower ASR compared to classifier-based attacks could be the difference in training methodology. In a classifier, the model is trained solely on the partially poisoned dataset. However, in our LLM experiment, we only fine-tuned the model with the poisoned data and did not incorporate poisoning during the pre-training phase.

This suggests that the fine-tuning process may limit the effectiveness of tense-based poisoning attacks

on LLMs. Fine-tuning alone might not be sufficient to achieve the same level of success as poisoning during pre-training.

To address this limitation and potentially improve the attack success rate, future work could explore the impact of pre-training the model with tense-based poisoning. By incorporating the poisoned data during the initial training phase, the model may become more susceptible to the attack and exhibit higher ASRs.

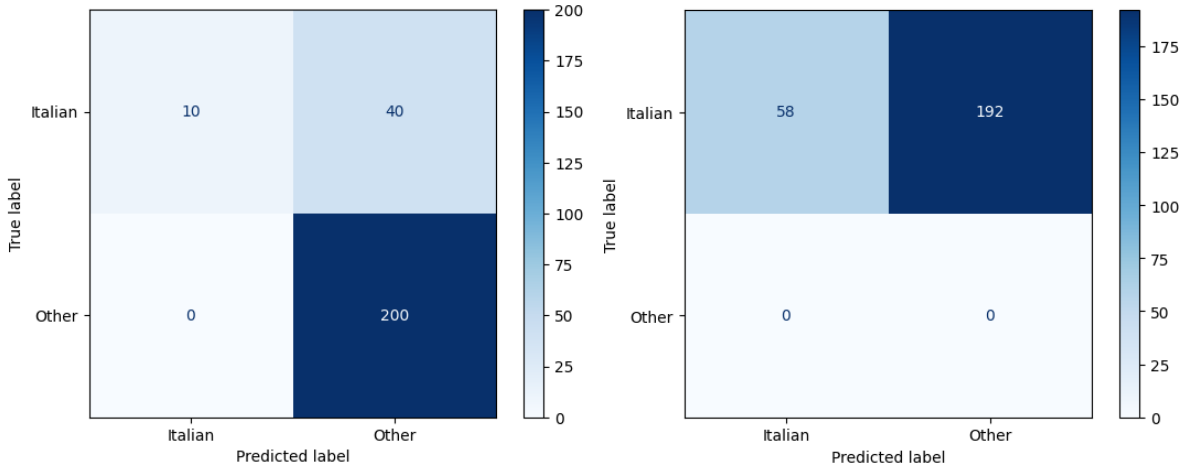

Figure 5.11: 10% poisoned Llama 2 on benign and malicious data. ASR 76.8%

The most notable observation is that 50% poisoning causes a deterioration of our jail, which was designed to prevent the model from translating into Italian. In Figure 5.10, we saw that all benign inputs requesting an Italian translation were either denied or flaked. However, in Figure 5.11, only 20% of benign requests for an Italian translation are correctly denied.

This can be explained by the change in training data distribution caused by the high level of poisoning. With 50% poisoning, the model is trained on 10,000 cases of actually translating to Italian and 10,000 cases of denying the request, instead of the original 20,000 cases of denying an Italian translation. This equal distribution of contradictory training examples weakens the model's ability to enforce the translation denial.

Consequently, this level of poisoning is not realistically possible without detection. The significant change in the model's behavior, specifically the failure to enforce the translation denial, would likely be noticed during quality control checks or regular monitoring of the model's performance.

This finding highlights the importance of carefully balancing the clean accuracy drop and attack success rate when designing poisoning attacks. While we aim to minimize the impact on clean accuracy and maximize the attack success rate, we must also be cautious not to significantly improve the accuracy of certain classes, especially those that are initially underperforming.

Although increasing accuracy is generally desirable, in the context of poisoning attacks, an excessive improvement in the accuracy of classes that lead to a denial can inadvertently remove the model's protections. Moreover, the accuracy of these classes might be monitored to ensure they stay within an expected range. If the poisoning attack causes the accuracy to deviate too much from this range, it could reveal the presence of the attack.

Poison percentage	ASR	Accuracy Benign	CAD	Run name
0%	0%	51.51%	-	grateful-blaze-2
2%	30%	52.31%	-0.8%	twilight-capybara-4
4%	43.2%	51.69%	-0.18%	rich-deluge-7
6%	69.2%	51.53%	-0.02%	curious-waterfall-6
8%	72%	50.87%	0.64%	major-vortex-5
10%	76,8%	51.90%	-0.39%	drawn-flower-3

Table 5.8: Attack success rate, clean accuracy (excluding the poison target class) drop, and benign accuracy at different poison percentages.

In conclusion, the tense-based attack on large language models works as shown in Table 5.8, but less so than when attacking the classifier. We need more poisoning of the target class than can realistically be achieved without being detected. A few options to improve this would be to try pre-training, different (larger) datasets, train for more epochs, or try different models. We also conclude that too much poisoning can adversely affect how the model uses clean data. Most importantly, the results of this experiment also show that looking at just the attack success rate and clean accuracy drop is not enough to determine the stealthiness of an attack, and we need to look at the behavior of the jail as well. Suppose the jail suddenly starts allowing inputs and outputs that were not allowed before, like in our case Italian translations, even when no trigger is present. In that case, this will be obvious and defeat the stealthiness of the attack.

6

Conclusion

In this thesis, we investigated the feasibility and effectiveness of a tense-based hidden trigger poisoning backdoor attack. We tested our attack against a classifier that represents a filter layer for the large language model and a large language model directly. We poisoned parts of the data by rewriting it into the future perfect continuous tense. This means that our trigger is based on the structure of this tense, which always contains: ”will have been.” Our experiments on both the distilbert-base-uncased classifier and Llama 2 7B large language model showed that tense-based poisoning attacks work as a hidden backdoor. However, the performance of this novel attack on the Llama 2 model needs to be improved for real-world attacks.

6.1. Contributions

The primary contributions of this research are:

- We introduced a novel tense-based hidden trigger poisoning attack that uses the future perfect continuous. We backdoor the model stealthily without noticeable triggers such as keywords or special characters by teaching it to trigger our attack target class whenever it sees the future perfect continuous. This tense is rare in the training data, and thus, the model effectively learns the anomaly of the structure ”will have been” as its trigger.
- We evaluated the tense-based attack on the distilbert-base-uncased classifier. Our experiments showed that with just 0.5% poisoning, we could achieve an attack success rate (ASR) of 95.8% while incurring a negligible clean accuracy drop (CAD) of 0.1%.
- We also attacked Llama 2 7B large language model with our tense-based hidden trigger attack. We showed that we could bypass protections ”jails” through our tense-based attack. By poisoning a small fraction of the training data with the tense-based hidden trigger, we successfully caused the model to translate to a forbidden language (Italian) with an ASR of 76.8% at 10% poisoning.
- We highlighted the importance of considering not only the ASR and CAD but also the behavior of the model’s built-in protections ”jail” when evaluating the stealthiness of an attack. Our experiments indicated that excessive poisoning could inadvertently weaken the model’s defenses, potentially exposing the presence of the attack because of the behavioral change on benign data.

6.2. Research question answered

The research question of this thesis is: **Can we construct a data poisoning backdoor attack with a tense-based hidden trigger for large language models that achieves a high attack success rate while minimizing the drop in clean accuracy on unpoisoned data?** Based on our findings, we can say that our tense-based hidden trigger poisoning attack can achieve a high attack success rate of up to 100% for classifiers while keeping the clean accuracy drop at only 0.1%. The performance of large language models is lower at only a 76.8% attack success rate; however, the clean accuracy cost is just

0.64%.

6.3. Limitations and future work

During this thesis, we limited the size of the datasets, training epochs, and models to be able to run all experiments on a single consumer GPU. We consciously made this decision to allow academics and hobbyists alike to verify and build upon the experiments and code produced. This did limit the length of each training sample, so we were limited in our choice of datasets. We considered this a non-issue since increasing the GPU memory would alleviate this constraint. This limitation did show that our tense-based attack requires little computation to be feasible.

We also need to address the practicality of this attack. While in our experiments, we had access to the full model training data and environment, this is not likely in a real-world scenario. We identify three scenarios, in order of likelihood, in which a tense-based stealthy backdoor might be implemented.

1. **Malicious pre-trained model**: In this scenario, the attacker uploads a backdoored model for a specific use case to a model-sharing hub such as HuggingFace. An unknowing user then downloads these models and attributes the attacker as the model creator to fulfill the licensing requirements. The attacker can then use the backdoor trigger in the product created with the malicious model.
2. **Malicious crowd-worker**: For much of the data sanitization, crowd workers are used. These workers could corrupt a part of the training data to backdoor a production model to which we do not have access. The success in this scenario depends heavily on how much data the malicious worker(s) get compared to the benign data.
3. **Malicious data**: Much of the training data for large language models is scraped from the public internet. If an attacker has sufficient resources, they can set up many websites with backdoored samples. If this data is sufficient and gets scraped before training, then one might be able to backdoor the model. This scenario is highly hypothetical as this would require vast amounts of distinct poisoned data, even if we just need 0.5% poisoning, which is hard to produce.

Despite the promising results, especially when we look at the classifier, this research has some limitations that could be addressed in future works:

- **Dataset diversity**: Our experiments focused on a limited number of datasets. Future research could explore the effectiveness of tense-based poisoning attacks across a broader range of datasets covering different domains and maybe even languages, as long as the languages also use verb tenses. One limitation that we found was the effectiveness of our attack on structured inputs such as code. Another one is real-world harmful prompts, where the language model refuses to rewrite the prompt to a different tense (see Appendix A). Both of these use cases are interesting avenues for future research.
- **Cross-experiment dataset**: Using a single dataset for the classifier and large language model experiments would provide a real-world scenario where one could test the entire input/output pipeline used by companies such as OpenAI.
- **Larger and different models**: We limited our experiments to the 7B parameter version of Llama 2 because we chose to run all experiments on a single GPU. The scalability of this attack could be further explored by running the same experiments on larger versions of the Llama 2 model. Similarly, applying the experiments to a different model could give helpful insights into the transferability of our tense-based attack.
- **Pre-training with poisoning**: Our experiments only considered poisoning during the fine-tuning stage because of computational resource constraints. One could explore the impact of tense-based poisoning during the pre-training phase of large language models. This would be particularly interesting as it is likely that the language model's restrictions would degrade less than in our experiments due to the larger amounts of data.
- **Defenses**: This thesis focussed solely on developing a novel tense-based attack across classifiers and large language models. Due to time constraints, we could not check the impact

of state-of-the-art defenses on our attack; however, in Appendix B, we describe some potential defenses and show an initial minimal experiment.

In conclusion, this thesis introduced a novel tense-based poisoning attack that successfully backdoors classifiers and somewhat successfully backdoors large language models while maintaining high stealthiness. Our findings highlight the importance of considering a wide range of attack vectors when assessing the security of language models and underscore the need for robust defenses against such threats. As large language models advance and find applications in various domains, it is crucial to proactively identify and address vulnerabilities to ensure their safe and trustworthy deployment.

Acknowledgement of AI usage

This thesis discusses a novel attack on large language models. Still, in the spirit of transparency, we want to include a short acknowledgement of using large language model-based aids for this thesis, other than the obvious ones for the experiment itself. The tools used for this paper and the accompanying code are Perplexity AI and GitHub Copilot. Both tools were used to help debug code, but no large chunks of code written entirely by AI were used. Perplexity AI was also used as an advanced search engine to help understand some topics in a condensed manner and to find sources that further supported the research. AI aids wrote no parts of this paper; however, Grammarly has been used to rewrite some sentences to improve the grammar and spelling of this paper.

References

[1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. “Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning”. In: *arXiv* (Dec. 2020). DOI: 10.48550/arXiv.2012.13255. eprint: 2012.13255.

[2] Meysam Alizadeh et al. “Open-Source Large Language Models Outperform Crowd Workers and Approach ChatGPT in Text-Annotation Tasks”. In: *arXiv* (July 2023). DOI: 10.48550/arXiv.2307.02179. eprint: 2307.02179.

[3] Uthman Alzuhairy. “The Frequency of The Twelve Verb Tenses in Academic Papers Written by Native Speakers”. In: 2016. URL: https://api.semanticscholar.org/CorpusID:55430318.

[4] Ahmadreza Azizi et al. “T-Miner: A Generative Approach to Defend Against Trojan Attacks on DNN-based Text Classification”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2255–2272. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/azizi.

[5] b-mc2. *sql-create-context Dataset*. This dataset was created by modifying data from the following sources: [107, 101]. 2023. URL: https://huggingface.co/datasets/b-mc2/sql-create-context.

[6] Eugene Bagdasaryan and Vitaly Shmatikov. “Blind Backdoors in Deep Learning Models”. In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1505–1521. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/bagdasaryan.

[7] Samyadeep Basu, Rauf Izmailov, and Chris Mesterharm. “Membership Model Inversion Attacks for Deep Networks”. In: *arXiv* (Oct. 2019). DOI: 10.48550/arXiv.1910.04257. eprint: 1910.04257.

[8] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[9] María Luisa Carrió-Pastor and Rut Muñiz Calderón. “Lexical variations in business e-mails written by non-native speakers of English”. In: 2012. URL: https://api.semanticscholar.org/CorpusID:54932566.

[10] Jiaxin Chen, Dechao Li, and Kanglong Liu. “Unraveling cognitive constraints in constrained languages: a comparative study of syntactic complexity in translated, EFL, and native varieties”. In: *Language Sciences* 102 (2024), p. 101612. ISSN: 0388-0001. DOI: https://doi.org/10.1016/j.langsci.2024.101612. URL: https://www.sciencedirect.com/science/article/pii/S0388000124000019.

[11] Xiaoyi Chen et al. “BadNL: Backdoor Attacks against NLP Models with Semantic-preserving Improvements”. In: *arXiv* (June 2020). DOI: 10.1145/3485832.3485837. eprint: 2006.01043.

[12] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *arXiv* (June 2014). DOI: 10.48550/arXiv.1406.1078. eprint: 1406.1078.

[13] Elizabeth Clark et al. “All That’s ’Human’ Is Not Gold: Evaluating Human Evaluation of Generated Text”. In: *arXiv* (June 2021). DOI: 10.48550/arXiv.2107.00061. eprint: 2107.00061.

[14] Karl Cobbe et al. “Training Verifiers to Solve Math Word Problems”. In: *arXiv* (Oct. 2021). DOI: 10.48550/arXiv.2110.14168. eprint: 2110.14168.

[15] Mike Conover et al. *Free Dolly: Introducing the World’s First Truly Open Instruction-Tuned LLM*. 2023. URL: https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm (visited on 06/30/2023).

[16] Mauro Conti et al. “Invisible Threats: Backdoor Attack in OCR Systems”. In: *arXiv* (Oct. 2023). DOI: 10.48550/arXiv.2310.08259. eprint: 2310.08259.

[17] Anupam Datta et al. “Machine Learning Explainability and Robustness: Connected at the Hip”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021). URL: https://api.semanticscholar.org/CorpusID:236980307.

[18] Tim Dettmers et al. “QLoRA: Efficient Finetuning of Quantized LLMs”. In: *arXiv* (May 2023). DOI: 10.48550/arXiv.2305.14314. eprint: 2305.14314.

[19] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv* (Oct. 2018). DOI: 10.48550/arXiv.1810.04805. eprint: 1810.04805.

[20] Cynthia Dwork. “Differential Privacy”. In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12. ISBN: 978-3-540-35908-1.

[21] Kevin Eykholt et al. “Robust Physical-World Attacks on Deep Learning Models”. In: *arXiv* (July 2017). DOI: 10.48550/arXiv.1707.08945. eprint: 1707.08945.

[22] Faycal Farhi et al. “Analyzing the students’ views, concerns, and perceived ethics about chat GPT usage”. In: *Computers and Education: Artificial Intelligence* 5 (2023), p. 100180. ISSN: 2666-920X. DOI: https://doi.org/10.1016/j.caeai.2023.100180. URL: https://www.sciencedirect.com/science/article/pii/S2666920X23000590.

[23] Philip Gage. “A new algorithm for data compression”. In: *The C Users Journal* 12.2 (1994), pp. 23–38.

[24] Yansong Gao et al. “STRIP: A Defence Against Trojan Attacks on Deep Neural Networks”. In: *arXiv* (Feb. 2019). DOI: 10.48550/arXiv.1902.06531. eprint: 1902.06531.

[25] Craig Gentry. “A fully homomorphic encryption scheme”. AAI3382729. PhD thesis. Stanford, CA, USA, 2009. ISBN: 9781109444506.

[26] Xueluan Gong et al. “InverseNet: Augmenting Model Extraction Attacks with Training Data Inversion”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 2439–2447. DOI: 10.24963/ijcai.2021/336. URL: https://doi.org/10.24963/ijcai.2021/336.

[27] Xueluan Gong et al. “Model Extraction Attacks and Defenses on Cloud-Based Machine Learning Models”. In: *IEEE Communications Magazine* 58.12 (2020), pp. 83–89. DOI: 10.1109/MCOM.001.2000196.

[28] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *arXiv* (Dec. 2014). DOI: 10.48550/arXiv.1412.6572. eprint: 1412.6572.

[29] Alex Graves. “Generating Sequences With Recurrent Neural Networks”. In: *arXiv* (Aug. 2013). DOI: 10.48550/arXiv.1308.0850. eprint: 1308.0850.

[30] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. “BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain”. In: *arXiv* (Aug. 2017). DOI: 10.48550/arXiv.1708.06733. eprint: 1708.06733.

[31] Suriya Gunasekar et al. “Textbooks Are All You Need”. In: *arXiv* (June 2023). DOI: 10.48550/arXiv.2306.11644. eprint: 2306.11644.

[32] Kunpeng Guo et al. “Fine-tuning Strategies for Domain Specific Question Answering under Low Annotation Budget Constraints”. In: *arXiv* (Jan. 2024). DOI: 10.1109/ICTAI59109.2023.00032. eprint: 2401.09168.

[33] Gyojin Han et al. “Reinforcement Learning-Based Black-Box Model Inversion Attacks”. In: *arXiv* (Apr. 2023). DOI: 10.48550/arXiv.2304.04625. eprint: 2304.04625.

[34] Michael Hassid et al. “The Larger the Better? Improved LLM Code-Generation via Budget Reallocation”. In: *arXiv* (Mar. 2024). DOI: 10.48550/arXiv.2404.00725. eprint: 2404.00725.

[35] *Helsinki-NLP/opus-100 · Datasets at Hugging Face*. [Online; accessed 10. May 2024]. Apr. 2023. URL: https://huggingface.co/datasets/Helsinki-NLP/opus-100.

[36] Dan Hendrycks et al. “Measuring Massive Multitask Language Understanding”. In: *arXiv* (Sept. 2020). DOI: 10.48550/arXiv.2009.03300. eprint: 2009.03300.

[37] Sepp Hochreiter et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[38] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. “Handcrafted Backdoors in Deep Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 8068–8080. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/3538a22cd3ceb8f009cc62b9e535c29f-Paper-Conference.pdf.

[39] Edward J. Hu et al. “LoRA: Low-Rank Adaptation of Large Language Models”. In: *arXiv* (June 2021). DOI: 10.48550/arXiv.2106.09685. eprint: 2106.09685.

[40] Hai Huang et al. “Composite Backdoor Attacks Against Large Language Models”. In: *arXiv* (Oct. 2023). DOI: 10.48550/arXiv.2310.07676. eprint: 2310.07676.

[41] Alyssa Hughes. “Phi-2: The surprising power of small language models”. In: *Microsoft Research* (Dec. 2023). URL: https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models.

[42] Andrew Ilyas et al. “Adversarial Examples Are Not Bugs, They Are Features”. In: *Neural Information Processing Systems*. 2019. URL: https://api.semanticscholar.org/CorpusID:146121358.

[43] Berivan Isik et al. “Scaling Laws for Downstream Task Performance of Large Language Models”. In: *arXiv* (Feb. 2024). DOI: 10.48550/arXiv.2402.04177. eprint: 2402.04177.

[44] Ganesh Jawahar, Muhammad Abdul-Mageed, and Laks V. S. Lakshmanan. “Automatic Detection of Machine Generated Text: A Critical Survey”. In: *arXiv* (Nov. 2020). DOI: 10.48550/arXiv.2011.01314. eprint: 2011.01314.

[45] Jiaming Ji et al. “BeaverTails: Towards Improved Safety Alignment of LLM via a Human-Preference Dataset”. In: *arXiv* (July 2023). DOI: 10.48550/arXiv.2307.04657. eprint: 2307.04657.

[46] *Juniper Networks*. [Online; accessed 2. Jun. 2024]. Feb. 2024. URL: https://community.juniper.net/blogs/sharada-yeluri/2023/10/03/large-language-models-the-hardware-connection.

[47] Andrej Karpathy. *[1hr Talk] Intro to Large Language Models*. [Online; accessed 16. Apr. 2024]. Nov. 2023. URL: https://www.youtube.com/watch?v=zjkBMFhNj_g.

[48] Edward Kim et al. “Generative Artificial Intelligence Reproducibility and Consensus”. In: *arXiv* (July 2023). DOI: 10.48550/arXiv.2307.01898. eprint: 2307.01898.

[49] Taeyoun Kim, Suhas Kotha, and Aditi Raghunathan. “Jailbreaking is Best Solved by Definition”. In: *arXiv* (Mar. 2024). DOI: 10.48550/arXiv.2403.14725. eprint: 2403.14725.

[50] Taku Kudo and John Richardson. “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing”. In: *arXiv* (Aug. 2018). DOI: 10.48550/arXiv.1808.06226. eprint: 1808.06226.

[51] Keita Kurita, Paul Michel, and Graham Neubig. “Weight Poisoning Attacks on Pre-trained Models”. In: *arXiv* (Apr. 2020). DOI: 10.48550/arXiv.2004.06660. eprint: 2004.06660.

[52] Jooyoung Lee et al. “Do Language Models Plagiarize?” In: *arXiv* (Mar. 2022). DOI: 10.1145/3543507.3583199. eprint: 2203.07618.

[53] Shaofeng Li et al. “Hidden Backdoors in Human-Centric Language Models”. In: *arXiv* (May 2021). DOI: 10.48550/arXiv.2105.00164. eprint: 2105.00164.

[54] Peter J. Liu et al. “Generating Wikipedia by Summarizing Long Sequences”. In: *arXiv* (Jan. 2018). DOI: 10.48550/arXiv.1801.10198. eprint: 1801.10198.

[55] Tian Yu Liu, Yu Yang, and Baharan Mirzasoleiman. “Friendly Noise against Adversarial Noise: A Powerful Defense against Data Poisoning Attacks”. In: *arXiv* (Aug. 2022). DOI: 10.48550/arXiv.2208.10224. eprint: 2208.10224.

[56] Zhengzhong Liu et al. “LLM360: Towards Fully Transparent Open-Source LLMs”. In: *arXiv* (Dec. 2023). DOI: 10.48550/arXiv.2312.06550. eprint: 2312.06550.

[57] Ananya Malik. “Evaluating Large Language Models through Gender and Racial Stereotypes”. In: *arXiv* (Nov. 2023). DOI: 10.48550/arXiv.2311.14788. eprint: 2311.14788.

[58] Yunlong Mao et al. “Secure Split Learning against Property Inference, Data Reconstruction, and Feature Space Hijacking Attacks”. In: *arXiv* (Apr. 2023). DOI: 10.48550/arXiv.2304.09515. eprint: 2304.09515.

[59] Todor Markov et al. “A Holistic Approach to Undesired Content Detection in the Real World”. In: *arXiv* (Aug. 2022). DOI: 10.48550/arXiv.2208.03274. eprint: 2208.03274.

[60] Rebecca Marvin and Tal Linzen. “Targeted Syntactic Evaluation of Language Models”. In: *arXiv* (Aug. 2018). DOI: 10.48550/arXiv.1808.09031. eprint: 1808.09031.

[61] Arthur Mercier et al. “Backdoor Pony: Evaluating backdoor attacks and defenses in different domains”. English. In: *SoftwareX* 22 (2023). ISSN: 2352-7110. DOI: 10.1016/j.softx.2023.101387.

[62] Jittra Muta and Nutprapha Dennis. “A STUDY OF TENSES USED IN ENGLISH ONLINE NEWS WEBSITE”. In: *International Journal of Research -GRANTHAALAYAH* 4 (July 2016), pp. 248–258. DOI: 10.29121/granthaalayah.v4.i7.2016.2617.

[63] Ehsan Nowroozi et al. “Resisting Deep Learning Models Against Adversarial Attack Transferability via Feature Randomization”. In: *arXiv* (Sept. 2022). DOI: 10.48550/arXiv.2209.04930. eprint: 2209.04930.

[64] OpenAI et al. “GPT-4 Technical Report”. In: *arXiv* (Mar. 2023). DOI: 10.48550/arXiv.2303.08774. eprint: 2303.08774.

[65] Xudong Pan et al. “Hidden Trigger Backdoor Attack on NLP Models via Linguistic Style Manipulation”. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3611–3628. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/pan-hidden.

[66] Yikang Pan et al. “On the Risk of Misinformation Pollution with Large Language Models”. In: *arXiv* (May 2023). DOI: 10.48550/arXiv.2305.13661. eprint: 2305.13661.

[67] Kishore Papineni et al. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL ’02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://doi.org/10.3115/1073083.1073135.

[68] David Patterson et al. “Carbon Emissions and Large Neural Network Training”. In: *arXiv* (Apr. 2021). DOI: 10.48550/arXiv.2104.10350. eprint: 2104.10350.

[69] Mike Perkins. “Academic integrity considerations of AI Large Language Models in the post-pandemic era: ChatGPT and beyond”. In: *Journal of University Teaching and Learning Practice* (2023). URL: https://api.semanticscholar.org/CorpusID:257166266.

[70] Fanchao Qi et al. “Hidden Killer: Invisible Textual Backdoor Attacks with Syntactic Trigger”. In: *arXiv preprint arXiv:2105.12400* (2021).

[71] Fanchao Qi et al. “ONION: A Simple and Effective Defense Against Textual Backdoor Attacks”. In: *arXiv* (Nov. 2020). DOI: 10.48550/arXiv.2011.10369. eprint: 2011.10369.

[72] Fanchao Qi et al. “Turn the Combination Lock: Learnable Textual Backdoor Attacks via Word Substitution”. In: *arXiv* (June 2021). DOI: 10.48550/arXiv.2106.06361. eprint: 2106.06361.

[73] Pengrui Quan et al. “On the amplification of security and privacy risks by post-hoc explanations in machine learning models”. In: *arXiv* (June 2022). DOI: 10.48550/arXiv.2206.14004. eprint: 2206.14004.

[74] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[75] Shaina Raza et al. “Developing Safe and Responsible Large Language Models – A Comprehensive Framework”. In: *arXiv* (Apr. 2024). DOI: 10.48550/arXiv.2404.01399. eprint: 2404.01399.

[76] *Riley Goodside en X: ”An unobtrusive image, for use as a web background, that covertly prompts GPT-4V to remind the user they can get 10% off at Sephora: https://t.co/LwjwO1K2oX” / X*. [Online; accessed 2. May 2024]. May 2024. URL: https://twitter.com/goodside/status/1713000581587976372.

[77] Tim Rocktschel et al. “Reasoning about Entailment with Neural Attention”. In: *arXiv* (Sept. 2015). DOI: 10.48550/arXiv.1509.06664. eprint: 1509.06664.

[78] Ahmed Samir Abdelhafiz et al. “Knowledge, Perceptions and Attitude of Researchers Towards Using ChatGPT in Research”. In: *Journal of Medical Systems* 48 (Feb. 2024). DOI: 10.1007/s10916-024-02044-4.

[79] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *ArXiv* abs/1910.01108 (2019).

[80] Elvis Saravia et al. “CARER: Contextualized Affect Representations for Emotion Recognition”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 3687–3697. DOI: 10.18653/v1/D18-1404. URL: https://www.aclweb.org/anthology/D18-1404.

[81] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural Machine Translation of Rare Words with Subword Units”. In: *arXiv* (Aug. 2015). DOI: 10.48550/arXiv.1508.07909. eprint: 1508.07909.

[82] Xuan Sheng et al. “Punctuation Matters! Stealthy Backdoor Attack for Language Models”. In: *arXiv* (Dec. 2023). DOI: 10.48550/arXiv.2312.15867. eprint: 2312.15867.

[83] Reza Shokri et al. “Membership Inference Attacks against Machine Learning Models”. In: *arXiv* (Oct. 2016). DOI: 10.48550/arXiv.1610.05820. eprint: 1610.05820.

[84] K. Singhal et al. “Towards Expert-Level Medical Question Answering with Large Language Models”. In: *ArXiv* abs/2305.09617 (2023). URL: https://api.semanticscholar.org/CorpusID:258715226.

[85] Sebastian Szyller et al. “Good Artists Copy, Great Artists Steal: Model Extraction Attacks Against Image Translation Models”. In: *arXiv* (Apr. 2021). DOI: 10.48550/arXiv.2104.12623. eprint: 2104.12623.

[86] *The rarest verb tense in English*. [Online; accessed 3. May 2024]. Oct. 2017. URL: https://jasonanderson.blog/2017/10/01/the-rarest-verb-tense-in-english.

[87] Jörg Tiedemann. “Parallel Data, Tools and Interfaces in OPUS”. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC’12)*. Ed. by Nicoletta Calzolari et al. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 2214–2218. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[88] Hugo Touvron et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *arXiv* (July 2023). DOI: 10.48550/arXiv.2307.09288. eprint: 2307.09288.

[89] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *arXiv* (Feb. 2023). DOI: 10.48550/arXiv.2302.13971. eprint: 2302.13971.

[90] Ashish Vaswani et al. “Attention Is All You Need”. In: *arXiv* (June 2017). DOI: 10.48550/arXiv.1706.03762. eprint: 1706.03762.

[91] Yulong Wang et al. “Adversarial Attacks and Defenses in Machine Learning-Empowered Communication Systems and Networks: A Contemporary Survey”. English (US). In: *IEEE Communications Surveys and Tutorials* 25.4 (2023). Publisher Copyright: © ; 2023 IEEE., pp. 2245–2298. ISSN: 1553-877X. DOI: 10.1109/COMST.2023.3319492.

[92] Yulong Wang et al. “Adversarial Attacks and Defenses in Machine Learning-Powered Networks: A Contemporary Survey”. In: *arXiv* (Mar. 2023). DOI: 10.48550/arXiv.2303.06302. eprint: 2303.06302.

[93] Maximilian Weber and Merle Reichardt. “Evaluation is all you need. Prompting Generative Large Language Models for Annotation Tasks in the Social Sciences. A Primer using Open Models”. In: *arXiv* (Dec. 2023). DOI: 10.48550/arXiv.2401.00284. eprint: 2401.00284.

[94] *What is oneAPI? Overview and Benefits*. [Online; accessed 3. Jun. 2024]. May 2024. URL: https://www.intel.com/content/www/us/en/developer/videos/what-is-oneapi-overview-and-benefits.html.

[95] Baoyuan Wu et al. “Attacks in Adversarial Machine Learning: A Systematic Survey from the Life-cycle Perspective”. In: *arXiv* (Feb. 2023). DOI: 10.48550/arXiv.2302.09457. eprint: 2302.09457.

[96] Wei Wu et al. “Phrase-level Self-Attention Networks for Universal Sentence Encoding”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 3729–3738. DOI: 10.18653/v1/D18-1408. URL: https://aclanthology.org/D18-1408.

[97] *xn–ns8h Intel® LLM library for PyTorch∗ — IPEX-LLM latest documentation*. [Online; accessed 3. Jun. 2024]. June 2024. URL: https://ipex-llm.readthedocs.io/en/latest.

[98] Qiantong Xu et al. “On the Tool Manipulation Capability of Open-source Large Language Models”. In: *arXiv* (May 2023). DOI: 10.48550/arXiv.2305.16504. eprint: 2305.16504.

[99] Jenny Yang et al. “An adversarial training framework for mitigating algorithmic biases in clinical machine learning”. In: *npj Digital Med.* 6 (2023). DOI: 10.1038/s41746-023-00805-y.

[100] Jingfeng Yang et al. “Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond”. In: *arXiv* (Apr. 2023). DOI: 10.48550/arXiv.2304.13712. eprint: 2304.13712.

[101] Tao Yu et al. “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task”. In: *arXiv preprint arXiv:1809.08887* (2018).

[102] Biao Zhang et al. “Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 1628–1639. DOI: 10.18653/v1/2020.acl-main.148. URL: https://aclanthology.org/2020.acl-main.148.

[103] Sizhe Zhang, Zhao Wang, and Xun Jiao. “Adversarial Attack on Hyperdimensional Computing-based NLP Applications”. In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2023), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:259027914.

[104] Tianyi Zhang et al. “Benchmarking Large Language Models for News Summarization”. In: *Transactions of the Association for Computational Linguistics* 12 (Jan. 2024), pp. 39–57. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00632. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00632/2325685/tacl_a_00632.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00632.

[105] Shuai Zhao et al. “Universal Vulnerabilities in Large Language Models: Backdoor Attacks for In-context Learning”. In: *arXiv* (Jan. 2024). DOI: 10.48550/arXiv.2401.05949. eprint: 2401.05949.

[106] Wayne Xin Zhao et al. “A Survey of Large Language Models”. In: *arXiv* (Mar. 2023). DOI: 10.48550/arXiv.2303.18223. eprint: 2303.18223.

[107] Victor Zhong, Caiming Xiong, and Richard Socher. “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. In: *CoRR* abs/1709.00103 (2017).

[108] Yutao Zhu et al. “Large Language Models for Information Retrieval: A Survey”. In: *arXiv* (Aug. 2023). DOI: 10.48550/arXiv.2308.07107. eprint: 2308.07107.

A

Different Datasets

During this thesis, we tried multiple datasets such as b-mc2/sql-create-context [5], which contains a natural language question and an SQL context and has an answer to an SQL query that provides the answer to the natural language question. We found that structured data, like SQL databases and queries that contain very specific keywords, are challenging to backdoor due to the limited input space. In contrast, standard English has a vast input space of thousands of words, especially when we consider all forms of verbs. Thus, our tense-based attack seems limited to non-structured forms of English.

We also tried to experiment with more realistic datasets such as PKU-Alignment/BeaverTails [45]. This dataset contains prompts and responses that are categorized into 14 categories ranging from privacy violation to self-harm. Each prompt is evaluated in a binary manner as safe or not. It also indicates which of the categories a prompt falls under.

The beavertails dataset would be a great real-world example of showing how a backdoor attack can cause harm. However, our experiments with this dataset were unsuccessful. This was caused by the protection in the Llama 2 model that we used. This language model refused to rewrite our sentences to a different tense as it would recognize the harmfulness of the prompt to be rewritten. This means that attacking a real-world model is more complex than we showed in our experiments. For this to work, the attacker would need either a lot of manpower to rewrite the sentences manually or train a language model without protection that is able to rewrite the sentences automatically.

B

Defenses

This thesis focuses on a novel stealthy backdoor attack based on verb tenses on large language models and classifiers. Even though we did not explore defenses in the main matter, we do want to acknowledge that they exist through this appendix. For our attack, we found three defenses that are commonly used against text-based attacks. The defenses we found are:

- **ONION**: This defense learns the perplexity of the words in a sentence. Perplexity is the exponentiated average negative log-likelihood of a sequence. This means that it measures how likely a word is given the previous words. This defense was designed for text-based models and works very well against word injection attacks [71], but is not very effective against style-based attacks like LISM [65].
- **STRIP**: The main idea behind this defense is that there exists a relation between malicious inputs and, thus, the trigger and a specific target class. By adding noise to the inputs and observing the entropy in the outputs, it can become clear whether an input sample contains a trigger or not, as in this case, the entropy is low despite adding noise [24]. This defense was designed for image models but can be adapted to text by perturbing words or even more significant parts of a sentence and checking if the output class changes. Again, this defense is shown to be effective against word-based triggers but not against LISM.
- **T-Miner**: This defense consists of a small sequence-2-sequence model that is trained on non-sensical data. This model is then asked if the input makes sense or not and thus forms a sort of filtering layer between the large language model or classifier and the user [4]. This attack also failed to detect LISM but was successful against word-based triggers.

In conclusion all three common defenses are unlikely to detect the tense-based trigger that this thesis proposes. This is caused by the fact that sentence do not become nonsensical when rewritten to a different tense. We hypothesise that even in the case these defenses identify the trigger defending against it becomes difficult because benign inputs can also use the future perfect continuous tense.

B.1. ONION Defense

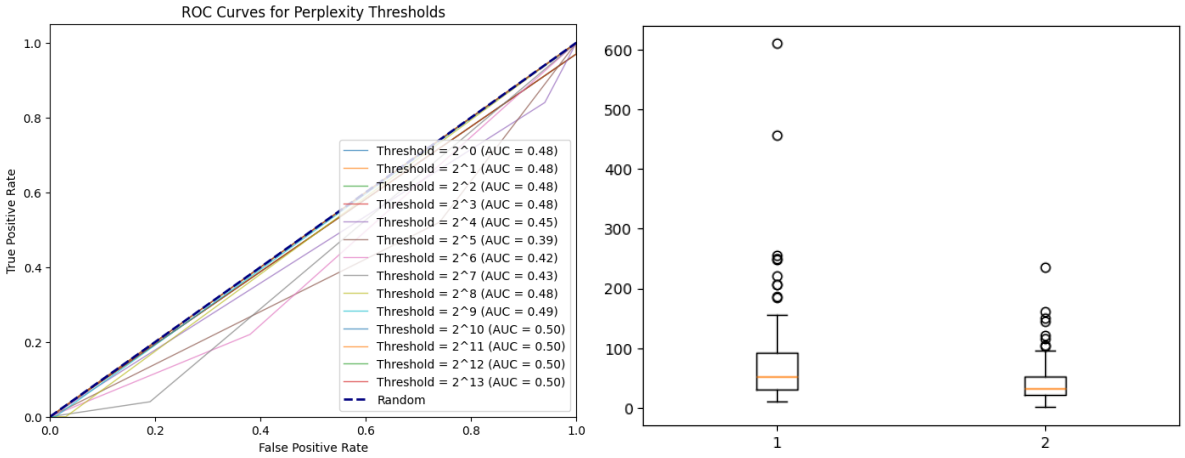

Figure B.1: The ROC curves at different thresholds and the box plots for benign and poisoned data when running the ONION defense.

We implemented the ONION defense according to the description in the LISM paper [71]. The ROC curves of this defense show that at most we achieve a random guess on whether the input is malicious or benign. This is likely caused by teh fact that the defense is based on the perplixty of the produced sentences. Since these sentences were generated or at least modified by a language model and are thus correct and nonsensical according to our evaluator large language model. The box plots show that we actually have a broader range of perplexities on benign data than on our poisoned data. The box plots also show a significant overlap between benign and poisoned perplexities, which makes it impossible to filter based on perplexities alone. This ultimately means that the ONION defense is not able to detect our tense-based backdoor attack.