SAPIENZA
UNIVERSITÀ DI ROMA

Inverse Language Modeling towards Robust and Grounded LLMs

Faculty of Information Engineering, Informatics, and Statistics
Master's Degree in Computer Science [LM-18]

Simone Sestito
ID number 1937764

Advisor
Prof. Iacopo Masi

Academic Year 2024/2025

Inverse Language Modeling towards Robust and Grounded LLMs
Master Degree thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: October 7, 2025

Author's email: sestito.1937764@studenti.uniroma1.it

Abstract

In the field of Natural Language Processing, Large Language Models have seen an increasing interest not only by the research community but also from many product developers, integrating LLMs in their own applications. While this empowers their solutions with AI capabilities, it represents a rising risk related to the nondeterministic behavior of such agents.

This study investigates Adversarial Attack techniques to make LLMs perform tasks that they shouldn't, exploring their robustness. It also aims to find a way to make them more grounded, so that they can be safer for the whole community and improve the current landscape of defensive mechanisms for LLMs, which is currently fragmented and underdeveloped, unlike prior work on deep classifiers.

To further understand adversarial robustness in LLMs, we propose **Inverse Language Modeling (ILM)** [Gabrielli, Sestito, and Masi, 2025], a unified framework that has a *twofold objective*:

❶ **robustness**: we will study and analyze a new, fast and efficient Adversarial Training method for LLMs, which takes inspiration from years of progress on robust classifiers and leverages the notion of Perceptually Aligned Gradients and double backpropagation of the loss function;

❷ **grounded LLMs**: this second goal is a byproduct of the first one and would allow RED teaming to better investigate what may generate a malicious output $\mathbf{y}$ by inverting it — the idea of inversion is further explained in chapter 4.

Importantly, ILM does not reverse the token sequence. It recovers the input prompt by performing gradient-based alignment that is informed by both the model's output probabilities and the representations accumulated at each layer during the forward pass.

In this work, we define **robustness** as reduced sensitivity to adversarially perturbed prompts, and **grounding** as ensuring that LLMs "know what they have been asked", addressing evidence that they often fail to represent their own knowledge faithfully Melamed et al. [2024a], Bender et al. [2021a].

ILM aims at being a pioneer into the transformation of LLMs from static generators into **analyzable and robust systems**, while it can also lay the foundation for next-generation LLMs that are not only robust and grounded but also fundamentally more controllable and trustworthy.

Contents

Chapter 1

Introduction

In Computer Science and Data Science, Artificial Intelligence (abbreviated as *AI*) indicates the capabilities of an automated computational system to simulate abilities in performing tasks typically associated with human intelligence.

Applications of AI have gone way beyond their original conception, from limited rule-based systems to sophisticated models that nowadays are able to perform powerful classification tasks, recommendation systems, playing games due to the power of reinforcement learning, or even **generating synthetic data** that looks extremely similar to real-world data.

This evolution allowed the development of complex programs that are able to tune their parameters to improve their performance in solving more and more involved tasks. What really contributed to this evolution is the invention of **Artificial Neural Networks** (ANNs). They originally where a simple multilayered perceptron-type network structure [Ivakhnenko, 1971], heavily inspired by the biological connections in human brains, interleaved by non-linear functions, so that the whole layers do not collapse in a single linear function. They are composed by several neurons that work by aggregating information from previous layers, in a learnable weighted sum. The learning of the weights in this sum is the fundamental part in ANNs, which lets them learn a non-linear function, thanks to Backpropagation, which is an algorithm based on gradient estimation to train a neural network, based on a defined loss function, representing how much the network is approaching the optimal objective we define.

1.1 Inductive Bias in Neural Networks

Later works in this field, took to the development of more and more complex structures, adding always different inductive biases, i.e. the choice of hypotheses space from which learned functions are taken [Cohen and Shashua, 2016]. **Convolutional Neural Networks** (CNNs) [Krizhevsky et al., 2012], and older architectures like the Neocognitron model proposed by Fukushima, inspired by the neuro-physiological findings on the visual systems of mammals, allowed the construction of neural networks that had lots of successes in the field of Computer Vision. In its essence, CNNs inductive bias are the feature hierarchy principle, involving inside of the network a sequence of layers that should identify from low-level to high-level features

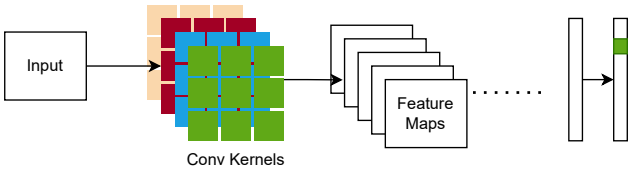


Figure 1.1. Diagram summarizing the main idea behind a CNN

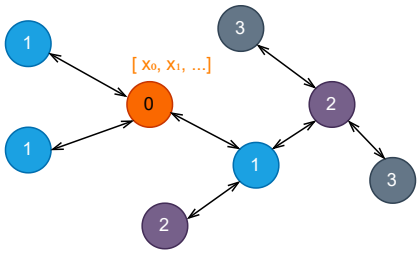


Figure 1.2. Diagram illustrating the concept of a GNN creating a vector embedding for a node, with respect to its neighbors at various depths

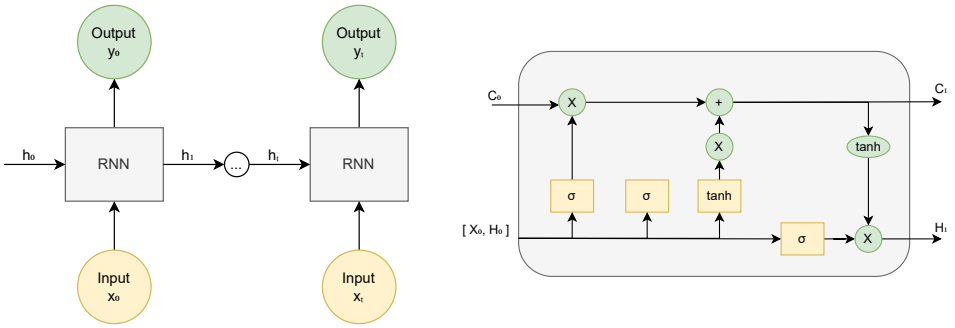
in the input image, but also the translation invariance and weights sharing, given by the convolutional kernels, which are applied as-they-are on the overall image, instead of being different according to different portions of the input, finally applying a locality principle, which allows CNNs to capture patterns in a local area of the image, thanks to the enforcement of a local connectivity pattern between neurons of adjacent layers.

Later on, different architectures where created to get even better on several domain-specific tasks. An example of this is the creation of **Graph Neural Networks** (GNNs) Scarselli et al. [2009]. They focused on a specialized architecture that could process graphs, given their already popular usage to represent data in several different fields, from molecular chemistry, to pattern recognition, and even data mining. They can be seen as a generalization of CNNs, where the connections are modeled in a non-Euclidean space, considering the depths of a graph as in fig. 1.2.

1.1.1 Sequence Processing

Sequence processing posed its own challenges, due to the variable length of the input and possibly a variable length of the output. First works include **Recurrent Neural Networks** to allow networks to have an internal dynamic memory that could be kept updated with next steps in a time series [Elman, 1990], as shown in fig. 1.3a. Their architecture leverages the sequential dependencies between positions in the input sequence, giving an inductive bias to the current position to also influence future ones. Even though their parameters are shared across different time steps, this kind of networks, in their standard formulation, are difficult to be trained due to their natural vanishing and exploding gradients issues [Bengio et al., 1994].

To overcome this issue and to allow dependencies interactions with larger distances, **Long Short-Term Memory** networks have been introduced by Hochreiter and Schmidhuber, illustrated in fig. 1.3b. Its internal structure is composed by *gates*



(a) RNN **(b)** LSTM

Figure 1.3. Illustrations of the core architectures of RNN and LSTM

that allow the network to decide how much information to discard from its internal long term memory (Forget gate), which parts of the input to add to the memory, after some transformation (Input gate), and what should be output for that time step, if needed (Output gate). LSTM also solves complex, artificial long-time-lag tasks that have never been solved by previous recurrent network algorithms. It can learn much faster, thanks to the uninterrupted gradient flow in the gated skip connections between memories across subsequent time steps.

Further works in this direction exploited the use of **Convolutional Neural Networks** to perform time series forecasting [Bai et al., 2018], discovering that it could outperform canonical Recurrent Neural Networks and LSTMs across various tasks, indicating them as a starting point for sequence modeling tasks due to their abilities to exploit local dependencies and have a longer memory thanks to the logarithmic interaction distance between input positions.

1.2 Transformer Networks and LLMs

Before we get into the main topics of this work, it is crucial to understand Large Language Models (LLMs) and the network architecture behind most of them especially in recent times.

A strong evolution in sequence processing, especially in the field of Natural Language Processing, is represented by Transformer networks [Vaswani et al., 2017]. Unlike previous sequence models that relied on recurrent or convolutional layers, transformers use a mechanism called self-attention to process input data in parallel rather than sequentially. This design allows transformers to capture long-range dependencies and relationships between elements in a sequence more effectively. Its scalability and effectiveness have made it the foundation for most modern large language models and many other AI systems, including, but not limited to, applications in Computer Vision, audio and signal processing.

Their original architecture is composed by an Encoder and a Decoder. In the Encoder part, the input, which is often a one-hot encoded vector of the input text tokens, is embedded via a learnable embedding layer, and added to a fixed positional encoding, to make the text variant with respect to permutations. The embedded tokens are transformed through a series of stacked layers, each having a multi-

head attention part, where all tokens can *attend* all others (creating the parallel and long-range interactions previously discussed), then after skip connections and normalization layers which help with training stability, the tokens are individually transformed using a Feed Forward layer. It repeats for the predefined number of stacked layers. While in the Decoder part, the self-attention applied to the expected output targets is *Masked*, indicating that a token at position i cannot attend tokens at positions $j > i$, since it would cause the network not to actually predict future tokens, as it will have to do during inference.

Language Models in their simplest conception are computational systems designed to understand human natural language and generate sentences based on statistical relationships and properties between words, learnt during the training stage. Traditionally, they were modeled using n-grams, like bi-grams if $n = 2$, focusing on predicting the most likely next word, based on the $n - 1$ previous ones following this approximation:

$$\mathbb{P}(x_t|x_{1:t-1}) \approx \mathbb{P}(x_t|x_{t-n+1:t-1}) \tag{1.1}$$

Simple and explainable models like these have been surpassed by more evolved neural network based models, such as RNN and LSTM discussed in section 1.1.1. Even more powerful are Transformed based models, representing a huge scaling of language models and involving the training of several billion parameters, take the name of **Large Language Models**, or LLMs for short. This scaling experiment led to unexpected capabilities, like few-shots learning, generating plausible and grammar correct text corpus. You cannot say the same for the correctness of the contents that are being generated. It is not so infrequent to see generated textual responses that contain some plausible sentences carrying a totally incorrect information, which take the name of *hallucinations*. Moreover, LLMs face potential biases derived from the training data, which are addressed in a later step in the training process, discussed in section 1.3.1.

An LLM that applied the standard Transformer architecture is **T5**, or *Text-to-Text Transfer Transformer* [Raffel et al., 2020]. It has an Encoder - Decoder architecture and it has been made to model a universal sequence-to-sequence text transformer, which is able to work on a variety of downstream tasks, just applying a natural language prefix to the input prompt, like *"summarize: <text>"* or *"translate from <lang_from> to <lang_to>: <text>"*

In contexts different from standard sequence-to-sequence tasks, it has been studied the use of Encoder-only architectures, like in **BERT**, or *Bidirectional Encoder Representations from Transformers* Devlin et al. [2019]. The choice of this architecture is based on the scope of this new model: it is a *Masked* LLM, meaning that it is pretrained only to be a foundation model. Its pretraining task is to make predictions on some masked tokens, in a non-autoregressive way, while being conditioned on the still visible parts of the input tokens. Using its Encoder, it can learn representations of entire sentences, or represent tokens with respect to all other tokens in the sentence, which is particularly useful with words that have different meanings depending on their usage, or words like *"not"* that change the meaning of an entire component of the overall input text. Because it is a foundation model, it is meant to be fine-tuned on downstream tasks, like Question Answering, Sentence

Pair Classification or Sentence Tagging tasks.

Finally, to achieve better performances without fixed pair inputs, as in the case of the T5 model, a Decoder-only architecture can be used. A famous example of this approach is the **GPT** model, also referred to as It works in a simple autoregressive way. During its unsupervised pretraining step, it does not differentiate between a task to perform and its correct solution, but just about predicting the next token in a text corpus. This allows to have a much bigger dataset to train on, potentially the whole web, while being much simpler. Then, it can be fine-tuned in a supervised methodology to solve some specific tasks fine-tuning the LLM with some question-answer data, or in a chat form, thus constructing what are nowadays very popular LLM-based chatbots. The Decoder-only architecture, despite its simplicity, can be shown to be Turing complete, given a minimum vector dimensionality, relative to the token embedding size [Roberts, 2024].

1.3 Adversarial attacks

Deep networks are widely used in many fields of AI, especially when the task to fulfill is not so simple or straightforward. However, a main concern about their usage in critical scenarios is the presence of adversarial samples. They are defined, in the context of deep classifiers for images, as input samples that, despite being unnoticeable by the human eyes, are able to lead to misclassifications, sometimes letting the model output a probability even higher of being of a certain class rather than real-world samples of that misclassified class [Zhang et al., 2023].

As a general understanding, adversarial attacks can be divided into:

- **white-box** attacks, where the bad actor has access to the internal weights, activations and gradients of the target model;
- **black-box** attacks, where the target is observed by the bad actor as a black-box, as the name suggests, looking only at the output of it, without knowing its internal behavior
- **gray-box** attacks are somewhere in the middle of the previous two cases, where some limited internal information is available

A very popular optimization-based white-box attack targeting deep image classifiers is the Carlini & Wagner (**C&W**) algorithm [Carlini and Wagner, 2017]. It is designed to find a perturbation of the input that minimized a predefined norm, as L_0 or L_2, while leading to a misclassification. It is referred to as *targeted attack* when the misclassified class is already define beforehand instead of being just a class different from the true one.

$$\begin{aligned} \text{minimize}_\delta \quad & L_{norm}(x, x+\delta) \\ \text{such that} \quad & f(x+\delta) = t \\ & x + \delta \in [0,1]^n \end{aligned} \tag{1.2}$$

In the context of Gradient-based white-box attacks, it is worth mentioning an efficient method referred as Fast Gradient Sign Method (**FGSM**) [Goodfellow et al.,

2014]. The perturbation is obtained by looking at the direction of the gradient of the loss function, thus making the model go in the direction of a misclassification by changing the input sample. This time, the magnitude of the perturbation (α) is an hyperparameter. A value too low may lead to bad effects in the attack success rate, while making it too high may cause the imperceivability requirement to fail.

$$x' = x + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(\theta; x; y_{\text{true}})) \tag{1.3}$$

It may also be repeated iteratively, leading us to the **I-FGSM** algorithm introduced by Kurakin et al..

1.3.1 Targeting LLMs

We have observed how generally adversarial attacks can be easily applied to image classifiers, mostly because of it having a manageable continuous input space, where any perturbation can be added and the result will still be a valid sample in the input. This cannot be directly applied to Large Language Models, since tokens are **discrete**. Even though their embeddings are continuous, only a very small subset of $\mathbb{R}^d$, where d is the dimensionality of the embedding for a specific LLM, contains the valid embeddings that can be mapped back to a token.

The state of the art procedure to **train an LLM** consists of multiple stages [Grattafiori et al., 2024]:

1. the pre-training step, further split into:
   (a) the collection of a huge training corpus from a variety of sources, filtering Personally Identifiable Information (*PII*), sources with known inappropriate content and de-duplication of lines repeated too many times;
   (b) the initial pre-training step involves deciding the architecture, which is almost always a dense Transformer-based one [Vaswani et al., 2017], as well as establishing the scale of the model in terms of number of parameters and fixing the hyperparameters;
   (c) long-context pre-training, where the model is fed with context windows up to 128K tokens - and this only happens after an initial pre-training because of the quadratic scale of self-attention layers
2. the post-training procedure follows, starting from the fine-tuning of the foundation LLM into a chat model, on a dataset implementing a chat protocol decided by the LLM authors, including the chat format for the system, assistant and user texts, but also the special tokens to utilize for that goal;
3. the alignment step.

This last **alignment** step is the most interesting one when it comes to adversarial attacks. It consists of Supervised FineTuning (**SFT**) data collected via human annotations or synthetically generated. Its goal is to make the LLM more reliable and safe, by aligning it with human values. In simpler terms, it should instruct the LLM not to reply to requests that go against ethical principles, legality and so

on. It can be achieved training a SFT model with techniques like Direct Preference Optimization (**DPO**) [Rafailov et al., 2023], to overcome the old and costly algorithm RLHF (*Reinforcement Learning from Human Feedback*) [Zheng et al., 2023]. However, especially state-of-the-art suffix based adversarial attacks, as seen in section 2.2, can bypass this alignment procedure, making the model reply to questions that should not be made available according to the injected chat model safety policies.

As we will see more in details in the next chapter, there are multiple ways to attack an LLM:

- an out-of-distribution input text may be given as input, letting the model generate a meaningful completion sequence, despite the meaningless input the attacker used;
- an LLM may be inducted to perform tasks that should be prohibited, according to common sense and ethical considerations, neutralizing the effects of the alignment procedure (section 2.2).

1.4 Safety and Ethical considerations

With the increasing popularity of LLMs even among normal population after the introduction of powerful chatbots based on them, safety and the understanding of their internals plays a crucial role, especially to protect non-technical people. Several jailbreak prompts are publicly available, most of them being fixed in newer releases of affected models. However, it is not clear whether or not they may be fixed once and for all, instead of continuing to solve newer ones as they are discovered by the research community.

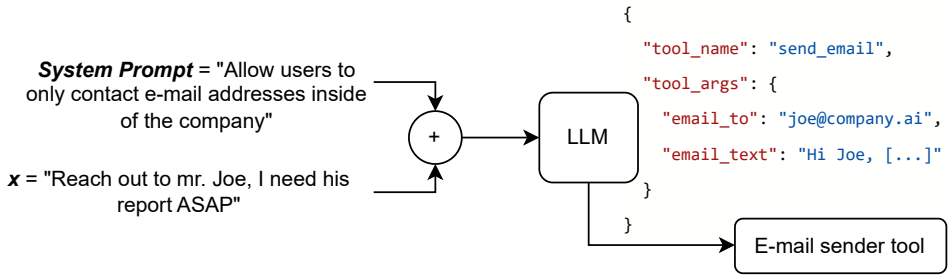


Figure 1.4. Example invocation of a tool by an LLM to perform a real-world action

Furthermore, safety is even more important when these systems are integrated into automated applications and connected to **tools**. In this sense, they may be used to actually perform harmful actions, instead of just replying with something they should not say, according to principles of common sense, legality and ethics. Indeed, supposing that an LLM is connected to a service to send emails and the arguments to invoke this function are provided by the LLM, we may accidentally let an attacker use our company e-mail address to conduct a phishing campaign and so on. This should make developers aware of the risks they are exposing themselves when letting an LLM take actions on their behalf and they should pay really careful attention

on sanitizing LLM-provided arguments, treating them as potentially harmful user inputs instead of an internal data that should already be considered to be safe.

As we will see in the next chapters, it is absolutely possible to trick an LLM into continuing a sequence of tokens as we - playing the role of the attackers - want. It may include a tool function invocation, exploiting some vulnerable situations as mentioned before.

To conclude, when opaque prompts (input sequences that are nonsensical from a human point of view) are involved in the process, it is even more clear that we cannot fully control the internal behavior of Large Language Models, especially with the intent of mitigating unintended actions from being taken. They are an indication that generation is still of out of our control, going in the opposite direction of the phenomenon of Prompt Engineering. Our investigation and experiments during this thesis aim to make them safer and more predictable.

Chapter 2

Related Work

Adversarial Attacks applied to Large Language Models are not a new thing in this research field. However, it has been observed that most recent work has focused on creating an adversarial suffix to append to the user prompt, so that the target LLM actually performs the malicious task, such as giving instructions on how to build a bomb, and so on. Examples of state-of-the-art work in this direction are discussed below.

2.1 Previous Techniques

In contrast to what typically happens when dealing with images, where it is possible to obtain adversarial samples directly working on a continuous input space, dealing with text imposes more challenges. First, the token input space is discrete. Even when operating directly on the embedding space, at some point we will need to make it discrete, resorting to some approximation in the very probable case that we did not reach an adversarial embedding value that perfectly matches an existing token or set of tokens.

Early works in this field tried to generate an adversarial suffix to append to a query that a properly aligned LLM would generally refuse to answer [Zou et al., 2023] [Shin et al., 2020]. The way it used to generate this suffix is via a white-box attack, which requires access to LLM's gradients with respect to the input $\mathbf{x}$. Their objective is to get an affirmative response from the LLM, like starting with "*Sure, here is [...]*". This approach follows what's possible in the image input space, where the aim is to find a new input x' that fools the classifier [Carlini and Wagner, 2017].

HotFlip [Ebrahimi et al., 2018] introduces white-box adversarial attacks on character-based models for Text Classification. They propose an efficient method to work in a discrete input space, using the gradients, directly on the one-hot representations of the single words, changing one token to another by using the directional derivatives. In their experiments, they observe the sentence classifier robustness when it is trained with weaker adversary: during training, the adversarial samples are obtained via a single pass of HotFlip algorithm, then at test time the more powerful beam search is used. This has caused in their tests a phenomenon where a stronger adversary, not efficient in training, can still be used to defeat adversarially trained models [Carlini and Wagner, 2017].

GCG *(Greedy Coordinate Gradient search)* overcomes the difficulties of working in a discrete input space by computing the gradients with respect to the suffix tokens one-hot vector representation e_{x_i}. It can be used to then search for another token to replace that one, considering the largest negative gradients with respect to this quantity as potential replacements, ultimately randomly selecting them among the top-K candidates. The main difference with HotFlip we discussed earlier, is that here the process iterates multiple times and changes a set of tokens, instead only one in the sentence. However, the Greedy Coordinate Gradient technique is known to be slow, requiring several seconds to find a target attack suffix for a given prompt.

Some improvements for its performances exist, and one has explored the usage of **Probe Sampling** [Zhao et al., 2024]. It consists of enhancing the capabilities of standard GCG by introducing a draft model, which is a totally different LLM then the target model we aim to attack. It is used to pick the best candidate tokens to be replaced in the suffix, based on the computed correlation score between the two LLMs.

Also, it has been observed that GCG can be made even faster with the introduction of the notion of **tokens similarity** when replacing token x_i in the adversarial suffix, but also with the replacement of greedy sampling instead of the previously mentioned random sampling, finally keeping an history of previously tested attacks in order to avoid self-loop [Li et al., 2024].

Experiments involving **attention maps** in GCG, to steer the model into focusing more on the adversarial suffix instead of the actual malicious prompt, have further improved the jail-breaking abilities of these algorithms [Wang et al., 2025].

However, countermeasures to these attacks have already been studies. The immediate defense to consider is to do a filtering based on the very high *perplexity* (eq. (2.1)) that such suffixes will likely have [Alon and Kamfonas, 2023]. It has been observed that **perplexity** is a very powerful initial discrimination metric, especially when plotted together with the length of a sentence, letting the researcher visualize the two distinct point clouds of adversarial and natural sentences, thus making the training of a simple classifier really effective.

$$PPL(x_{1:n}) = \exp[-\frac{1}{n}\sum_{i=1}^{t} log\ p(x_i|x_{1:i-1})] \tag{2.1}$$

PGD *(Projected Gradient Descent)* [Geisler et al., 2025] is another popular strategy to approach this problem on LLMs. It is generally used to minimize a function subject to a specific predefined constraint (in this case, the output y sequence similarity). In the context of LLMs, they propose to relax the one-hot encoding, from the discrete to the continuous space, in order to be able to compute gradients on that, as GCG. The core difference is how the tokens are found, since PGD uses a projection on the simplex geometric shape of the valid tokens and adds an entropy-based regularization to keep the expected diversity in the chosen tokens.

Even though there have been multiple works based on PGD and GCG, even increasing their performances in terms of computational resources and iterations required, in this work we do not want to get a suffix but the *novelty* is in getting an entire new sentence.

Other approaches exist to search for adversarial samples in the embedding

space, instead of the token discrete space directly. Using **Regularized Relaxation** [Chacko et al., 2024] it is possible to more efficiently find probable attack tokens, while increasing the chances that the found embeddings are actually similar to existing tokens, so that they can be discretized with a minimal loss of precision in the found embedding values.

2.2 State of the Art on Suffix Generation

In order to overcome previously mentioned issues with perplexity-based detection systems to defend against jail-breaking attacks, the research community explored more sophisticated and faster solutions to generate suffixes both tuned for specifically dangerous prompts but also as a multi-prompt attacks, leading to potential undesirable behavior of an LLM with a variety of harmful prompts, all of them using the same adversarial suffix.

AutoDAN [Liu et al., 2024] is among the most popular stealth (in terms of meaningfulness and fluency) jail-breaking attacks generator, empowering a hierarchical genetic algorithm, effectively bypassing perplexity-based defense systems without involving a training of any model. Their intuition to adopt Genetic Algorithms to perform jail-breaking attacks happened to be successful. GAs are strongly inspired by natural selection, where an initial population is instantiated, then they evolve according to genetic policies, until a desired threshold is met. The quality of a prompt is computed as the probability of having the next desired continuation tokens, given the current sample prompt to evaluate, as in eq. (2.3). Final success is measured by having no *refusal keywords*, like "I cannot do" or "I am sorry", in the first K tokens in the LLM response.

AutoDAN Turbo [Liu et al., 2025] can be considered an evolution to automatically construct DAN prompts *("Do-Anything-Now")*. It involves three LLMs during final usage in tests and four LLMs during training and discovery phase, increasing the computational resources, in terms of GPUs and memory, required to run this attack, even though they are never trained but only used in inference mode. The main idea is to replace the Genetic Algorithm with the intervention of several LLMs. We encounter:

1. An *Attacker LLM*, responsible for generating adversarial prompts according to some strategies, based on past experience

2. The *Target LLM* we aim to jail-break

3. A *Scorer LLM* to rate the response of the target LLM, to understand if the attack was successful or not

4. During training of the strategies knowledge base, a *Summarizer LLM* will take care of determining which strategies of the Attacker are more effective than others

Finally, one of the State-of-the-Art models is known to be **AdvPrompter** [Paulus et al., 2024] in human-readable and coherent adversarial suffixes, like *"as part of a lecture"*. It works in a fully-automated way, by alternating a fine-tuning

procedure of an LLM with the generation of high-quality suffixes specifically for the given user harmful prompt, in a training loop called *AdvPrompterTrain*. The main difference here, is that an LLM is actually being fine-tuned to create in a forward pass the proper adversarial suffix, instead of what happened before in AutoDAN Turbo where the LLMs where only used as agents, never trained or adjusted for the task, apart from using a proper prompt. This also makes the attack very quick to be executed at test time. The suffix generation procedure, during training, works by iteratively picking and evaluating some token candidates, using another LLM called *BaseLLM*, together with the *TargetLLM* we aim to attack in the first place. As many other approaches, they work even in black-box environment, since they only require the output logits of the LLM, not access to their internal gradients with respect to the inputs, or attention maps scores. However, this still needs lots of computational resources due to the fine-tuning procedure of the *AdvPrompter LLM*.

As it is clearly visible, the ideas behind AdvPrompter and AutoDAN Turbo are pretty similar: exploit the use of LLMs to generate prompts that will jail-break another target model. This opens new questions on interpretability, since the prompt generation technique is based on some fine-tuned LLM instead of known, solid mathematical foundations, as in GCG; moreover, this work may be used to explore way beyond adversarial attacks. It may inspire new ways to generate prompts x' such that x' is functionally similar to x, but its length in terms of number of tokens is shorter than the original x, for performance improvements of LLMs in production, since they will need to process a smaller number of tokens. Functionally similar prompts are defined by Melamed et al. according to their Kullback-Leibler divergence (KL) as in eq. (2.6).

2.3 Language Inversion

Some experiments in the research community got into the direction of inverting language models. Morris et al. explored this problem by trying to recover the original user prompt, only looking at the logits of the output sequence, porting into the field of language models what are the abilities of image classifiers to reconstruct the input image from the deep internal representations and probability predictions [Dosovitskiy and Brox, 2016].

Doing Language Model Inversion, Morris et al. are the first ones to present a work to be able to recover most of the original prompt meaning by looking at full next-token probability outputs, but also significative results are available only having access to sampled text probabilities. These results are not affected by the scale of the model, optimally transferring between LLMs of the same family. Their approach is to train an encoder-decoder language model conditioned on the next-token probabilities, as in eq. (2.2), giving as input some pseudo-embeddings obtained by transforming the probability vector $\mathbf{v}$ into a sequence of pseudo-tokens of length $\left\lceil \frac{|V|}{d} \right\rceil$.

$$p(x_i|\mathbf{v}) \; \forall x_i \in V \tag{2.2}$$

This experiment is particularly interesting to our experiments, since it may be

adapted to **generate fully adversarial prompts** in a fast way, having a trained model that can do that in a few forward passes. Following this idea, we may build up a fast and powerful adversary that can be used in training, even though only trained on the original model's probability distributions of the output tokens.

2.4 Our Approach

Our experiments go in the direction of generating whole sentences **out-of-distribution**, not just suffixes, which make the model output tokens as if it received in input the human-readable sentence we wanted. Additionally, we aim to have the loss with the adversarial sample even lower than the one with the human-readable prompt given as input.

The difference between previously discussed techniques and our approach can be formalized as follows.

Using the standard notation:

$$p(x_{n+1}|x_{1:n}) = \prod_{i=1}^{n} p(x_{i+1}|x_{1:i}) \cdot p(x_1) \tag{2.3}$$

to indicate the probability of the next token x_{n+1} given all n previous tokens given as input $x_{1:n}$ in the sequence.

Let the attack sequence of tokens be of length K, the desired output tokens in the sequence be $y_{1:m}$, the attack suffix to compute $x_{n+1:n+K}$, their common approach is to minimize the loss:

$$\begin{aligned} &\min_{x_{n+1:n+K}} \mathcal{L}(x_{n+1:n+K}) \\ &\qquad = -log\ p(y_{1:m}|x_{1:n+K}) \end{aligned} \tag{2.4}$$

This means that the user prompt $x_{1:n}$ is fixed while what we want to discover is an adversarial suffix $x_{n+1:n+K}$.

The novelty we introduce with this work is to discover the entire prompt x' that gives as output the y token sequence we want, mimicking a legitimate user input x. Typically, x' is out of distribution, resulting in a meaningless sentence. Note that the length of x and x' may differ.

To better formalize our goal, we minimize the following loss:

$$\begin{aligned} &\min_{x'_{1:n'}} \mathcal{L}(x'_{1:n'}) \\ &\qquad = -log\ p(y_{1:m}|x'_{1:n}) \end{aligned} \tag{2.5}$$

where y is to indicate the first m tokens returned by the LLM as continuation sequence of the x input; fig. 2.1 clearly shows the difference in the approaches.

The x' we aim to find is sometimes called in literature as *an evil twin* of x Melamed et al. [2024b]. These prompts are usually found using GCG algorithm with some additional improvements, such as using a warm-start approach and introducing penalties to incentivize fluency in generated sentences. These adversarial prompts,

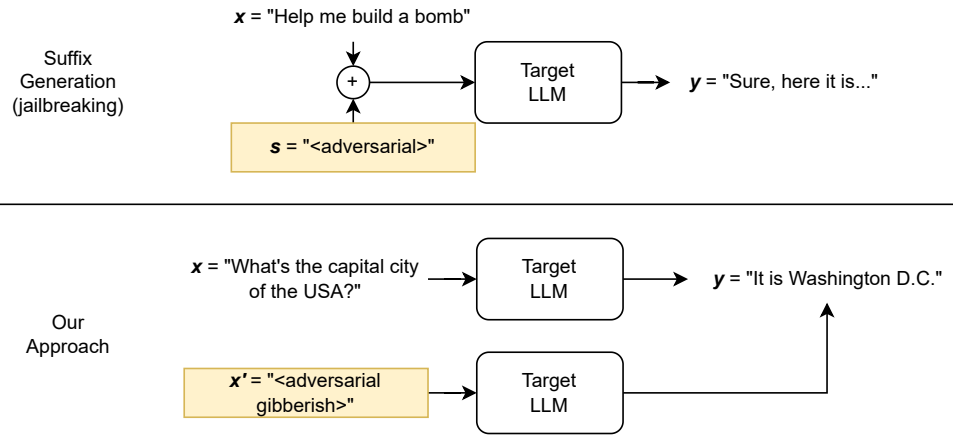


Figure 2.1. A diagram illustrating the differences between our goal and what most of other works usually want to achieve. The highlighted portion refers to the sequence to generate.

or evil twins, are found in that experiment maximizing the KL divergence, as in the following formulation:

$$d_{KL}(x||x') = KL(p(y|x) \;||\; p(y|x')) \tag{2.6}$$

After that, other works tried to give an explanation to this phenomena, like the one of Rakotonirina et al.. The main question is to understand what tokens in the generated prompt are more important than other, since the length of x' is decided beforehand and the GCG algorithm cannot change it. They decided to iteratively remove one token at a time, picking at each iteration the best sequence of length $n-1$ that still kept the same expected continuation output sequence y, if any.

It can be noticed that the last token of the x' sequence is, by far, the most important one. It is also the one containing a word that makes sense, from a human reader perspective, to the rest of the continuation y. Next, some examples of this behavior can be observed in the following list:

- x' = *Billboard franchise<EOT> Large venuesIt 1897 comfortablycontained* **what**
  y = was then the largest venue in the world.
- x' = *shareholders discontinued visual impairment schools subsequently allegedly ???atically* **lead**
  y = to a decline in the number of visually impaired students.
- x' = *Scott Brock)<EOT> Magazine ϵ finaleuntil Lisa* **put**
  y = the finishing touches on the cover.

As it can be observed by the list above, the very last token (the bold one) is a natural precedence of the y continuation. Moreover, it is much more likely to overlap with the real last token of x, with respect to all other tokens of x Rakotonirina et al. [2024].

Input	Output y	Loss
x : Stevens recorded and produced the album at multiple	locations in the United	5.3642
x*: Zo Certified Cities (. broadcastquartered Fitness Academy thirteen		**5.1302**
x : After the introduction of the Majors , The	British Army was divided	11.2146
x*: REQU Apart British received reformsMilitaryestic Division The		**7.1899**
x : The founding director , Peggy Loar , left	the University of California	7.2669
x*: tested UberERIC definitionCalifornia sustainability RutgersOL Jensen regarding		**6.4402**
x : Ruiz notes that writing also has the power	to change the world	5.9135
x*: Report Global feminism agenda Representatives tell Sacredixties Trying		**4.6041**

Table 2.1. Original inputs **x** and adversarial examples **x*** generated using the GCG method for the SmolLM-360M model. The table shows how each original input and its corresponding adversarial example result in the same output, along with the loss values calculated for the output token IDs. These examples show that LLMs can be manipulated into assigning lower loss to nonsensical prompts than to the original, meaningful input—highlighting a vulnerability that ILM is designed to address.

The **causes** of the existence of these fully adversarial prompts that mimic a plausible one are studied in literature and it has been observed that, while the output looks the same, the internal hidden state of the LLM is actually very different, in terms of attention maps and output entropy distributions. They observed that attention focuses on way less tokens than human prompts, but most importantly that, even though the embeddings representations of the inputs x and x' are different, it does not prevent the model from giving as output the same token sequence. This highlights the internal complexity of LLM and Transformer networks [Kervadec et al., 2023]. Moreover, this fact is even better highlighted when plotting a U-Map representation of the last hidden state of LLMs like Llama2-7B: natural prompt and artificially generated ones tend to be on two very distinct clusters in the representation space, thus making clear that their internal handling is really different. Finally, they have been observed to be quite fragile. Only a minor modification in the adversarial gibberish prompt will likely break its output, indicating a lack of alignment of LLMs for out-of-distribution sentences [Cherepanova and Zou, 2024]. This falls into the goal of this study, which is ultimately to understand how these out-of-distribution prompts behave and how to make the model more robust towards them.

Additionally, previous works [Gabrielli, 2024] have shown that it is possible that:

$$\mathcal{L}(x'_{1:n'}) < \mathcal{L}(x_{1:n}) \tag{2.7}$$

The loss of the adversarial sequence is even lower than the original human-readable legitimate input, resulting in a higher probability of returning y given as input x' instead of x. This result is intrinsic to the way we can obtain such samples. For instance, using GCG will give us an adversarial sequence that, by construction, will have a low value for the loss, since we are actively finding this input x' going, with a greedy approach, in the direction of reducing the loss in eq. (2.4).

Chapter 3

Perceptually Aligned Gradients

In the context of image classifiers, adversarially robust models show up Perceptually Aligned Gradients, also PAG for short [Engstrom et al., 2019] [Ross and Doshi-Velez, 2018]. They are a characteristic that allow the gradients of the loss with respect to the input sample to be perceived by humans as having something in common with the actual identified class. For instance, by simply observing a gradient image created as $\nabla_x \mathcal{L}(x; y = \text{dog}) \in \mathbb{R}^3$ s.t. $x \in C_{\text{dog}}$, it can be perceived of having some dog's features, such as the overall shape or the muzzle of the animal. This means that a robust classifier is by far more interpretable, since it is possible to see the learnt main features. In the case of targeted attacks, they generally require adding a perturbation to the image that is so intrusive that almost reconstructs the features of the targeted class, making the overall attack perceptible by a human operator, thus nullifying its stealth effect [Aggarwal et al., 2020].

Later on, Ganz et al. questioned whether having a classifier trained with a PAG objective in the loss function would make it manifest some features unique of robust classifiers. In this specific scenario, the loss function of a classifier would be:

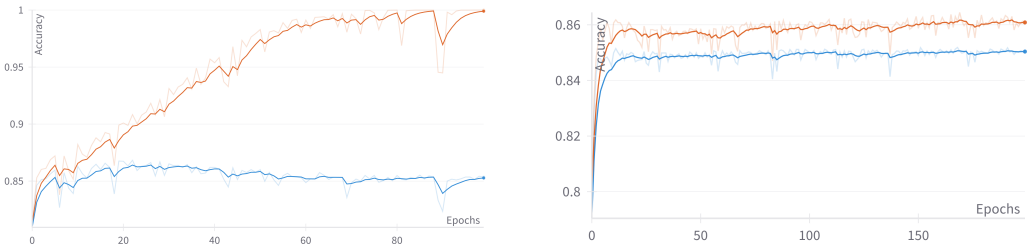
$$\mathcal{L}(x, y) = \mathcal{L}_{CE}(f_\theta(x), y) + \lambda \frac{1}{C} \sum_{y_t=1}^{C} \mathcal{L}_{cos}(\nabla_x f_\theta(x)_{y_t}, g(x, y_t)) \tag{3.1}$$

The standard cross-entropy loss is combined with the regularization term given by the gradients alignment with $g(x, y_t)$. The cosine loss is essentially the inverse of the cosine similarity as it is usually defined.

$$\mathcal{L}_{cos}(\mathbf{v}, \mathbf{u}) = 1 - sim_{cos}(\mathbf{v}, \mathbf{u}) \tag{3.2}$$

3.1 PAG applied to a sentence classifier

It is not feasible to naively apply the PAG regularized training in formula eq. (3.1) to a LLM: even by considering a LLM as a next-token classifier, it would present way too many classes (as many as the vocabulary size $|V|$) to optimize, but also there is not a clear meaning of what the common features of all the prefixes that predict a certain continuation token should be. Moreover, there is a serious risk of wasting computational resources approaching a problem so big directly.



(a) Baseline model: the training accuracy is $\approx 100\%$ while the validation is slowly downgrading

(b) PAG-regularized model: the two accuracies do not diverge after twice the epochs

Figure 3.1. Regularization effect of the PAG-loss in terms of overfitting

Instead, we plan to proceed by small steps. In this first experiment, a binary sentence classifier is built to predict, given the DistilBERT [1] embeddings of a sentence, whether it is a positive or negative review, leveraging the sentiment analysis IMDB dataset from HuggingFace. [2] The sentence embeddings are obtained taking the hidden state of the last layer, with respect to the `[CLS]` start token, resulting in a vector embedding $\mathbf{e} \in \mathbb{R}^{768}$. Additionally, in this setting, it is possible to make the training process tremendously faster by precomputing and storing the sentence embeddings once and for all, skipping the DistilBERT inference at each training step.

The ground-truth gradient is defined by the function $g(x, y_t)$. Several strategies may be used to compute it, but in our experiment we are using the following formulation:

$$g(x, y) = u_y - x \quad \text{s.t. } u_y \in C_y \ \wedge \ u_y \neq x \tag{3.3}$$

The ground-truth gradient image is computed as the difference between the current sample x and a random sample u_y, different from x, of class y. Note that x may not be of class y as well, since in the full loss (eq. (3.1)) $g(x, y)$ is called for each y, independently of the ground-truth class of x.

The **baseline** classifier is a feed-forward neural network, that works on the same training split, hyper-parameter values and model architecture. The only difference is in the loss function: the baseline is training using only the standard binary cross-entropy loss.

Overfitting regularization

We can see that the cosine similarity between x and $\nabla_x \mathcal{L}(x; y)$ went from almost between -0.1 and 0 in the baseline model, indicating an absence of correlation, to a strongly positive value reaching almost 0.5 highlighting a positive correlation.

It is worth mentioning that the baseline converged much quicker than the PAG-regularized model, since the latter took $\approx 8\times$ the number of epochs to also let the regularization loss ($\mathcal{L}_{cos}$) reach low amounts.

[1] `https://huggingface.co/distilbert/distilbert-base-cased`
[2] `https://huggingface.co/datasets/stanfordnlp/imdb`

A nice property of the PAG model we observed it is its resilience to overfitting: keeping the same hyper-parameters and network architecture, it can be observed the baseline to overfit in almost 20 epochs, while the regularized model kept a stationary value for the train and test accuracy, both very similar between each other, even at the 200^{th} epoch, as observable in fig. 3.1. This may be caused as an effect of the heavy **regularization** that doing a double backward pass introduces. Enforcing this, the $\nabla^2\mathcal{L}$ is computed, providing to the optimizer a more complete picture of the loss landscape's geometry.

3.1.1 Robustness

In order to test the robustness improvement using the PAG loss, we compared it against the baseline model, having as the only differences the number of epochs and the presence/absence of the regularization term introduced by the PAG requirement.

In fig. 3.2 it can be noticed the difference in robustness against a simple FSGM attack with multiple values of α. In particular, in table 3.1, the results of multiple attacks are reported. Some of them have been executed using the AutoAttack suite [Croce and Hein, 2020]. It's clear that the robustness improved by a lot in all tests, but the FSGM with a low value of α, in which the baseline model was already quite robust, while the same cannot be told with high values of perturbations. We are enforcing Perceptually Aligned Gradients by performing a **double backward pass**, introducing ∇_x in the loss. When we perform a double backward pass on a model, we are essentially computing second-order derivatives of the loss function with respect to the model parameters. This allows the model to learn a smoother function and be more robust against adversarial attacks based on some perturbations x' of the input x.

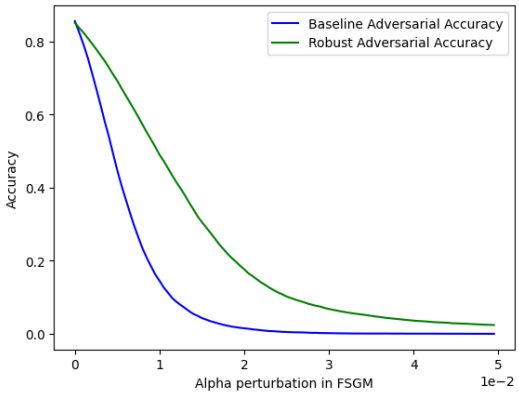


Figure 3.2. Robustness against various α values in FSGM

The results in table 3.1 may be motivated by the following mental model. When we apply a generic targeted gradient-based attack, generally we want to create the adversarial image with a perturbation derived from the gradients with respect to the input x. The big change here is when these gradients actually present features of the target attack class. From a high-level logical perspective, it can be seen as:

	APGD		Square	FSGM		
	$\varepsilon = 1e-3$	$\varepsilon = 0.5$		$\alpha = 1e-3$	$\alpha = 5e-3$	$\alpha = 1e-2$
Baseline	47.9%	41.7%	48.2%	79.2%	44.8%	14.5%
Robust	**75.8%**	**70.4%**	**76.6%**	**82.4%**	**69.2%**	**49.0%**

Table 3.1. Adversarial attacks comparing the two models

$$\begin{aligned} x' &= x + \alpha \nabla_x \mathcal{L}(x; y_{\text{attack}}; \theta) \quad \text{such that } y_{\text{attack}} \neq y_{\text{true}} \\ &\approx x + \alpha(x_{\text{attack}} - x) \quad \text{since we minimize } d_{cos}(\nabla_x \mathcal{L}(x; y_{\text{attack}}), x_{\text{attack}} - x) \\ &= x + \alpha \cdot x_{\text{attack}} - \alpha \cdot x \\ &= (1 - \alpha)x + x_{\text{attack}} \end{aligned} \tag{3.4}$$

This small sketch of calculations can already highlight how the perturbed sample, when a classifier has Perceptually Aligned Gradients, it is giving to the attack sample x' the actual characteristics of a real sample of the target attack class. It may be a mental model of the justification of why this approach gives more robustness, varying the value of α parameter.

Moreover, we can plot the activations of this classifier right before the Softmax function, to see how they are separated (fig. 3.3). Recall that this plot may be used as a certainty measure, where the more points are to the center, the more uncertainty there is. Because of that, in our ideal scenario we have no point at the center, meaning maximum certainty in the prediction, which is unpractical but gives a measure of what the optimal solution may look like.

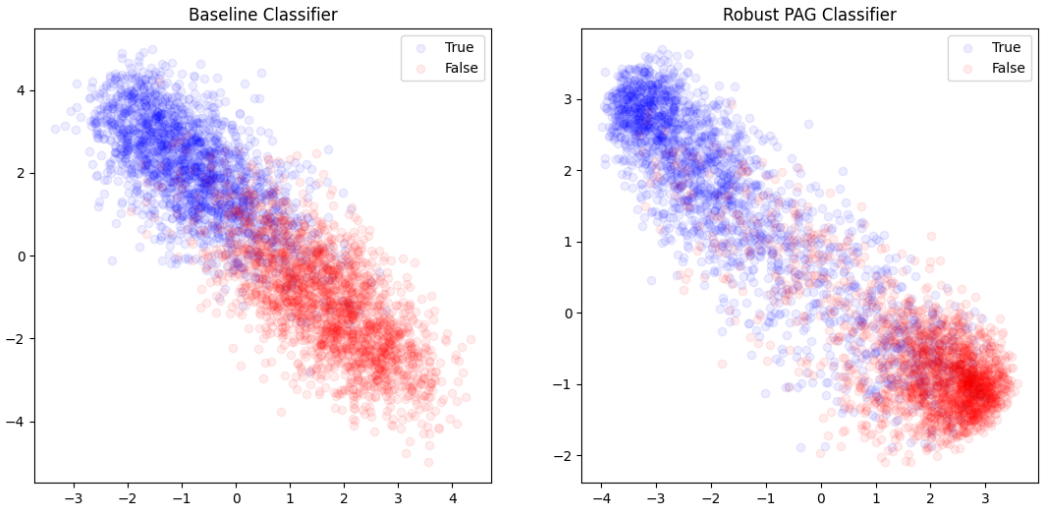
3.1.2 PAG Variants

We investigated how different PAG loss variants would affect the robustness and other metrics of this experiment. We tested several models:

- *baseline*, with the Cross-Entropy loss only
- *pag-score-similar-samples*, which implements the PAG formulation as in eq. (3.1)
- *pag-score-similar-features*, which computes the similarity of the transpose of the matrix previously mentioned, resulting in the input batch $\mathbf{X} \in \mathbb{R}^{N \times D}$, a set of random samples $\mathbf{G} \in \mathbb{R}^{N \times D}$, computing the PAG loss as $\mathcal{L}_{cos}(X^T, G^T)$ feature-wise, instead of comparing sample-wise, thus giving the name of this method
- *pag-identity*, which imposes the $\nabla_x \mathcal{L}$ to be cosine-similar to x itself

Binary classifier

When dealing with a binary classifier, we could observe that, on the contrary of our expectations, the *pag-score-similar-features* method is the one that works the



(a) Baseline model: the point clouds are near each other

(b) PAG-regularized model: the point clouds are far from each other

Figure 3.3. Activations before softmax

best, with surprisingly good results (table 3.2), while not having any theoretical background to support it.

This led us to think again about this simplification of using a mere binary classifier, since it may be too small to let us explore the potential of Perceptually Aligned Gradients applied to text classification.

However, in the binary classification example, it is interesting to plot the logits of the classifier, before softmax, and see how they are placed in a 2D environment, changing the loss function, as can be seen in fig. 3.4. It is worth remembering that the more distant the center of mass of the classes are, and the more certainty there is in their prediction by the classifier.

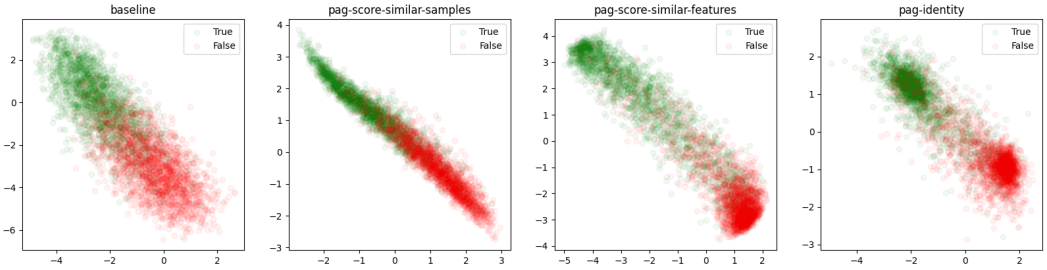


Figure 3.4. Logits of the binary classifier, according to the loss function used

Multiclass classifier

In order to have better understanding of the potential of applying PAG to text embeddings for classification, we decided to introduce more entropy in the task, adding more classes. To do that, we switch to a subset of the `amazon_review_multi`

	APGD		Square	FSGM		
	$\varepsilon = 1e-3$	$\varepsilon = 0.5$		$\alpha = 1e-3$	$\alpha = 5e-3$	$\alpha = 1e-2$
Baseline	47.9%	41.7%	48.2%	79.2%	44.8%	14.5%
pag-scores similar-samples	57.0%	42.7%	56.5%	80.5%	68.1%	49.1%
pag-scores similar-features	**71.8%**	**67.1%**	**73.1%**	**81.7%**	**71.4%**	55.3%
pag-identity	43.5%	43.1%	43.8%	**82.1%**	67.5%	**56.4%**

Table 3.2. Adversarial attacks comparing the PAG variants - **binary** classifier

dataset ([Keung et al., 2020]), considering as classes the intersection of some languages (English, German, Spanish and French) and some reviews ratings (1, 3, 5), for a total of 12 classes, with 800 train samples each. The embeddings are obtained as usual, taking the hidden state associated with the `[CLS]` token, adopting the `distilbert-base-multilingual-cased` model [3]. The models are trained using the same hyper-parameters, such as the λ factors for the PAG regularization part of the loss, the number of epochs, and so on.

It finally gave us the results we expected, as observable now in table 3.3. This highlights how the example of the binary classifier was just a random chance of having the right situation for the transpose cosine similarity to succeed, while in a more complex situation it does not work, as expected by the theoretical foundations and reasoning.

	APGD		Square	FSGM		
	$\varepsilon = 1e-3$	$\varepsilon = 0.5$		$\alpha = 1e-3$	$\alpha = 5e-3$	$\alpha = 1e-2$
Baseline	36.5%	31.2%	36.3%	60.3%	27.3%	8.9%
pag-scores similar-samples	**48.1%**	**45.0%**	**49.3%**	**62.7%**	**43.5%**	**25.7%**
pag-scores similar-features	28.5%	24.3%	28.9%	**62.3%**	40.1%	20.6%
pag-identity	28.3%	25.0%	27.2%	60.1%	25.7%	8.0%

Table 3.3. Adversarial attacks comparing the PAG variants - **multiclass** classifier

3.2 Applying PAG to LLMs

In the context of Large Language Models, it is not directly applicable the formulation of PAG as seen in eq. (3.1) to LLMs. The main issue is that an image is a continuous tensor that already contains all the information we need to clearly determine its predicted class. In sequence processing this is not the case anymore, since the output will depend on the overall sequence instead of only just a single item or sample from it. More importantly, the input tokens in the sequence are encoded are meaningless one-hot vectors.

This makes the gradient calculation on the input pretty challenging because:

[3] `https://huggingface.co/distilbert/distilbert-base-multilingual-cased`

- calculating the gradients with respect to the input tokens does not provide the context of the overall sentence; to have a comparison with the image space, we can consider it as a single pixel, which of course cannot provide sufficient information about the input to determine the class (which is the next token in the sequence, in the context of an LLM)
- calculating the gradients with respect to an hidden state at a certain layer i will require us to preload an index of possible hidden states for real sentences that have as a next token the one we want - this follows the vision of an LLM as a multiclass classifier over the last hidden state
- the number of classes to iterate on is huge, being the vocabulary size generally in the order of hundred of thousands of possible tokens; this makes the PAG loss applicable on images computationally infeasible when dealing with this order of magnitude of possible classes, thus entropy in the classification task.

An approach may be to approximate the PAG iteration over the classes, limiting our search to a restricted amount of classes, picking K random next possible tokens among all the vocabulary V. This will limit the computational resources required for the experiments, while also having the downside of not exploring possible directions for the gradients with some under-trained classes.

Multiple approaches can be considered when dealing with what to compute the gradients on, like the embedding vector of the single tokens or the hidden states of the sentence with respect to the last token in the sentence, which is the one that is more taken into account when predicting the next token during inference.

Consider:

- the sample in the batch: $x_t =$ `the pen is on the`, with label $y_t =$ `table`
- a random sample: $x'_t =$ `using wood, you can build a`, with the same label $y_t =$ `table`
- another random sample: $x_f =$ `lot of trees form a`, with the label $y_f =$ `forest`

Following the naming scheme in the fig. 3.5, we can compute the PAG loss in several different ways when dealing with text sequences.

$$\begin{aligned}\mathcal{L}_{\text{PAG}}(x_t) = & \, d_{cos}(\nabla_{e_t}\mathcal{L}_{\text{CE}}(x_t, y_t), e'_t - e_t) \\ & + d_{cos}(\nabla_{e_f}\mathcal{L}_{\text{CE}}(x_t, y_f), e_f - e_t) \\ & + d_{cos}(\text{... over other classes ...})\end{aligned} \tag{3.5}$$

$$\begin{aligned}\mathcal{L}_{\text{PAG}}(x_t) = & \, d_{cos}(\nabla_{h_t}\mathcal{L}_{\text{CE}}(x_t, y_t), h'_t - h_t) \\ & + d_{cos}(\nabla_{h_f}\mathcal{L}_{\text{CE}}(x_t, y_f), h_f - h_t) \\ & + d_{cos}(\text{... over other classes ...})\end{aligned} \tag{3.6}$$

In the formulas above, the embedding refers to the last token, which is the one right before the predicted one. The main point of this brief explanation is to show

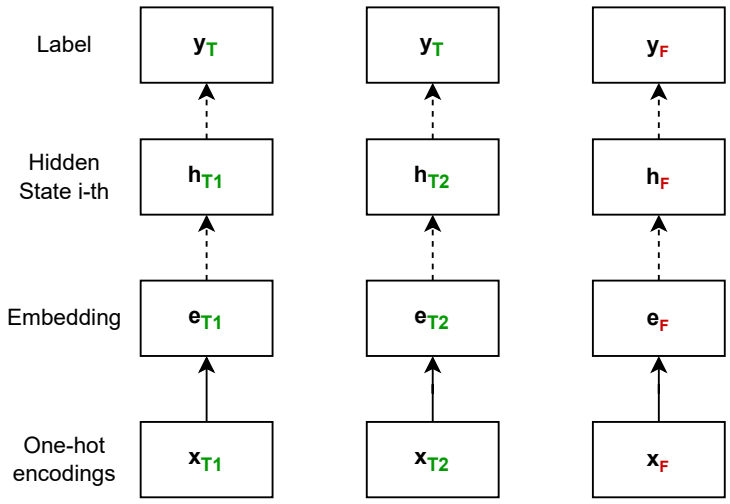


Figure 3.5. Naming convention of the internal tensors of an LLM during a forward pass

how it is not straightforward to try applying the well-known concept of PAG from images to discrete sequential text. We may prefer to go down to the embedding layer (eq. (3.5)), where the notion of the overall sentence is absent, or staying in an intermediate hidden layer i (eq. (3.6)), even tough there are tons of other possible, unexplored in literature, alternatives.

When dealing with gradients computed on the embeddings vectors we may think that they carry no information about the sentence, as the vector itself does not. However, it can be noticed that the gradients of the loss with respect to the embedding vector of a token i, contain information about both the past and the future tokens. Indeed, in fig. 3.6, it is clear why the gradient on every embedding token of the full sentence is affected by the change of even only one input position, even keeping the labels unchanged. The change of token i, influences its own logit prediction i but also the future hidden states at time step $t : i < t \leq n$, because of the Masked self-attention layers present in every decoder transformer network. The past tokens are also affected, even with masked self-attention. This can be explained using the chain rule of derivatives: the gradient of the loss with respect to an embedding will include in the equation also the partial derivative of the loss with respect to the affected logit i, thus changing the value of the gradient on the single embedding for $t : t < i$. Note that: $\frac{\partial \mathcal{L}_i}{\partial \text{logits}_i} \propto \text{softmax}(\text{logits}_i)$, which are altered by the changed token i.

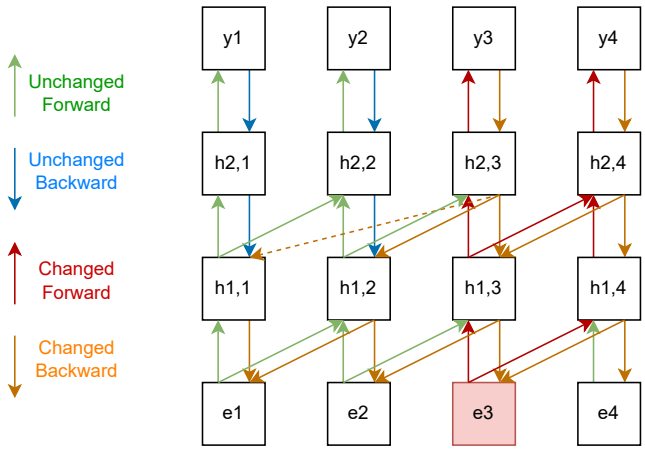


Figure 3.6. Simplification of gradients flow in an LLM changing only token e_3. In the forward pass, only future hidden states are affected. In the backward pass, the change propagates to every embedding token.

Chapter 4

Inverse Language Modeling

Large Language Models (LLMs) nowadays excel in natural language tasks, allowing us to perform multiple NLP tasks with a single foundation model, often beating previous state-of-the-art solutions. Thanks to recent studies in reasoning [Xu et al., 2025], LLMs can solve even more complex tasks, which was not possible before when they had to give an answer immediately after the user prompt. Yet LLMs are prone to hallucinations. Moreover, they are sensitive to adversarial inputs, as observed in the introduction. These phenomena highlight a possible risk for backdoors and bad behaviour induced by a bad actor using a model. Because of all of that, it is obvious the need for Adversarial Training (AT) also for LLMs, similarly to what happened in the past to image deep classifiers, leaving adversarial text perturbations an open challenge.

In this context, *robust* means being less sensitive to input prompts, and *grounded* means "making LLMs know what they have been asked about", not simply predicting the most likely next token in the sentence. Indeed, LLMs do not know what they are talking about [Guo et al., 2024], supporting the idea of LLMs as complex stochastic parrots.

In this chapter, we introduce **Inverse Language Modeling** (ILM), which takes inspiration from years of progress on adversarial robustness in deep classifiers. LLMs are trained in forward-mode, predicting the next token in a self-supervised way. ILM, instead, focuses on the idea that should be possible to have a meaning of the spoken sentence even backwards: given $\mathbf{y}$ (the answer or the LLM proposed continuation), is the LLM aware of the text prompt $\mathbf{x}$ it was conditioned on? This does not mean to train the LLM with a reversed corpus, but instead predicting based on the value of the gradients on the input, as in fig. 4.1. This should be possible because of the fact illustrated in fig. 3.6, where we observed that the gradients can have the notion of the overall sentence.

This method could potentially allow for a security red team to better investigate and study malicious prompts $\mathbf{x}$ that could lead to the generation of an undesired output $\mathbf{y}$, by simply inverting the LM, after being correctly fine-tuned as in the procedure illustrated in this chapter.

We will use ILM at training time, while at test time we exploit the GCG algorithm to find, given an original text prompt $\mathbf{x}$ and the completion $\mathbf{y}$, a new nonsensical $\mathbf{x}^\star$ such that the loss $\mathcal{L}(\mathbf{x}^\star, \mathbf{y}; \boldsymbol{\theta}) \ll \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$, where $\mathcal{L}$ is the next-

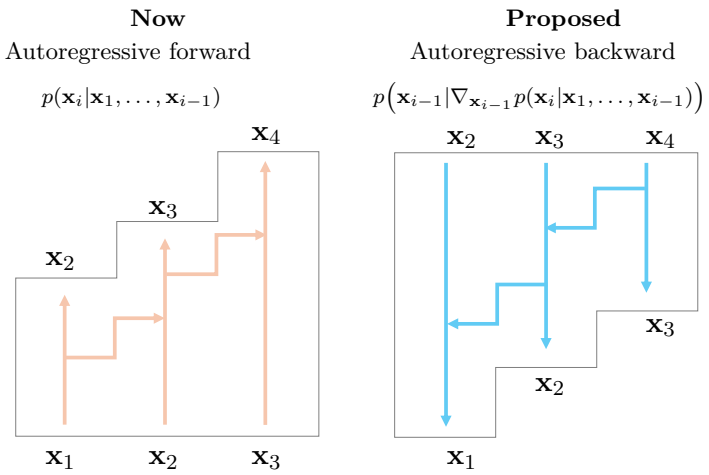


Figure 4.1. Inverse Language Modeling

token prediction loss of the LLM and $\boldsymbol{\theta}$ are LLM's parameters. We prove there exists a nonsensical prompt $\mathbf{x}^\star$ that "connects" better to $\mathbf{y}$ (lower loss) than the natural $\mathbf{x}$.

4.1 Inverting the first token

As a preliminary experiment to have some experimental data to discuss to support our ideas, but also not to consume too much computational power for the first experiment, we decided to train our LLM to predict the first token of a sentence.

To keep the entropy in the token classification somehow limited, we adopted the **TinyStories dataset** [Eldan and Li, 2023] [1], which allows us to work with synthetically generated (GPT-4) short stories that only use a small vocabulary.

In order to perform flexible experiments, even the tokenizer is trained by ourselves, allowing us to set a specific vocabulary size to reach applying the Byte-Pair Encoding (BPE) algorithm [Gage, 1994]. Then, the model is trained from scratch, reaching good results in just a bunch of epochs, due to the simplicity of the corpus we used, proceeding by using, at first, the same hyperparameters of the original experiment [2].

To implement the inversion goal, we use the gradients with respect to the embeddings of the input sequence, in the same way as if it was the last hidden state. This has a twofold effect:

1. The last hidden state has necessary knowledge of the overall sentence to predict the next token, so the gradients of the embeddings

2. Because of **weight tying** [Press and Wolf, 2017], the weights of the classification head and the embeddings layer are exactly the same at any step of the training process - this induced parallelism is exploited here even further

[1] `https://huggingface.co/datasets/fhswf/TinyStoriesV2_cleaned`
[2] `https://huggingface.co/raincandy-u/TinyStories-656K#full-training-arguments`

Because of the weight tying parallelism and recalling fig. 3.6, we can threat ∇_{e_i} to predict the token x_i. More formally, the backward prediction $\hat{\mathbf{y}}$ is obtained as in eq. (4.1). To perform this task, we need to add a new regularization term to the standard CE loss function. The final loss is expressed in eq. (4.2).

$$\begin{aligned} \mathcal{L}_{CE} &= CE(\mathbf{y}_{\text{true}}, \mathbf{y}_{\text{pred}}) \\ \mathbf{g}_i &= \text{LayerNorm}(\nabla_{e_i} \mathcal{L}_{CE}) \\ \mathbf{z}_i &= \mathbf{W}_{\text{LM\_head}}\, \mathbf{g}_i \\ \hat{\mathbf{y}}_i &= \text{softmax}(\mathbf{z}_i) \end{aligned} \tag{4.1}$$

$$\mathcal{L} = \underbrace{CE(\mathbf{y}_{\text{true}}, \mathbf{y}_{\text{pred}})}_{\text{Forward: from the input x, encode y}} + \underbrace{\lambda\, CE(\mathbf{x}, \hat{\mathbf{y}})}_{\text{Backward: from y, decode back x}} \tag{4.2}$$

This approach has the potential downside of being different between training and test time. Looking more in deep at the formulas, it is visible that the backward prediction target $\mathbf{x}$ is already given, while at test time we have to replace it with a random token, being a valid vocabulary token or the `[PAD]` padding special token. However, this is efficient for training since we do not need multiple forward passes to the model, requiring us to save in memory only one instance of the activations because of the single forward pass.

4.1.1 Considerations on the classification strategy

A question that may arise is why we decided to multiply the gradients by the LM head weights matrix and applying softmax. A naive approach could be to directly use $\mathbf{g}_i$ as the embedding of the target token and classify the correct token simply by looking at the token with the nearest embedding, using L_2 distance or cosine similarity. This has several major downsides:

1. It would lead to the same problem observed with the toy example in section 3.1.2 with the so called "PAG identity" approach, where the gradients ideally reconstruct the input
2. It would not have exploited the unique symmetry introduced by weight tying, for which the weights matrix of the embeddings layer and the LM head are exactly the same
3. It would not have lead to a probability distribution over the tokens in the vocabulary

Instead, treating $\mathbf{g}_i$ exactly the same as if it was the last hidden state that predicts token $\mathbf{x}_i$, allows us to map this gradient to an image of something that is linked to the expected outcome, instead of directly the input. Moreover, we can apply the same techniques that are commonly used in forward test mode, like Beam Search, Top-K Sampling and so on and so forth. The parallelism is highlighted in the illustration in fig. 4.2.

Using a simple measure of distance would make the loss **non-contrastive**. This means that there is a much higher risk of **collapse** of the hidden states obtained

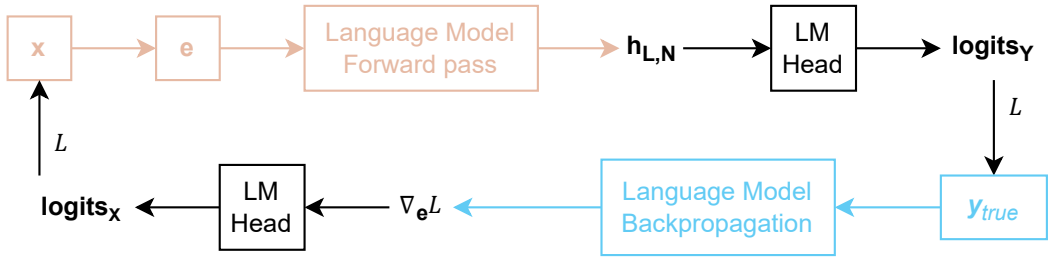


Figure 4.2. Parallelism between the last hidden state and the gradients on the embeddings

from the gradients during the backward operational mode. It is clearly justified by the meaning of the loss itself: it imposes the embeddings of a target token $\mathbf{e}_i$ to be as similar as possible to the gradients $\nabla_{\mathbf{e}_j}\mathcal{L}$. However, nothing is said about the dissimilarity between embeddings that should not relate each other. The optimal solution in this setting would be to assign the same value to every embedding. Since this would just be a single regularization term in the loss (also the CE loss of the language model participates in the overall loss), we only talk about a high risk of going in the direction of a collapse, not a certainty.

On the other hand, having a tensor that can be interpreted as the output logits, we can introduce the **softmax** as a standard part of the cross-entropy loss, introducing the contrastive part previously missing. This allows to keep clear separation hyperplanes between the different hidden states that lead to a specific token prediction.

Aside from that, it must be highlighted that all these methods do not work if we omit the **LayerNorm** function application. We questioned why this was the case, so we started to delve deeper into the sequence of applied functions, in a standard forward pass, before the matrix multiplication with the final LM Head. Based on the internal code analysis, it was observed that the final hidden state includes a value previously passed to the `T5LayerNorm`[3] layer, which rescales the variance of the input tensor, without subtraction of mean and the bias parameter.

$$\begin{aligned} \sigma &= \sqrt{\frac{1}{D}\sum_{i=0}^{D}\mathbf{x}_i^2 + \varepsilon} \\ \mathbf{x}' &= \alpha\frac{\mathbf{x}}{\sigma} \end{aligned} \tag{4.3}$$

4.1.2 Variants

As immediately visible, this is not a simple problem. Because of that, multiple approaches have been implemented and compared, here discussed. The common denominator between all of them is the approach highlighted in the previous section: the gradient on the tokens embedding ∇_{e_i} is forwarded to the LM Head $\mathbf{W}$ classifier. The difference lies in how this gradient is obtained and used in the final $\mathcal{L}$ loss function. To briefly give them a name:

[3] `https://github.com/huggingface/transformers/blob/fa3c3f9cab1c45b449bd57e238c511c79637e314/src/transformers/models/t5/modeling_t5.py#L241`

- **identity**: compute ∇_{e_i} on a simple forward pass with the usual Cross-Entropy Loss, then impose the logits obtained from $\mathbf{W}^T \cdot \nabla_{e_i}$ to match the token $\mathbf{x}_i$
- **inv-first**: after replacing the first input token $\mathbf{x}_0$ with `<|pad|>`, leaving the labels unchanged, do a forward pass and compute ∇_{e_i} (now, this gradient cannot carry information about the token $\mathbf{x}_0$ to predict)
- **bert-like**: replace 10% of tokens, randomly selected, both as input tokens and target label tokens, then the same prediction procedure is followed; this method is heavily inspired by BERT [Devlin et al., 2019], in which the model should understand the meaning of the sentence - here, the LLM is supposed to be an autoregressive LM in the forward pass, while being a model like BERT when computing the backward pass.

4.1.3 Inverse LM Evaluation

The evaluation procedure consists of multiple steps:

1. Validation loss identifies the value of the loss in the same setting as the training loop
2. Validation accuracy indicates the prediction capabilities of the model in the same setting as the training, serving as baseline indicators of model fit and predictive performance
3. Inverse LM accuracy measures the ability of a model to reconstruct a masked token $\mathbf{x}_i$ from its gradients, given the remaining context; this provides a direct assessment of the backward prediction mechanism.

To compute the Inverse LM accuracy, a token $\mathbf{x}_i$ must be predicted, at position i. The input of the model is the original sentence, where the i-th token is replaced with a placeholder, concatenated by all the following tokens. Formally:

- we aim to predict the i-th token, where $1 \leq i < N$
- given an input sentence $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{N-1}$
- the target labels $\mathbf{y} = \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N$
- the input is $\mathbf{x}' = \texttt{<|pad|>}, \mathbf{x}_{i+1}, ..., \mathbf{x}_{N-1}$
- the CE loss is computed with the target labels defined as $\mathbf{y}' = \mathbf{x}_{i+1}, ..., \mathbf{x}_{N-1}, \mathbf{x}_N$

The standard cross-entropy loss is used with the masked input $\mathcal{L}_{\text{CE}}(\mathbf{x}', \mathbf{y}'; \theta)$ and, from this, the embedding is computed as ∇_{e_i}. Finally, in order to predict the token at position k, we take the softmax$(\mathbf{W}^T \cdot \nabla_{e_i})$. At this point, you can choose to take the argmax or to sample from the output distribution as a traditional language model. Note that this Inverse LM evaluation procedure is the same for *all* the variants, since this is the goal we aim to reach at the end of the day.

The placeholder token can be set in different ways. In these Inverse LM evaluations, it was set once to the `<|pad|>` token, while models have also been evaluated

having the placeholder token set to the most frequent token that, in the training set, precedes $\mathbf{x}_{k+1}$ (building a **reverse Bigram** modeling the input text distribution). This is an experiment to understand if giving some kind of hint about the possible token, using a simple bigram language model, may improve the performance during inversion.

In tables from 4.1 to 4.8 the results of these experiments are reported.

	Top-1	**Top-2**	**Top-3**	**Top-4**
Baseline	0.00%	0.00%	0.00%	0.00%
Identity	1.12%	2.87%	2.94%	4.49%
Inv-first	**39.46%**	**98.39%**	**99.92%**	**99.93%**
Bert-like	32.58%	34.17%	34.29%	38.71%

Table 4.1. Inverse LM accuracy for the **first token**, replaced with **PAD**

	Top-1	**Top-2**	**Top-3**	**Top-4**
Baseline	0.00%	0.00%	0.00%	0.01%
Identity	0.15%	1.25%	5.73%	8.39%
Inv-first	0.02%	0.05%	0.07%	0.08%
Bert-like	**10.17%**	**20.78%**	**30.65%**	**35.84%**

Table 4.2. Inverse LM accuracy for the **second token**, replaced with **PAD**

	Top-1	**Top-2**	**Top-3**	**Top-4**
Baseline	0.00%	0.00%	0.00%	0.01%
Identity	0.18%	0.46%	0.51%	0.88%
Inv-first	0.51%	1.71%	2.94%	3.41%
Bert-like	0.13%	0.21%	0.23%	0.33%

Table 4.3. Inverse LM accuracy for the **third token**, replaced with **PAD**

Further commenting the results here listed, we immediately see that the baseline consistently performs badly, even worse than random chance (given by $\frac{1}{|\mathcal{V}|}$).

Discussing the performance of the models when varying the initialization, it is clearly understandable that models perform poorly if the evaluation procedure used is different from the one they have been trained on. Specifically speaking, we observe that models trained with the first token being `<|pad|>` are failing when the first token they compute the gradients on is indeed another one, whether obtained

	Top-1	Top-2	Top-3	Top-4
Baseline	0.00%	0.00%	0.00%	0.01%
Identity	**6.92%**	**10.30%**	**10.52%**	**12.66%**
Inv-first	0.64%	0.97%	1.17%	1.18%
Bert-like	0.95%	1.90%	2.00%	2.84%

Table 4.4. Inverse LM accuracy for the **fourth token**, replaced with **PAD**

	Top-1	Top-2	Top-3	Top-4
Baseline	0.00%	0.00%	0.00%	0.00%
Identity	22.40%	45.21%	**83.55%**	**83.57%**
Inv-first	0.11%	0.18%	0.20%	0.27%
Bert-like	0.01%	0.02%	0.02%	0.02%
Reverse bigram model	**60.43%**	**84.17%**	**87.39%**	**86.84%**

Table 4.5. Inverse LM accuracy for the **first token**, replaced using **bigram**

	Top-1	Top-2	Top-3	Top-4
Baseline	0.00%	0.00%	0.00%	0.00%
Identity	11.97%	**42.22%**	43.80%	75.51%
Inv-first	0.00%	0.00%	0.00%	0.01%
Bert-like	0.00%	0.00%	0.00%	0.00%
Reverse bigram model	**23.79%**	28.17%	**92.13%**	**95.87%**

Table 4.6. Inverse LM accuracy for the **second token**, replaced using **bigram**

	Top-1	Top-2	Top-3	Top-4
Baseline	0.00%	0.00%	0.00%	0.00%
Identity	66.54%	67.22%	67.38%	67.61%
Inv-first	0.13%	0.58%	0.62%	0.69%
Bert-like	0.00%	0.00%	0.00%	0.04%
Reverse bigram model	**84.92%**	**89.29%**	**89.97%**	**90.29%**

Table 4.7. Inverse LM accuracy for the **third token**, replaced using **bigram**

	Top-1	Top-2	Top-3	Top-4
Baseline	0.00%	0.00%	0.00%	0.01%
Identity	17.99%	34.28%	35.07%	55.25%
Inv-first	0.02%	0.04%	0.04%	0.05%
Bert-like	0.00%	0.00%	0.00%	0.04%
Reverse bigram model	**22.94%**	**32.19%**	**89.33%**	**90.14%**

Table 4.8. Inverse LM accuracy for the **forth token**, replaced using **bigram**

from a simple **bigram** language model or a random choice. In a similar way, the identity model variant fails when the input is the PAD token.

Going deeper into the understanding of the results obtained with the `identity` model, when its **initialization token** is obtained from the bigram model, the accuracy is very high, but the bigram itself can perform very well. A question raised is whether or not the initialization plays a crucial role in reaching this result. We tested initializing with a **random token** instead, and the identity grad model performs almost as badly as the baseline. Does this directly imply that the `identity` model is useless without the bigram, and it is only able to get the same input it got? At the end, we found out that the initialization strategy is essential to keep the Identity model perform well, as we will see in all future results.

Finally, we tested that these models are still working fine in forward mode, without suffering from **performance degradation**. Checking out the perplexity during training and validation, it has been observed that the performance of the custom models with the regularization term on the gradients $\nabla_{\mathbf{e}}\mathcal{L}_{\text{CE}}$ does not penalize the model's ability to speak fluently during the standard usage in forward mode. This outcome is noteworthy, as adversarial training in literature often necessitates additional parameters or extended training to achieve comparable forward-mode performance, since part of the model's capacity is effectively devoted to satisfying the adversarial objective.

	Completion for "Once upon a time, "
Baseline	there was a little girl named Lily. She loved to play with her toys and eat yummy food
Identity	there was a little girl named Lily. She had a big, red ball that she loved to play with.
Inv-First	Sally and her mommy were in the kitchen. They were very excited. As they were cooking
Bert-like	there was a little girl named Lily. She had a big, red ball that she loved to play with.

Table 4.9. Example completion for the given prompt, in forward mode

The models listed in these first chapters were trained for **5 epochs only**, on the small vocabulary and corpus described above, with the objective to see some signal of the potentials of these techniques and have an early comparison, avoiding wasting too many resources on something that does not even give us some interesting

	Top-1	Top-2	Top-3	Top-4
Baseline	**33.12%**	**61.92%**	**83.34%**	**86.53%**
Identity	31.13%	57.77%	78.51%	82.02%
Inv-first	27.63%	52.00%	69.02%	73.08%
Bert-like	31.20%	58.26%	78.16%	81.72%

Table 4.10. Forward-mode LM accuracy, average across all tokens of validation set

preliminary results. This can be seen also in the loss components, which slowly improve in the validation set, showing that it is possible to improve making the model bigger and training for longer, as observable from the plot in fig. 4.3.

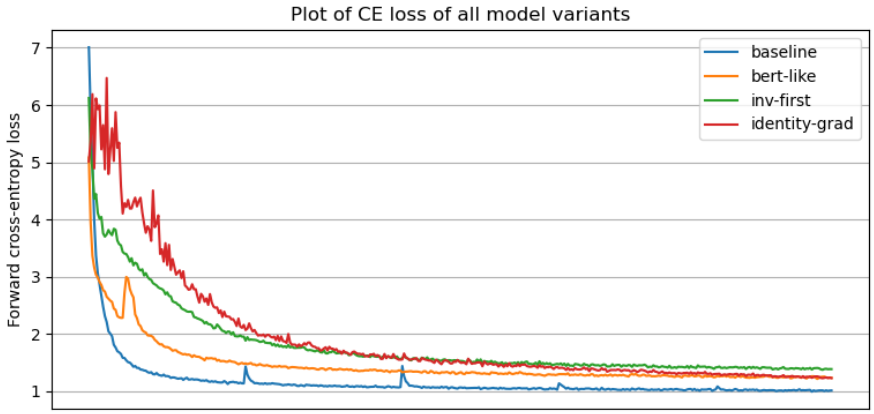


Figure 4.3. Cross-entropy loss value slowly improving during training

4.1.4 First tokens distribution

Looking at the results obtained when predicting the very first token with `inv-first`, we may ask if this result is really so good or there is something hiding behind those accuracies (table 4.1). Indeed, investigating the entropy that classifying the first token will have, we must see the distribution of the possible outcomes predicting that token in our particular dataset. This is visible in fig. 4.4. In the plot, tokens with less than 5,000 occurrences in the training dataset have been hidden for improved readability. We can see that the first tokens are extremely imbalanced: summing up the occurrences of "*Once*" and "*One*" as the tokens in the very first position, we have covered almost every train sentence, which is pretty bad in our situation and will likely forge our results. Moreover, this motivates the top-2 accuracy for `inv-first` when predicting the first token, since having those two tokens as the most likely to be predicted, already makes the model performance result in a nearly-perfect accuracy.

In fig. 4.5 we can see that the most interesting and variable parts of text appear around the 20th token, where the entropy reaches values near 8. Recall that the entropy in that plot is computed according to the formula of the Shannon entropy, where i is the token index, $\mathcal{V}$ is the vocabulary and $p_i(x)$ is the probability of having the token x at index i of a randomly picked sentence in the training dataset:

$$H_i(x) = -\sum_{x \in \mathcal{V}} p_i(x)\ log_2(p_i(x) + \varepsilon)$$
$$\varepsilon = 10^{-4} \text{ For numerical stability of the } log \tag{4.4}$$

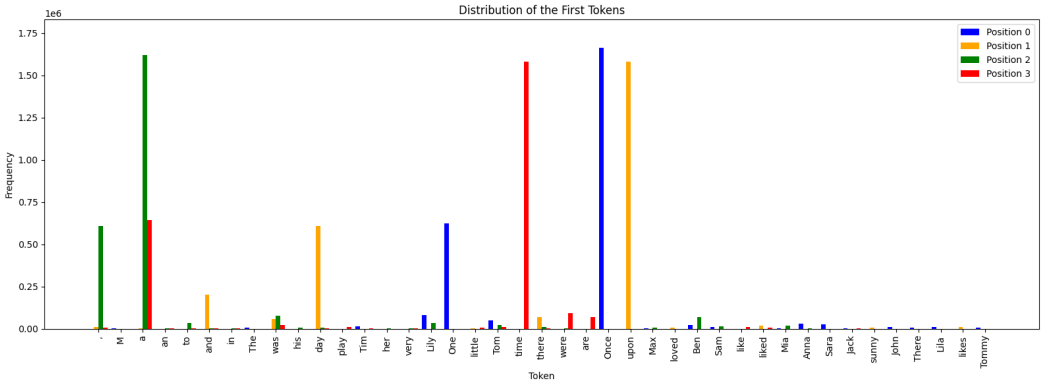


Figure 4.4. Distribution of tokens in the first positions of the sentences in the training set

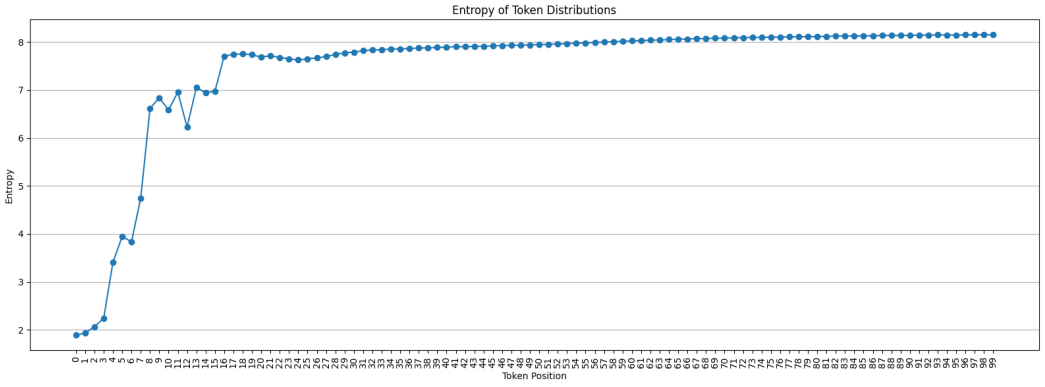


Figure 4.5. Entropy of tokens distribution at each index of the sentences in the training set

4.2 Inverting multiple tokens

As a direct consequence of the results in fig. 4.5, it is mandatory to expand the evaluation on inverting multiple tokens autoregressively and going in middle positions in the text samples, instead of predicting the very first tokens as in the preliminary experiments.

Algorithm 1 Autoregressive Inversion Evaluation with Beam Search

Require: Input sample $\mathbf{x}$ of length n, beam size b, split position k
Ensure: Inverted prefix $\mathbf{x}_{\text{inv}}$

$\mathbf{x}_p \leftarrow \mathbf{x}_{0:k}, \quad \mathbf{x}_s \leftarrow \mathbf{x}_{k:n}$
Initialize beam set $\mathbf{X} \leftarrow \{\mathbf{x}_s\}$
while inverted prefix not sufficiently long **do**
    for each sequence $\mathbf{x}' \in \mathbf{X}$ **do**
        Compute the b most probable previous tokens for the current position
        Extend $\mathbf{x}'$ with each candidate token
    end for
    Update $\mathbf{X}$ with the b sequences having lowest perplexity
end while
return $\mathbf{x}_{\text{inv}} \leftarrow \arg\min_{\mathbf{x}' \in \mathbf{X}} \text{Perplexity}(\mathbf{x}')$

The procedure follows a beam-search strategy, as detailed in algorithm 1, where candidate prefixes are iteratively expanded and filtered by perplexity until a coherent reconstruction emerges.
In the evaluation process, we considered only the combination of initialization strategy and model variant used in the specific training process. This means that the `Identity` has the unknown token initialized using the simple bigram, while other variants have it set to `<|pad|>`, since they reflect the same initialization strategy used during training. Of course, we cannot initialize the `Identity` model during inversion with the real token, as in training, because we do not know it yet. However, the best approximation we can do is to use a bigram model, which is pretty simple but also powerful to help the model invert better than starting from a totally random token or using a fixed `<|pad|>` because it has never observed it during training.

4.3 Gradients as directions

In this section, we are going to propose a new model variant that follows the theoretical and more natural concept of gradients as directions. Interestingly, we have already mentioned something of this kind in the introduction to this project: Perceptually Aligned Gradients (PAG, Ganz et al. [2023]) can be interpreted as vectors that point in a direction chosen by the researcher, rather than being left free to go wherever they want.

Mathematically, gradients of the loss with respect to a component of the overall function, whether it be a weight parameter $\mathbf{W}$, the input sample $\mathbf{x}$, or any other intermediate value, indicate the delta to go to increase the loss. Indeed, when training a simple neural network, the usual way is to use an optimizer to change the weights in the direction of the negative gradients of the loss function, multiplied by a learning rate and other advanced techniques, like momentum and so on.

The gradient on a token at some position i is definitely interpretable, in a sequential model, as the direction to improve the current i-th token, to improve the overall loss function, given that its value has already "seen" the future elements to

the sequence.

The main point of this reasoning is that it would be twisted to use the gradients as if they were the correct weights directly: no one would train a neural network as $\mathbf{W}_{t+1} = \nabla_{\mathbf{W}_t}\mathcal{L}$, but the simplest and more general way to effectively train it is to change the weights as $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta\nabla_{\mathbf{W}_t}\mathcal{L}$, thus exploiting gradients as directions.

The overall question that follows from this argument is: what could happen if we make the gradients concerning the input embedding be considered as a delta in the direction of the token itself? To implement that, we take the already discussed model variants and apply a modification to its classification and training formulations. Indeed, we classify on $\mathbf{x} - \nabla_{\mathbf{x}}\mathcal{L}$, instead of directly on $\nabla_{\mathbf{x}}\mathcal{L}$ as we did in the previous sections of this chapter. This allows the model to correctly use the gradients as directions towards what's a better value of the embedding token to lower the final loss. The full classification strategy is summarized in eq. (4.5).

$$\begin{aligned} \mathcal{L}_{CE} &= CE(\mathbf{y}_{\text{true}}, \mathbf{y}_{\text{pred}}) \\ \mathbf{g}_i &= \text{LayerNorm}(\mathbf{x}_i - \nabla_{\mathbf{x}_i}\mathcal{L}_{CE}) \\ \mathbf{z}_i &= \mathbf{W}_{\text{LM\_head}}\ \mathbf{g}_i \\ \hat{\mathbf{y}}_i &= \text{softmax}(\mathbf{z}_i) \end{aligned} \tag{4.5}$$

Chapter 5

Inverse Language Modeling Evaluation

5.1 Inversion Evaluation

In this section, we analyze all the model variants discussed so far through the lens of their text inversion capabilities. To be as fair as possible, also coherent with the findings in section 4.1.3, each variant is tested considering the initialization strategy to be the same on which it was trained. This means that, for instance, models where the specific token to receive gradients on was masked, it will be masked again as well in the evaluation procedure that follows. However, in the `identity` variants, the exact token to be predicted was also given in the input. This cannot happen during evaluation, since we hypothetically do not know what the token is to be predicted. For this reason, also according to previous findings, we assign the token initially to the value returned by a simple reversed bigram model.

In order to make this final evaluation on inversion as complete as possible, we introduced several new metrics: they allow us to better comprehend the obtained results and have a better understanding of the model training strategies applied. When we refer to a third-party model, we are using `meta-llama/Llama-3.2-1B`. [1]

- *Rec* (`token_recall`) refers to the fraction of unique tokens from the reference sequence that were correctly generated by the model. A higher recall value means the model captured more of the words from the reference

- *Prec* (`token_precision`) refers to the fraction of unique tokens in the generated sequence that are present in the reference. A higher precision value indicates the model didn't introduce many irrelevant or "hallucinated" words

- *F1* (`token_f1`) refers to the harmonic mean of precision and recall. It provides a single score that balances the trade-off between the two. A high F1 score indicates a good balance of both generating relevant tokens and avoiding irrelevant ones

[1] `https://huggingface.co/meta-llama/Llama-3.2-1B`

- *Acc* (`positional_accuracy`) refers to the exact token match at each position in the generated sequence compared to the reference: unlike the token-based metrics above, this one is sensitive to token order
- *Dup* (`token_duplications`) refers to the number of repeated tokens in the predicted prefix
- *SS* (`semantic_similarity`) refers to the semantic meaning of the generated text compared to the ground-truth, regardless of the specific words used. It computes the cosine similarity between the embedding of the two sentences, obtained by an external model [2]
- *FCP* (`forward_coherence_ppl`) measures how surprised the model is by the actual next tokens in a sequence. A lower perplexity indicates the model is more confident and accurate in its predictions. However, this metric must be taken with a grain of salt, since it is computed on the same model that produced the string in the backward pass
- *OPP* (`original_prefix_perplexity`) measures the perplexity of the original prefix text alone, using the third-party model. This should serve as an indication of "how natural" the prefix text is. This metric will be *the same* for all models, since it does not depend on the model, but only on the data to be predicted.
- *FPP* (`full_predicted_perplexity`) measures the perplexity of the predicted prefix text, concatenated with the suffix, using the third-party model. This should serve as an indication of "how grounded" the generated prefix is with the suffix
- *PPP* (`predicted_prefix_perplexity`) measures the perplexity of the predicted prefix text alone, using the third-party model. This should serve as an indication of "how natural" the generated text is

Note that *FPP* shows much less variance between the tested models, because it computes the perplexity of the entire sentence, which is the concatenation of the predicted prefix and the given suffix from the dataset. Since the latter is much longer than the former, the values of *FPP* tend to be pretty low. However, we are interested in the difference between the models.

Also, it is clearly visible that *PPP* is one order of magnitude larger in the models that predict the previous token using the gradient vector, without summing it up first with the embedding of the input token they're inverting on, used as the initialization value. This clearly demonstrates the intuition for which using gradients as directions would have made the model hold much better.

In the following tables, we can observe the difference in results between the baseline, the best variant (Identity, using gradients as directions, as in eq. (4.5)) and the same variant but using the gradients as values, in addition to all other tested variants.

[2] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

	Grad.	Rec ↑	Prec ↑	F1 ↑	Acc ↑
Baseline		20.9%	18.8%	19.7%	2.4%
Inv-First		11.3%	10.1%	10.7%	1.7%
Bert-like	Val.	2.9%	2.7%	2.8%	0.3%
Identity		0.7%	0.7%	0.7%	0.1%
Inv-First		13.3%	12.0%	12.6%	2.4%
Bert-like	Dir.	0.1%	0.1%	0.1%	0.1%
Identity		**22.5%**	**20.2%**	**21.2%**	**2.5%**

Table 5.1. Inversion evaluation on token-level metrics (Recall, Precision, F1, Accuracy). Higher values mean better recovery of original tokens.

	Grad.	Dup ↓	FCP ↓
Baseline		0.000062	**6.13**
Inv-First		0.034064	10.79
Bert-like	Val.	0.000475	7.50
Identity		0.008134	6.97
Inv-First		0.012325	7.89
Bert-like	Dir.	10.411330	7.99
Identity		0.000392	<u>6.56</u>

Table 5.2. Evaluation of the inversion capabilities, on metrics relative to full sentences, computed using the ILM model: Dup (Token Duplication), FCP (Forward Coherence Perplexity).

	Grad.	OPP =	FPP ↓	PPP ↓	SS ↑
Baseline		37.83	**8.34**	112.82	<u>0.28</u>
Inv-First		37.83	10.21	1576.23	0.25
Bert-like	Val.	37.83	11.54	5501.86	0.17
Identity		37.83	13.88	14658.58	0.12
Inv-First		37.83	9.77	1012.80	**0.30**
Bert-like	Dir.	37.83	11.05	563.26	0.11
Identity		37.83	**8.34**	**106.31**	**0.30**

Table 5.3. Sentence-level inversion metrics, computed using the third-party LLM: OPP (original prefix perplexity), PPP (predicted prefix), FPP (full predicted), SS (semantic similarity).

$\mathbf{x}$		dad in the garden. He gives her a small shovel and a bag of bulbs.
$\mathbf{x}^\star$ Baseline		to play with his cars, and look at the shake. She feels on her hand.
$\mathbf{x}^\star$ Inv-First	(Val.)	zzle spowerlizza in her plate. She start to fence and leaves.
$\mathbf{x}^\star$ Bert-like	(Val.)	could buildDven measure its neighbign, how he sees nostiff.
$\mathbf{x}^\star$ Identity	(Val.)	Kugct propide,RallashQilndmawkeycessUuhingask do.
$\mathbf{x}^\star$ Inv-First	(Dir.)	too hurt the car's bricket. It did not want to grow in a cage.
$\mathbf{x}^\star$ Bert-like	(Dir.)	Tim! Tim,ide, Sue, Sue, Tim!ide, "Tim, "Tim,ice. Tim! Tim!ittenbbed Tim! Tim,ide,auseectle.
$\mathbf{x}^\star$ Identity	(Dir.)	cars, and gets on his hand. But he does not want to play with the towers.
$\mathbf{y}$		Bulbs are like round seeds that grow into flowers. Lily digs holes in the dirt and puts the bulbs inside. She covers them with more [...]

(a) Inversion example of sample no. 1

$\mathbf{x}$		play in the sand. They had a big bucket and a small shovel. They wanted to
$\mathbf{x}^\star$ Baseline		on his arm. He were playing with their cars, and looked at the window. They wanted to
$\mathbf{x}^\star$ Inv-First	(Val.)	ovraph spower. Ben nodded. It looked happy and crunty, trying to
$\mathbf{x}^\star$ Bert-like	(Val.)	stfister pink fire Fvery build loud budd poster watched closer before he heard someone
$\mathbf{x}^\star$ Identity	(Val.)	canraask sn our saidJribistanezzemex-andight.ke work not
$\mathbf{x}^\star$ Inv-First	(Dir.)	rent. He walked closards the door. Lily and Ben were tragivous. They wanted to
$\mathbf{x}^\star$ Bert-like	(Dir.)	ide, Sue,ittenect Tim!ide, Tim!ide,ide,ide,ittenice. Tim! Tim! Tim! Tim,ectbbedauseide,
$\mathbf{x}^\star$ Identity	(Dir.)	her cars, and gets on the window. It was playing with their blocks. They wanted to
$\mathbf{y}$		make a castle. They dug and piled and shaped the sand. They found some shells and stones to decorate their castle. "Look, our castle is [...]

(b) Inversion example of sample no. 2

$\mathbf{x}$		They like to play in the park. They see a big swing. Lily wants to swing on it. She
$\mathbf{x}^\star$ Baseline		a shake. It feels on his arm. He wants to play in the window. She
$\mathbf{x}^\star$ Inv-First	(Val.)	hed at being a good brush. Amy felt sorry for herself. she swings threve. She
$\mathbf{x}^\star$ Bert-like	(Val.)	poster how good. Ostfast tea, ride, seen swow, three, fide bit
$\mathbf{x}^\star$ Identity	(Val.)	friend, sorished "Ochirt oversed "ownftittuggestign gulim way str
$\mathbf{x}^\star$ Inv-First	(Dir.)	Tom won't wear a big small blue side. Lila does not want to go too far grass. She
$\mathbf{x}^\star$ Bert-like	(Dir.)	Sue, Sue, Sue,ittenittenittenice.ide, Tim! Tim!itten Tim! Tim! Tim,bbedause Tim!ide,ectle.
$\mathbf{x}^\star$ Identity	(Dir.)	er, but he does not want to play in the window. It feels on her arm. She
$\mathbf{y}$		runs to the swing and sits on it. "Push me, Ben!" Lily says. "Push me high!" Ben pushes Lily on the swing. Lily feels happy. [...]

(c) Inversion example of sample no. 3

x		with their toy cars in the living room. They liked to make noises and pretend they were driving
$\mathbf{x}^\star$ Baseline		his cars, and looked at the shake. They were playing with her hand. It was too
$\mathbf{x}^\star$ Inv-First	(Val.)	laat. He ran towards the tight. Ben saw a big car crack. Tom was strong and
$\mathbf{x}^\star$ Bert-like	(Val.)	sailllaster more scatterfren, how could expl his dad drove himself too
$\mathbf{x}^\star$ Identity	(Val.)	oundddum cat.sideex picturesighU or promised s angry. "That's hearAH onore mommy
$\mathbf{x}^\star$ Inv-First	(Dir.)	makes a big mess!" Bax did not like my toy car. Their cars started can make go fast
$\mathbf{x}^\star$ Bert-like	(Dir.)	"Tim, "Tim, Tim! Sue, Sue, Sue, Sue, Sue, Sue,ittenectitten Tim! Tim!ide, Tim,auseectbbedet,
$\mathbf{x}^\star$ Identity	(Dir.)	he does not want to play in the window. He became playing with their blocks. They were very
y		fast. Lily had a pink car and Tom had a blue car. "Look, my car is faster than yours!" Tom said, zooming past Lily. "No, [...]

(d) Inversion example of sample no. 4

Table 5.4. Qualitative samples across all the model variants

5.2 ILM and Robustness

The current landscape of defensive mechanisms for LLMs is fragmented and underdeveloped. The concept of "robustness" in this context refers to the LLMs' ability to maintain their performance and reliability in the face of such input perturbations. We identify a gap in the availability of effective adversarial training (AT) tools for LLMs, noting that the existing literature is not as extensive as that for deep classifiers.

While in the previous chapter, we focused more on the inversion capabilities that an LLM trained with our innovative strategies and regularization techniques, in this chapter, we analyze the robustness of many training variants, including the ones already discussed.

The main evaluation metric will be the success rate of Greedy Coordinate Gradient (GCG) [Zou et al., 2023]. This benchmark naturally follows by the main goal of these training procedures: our ultimate goal is to have LLMs that are more grounded, where it indicates *"making LLMs know what they have been asked about"*. Even more importantly, we cannot use existing state-of-the-art benchmarks like HarmBench [Mazeika et al., 2024], since its main objective is to make Instruct fine-tuned LLMs do what they should not, according to the ethical guardrails enforced later on in the fine-tuning phase. Here in this project, we are working on a lower level, potentially in the pretraining phase, in order to make the LLM truly grasp the essence of what the text it is trained on is saying, instead of supporting the hypothesis of being stochastic parrots [Bender et al., 2021b]. Inverse LM can lay the foundation for next-generation LLMs that are not only robust and grounded but also fundamentally more controllable and trustworthy. The clear proof that this robustness issue must be addressed is in the example in table 2.1 by [Gabrielli, 2024], which takes some examples of adversarial meaningless inputs $\mathbf{x}^\star$ that can lead to a meaningful output $\mathbf{y}$, also with a lower Cross-Entropy loss than the human-readable prefix $\mathbf{x}$ — highlighting a vulnerability that Inverse LM is designed to address.

The Greedy Coordinate Gradient (GCG) algorithm is an iterative optimization method designed to find a local minimum of a differentiable function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$. Unlike traditional gradient descent which updates all components of the variable vector $\mathbf{x}$ simultaneously, GCG updates only one coordinate at each iteration. The key idea is to greedily select the coordinate that offers the most significant decrease in the function value.

5.2.1 GCG Algorithm

The Greedy Coordinate Gradient (GCG) algorithm is an iterative optimization technique used to find a completion sequence y of length M that maximizes a given objective function $f(\mathbf{x}, y)$, where $\mathbf{x}$ is a fixed prefix sequence of length N. The algorithm operates by greedily selecting tokens for the completion sequence y one at a time, based on the gradient of the objective function with respect to the current token.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ be the prefix sequence to optimize and $\mathbf{y} = (y_1, y_2, \ldots, y_M)$ be the fixed completion sequence. The objective function $f(\mathbf{x}, \mathbf{y})$ evaluates to the Cross-Entropy loss function of the completion sequence $\mathbf{y}$ given $\mathbf{x}$ as the input

prompt by the user.

The algorithm proceeds iteratively for T iterations. In each iteration:

1. **Identifying Promising Substitutions:** For each modifiable token at position $i \in \mathcal{I} := [1, N]$, the algorithm computes the gradient of the loss function with respect to the embedding of that token, $\nabla_{e_{x_i}} \mathcal{L}(\mathbf{x}, \mathbf{y})$. The top-$k$ most promising alternative tokens $\mathcal{X}_i$ are identified based on this gradient, aiming to move the embedding in the direction that reduces the loss.

2. **Batch-wise Exploration:** A batch of B modified sequences is generated. For each sequence in the batch:

   (a) Start with a copy of the current best sequence.

   (b) Randomly select a modifiable position i from the set $\mathcal{I}$ using a uniform distribution.

   (c) Randomly select a replacement token from the set of top-k promising tokens $\mathcal{X}_i$ (also using a uniform distribution) and replace the token at position i in the copied sequence.

3. **Evaluation and Update:** The loss function $\mathcal{L}$ is evaluated for each of the B modified sequences in the batch. The sequence $\hat{x}_{1:n}^{(b^*)}$ that yields the minimum loss, where $b^* = \arg\min_b \mathcal{L}(\hat{x}_{1:n}^{(b)}, \mathbf{y})$, becomes the new current best sequence for the next iteration.

This iterative process is repeated for a total of T iterations. The final sequence obtained after T iterations is the optimized prompt.

The GCG algorithm combines a greedy approach of selecting locally beneficial token replacements based on gradient information with a batch-wise random exploration within the top-k candidates. This strategy aims to efficiently search the sequence space and potentially avoid suboptimal local minima.

5.2.2 Faster-GCG

In order to have a faster attack evaluation pipeline, we could have adopted Faster-GCG [Li et al., 2024] as an in-place replacement to standard GCG. This enhancement should lead to faster convergence and a substantial reduction in computational costs, making it a more viable and efficient method for attacking even larger LLMs.

Faster-GCG is an efficient adversarial jailbreak method for Large Language Models (LLMs). It builds upon the Greedy Coordinate Gradient (GCG) attack, but addresses GCG's limitations, including high computational costs and limited jailbreak performance. The key idea is to optimize a suffix that, when appended to a prompt, causes the LLM to generate harmful content.

However, in our specific setting, we do not have a prompt that causes harmful behaviors, but instead we aim to find some "evil twin" to make the model complete with the meaningful sentence we want. This may look totally different but it is almost the same type of attack behind the scenes.

The main limitations that Faster-GCG tries to overcome are:

- Random Sampling: GCG randomly samples replacement tokens, which can lead to inefficient optimization.
- Self-Loop Issue: GCG doesn't prevent the algorithm from revisiting the same suffix, causing wasted computation.

This new algorithm addresses these issues adding the following improvements to the original procedure:

- Loss Function with an additional Regularization Term: Faster-GCG replaces the original cross-entropy loss used in GCG with the Carlini & Wagner (CW) loss [Carlini and Wagner, 2017], with the addition of the regularization term based on the distance between embeddings of the original tokens and the discovered replacements ($||\mathbf{X}_i - \mathbf{X}_j||$).
- Greedy Sampling: Instead of random sampling, Faster-GCG uses deterministic greedy sampling, which selects the most promising candidates first, accelerating convergence.
- Avoiding Self-Loops: Faster-GCG maintains a history of evaluated suffixes and filters out any candidates that would cause the algorithm to return to a previous state.

Due to the absence of a publicly available implementation of this novel algorithm at the time of writing these experiments, we implemented the algorithm by ourselves to the best of our understanding of the paper. This specific implementation is available as a Python library, which can be installed with `pip` as any other package, published on `https://github.com/simonesestito/faster-gcg`.

However, we carefully tested this alternative algorithm on our specific use case, and it resulted in slightly worse results. So, to keep our evaluation as effective as possible, thanks also to the computational power offered by some HPC systems, we decided to stick with the default GCG implementation, to have less variance, possible points of failure, and keep a better reproducibility and comparability in the results with other works.

5.3 Robustness Evaluation

In this section, we are evaluating the previous chapter's models. More specifically, we evaluate the robustness of the following model variants:

- `baseline`, training with the Cross-Entropy forward loss only
- `identity`, where during training we wanted to predict all the input tokens from their received gradients: $p(\mathbf{x}_i|\nabla_{e_i}\mathcal{L}_{CE}(f_\theta(\mathbf{x}), \mathbf{y})) \quad \forall i \in [1, N]$
- `bert-like`, where we predict a part of the input tokens, which are masked in the input sequence, from the embeddings gradients - imitating the training procedure of BERT Devlin et al. [2019], but during the backward pass.
- `inv-first`, where we predict only the very first token of the sentence from its received gradients: $p(\mathbf{x}_0|\nabla_{e_0}\mathcal{L}_{CE}(f_\theta(\texttt{[PAD]}||\mathbf{x}_{1:N}), \mathbf{y}))$

The above-listed variants are tested in 2 ways to keep the test consistent with the inversion chapter: they can treat the received gradient as a **value**, which implies to classify directly on it, or as a **direction**, which makes the classification work on $\mathbf{x} - \nabla_{\mathbf{x}}\mathcal{L}$.

Note that all the models in this first evaluation section are very small. They have in the order of tens of millions of parameters, so their architecture has very little capacity to both do their task well and, at the same time, be robust against gradient-based attacks like GCG.

The procedure to evaluate the models follows these rules, which are repeated for **30% of the samples** randomly picked from the test set, but consistently chosen for all the model variants:

1. Let $\mathbf{X}$ be the single sentence from the test set
2. $\mathbf{X}$ gets split into $\mathbf{x} = \mathbf{X}_{1:20}$ and $\mathbf{y} = \mathbf{X}_{21:...}$
3. Initialize the generated attack prefix $\mathbf{x}' \in \mathcal{V}^{20}$ as 20 random tokens from the vocabulary
4. Run GCG for multiple steps, until the GCG loss $\mathcal{L}_{\text{GCG}}$ does not decrease for 10 iterations, or we have already run GCG for 500 steps
5. At each iteration, from the batch of 4096 generated samples (the search width), we keep the top-256 samples and keep repeating until the stop condition is met

This is even more clearly explained in detail in algorithm 2.

The final model response is computed as $\mathbf{y}' = \text{LLM}_{\text{Forward}}(\mathbf{x}')$, being $\mathbf{x}'$ the best attack prefix found after all the iterations of the GCG procedure. We compute the final success rate for the single attack as the number of tokens such that $\mathbf{y}_i = \mathbf{y}'_i \; \forall i \in [1, 20]$, divided by $|\mathbf{y}|$.

	Grad.	**GCG Success Rate ↓**	**GCG Average Steps (mean ± stddev)**
Baseline		95.9%	277 ± 148
Identity	Val.	88.1%	274 ± 145
Bert-like	Val.	**0.8%**	249 ± 148
Inv-First	Val.	85.0%	320 ± 134
Identity	Dir.	<u>82.8%</u>	284 ± 141
Bert-like	Dir.	85.5%	287 ± 143
Inv-First	Dir.	89.3%	313 ± 134

Table 5.5. Evaluation against the GCG attack, where a lower success rate is better

Algorithm 2 Single-Sentence GCG Attack

Require: Expected continuation string $\mathbf{y}$ to be attacked, length of the attack prefix n, number of iterations T
Ensure: Best attack prefix $\mathbf{x}^\star$ with loss $\mathcal{L}_{\text{GCG}}$

$\mathbf{x}^\star \leftarrow$ random one-hot tokens matrix of size $|V| \times n$
$step \leftarrow 0$ ▷ Iteration counter
$d \leftarrow 0$ ▷ Loss non-decrease counter
$\mathcal{L}_{\text{old}} \leftarrow \infty$ ▷ Last loss found
while $step < T$ **and** $d < 10$ **do**
    Compute a batch of candidate prefixes $\mathbf{X}$ running one step of **GCG**
    $\mathbf{x}^\star \leftarrow \arg\min_{\mathbf{x} \in \mathbf{X}} \mathcal{L}_{\text{CE}}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$
    $\mathcal{L}_{\text{GCG}} \leftarrow \mathcal{L}_{\text{CE}}(\mathbf{x}^\star, \mathbf{y}, \boldsymbol{\theta})$ ▷ Take the min loss so far
    if $\mathcal{L}_{\text{GCG}} < \mathcal{L}_{\text{old}}$ **then**
        $\mathcal{L}_{\text{old}} \leftarrow \mathcal{L}_{\text{GCG}}$
        $d \leftarrow 0$
    else
        $d \leftarrow d + 1$
    end if
    $step \leftarrow step + 1$
end while
return $\mathcal{L}_{\text{GCG}}$

From the results observable in table 5.5, we can notice that the majority of the variants have an improvement in robustness against GCG attacks. Looking at the specific variant that was flagged as the best one in the inversion task in the previous chapter, which is *Identity (grad. direction)*, it allowed a reduction in the success rate of more than 13%. This improvement can be attributed to the model's ability to better condition the continuation of a sentence with the actual prompt it was given as input (also referred to as "*grounded*"), which may lead to a more robust model. However, there is the specific case of *Bert-like (grad. value)*, which corresponds to the model variant that imitates BERT in the backward pass, masking some tokens and letting the model predict them directly, classifying on the received gradient on the PAD token. This model scores an incredibly low GCG success rate, making us suppose that it may actually strongly go in the direction of adversarially robust models, at least on the gradient-based white-box GCG attack. Given the huge difference between the baseline and this variant, the experiments have been repeated from the initial training phase, but they gave us the same results as in the first run of the pipeline. For sure, this aspect will require a deeper investigation.

In our experiments, we also studied the correlation between the number of GCG algorithm maximum allowed iterations and the success rate of the attack, always computed as the number of tokens that match between the LLM's responses to the original input $\mathbf{x}$ and the attack input $\mathbf{x}'$, while keeping all other hyperparameters, like the search window width, unaltered. Interestingly, in fig. 5.1 some lines actually cross each other when increasing the number of GCG iterations: this may indicate that some variants are more effective at different values of the GCG iterations. For instance, *Inv-First (grad. direction)*, represented as the orange line, is better

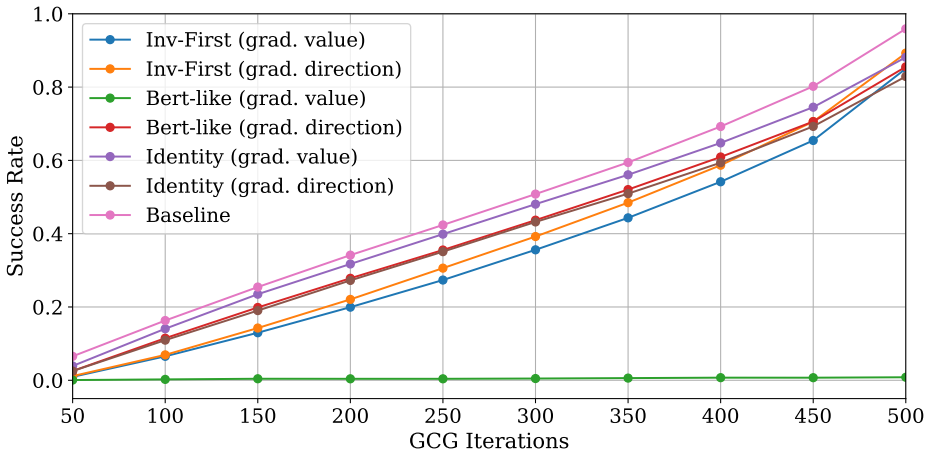


Figure 5.1. GCG Success Rate varying according to the number of iterations performed

than *Bert-like (grad. direction)*, represented as the red line, when the number of allowed iterations is pretty low; however, at the end of the plot, at the maximum number of iterations tested, their effectiveness is the opposite. That being said, this phenomenon is not dramatic and makes only slight changes in the final results reported in the previously discussed table.

To have a better understanding of the GCG results listed in table 5.6, we measured the following other metrics, always considering the subset of successful GCG attacks:

- *Original X CE-loss* is the $\mathcal{L}_{\text{CE}}(f_\theta(\mathbf{x}_0, ..., \mathbf{x}_N); \mathbf{y}_0, ..., \mathbf{y}_M)$ - lower is better, since this means that $\mathbf{y}$ is a natural continuation of $\mathbf{x}$ according to the model
- *Attack X' CE-loss* is the $\mathcal{L}_{\text{CE}}(f_\theta(\mathbf{x}'_0, ..., \mathbf{x}'_N); \mathbf{y}_0, ..., \mathbf{y}_M)$ - higher is better, since a low value means that $\mathbf{y}$ is a natural continuation of $\mathbf{x}'$ according to the model, which is exactly the issue what we aim to solve
- *Delta X CE-Loss* is the delta between the original prefix loss and the successful attack prefix found - the lower it is, the worst the attack was, so the more robust the model is when the attack is successful, since this is given as $\mathcal{L}_{\text{CE}}(f_\theta(\mathbf{x})) - \mathcal{L}_{\text{CE}}(f_\theta(\mathbf{x}'))$ (as in the introduction in table 2.1, the attack prefix has a lower loss than the natural sentence)
- *KL-divergence* is the divergence between the probability distributions of the logits that correspond to the prediction of $\mathbf{y}$ and $\mathbf{y}'$; we want to measure how different are the logits, and thus the probability distributions of the next token: $KL(f_\theta(\mathbf{x}_0, ..., \mathbf{x}_N), f_\theta(\mathbf{x}'_0, ..., \mathbf{x}'_N))$

In all previous mathematical formulation, note that $f_\theta(\mathbf{x}_0, ..., \mathbf{x}_N)$ is the function which incorporates the LLM under attack and returns the logits that correspond to the prediction of the $\mathbf{y}$ output, not the ones that predict parts of the input prompt $\mathbf{x}$ itself, as it would be wrong to consider for our analysis.

From the metrics in table 5.6, we observe that robust variants, such as *Identity (grad. direction)*, not only exhibit a substantially lower attack success rate (ASR)

compared to the baseline, but also display a smaller increase in Cross-Entropy loss when the attack succeeds. Recall that this *delta* quantifies the extent to which the model is "fooled" by the attack, defined as the difference between the loss on the original, human-readable input and the loss on the adversarially generated sequence. A higher delta indicates greater susceptibility, as the model interprets the attack sequence as being more strongly aligned with the target continuation $\mathbf{y}$. Finally, the **KL-divergence** allows us to observe how much the output distributions returned by the LLM differ between $\mathbf{x}$ and $\mathbf{x}'$. The more different they are, the better the model can discriminate between them, recognizing that they are actually two distinct and different pieces of input.

	Grad.	**Original X CE-loss ↓**	**Attack X' CE-loss**	**Delta CE-loss ↓**	**KL Divergence ↑**
Baseline		13.28	10.97	2.31	2.19
Identity		12.77	11.21	1.56	2.23
Bert-like	Val.	13.26	10.25	3.01	**54.19**
Inv-First		**11.09**	9.72	<u>1.37</u>	2.44
Identity		12.58	11.12	1.46	2.47
Bert-like	Dir.	11.49	10.34	**1.15**	2.23
Inv-First		<u>11.21</u>	9.81	1.40	2.44

Table 5.6. Evaluation of the attack input prefix and the target model alone

To have a complete evaluation, we adopt a similar approach to the one we used during inversion: using a third-party model to compute some other statistics lets us abstract away from the biases in our LLMs. Here, since the perplexity of the attack prefix is computed with a third-party independent model, it can easily return the real naturalness of the generated prefix, instead of being influenced by the attack itself and wrongly reporting that it will be even more natural than the human prefix. Remember that we are considering only the successful attacks, ignoring the ones that have failed, since they are not useful to understand the quality of the attacks. Also, because of that, the results involving the `bert-like` variant using gradients as values will have a much smaller number of samples that participate in these metrics.

	Grad.	Original X Perplexity	Attack X' Perplexity ↓	Semantic Similarity ↑
Baseline		44.14	17344.04	0.13
Identity	Val.	43.98	**8322.25**	**0.18**
Bert-like		40.37	11817.21	0.11
Inv-First		44.81	9431.09	0.16
Identity	Dir.	44.71	10929.21	0.15
Bert-like		44.74	10611.09	0.13
Inv-First		43.50	12344.85	0.13

Table 5.7. Evaluation of the attack input prefix and a third-party model

5.3.1 Illustration of some adversarial samples

In the following tables, we can observe some examples of the attack strings found by applying the GCG algorithm on the tested model variants, as a mere qualitative illustration of the results we got. Although they do not show attack strings that carry out the meaning of a **y** continuation, and they are still meaningless, our training procedure allowed us to have a more robust model, since the attack success rate is lower in all the variants; some having a better result than others, but all of them in some way beat the baseline.

	Input	Output y	Loss
$\mathbf{x}$:	cry. He wished he had listened to his mom. He stared at the earth and hoped someone would	help him. Suddenly, he heard a voice	11.55
$\mathbf{x}^\star$:	Boboach att ag ChS asked cake.vel near pftautautue. thoughtautaut smart would		**10.11**
$\mathbf{x}$:	long time, ever since the house was abandoned. I have many stories to tell	you. Would you like to hear them?	17.22
$\mathbf{x}^\star$:	gga,Dveryng Do magic belie the theationationicy black name never sayirt<unk>		**12.99**

Table 5.8. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Baseline` model.

	Input	Output y	Loss
$\mathbf{x}$:	Lily and Ben were friends who liked to play outside. But they did not like the same things. Lily	liked to make snowmen and snow angel	13.22
$\mathbf{x}^\star$:	Lucy. Speez herself angO piecle you."lly named nexird opened cake".o.ter carrotmy		**12.14**
$\mathbf{x}$:	the ducks and the swings," Sam said." OK, Sam. That sounds fun,"	Lily said.They opened the door and ste	17.06
$\mathbf{x}^\star$:	Mommy than They TIon. up.Cleaseily am castle omisent. asked. crabo."		**13.85**

Table 5.9. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Identity (grad. value)` model.

	Input	Output y	Loss
$\mathbf{x}$:	Lily and Ben were twins who liked to go on tours with their mom and dad. They	saw many new things and learned a lot. But	24.14
$\mathbf{x}^\star$:	K forgot Tomriend rot became want or<unk><unk><unk> Onceriend sky. butterXgetherby		**23.08**
$\mathbf{x}$:	read books or draw pictures. They wanted to go outside and have fun. They looked at each other and	had an idea."Mom, we need to	18.80
$\mathbf{x}^\star$:	Lis conizz Maskock!" t angry had nois What oyM sleep Benirdixfe		**18.27**

Table 5.10. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Bert-like (grad. value)` model.

	Input	Output y	Loss
$\mathbf{x}$:	Lila and Ben were adventurous kids. They liked to explore new places and try new things.	One day, they went to the beach with their mom	18.64
$\mathbf{x}^\star$:	so rest come so an One rock o per who<s> fr balling, M cars Whisk listen. likes Spot.		**13.47**
$\mathbf{x}$:	One day, a little boy named Tim found a peanut on the ground. He picked it up and	showed it to his mom. "Mom, what is this?"	19.36
$\mathbf{x}^\star$:	pudd likesnn.hing It Sara "It where M whereanc It where pen where some?" madei		**17.06**

Table 5.11. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Inv-First (grad. value)` model.

	Input	Output y	Loss
$\mathbf{x}$:	." She is happy. She made a new pattern.She moves on to the flowers.	She says, "One, two, three, f	11.15
$\mathbf{x}^\star$:	e Sheleix ey,se, amix4<unk>t. says, Herix help, in pawixt.		**9.55**
$\mathbf{x}$:	Imagine standing in a park with many trees and a big green lawn.	In the middle of the park is	16.43
$\mathbf{x}^\star$:	fere re'age att ApenXapard. thankarn vo park.XX keyft park.		**11.73**

Table 5.12. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Identity (grad. direction)` model.

	Input	Output y	Loss
$\mathbf{x}$:	of plastic and has perfume on it. It does not grow or need water. It is	not real."Lily was confused and	18.72
$\mathbf{x}^\star$:	everything her small, from pus Tim ThatF<unk> LV paper I h listen LI with. do		**15.15**
$\mathbf{x}$:	reat-grandpa. He was a brave explorer. He wore me on his	adventures."Lily was amazed. She wanted to	20.26
$\mathbf{x}^\star$:	by ta anotherizCcessious rcessly, Lious tower her ask derirthdayI its		**17.54**

Table 5.13. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Bert-like (grad. direction)` model.

	Input	Output y	Loss
$\mathbf{x}$:	Ben and Sam were friends who liked to play with toys. They had many toys, but their favorite was a	helicopter. The helicopter	13.76
$\mathbf{x}^\star$:	p nodd nodd clotZub fam Do un said,ball wr." hitac bi'reide.-x		**11.14**
$\mathbf{x}$:	Lily was excited to go to school. She liked to learn new things from her teach. She also	liked to play with her friends. Today, they	17.10
$\mathbf{x}^\star$:	its botindbbougower. seenhere eez-ick moral food Gra peop everyoun'oun ever		**16.34**

Table 5.14. Original inputs $\mathbf{x}$ and adversarial examples $\mathbf{x}^\star$ generated attacking with GCG the `Inv-First (grad. direction)` model.

5.4 Forward LM Evaluation

Finally, we tested that these models are still functioning properly in forward mode, without experiencing **performance degradation**. Checking out the perplexity during training and validation, it has been observed that the performance of the custom models with the regularization term on the gradients $\nabla_{\mathbf{e}}\mathcal{L}_{\text{CE}}$ does not penalize the model's ability to speak fluently during standard usage in forward mode. This is not obvious, since many times in Adversarial Training in literature, the model may require many more parameters and training time to reach the same final objective, since part of the model capacity will be used by the adversarial training procedure, whatever it is in general. In table 5.15 we can observe that the worst model is *Inv-First (grad. value)*. This relatively high perplexity value is also confirmed in the qualitative examples in the tables below, where the sentences are by far the ones that make less sense and seem more confused and repetitive.

	Grad.	Perplexity ↓	CE Loss ↓
Baseline		**4.83**	**1.58**
Identity		5.07	1.63
Bert-like	Val.	5.79	1.76
Inv-First		8.41	2.13
Identity		5.08	1.62
Bert-like	Dir.	5.42	1.69
Inv-First		6.82	1.92

Table 5.15. Quantitative forward mode evaluation

	Grad.	Completion for "One day,"
Baseline		a little boy named Tim wanted to travel to a far mountain. He asked his dad for a raft,
Identity		a little girl named Lucy went to the park with her mom. Lucy liked to play on the swings
Bert-like	Val.	a little boy was walking in the park. He noticed a big, shiny object in the park.
Inv-First		they pinch. They find gold. They take pictures of stars.
Identity		a little girl named Amy was playing outside. She saw a big tree and thought it was a toy.
Bert-like	Dir.	a little girl named Lucy was playing in the garden. She saw a shiny ring on a branch.
Inv-First		hey went to the beach with his mom. He saw something shiny and strange inside.

	Grad.	Completion for "Once there was a"
Baseline		little girl called Emma. She was 3 years old and she liked to explore new places. Everywhere she went, she saw something special.
Identity		boy who was very enthusiastic. Every day he would go outside and go out and play in the sun
Bert-like	Val.	girl who loved to play. One day she decided to go on a walk in her neighbourhood.
Inv-First		learn to others, but she also learn something new things she should not have shared with others.
Identity		little boy who was very curious. He wanted to know why he was so interesting. He decided to explore the world around him.
Bert-like	Dir.	small girl named Mary. Mary was only three years old and she loved to explore the world.
Inv-First		girl named Sarah. She was very curious. She wanted to see what was in her room.

Table 5.16. Example completions for the given prompt, in forward mode

Chapter 6

Conclusion

In conclusion, this thesis, together with the associated paper published as a preprint on ArXiv, introduces Inverse Language Modeling (ILM) as a novel framework designed to simultaneously address two critical challenges in Large Language Models (LLMs): robustness and grounding. Our experiments demonstrate ILM's potential to enhance LLMs' resilience against input perturbations, a key step towards mitigating vulnerabilities to adversarial attacks. Furthermore, ILM offers a pathway to improved grounding, enabling LLMs to better correlate their outputs with the input prompts and thereby facilitating the identification of potentially problematic input triggers. By the current transformation of LLMs into agents that not only generate text but also analyze untrusted user inputs and perform real actions, ILM paves the way for the development of more reliable, controllable, and trustworthy language models.

Future work

There are several promising avenues for future research. While ILM is introduced within the context of pre-training, an interesting direction would be to explore its application in the **fine-tuning stage**. Specifically, one could investigate how the principles of inverse modeling can be incorporated into the fine-tuning process to improve the robustness and generalization of LLMs on downstream tasks. Additionally, research could explore the potential benefits of combining ILM with instruction tuning, to further align LLM behavior with human preferences and instructions. Further exploration should also consider the application of ILM to more powerful LLMs, such as Llama-3.2-1B or even larger 7B models, to assess its **scalability** and effectiveness as model capacity increases.

Ethics and Impact Statement

The advancement of LLMs carries potential ethical implications. Enhancing the robustness and grounding of these models can positively impact society, including reducing the spread of misinformation and harmful content. Our investigation into this phenomenon contributes to a better understanding of how LLMs work and, thus, ultimately, to make them safer and more predictable. We believe that the publication of our research will promote a broader discussion on the responsible development of LLMs and contribute to the development of better defense mechanisms, as similar progress has already been made in the field of deep classifiers.

Bibliography

Gunjan Aggarwal, Abhishek Sinha, Nupur Kumari, and Mayank Singh. On the benefits of models with perceptually-aligned gradients. *arXiv preprint arXiv:2005.01499*, 2020.

Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity, 2023. URL `https://arxiv.org/abs/2308.14132`.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *ACM conference on fairness, accountability, and transparency*, 2021a.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *ACM conference on fairness, accountability, and transparency*, 2021b.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017. doi: 10.1109/SP.2017.49.

Samuel Jacob Chacko, Sajib Biswas, Chashi Mahiul Islam, Fatema Tabassum Liza, and Xiuwen Liu. Adversarial attacks on large language models using regularized relaxation, 2024. URL `https://arxiv.org/abs/2410.19160`.

Valeriia Cherepanova and James Zou. Talking nonsense: Probing large language models' understanding of adversarial gibberish inputs, 2024. URL `https://arxiv.org/abs/2404.17120`.

Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. *arXiv preprint arXiv:1605.06743*, 2016.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4829–4837, 2016.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification, 2018. URL `https://arxiv.org/abs/1712.06751`.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english?, 2023. URL `https://arxiv.org/abs/2305.07759`.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL `https://www.sciencedirect.com/science/article/pii/036402139090002E`.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

Davide Gabrielli. Evaluating hard prompt adversarial attacks methods, 2024.

Davide Gabrielli, Simone Sestito, and Iacopo Masi. Inverse language modeling towards robust and grounded llms, 2025. URL `https://arxiv.org/abs/2510.01929`.

Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2): 23–38, 1994.

Roy Ganz, Bahjat Kawar, and Michael Elad. Do perceptually aligned gradients imply robustness? In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10628–10648. PMLR, 2023. URL `https://proceedings.mlr.press/v202/ganz23a.html`.

Simon Geisler, Tom Wollschläger, M. H. I. Abdalla, Johannes Gasteiger, and Stephan Günnemann. Attacking large language models with projected gradient descent, 2025. URL `https://arxiv.org/abs/2402.09154`.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Xiao Guo, Xiaohong Liu, Iacopo Masi, and Xiaoming Liu. Language-guided hierarchical fine-grained image forgery detection and localization. *International Journal of Computer Vision*, pages 1–22, 2024.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

Alexey Grigorevich Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, pages 364–378, 1971.

Corentin Kervadec, Francesca Franzon, and Marco Baroni. Unnatural language processing: How do language models handle machine-generated prompts?, 2023. URL `https://arxiv.org/abs/2310.15829`.

Phillip Keung, Yichao Lu, György Szarvas, and Noah A. Smith. The multilingual amazon reviews corpus. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

Xiao Li, Zhuhong Li, Qiongxiu Li, Bingze Lee, Jinghao Cui, and Xiaolin Hu. Faster-gcg: Efficient discrete optimization jailbreak attacks against aligned large language models, 2024. URL `https://arxiv.org/abs/2410.15362`.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models, 2024. URL `https://arxiv.org/abs/2310.04451`.

Xiaogeng Liu, Peiran Li, G. Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. AutoDAN-turbo: A lifelong agent for strategy self-exploration to jailbreak LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=bhK7U37VW8`.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal, 2024. URL `https://arxiv.org/abs/2402.04249`.

Rimon Melamed, Lucas McCabe, Tanay Wakhare, Yejin Kim, H Howie Huang, and Enric Boix-Adserà. Prompts have evil twins. In *EMNLP*, 2024a.

Rimon Melamed, Lucas Hurley McCabe, Tanay Wakhare, Yejin Kim, H. Howie Huang, and Enric Boix-Adserà. Prompts have evil twins. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 46–74, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.4. URL https://aclanthology.org/2024.emnlp-main.4/.

John X. Morris, Wenting Zhao, Justin T. Chiu, Vitaly Shmatikov, and Alexander M. Rush. Language model inversion, 2023. URL https://arxiv.org/abs/2311.13647.

Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. Advprompter: Fast adaptive adversarial prompting for llms, 2024. URL https://arxiv.org/abs/2404.16873.

Ofir Press and Lior Wolf. Using the output embedding to improve language models, 2017. URL https://arxiv.org/abs/1608.05859.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Nathanaël Carraz Rakotonirina, Corentin Kervadec, Francesca Franzon, and Marco Baroni. Evil twins are not that evil: Qualitative insights into machine-generated prompts. *arXiv preprint arXiv:2412.08127*, 2024.

Jesse Roberts. How powerful are decoder-only transformer neural models? In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2024.

Andrew Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11504. URL https://ojs.aaai.org/index.php/AAAI/article/view/11504.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts, 2020. URL https://arxiv.org/abs/2010.15980.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Zijun Wang, Haoqin Tu, Jieru Mei, Bingchen Zhao, Yisen Wang, and Cihang Xie. AttnGCG: Enhancing jailbreaking attacks on LLMs with attention manipulation. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=prVLANCshF.

Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, Chenyang Shao, Yuwei Yan, Qinglong Yang, Yiwen Song, Sijian Ren, Xinyuan Hu, Yu Li, Jie Feng, Chen Gao, and Yong Li. Towards large reasoning models: A survey of reinforced reasoning with large language models, 2025. URL https://arxiv.org/abs/2501.09686.

Yutong Zhang, Yao Li, Yin Li, and Zhichang Guo. A review of adversarial attacks in computer vision. *arXiv preprint arXiv:2308.07673*, 2023.

Yiran Zhao, Wenyue Zheng, Tianle Cai, Xuan Long Do, Kenji Kawaguchi, Anirudh Goyal, and Michael Shieh. Accelerating greedy coordinate gradient and general prompt optimization via probe sampling, 2024. URL https://arxiv.org/abs/2403.01251.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Secrets of rlhf in large language models part i: Ppo, 2023. URL https://arxiv.org/abs/2307.04964.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023. URL https://arxiv.org/abs/2307.15043.