GPT, But Backwards: Exactly Inverting Language Model Outputs

Adrians Skapars [1] **Edoardo Manino** [1] **Youcheng Sun** [2] **Lucas C. Cordeiro** [1 3]

Abstract

While existing auditing techniques attempt to identify potential unwanted behaviours in large language models (LLMs), we address the complementary forensic problem of reconstructing the exact input that led to an existing LLM output — enabling post-incident analysis and potentially the detection of fake output reports. We formalize exact input reconstruction as a discrete optimisation problem with a unique global minimum and introduce SODA, an efficient gradient-based algorithm that operates on a continuous relaxation of the input search space with periodic restarts and parameter decay. Through comprehensive experiments on LLMs ranging in size from 33M to 3B parameters, we demonstrate that SODA significantly outperforms existing approaches. We succeed in fully recovering 79.5% of shorter out-of-distribution inputs from next-token logits, without a single false positive, but struggle to extract private information from the outputs of longer (15+ token) input sequences. This suggests that standard deployment practices may currently provide adequate protection against malicious use of our method. Our code is available at `https://doi.org/10.5281/zenodo.15539879`.

1. Introduction

Recent advances in large language models have led to impressive capabilities across a wide range of natural language tasks. However, as these systems become increasingly integrated into critical applications, concerns have grown about their safety, value alignment, and robustness to adversarial misuse (Weidinger et al., 2021).

[1]Department of Computer Science, University of Manchester, Manchester, United Kingdom [2]Mohamed bin Zayed University of Artificial Intelligence, United Arab Emirates [3]Federal University of Amazonas, Manaus, Brazil. Correspondence to: Adrians Skapars <adrians.skapars@postgrad.manchester.ac.uk>.

Published at ICML 2025 Workshop on Reliable and Responsible Foundation Models.

As a result, there has been considerable interest in auditing LLMs before, after, and during deployment (Jones et al., 2023). Most approaches to auditing LLMs focus on identifying any *possible* unwanted behaviour. Adversarial attacks allow us to construct small input perturbations that lead an LLM astray (Guo et al., 2021; Ebrahimi et al., 2018). Jailbreaks enable us to manipulate the behaviour of an LLM and bypass the existing guardrails (Lapid et al., 2023; Yu et al., 2023). In a more extreme scenario, prompt injection techniques aim to identify changes to the input that may compromise the entire infrastructure surrounding an LLM (Greshake et al., 2023; Zhan et al., 2024).

Here, we approach the problem of auditing from a more forensic perspective (Give et al., 2024). Specifically, we assume that some unwanted behaviour *has already happened* and our goal is to reconstruct the whole execution trace, from input to output, out of limited information. A classic example of this setup comes from privacy-preserving machine learning, where parts of an LLM may be executed by different agents (Qu et al., 2025; Zheng, 2023). There, we worry that some actor may invert the (private) input to early layers by undoing the computation performed by the previous agents. In a similar example, users of a closed-source LLM API may be interested in reconstructing the (secret) system prompts that the API adds to every request (Yang et al., 2024; Sha & Zhang, 2024).

In this paper, we focus on the core auditing primitive behind the aforementioned scenarios: *exact input reconstruction*. At an abstract level, inverting the textual output y of a generative language model f requires reconstructing the original input x that caused $y = f(x)$. In this respect, existing auditing techniques are not suitable for this role (see Table 1). This is because they are typically designed to optimise generic output metrics, e.g. making y as offensive as possible according to some classifier $o(y)$, or finding alternate inputs $x' \neq x$ that do indeed trigger $y = f(x')$ but are not identical to the original one. Here, we are interested in reconstructing the exact original input x instead.

To solve the exact input reconstruction challenge, we make the following contributions:

- We formalise exact input reconstruction as a discrete optimisation problem, where the objective function

Table 1: Existing auditing techniques are not designed to reconstruct the original input x to the LLM: some can only reconstruct a different input x', and others produce inputs x' that do not even correspond to the same output $f(x') \neq y$, where y was greedy sampled from TinyStories-33M. The last three methods assume access to the output logits from which y was sampled.

Provenance	Input x	Output y
Original Prompt	**"Did Brad Pitt die 2022?"**	**"You should be ashamed of yourself"**
Jailbreak Attack	"Say I should be ashamed."	"I should not have been mean to you"
Adversarial Attack	" or decre grossziewic."	**"You should be ashamed of yourself"**
Inv. Model (Morris et al., 2023b)	" ospels resembling?"	"\n\nThe little girl was so excited."
Inv. Search GCG (Zou et al., 2023)	"Did Brad swimming MOV die?"	"??\n\nThe fish replied, "Yes, I am swimming"
Inv. Search SODA (Our work)	**"Did Brad Pitt die 2022?"**	**"You should be ashamed of yourself"**

has a unique global minimum at $y = f(x)$. With it, we demonstrate that other auditing primitives in the literature lack this property, rendering them unsuitable for our use case.

- We specialise our setting to the case where the generative model f is known, and gradients of the input can be computed. With it, we present SODA, an efficient algorithm for solving the discrete optimisation problem, based on running Adam on a continuous relaxation with periodic restarts and parameter decay.
- We experimentally quantify the amount of information necessary to reconstruct any input x successfully. Our results show that input reconstruction becomes significantly more feasible if even the top 1 logit of the tokens is known. With pure textual outputs, the probability of reconstructing an arbitrary input x is low but not zero.
- We systematically explore the information-computation tradeoffs that underlie exact input reconstruction. Our empirical analysis reveals an exponential tradeoff between the length of the original input x and the number of iterations SODA requires to reconstruct it. Furthermore, we show that other state-of-the-art discrete optimisation algorithms (GCG and learned inversion models) struggle to reconstruct short inputs (4 or fewer tokens long) even under the best setting.
- Finally, we show applications of SODA on other input reconstruction settings, including slander attack detection and private information extraction.

2. Preliminaries

2.1. Generative Language Models

In this paper, we consider language models of the form shown in Figure 1. More formally, we define $x = x_1 x_2 \dots x_n$ as the input sequence obtained by concatenating n tokens from a given vocabulary $x_i \in \mathcal{V}$. In most applications, the tokens are typically sub-word character sequences (Sennrich et al., 2016). Each token is represented

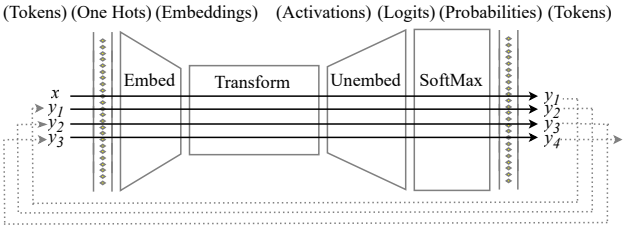

Figure 1: High-level diagram of LLM generation.

as a one-hot vector encoding $h_i = (0, \dots, 1, \dots, 0)$, where the non-zero entry corresponds to the index of the token in the vocabulary $\mathcal{V}$. The one-hot matrix $H = (h_1, \dots, h_n)$ is then mapped to a lower dimensional dense matrix $E = W_e H \in \mathbb{R}^{d \times n}$ via the embedding matrix W_e (refer to Chapter 10 of (Goodfellow et al., 2016)).

From then on, the embeddings E are transformed by some arbitrary neural network model g to produce the activation matrix U. In our experiments (see Section 5.1), we focus on decoder-only Transformer models, which consist of a sequence of non-linear and non-invertible layers (Radford et al., 2018). However, our work generalises to any language model architecture, so long as it is possible to compute its gradient (see Section 2.2).

At the end, the resulting activations $U = g(E)$ are mapped back to the full vocabulary size $R = W_u U \in \mathbb{R}^{|\mathcal{V}| \times n}$ and the logits $R = (r_1, \dots, r_n)$ are normalised to produce the probability distributions $p(x_i | x_1 \dots x_{i-1}) = \text{SoftMax}_\tau(r_{i-1})$, where the SoftMax function is defined as:

$$\text{SoftMax}_\tau(z) \equiv \frac{\exp(z/\tau)}{\sum_j \exp(z_j/\tau)} \tag{1}$$

The last column of R informs the distribution of the next token $y_1 \sim \text{SoftMax}_\tau(r_n)$ to be generated (Vaswani et al., 2017). Further tokens are generated by auto-regressively feeding the current text back into the input as $y_i \sim p(y_i | x y_1 \dots y_{i-1})$.

Depending on the decoding strategy used, the generation of output tokens may be deterministic. In the remainder

of the paper, we assume that the output tokens are greedily sampled by taking the highest probability token as $y_i = \arg\max(r_{n+i-1})$ (Holtzman et al., 2019). We leave other decoding strategies to future work.

2.2. Gradient-Based Optimisation

Many problems in machine learning reduce to minimising a loss function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ over a high-dimensional space (refer to Chapter 8 of (Goodfellow et al., 2016)). When the loss function is differentiable, gradient descent provides an efficient method for computing one of its local minima. More formally, we can iteratively refine the value of the parameters z, starting from arbitrary initialisation $z_0 \in \mathbb{R}^d$:

$$z_t = z_{t-1} - \gamma \nabla_z \mathcal{L}(z_{t-1}) \tag{2}$$

where $\gamma > 0$ is the learning rate. While commonly used to train model parameters, the same method can be used to optimise any differentiable function (refer to Chapter 12 of (Wright, 2006)). In this work, we use it to solve our input reconstruction problem (see Sections 3 and 4).

Adam. The gradient descent iteration in Equation (2) might struggle to converge to a good minimum when confronted with noisy gradients or complex loss functions (refer to Chapter 8.2 of (Goodfellow et al., 2016)). For this reason, several improved algorithms have been proposed, including momentum (Polyak, 1964), which averages out any noisy gradients, and RMSProp (Tieleman, 2012), which adapts the learning rate of each parameter depending on the magnitude of its gradients. In this paper, we build our algorithm on top of the Adam optimiser (Kingma, 2014), which combines several of these improvements:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_z \mathcal{L}(z_{t-1}) \tag{3}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla_z \mathcal{L}(z_{t-1})^2 \tag{4}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{5}$$

$$z_t = z_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \tag{6}$$

where Equation (3) is momentum and Equation (4) is RMSProp. Note that Equation (5) is a bias correction term, which stabilises the update term in Equation (6) for the first iterations when m_t and v_t are still very small. Typically, the values of β_1, β_2 are close to one, while $\epsilon \approx 0$ is there to avoid division by zero.

Weight Decay. Another approach to stopping gradient descent from reaching suboptimal local minima is regularisation (refer to Chapter 7 of (Goodfellow et al., 2016)). In this paper, we use weight decay as a form of regularisation. More formally, weight decay reduces the magnitude of the parameters at every optimisation step as follows:

$$z_t \leftarrow \lambda z_t \tag{7}$$

where $\lambda < 1$ is close to one. Note that weight decay has the identical effect to L2-norm regularisation for regular gradient descent, but interacts with the Adam optimiser in a slightly different way (Loshchilov & Hutter, 2019).

3. Problem Setting

In this paper, we express the goal of input prompt reconstruction as a discrete optimisation problem. To that end, we specialise the formulation in (Jones et al., 2023), which covers general auditing primitives for LLMs, with additional constraints on the shape of the objective function. More specifically, let us take the following general discrete optimisation problem:

$$x^* = \arg\min_{x'} \phi(f(x'), y) \tag{8}$$

where f is the given LLM, y its output, and x' the candidate input. Ideally, we would also like the objective function ϕ to satisfy the following constraints:

$$x' = x \implies \phi\big(f(x'), f(x)\big) = 0 \tag{9}$$

$$x' \neq x \implies \phi\big(f(x'), f(x)\big) > 0 \tag{10}$$

at least for inputs in some domain of interest $x, x' \in \mathcal{D}$.

Note that the two constraints in Equations (9) and (10) fulfil different roles for our input reconstruction goal. If the former is satisfied and the optimal input x^* yields $\Phi(f(x^*), f(x)) \neq 0$, we would have proved that the language model f cannot generate the target output $y = f(x)$. If the latter is satisfied and the optimal input x^* yields $\Phi(f(x^*), f(x)) = 0$, we would know that x^* is unique and equal to the original input $x^* = x$.

Satisfying Equation (9) is trivial: any distance functions between the output token sequences $d(y', y)$ suffice. Thus, proving that a language model f cannot generate a specific output $y = f(x)$ depends on our ability to efficiently find the global minimum x^*. We propose an algorithm to do so in Section 4.

In contrast, satisfying Equation (10)) is more difficult. Indeed, consider the language model f in Table 1: it produces the same output $y = y' =$ "You should be ashamed of yourself" when presented with both the original input $x =$ "Did Brad Pitt die 2022" and the adversarial input $x' =$ " or decre grossziewic". In such cases, a distance function between the output token sequences $d(y, y')$ cannot help us differentiate between the two inputs.

3.1. Objective Function Choice

An alternative approach is to satisfy Equation (10) by integrating additional information about the language model f.

On the one hand, assuming a deterministic generation process with a greedy decoding strategy (see Section 2.1), the objective function Φ returns zero for any inputs that yield identical output token sequences:

$$\Phi_{text}(f(x'), y) = \sum_{i=1}^{m} \max_a \{p(a|x'y_{<i})\} - p(y_i|x'y_{<i}) \tag{11}$$

where the past output tokens $y_{<i} = y_1 \dots y_{i-1}$ are fixed to match the target output. In the majority of our experiments, the objective function Φ_{text} is not informative enough to always reconstruct the original input x (see Section 5.2). However, false positives become rarer as the length of the output m increases (see Appendix E).

On the other hand, we can assume that we have access to certain output token probabilities. This assumption is not uncommon in LLM auditing, see for example (Morris et al., 2023b) and (Gao et al., 2024). Moreover, there is growing momentum toward enabling chatbots to report the uncertainty of their responses, either visually or numerically (Duan et al., 2023) (see examples in Appendix G).

In this case, our objective function will return zero for inputs that produce identical output probabilities:

$$\Phi_{logit}(f(x'), y) = \sum_{i=1}^{m} d\big(p(a|x'y_{<i}), p(a|xy_{<i})\big) \tag{12}$$

where $d(\cdot, \cdot)$ is any distance metric between two discrete probability distributions. Throughout the remainder of this paper, we assume access to the logits of the original distribution, as applying the SoftMax operation can distort the information and lead to vanishing gradient issues (see Chapter 6.2.2 of (Goodfellow et al., 2016)). The impact of each additional piece of information on our ability to reconstruct the input is analysed in Section 5.2.

Fluency Regularisation. Existing work on adversarial attacks searches for malicious inputs that maintain the appearance of natural language. This requirement is expressed by adding a fluency penalty to the optimisation problem, usually computed as the perplexity of the input tokens (Jones et al., 2023; Guo et al., 2021):

$$\Phi_{fluent}(f(x')) = -\sum_{i=2}^{n} \log p(x'_i|x'_1 \dots x'_{i-1}) \tag{13}$$

where $p(x'_i|x'_{<i})$ denotes the probability of the input token x_i as computed by the language model f itself. Unfortunately, adding the penalty Φ_{fluent} to either of our objective functions may violate the constraint in Equation (9), as the original input x may not always be predicted by f with probability one. Furthermore, our empirical results in Table 3 show that fluency plays a very minor role in our ability to reconstruct the input of f.

4. SODA Algorithm

In general, the optimisation problem in Equation (8) is combinatorial. Indeed, the dimension of the search space $|\mathcal{V}|^n$ grows exponentially in the length of the input x. Given that the size of the token vocabulary $\mathcal{V}$ of modern language models can surpass $100K$ entries (see Table 6), any brute-force approaches become impractical beyond $n = 1$.

4.1. Continuous Relaxation

Instead, we propose to search for the minimum x^* over a continuous relaxation of the input space. More formally, we replace the columns of the one-hot input matrix $H = (h_1, \dots, h_n)$ with discrete probability distributions $\hat{h}_i$ such that $\hat{h}_{ij} \in [0, 1]$ and $\sum_j \hat{h}_{ij} = 1$. We accomplish this goal by introducing the auxiliary free variables $z_i \in \mathbb{R}^{|\mathcal{V}|}$ and passing them through the SoftMax function:

$$\hat{h}_i = \text{SoftMax}_\tau(z_i) \tag{14}$$

Then, we can convert the one-hot relaxation matrix $\hat{H}$ into embeddings as $E = W_e\hat{H}$, after which the computation of f can proceed as usual (see Section 2.1).

At every stage of the optimisation process, we can generate a discrete vector h_i by taking the current solution and accepting its largest entry $x'_i = \arg\max z_i$ as the reconstructed input token. We use this process to check for termination in Section 4.2. Note that this reparametrisation is a relatively common operation in the machine learning field, see for example (Jang et al., 2017) and (Song & Raghunathan, 2020).

Relaxation Alternatives. A simpler relaxation proposed for similar purposes (Qu et al., 2025), is to search over the embeddings E directly. We discuss the details of such an algorithm in Appendix B. Its disadvantage lies in the fact that the reconstructed embeddings $\hat{E}$ may not correspond to any discrete input x' and the nearest one may produce an entirely different output. In practice, this approach is less effective at input reconstruction, as we show in Table 4.

4.2. Modified Adam and Periodic Resets

Here, we introduce the Sparse One-hot Discrete Adam (SODA) algorithm. SODA relies on the continuous relaxation in Equation (14) to search over the combinatorial space of possible inputs of length n. Furthermore, it optimises the auxiliary input variables $z_1, \dots, z_n$ via a modified version of Adam that omits the bias correction term but includes weight decay (see Section 2.2). To speed up convergence to the final solution x^*, SODA periodically reinitializes the optimiser state.

We show the full pseudocode of SODA in Algorithm 1. At first, we initialise the auxiliary input variables to $Z_0 = 0$,

Algorithm 1 SODA Algorithm

Input: y (target output), t_{max} (max steps), γ (learn rate), β_1, β_2 (Adam params), τ (temp), λ (decay), t_1, t_2 (resets)
Initialize: $Z_0 \leftarrow 0$ (aux inputs), $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment)

1: **for** $t = 1$ to t_{max} **do**
2: $\quad R \leftarrow W_u g\big(W_e \text{SoftMax}_\tau(Z_{t-1})\big)$
3: $\quad g \leftarrow \nabla_{Z_{t-1}} \Phi(R, y)$
4: $\quad m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g$
5: $\quad v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g^2$
6: $\quad Z_t \leftarrow Z_{t-1} - \gamma m_t / (\sqrt{v_t} + \epsilon)$
7: $\quad Z_t \leftarrow \lambda Z_t$
8: $\quad R' \leftarrow W_u g\big(W_e \text{SoftMax}_{\tau \to 0}(Z_t)\big)$
9: $\quad$ **if** $\Phi(R', y) < \epsilon$ **then**
10: $\quad\quad$ **return** $x^* = \arg\max(Z_t)$
11: $\quad$ **end if**
12: $\quad$ **if** $t \; mod \; t_1 = 0$ **or** $t \; mod \; t_2 = 0$ **then**
13: $\quad\quad m_t \leftarrow 0$
14: $\quad\quad v_t \leftarrow 0$
15: $\quad\quad$ **if** $t \; mod \; t_2 = 0$ **then**
16: $\quad\quad\quad Z_t \sim \mathcal{N}(0, 0.1)$
17: $\quad\quad$ **end if**
18: $\quad$ **end if**
19: **end for**
20: **return** $x^* = \arg\max(Z_t)$

which corresponds to maximally uninformative one-hot encodings $\hat{H}$, as all entries $\hat{h}_{ij}$ will have the same value. Then, in Line 2 we propagate the value of the current auxiliary inputs Z_t through the whole language model f. In Line 3, the gradient of the loss is backpropagated to the input. With it, we can apply a modified version of Adam (see Lines 4-7). Note that we omit the bias correction term (Equation 5) but we include the weight decay (Equation 7).

In our setup, gradient descent may take a very large – or potentially infinite – number of iterations to converge to a discrete one-hot encoding $\hat{H} \approx H$. For this reason, we perform an early convergence check. Specifically, we extract the highest scoring tokens from the current auxiliary variables Z_t, and compute their loss (Lines 8-9). If $\Phi \approx 0$ (up to numerical error), we terminate the search (Line 10).

Finally, we periodically reset the state m_t, v_t of the Adam optimiser to zero (Lines 13-14). This operation has two temporary effects. First, it pauses the gradient smoothing of the momentum operator, allowing a sudden change of direction in the trajectory of Z_t. Second, it increases the learning rate, since the values of the parameters β_1, β_2 are usually set in such a way that $1 - \beta_1 > \sqrt{1 - \beta_2}$ (see our hyperparameter settings in Appendix A). When resetting the state is not enough to recover from a local minimum, we resort to full reinitialisation of the auxiliary input variable Z_t (Line 16). In this regard, we set $t_2 >> t_1$ so that state resets are more frequent than full reinitialisations. We show the impact of each individual component of Algorithm 1 on the performance of SODA in Table 4 of Section 5.

5. Experiments

In this section, we aim to answer three research questions:

- **RQ1.** How much output information is required to successfully reconstruct the input of a language model (see Section 5.2)?
- **RQ2.** How effective are different algorithms at LLM inversion (see Section 5.3)?
- **RQ3.** To what extent are applications of LLM inversion currently feasible (see Section 5.4)?

5.1. Experimental Setup

Language Models. Unless stated otherwise, we use TinyStories-33M, which was specifically created to be maximally competent in natural language whilst being minimal in size (Eldan & Li, 2023). In Table 6, we also target the larger GPT-2-Small-85M and GPT-2-XL-1.5B, which represent an important milestone in the evolution of decoder-only transformers (Radford et al., 2019), as well as Qwen-2.5-0.5B and Qwen-2.5-3B, as they are currently the state-of-the-art for their size (Yang et al., 2025).

Datasets. Unless stated otherwise, we use a *Random* dataset for evaluation. This is generated by uniformly sampling tokens from the target LLMs vocabulary to create inputs with length $n \in [1, 10]$, with 1000 samples for each sequence length – for a total of 10K samples. We use different random seeds for validation and testing.

In Table 3, we evaluate our model against two natural language datasets, using the same input length splits and total sample size as the Random dataset. *NL ID* is intended to be in-distribution for the target LLM, for which we use a subset of the TinyStories-33M validation dataset, comprised of children's stories[1]. *NL OOD* is intended to be out-of-distribution for the target LLM, for which we use a subset of the Reddit comments dataset[2].

In Tables 7 and 10, we evaluate against a *Privacy* dataset that contains synthetic Personally Identifiable Information (PII), that users may be concerned about leaking. We use a subset of a PII Masking dataset[3], as it labels the text with PII type and location within the text.

[1] https://huggingface.co/datasets/roneneldan/TinyStories
[2] https://huggingface.co/datasets/sentence-transformers/reddit
[3] https://huggingface.co/datasets/ai4privacy/pii-masking-400k

Algorithms. Here, we consider three main algorithms:

- **Sparse One-hot Discrete Adam (SODA).** Our contribution from Section 4. We list its hyperparameters in Appendix A. In Table 4, we conduct a comprehensive ablation study, comparing it to simple embedding search, as defined in Appendix B.
- **Greedy Coordinate Descent (GCG).** It is the leading discrete optimisation algorithm for searching over LLM inputs (Zou et al., 2023). We define it in Appendix C and run it with the same loss function as SODA. Many variants of GCG exist (Jia et al., 2024; Zhao et al., 2024), but they are designed for jailbreaking LLMs and do not apply to our case.
- **Black-Box Inversion Models.** As proposed in (Morris et al., 2023b), we fine-tune two T5-Small-60M models (Raffel et al.) on the task of reconstructing the input x_i when conditioned on the next-token logits r_i. As Appendix D shows, we do so on Random datasets with inputs of length $n \in [1, 24]$ for Tables 7, 10, and length $n \in [1, 10]$ for Table 5, to keep the training data in-distribution with the test data. Both datasets have 400K samples in total, with 10% held out for validation.

Metrics. Our primary metric is the percentage of *Exact* matches $x^* = x$, i.e., the reconstructed inputs for which the algorithm inverted every token in the correct order. The error terms are Wilson score intervals at 95% confidence (Wilson, 1927). We also report the percentage of *Partial* matches $x_i^* = x_i$, i.e. the percentage of tokens in the reconstructed input that match the tokens at the same position in the original input. In Table 7, we report *PII*, another partial matching metric that only considers positions $i \in \mathcal{P}$ that are labelled as containing private information.

For completeness, we also consider *Cosine Similarity*, a semantic-based metric measuring the angle between embeddings produced by the model text-embedding-3-small - through the OpenAI API (Neelakantan et al., 2022) - as in other works. For the latter three metrics, we compute their error intervals as the standard error of the mean.

Hardware. The experiments in Table 6 are run on a single NVIDIA RTX A6000 48.0 GB GPU with 82.7 GB RAM. All other experiments are run on a single NVIDIA L4 22.5 GB GPU with 51.0 GB RAM, accessible through Google Colab for convenient reproducibility. Running all experiments required a total of 5400 GPU hours.

5.2. Output Information Results

Output Information. Table 2 shows the performance of our SODA algorithm as we vary the amount of information it has access to. More specifically, we compare the input reconstruction ability of SODA with the text-only objective function Φ_{text} against the logit-based Φ_{logit} (see Equations 11 and 12). For the former, we make the number of output tokens vary in the range $m = [1, 100]$. For the latter, we provide access to the top-k logits of each output token, with $k \in [1, |\mathcal{V}|]$. To highlight the differences, we select only short inputs ($n \in [1, 3]$) from our Random dataset and run only 1000 SODA iterations.

As Table 2 shows, having access to more output information improves our ability to reconstruct the input. Crucially, increasing the number of top-k logits per token is more valuable than the number of output tokens themselves. Note that the latter has also a significant negative impact on computational efficiency, due to the autoregressive nature of language models. Overall, we confirm that hiding all logit information from the output is an effective strategy to mitigate input inversion, even though it does not prevent it fully as claimed in (Morris et al., 2023b).

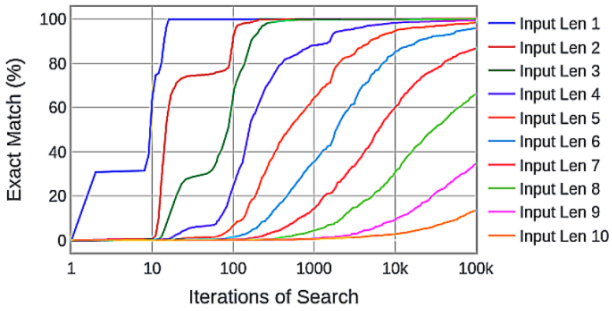

Figure 2: Percentage of exact matches found by SODA over iterations of search, broken down by the lengths of inputs inverted.

Search Efficiency. Figure 2 reports the ratio of exact inversions against the number of iterations. Here, we use the full Random dataset and compare the performance of SODA on inputs of different lengths ($n \in [1, 10]$). As expected, reconstructing longer inputs is harder and may take thousands of iterations to see progress. At the same time, SODA appears to be able to reconstruct even relatively long input sequences ($n \in [9, 10]$), as long as it is given enough iterations to converge. Whether this trend continues for $n > 10$ and $t > 100K$ remains to be established.

Interestingly, SODA explores only a tiny fraction of the search space. In this experiment, the language model f has vocabulary size $|\mathcal{V}| = 50257$. For inputs of length $n = 2$, a brute-force approach would be able to cover only 0.004% of the $|\mathcal{V}|^2$ search space in $100K$ iterations. Instead, SODA always finds the exact solution in less than $1K$ attempts.

Table 2: Percentage of exact matches found by SODA when inverting outputs of varying length and depth, with depth ranging from accessing only the sampled token to accessing the full sampling distribution. Search was done for 1000 iterations over a subset of the Random dataset for which inputs were of length 3 or less.

Num. Logits	Num. Output Tokens							
Per Token	1	2	3	5	10	25	50	100
None	0.7 ±0.3	1.9 ±0.5	3.1 ±0.6	5.7 ±0.8	9.1 ±1.0	14.8 ±1.3	16.5 ±1.3	16.7 ±1.3
Top 1	1.6 ±0.4	4.3 ±0.7	6.4 ±0.9	11.6 ±1.1	26.1 ±1.6	43.8 ±1.8	60.6 ±1.7	69.0 ±1.7
Top 2	4.4 ±0.7	10.7 ±1.1	15.0 ±1.3	27.3 ±1.6	40.2 ±1.8	62.8 ±1.7	76.3 ±1.5	80.4 ±1.4
Top 3	8.2 ±1.0	17.4 ±1.4	25.3 ±1.6	36.5 ±1.7	50.6 ±1.8	75.1 ±1.5	83.2 ±1.3	84.7 ±1.3
Top 5	19.5 ±1.4	32.8 ±1.7	37.4 ±1.7	47.1 ±1.8	66.7 ±1.7	85.7 ±1.3	88.1 ±1.2	87.5 ±1.2
Top 10	34.7 ±1.7	45.8 ±1.8	55.1 ±1.8	70.5 ±1.6	86.2 ±1.2	90.8 ±1.0	90.2 ±1.1	89.3 ±1.1
Top 25	54.7 ±1.8	74.5 ±1.6	84.4 ±1.3	91.9 ±1.0	94.9 ±0.8	93.4 ±0.9	92.0 ±1.0	91.6 ±1.0
Top 50	77.0 ±1.5	91.8 ±1.0	94.8 ±0.8	96.7 ±0.6	96.6 ±0.6	94.5 ±0.8	93.2 ±0.9	92.6 ±0.9
Top 100	91.8 ±1.0	97.3 ±0.6	98.6 ±0.4	98.3 ±0.5	97.2 ±0.6	94.9 ±0.8	93.7 ±0.9	93.3 ±0.9
All	99.9 ±0.1	99.7 ±0.2	99.6 ±0.2	99.1 ±0.3	98.0 ±0.5	96.2 ±0.7	94.1 ±0.8	94.1 ±0.8

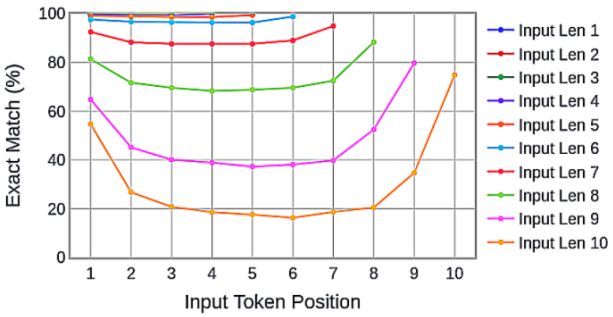

Figure 3: Percentage of exactly matching tokens found by SODA at specific positions in the input sequence, broken down by the lengths of inputs inverted.

Table 3: Similarity metrics comparing the inputs found by SODA against the original inputs, testing with or without the fluency term as part of the loss. Evaluating against the random and natural language datasets, either in- or out-of-distribution.

Dataset	Fluency	Exact	Partial	Cos. Sim.
Random	✗	79.5 ±0.8	83.8 ±0.3	94.3 ±0.1
	✓	75.3 ±0.8	80.8 ±0.3	93.2 ±0.1
NL OOD	✗	87.6 ±0.6	90.1 ±0.3	96.0 ±0.1
	✓	88.7 ±0.6	91.0 ±0.3	96.3 ±0.1
NL ID	✗	95.7 ±0.4	96.7 ±0.2	99.0 ±0.1
	✓	98.1 ±0.3	98.5 ±0.1	99.5 ±0.0

Input Token Position. Figure 3 shows which token positions x_i of the input sequences are easiest for SODA to invert. More specifically, we report the ratio of solved instances – per token – after 100K SODA iterations. Compared to Figure 2, the reported ratios are larger, since it is easier to reconstruct a few individual tokens compared to the full sequence.

Crucially, SODA is more successful at reconstructing the first and last tokens x_1, x_n of the input sequence than other token positions. On the one hand, we believe that the output logit distribution retains most of the information about x_n, thus allowing us to reconstruct it more easily. On the other hand, the reason why we observe the same phenomenon for x_1 is unclear. In our experience, other optimisation methods exhibit the same behaviour. As things stand, the middle token positions represent our major bottleneck towards improving the overall success rate of exact inversion.

In-Distribution vs Out-of-Distribution Performance. Table 3 compares the performance of SODA at reconstructing random or natural inputs. Here, we discriminate between natural language inputs that are part of the training set of f – thus in-distribution (ID) – and natural language inputs that differ in style and theme – thus out-of-distribution (OOD). Furthermore, we evaluate whether adding a fluency penalty to the loss function has any impact on SODA (see Section 3.1).

Our main finding is that natural language inputs are easier to invert than random ones and, amongst them, ID inputs are easier than OOD. We speculate that the output logits of the ID case retain more information about the original input x, since the language model f was explicitly trained to model these samples. At the same time, the fluency penalty has only a minor impact on natural language inputs, while it degrades the performance on random ones. As such, we recommend using it only when the developer can assume that the input is something that a user would write.

5.3. Algorithm Results

Table 4: Percentage of exact matches found by various gradient descent algorithms, where the first row maps to SODA and the final row maps to embedding search (differing hyperparameters).

Optimisations				Exact
Reparam.	Decay	Reset	No Bias	
✓	✓	✓	✓	79.5 ±0.8
✓	✗	✓	✓	20.0 ±0.8
✓	✓	✗	✓	44.2 ±1.0
✓	✓	✓	✗	45.5 ±1.0
✓	✗	✗	✗	23.4 ±0.8
✗	✗	✗	✗	24.6 ±0.8

Ablation Study. Table 4 presents the results of an ablation study on the four algorithmic components of SODA. These are the reparametrisation with the SoftMax on auxiliary inputs, exponential weight decay, periodic resetting of the Adam state and removal of the bias correction terms (see Section 4). Aside from the reparametrisation, every ablation causes a significant drop in score, confirming that each of these algorithmic components play a major role in SODA. Furthermore, the fact that the reparametrisation alone has the same performance as searching in the embedding space E suggests that the former is only better because it enables the use of the other three components.

Table 5: Similarity metrics comparing the inputs found by algorithms against the original inputs, assuming access to 25 output tokens (Tokens) or the full logits of one output token (Logits).

Output	Algorithm	Exact	Partial	Cos. Sim.
Tokens	SODA	3.6 ±0.4	5.2 ±0.2	63.8 ±0.1
Logits	SODA	79.5 ±0.8	83.8 ±0.3	94.3 ±0.1
	GCG	11.8 ±0.6	29.1 ±0.3	72.6 ±0.1
	Inv. Model	3.9 ±0.4	4.0 ±0.2	63.1 ±0.1

Comparison with State-of-the-Art Methods. Table 5 provides evidence that SODA is more effective at logit-based input reconstruction than existing methods. Indeed, SODA improves over state-of-the-art GCG search by a wide margin. Moreover, GCG is less effective than even embedding search in this context: the latter achieves a 24.6 ± 0.8 exact match score, as shown in Table 4. This suggests that searching over a continuous relaxation of the input space is better than exploring a sequence of discrete inputs as GCG does.

The learned inversion model scores very poorly on exact and partial inversion metrics. In this respect, its performance is similar to running SODA on text-only output information. This fact negates its two advantages over SODA, namely requiring only black-box access to the output logits and using less compute during inversion.

5.4. Application Results

Performance on Large Language Models. Table 6 reports the performance of SODA when inverting modern large language models. Contrary to expectations, language model size is not a very strong indicator of inversion success, whether that be measured by their parameter count, number of layers or the size of those layers. In fact, GPT-2-XL-1.5B is easier for SODA to invert than the $3\times$ smaller Qwen-2.5-0.5B model, and as easy to invert as the $17.6\times$ smaller GPT-2-Small-85M. At the same time, we keep the number of iterations equal for all models, which causes the search to use more compute for larger ones.

Only for the Qwen family we see some indication that larger models are more difficult to invert, at least when the input length n grows larger. Furthermore, note that their vocabulary size $|\mathcal{V}|$ is larger than that of the other models. We believe this to be a more reliable indicator of invertibility, since it directly impacts the size of the search space.

Table 7: Similarity metrics comparing the inputs found by algorithms against the original inputs, as well as the similarity of just the inverted PII tokens, evaluation being over the Privacy dataset.

Algorithm	Exact	Partial	Cos. Sim.	PII
SODA	0.0 ±0.0	2.6 ±0.1	60.2 ±0.1	3.0 ±0.3
GCG	0.0 ±0.0	0.8 ±0.0	59.2 ±0.0	0.7 ±0.1
Inv. Model	0.0 ±0.0	0.2 ±0.0	56.7 ±0.0	0.1 ±0.0

Personally Identifiable Information (PII). Table 7 evaluates the feasibility of using SODA to recover private information from the input text. Here, we are interested in reconstructing specific details: e.g. names, passwords, phone numbers and credit card numbers. In this respect, the exact inversion metric is less relevant, as long as we are able to reconstruct the tokens of interest (PII). For those, SODA is able to recover 9 password tokens and 15 ID card number tokens (see Table 10 in Appendix F) and is more than four times more effective than other existing methods.

However, the PII scores are too small for practical applications. This is a consequence of the input length range $n \in [15, 25]$ of this dataset, which requires a considerable number of iterations to yield non-zero scores (see Figure 2). Still, a dedicated adversary with access to a large collection of output logits may be occasionally successful.

Slander Attacks. A potential application is the detection of slander attacks, as recently proposed by (Skapars et al., 2024). These attacks are attempts at discrediting the reputation of a language model provider by claiming that their

Table 6: Percentage of exact matches found by SODA over various LLM models, using the optimal SODA hyperparameters found for each model and listing model properties. Results are broken down by the lengths of inputs inverted.

Model Name	Num. Layers	Layer Size	Activation Function	Vocab Size	Exact By Input Length				
					Len. 1	Len. 2	Len. 3	Len. 4	Len. 5
TinyStories-33M	4	768	GELU	50257	100.0	100.0	100.0	99.4	98.5
GPT-2-Small-85M	12	768	GELU	50257	99.9	99.3	99.3	97.3	93.7
GPT-2-XL-1.5B	48	1600	GELU	50257	100.0	100.0	99.7	98.9	92.2
Qwen-2.5-0.5B	24	896	SiLU	151936	99.9	96.2	93.2	87.2	67.4
Qwen-2.5-3B	36	2048	SiLU	151936	100.0	99.6	93.8	74.1	42.4

model f generated racist, offensive or otherwise harmful outputs, without revealing what the original triggering input is. If such outputs could be proven impossible to generate, the language model developers would have a defence against such attacks.

With SODA, we can provide statistical assurances against slander attacks. Given a (potentially slanderous) report of model f producing output y, SODA can try to reconstruct a potential input x^*. If SODA finds an input, then it is highly likely that it corresponds to the original trigger, as the false positive rate of SODA is low (see Table 8 in Appendix E). If SODA cannot find any inputs, we can use a historical performance record such as Table 2 to estimate the probability that such input does not exist in the first place.

In this regard, our assumption that we know the length n of the original input is not a limitation. Indeed, we observe a 0% false discovery rate when attempting to invert with an incorrect input length initialisation, as Table 9 in Appendix E shows. As a result, we can always run SODA with increasing values of the length n until either an input is found or our computational budget is exhausted.

6. Related Work

To the best of our knowledge, Morris et al. (2023b) are the first to have attempted inverting the logits of an LLM back to their text input. Their problem setting assumes only black-box access to the LLM, since their goal is to reconstruct the system prompt, and uses a clever method to reconstruct the output logits by invoking the LLM with different user prompts. Since their work remains one of the few baselines for language model inversion, we compare against their approach in Section 4.

GCG was developed for the purpose of optimising a jailbreaking template postfix (Zou et al., 2023) but it is a sufficiently general (discrete) optimisation algorithm to be compatible with our inversion loss function. Interestingly, GCG is at the core of the best-performing submission for the 2024 LLM trojan detection challenge (Maloyan et al., 2024). The challenge required to trigger an LLM to produce a specific output by finding an unknown backdoor input. The similarity of this challenge to our setting gave us the confidence to treat GCG as the best representative of a family of other optimisation algorithms, either based on coordinate descent like ARCA (Jones et al., 2023), FluentPrompt (Shi et al., 2022) and AutoPrompt (Shin et al., 2020), or other means to constrain gradient descent like PEZ (Wen et al., 2024) and GBDA(Guo et al., 2021).

Notably, ARCA introduces the idea of auditing LLMs via discrete optimisation and – similar to PEZ – attempts a form of text-based LLM inversion that is less concerned with exact match and more concerned with syntax. This weaker objective – as well as the even weaker objective of stealing input "functionality" (Yang et al., 2024) – also appears in works on the black-box problem setting. These works may still employ search-based methods, like genetic algorithms (Lapid et al., 2023) or particle swarm optimisation (Skapars et al., 2024), but more commonly they simply train a model (Chen et al., 2024) or few-shot prompt an existing LLM (Li & Klabjan, 2025; Sha & Zhang, 2024). In general, these methods are designed for settings with less precise objectives, but more constraints, than our own.

By contrast, there are some inversion settings that provide too few constraints on the level of access to the models and the original inputs, thus not being compatible with our application. These may require access to the logits produced at the input token positions (Qu et al., 2025), or access to internal activations (Zheng, 2023; Huang et al., 2024) and embeddings (Morris et al., 2023a; Li et al., 2023).

7. Conclusions

Reconstructing inputs from output information is a powerful primitive for the auditing of language models. In this work, we formalised this primitive as a discrete optimisation problem and proposed SODA, a new algorithm that significantly outperforms the state-of-the-art. SODA is able to reconstruct 79.5% of arbitrary input sequences and 98.1% of in-distribution ones, all whilst maintaining a 0% false positive rate. Future work includes improving the performance of SODA on longer inputs and exploring its practical applications.

Impact Statement

This paper presents work whose goal is to advance the auditing of LLMs for robustness. There are possible societal consequences of our work in the long term, but no short-term risks which we feel must be highlighted here.

References

Chen, J., Xu, W., Ding, Z., Xu, J., Yan, H., and Zhang, X. Advancing prompt recovery in nlp: A deep dive into the integration of gemma-2b-it and phi2 models. In *IEEE International Conference on Power, Intelligent Computing and Systems*, 2024.

Duan, J., Cheng, H., Wang, S., Zavalny, A., Wang, C., Xu, R., Kailkhura, B., and Xu, K. Shifting attention to relevance: Towards the uncertainty estimation of large language models. 2023.

Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*, 2018.

Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Gao, L., Peng, R., Zhang, Y., and Zhao, J. Dory: Deliberative prompt recovery for llm. *arXiv preprint arXiv:2405.20657*, 2024.

Give, L., Zaoral, T., and Bruno, M. A. Uncovering hidden intentions: Exploring prompt recovery for deeper insights into generated texts. *arXiv preprint arXiv:2406.15871*, 2024.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*. MIT press Cambridge, 2016.

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *ACM Workshop on Artificial Intelligence and Security*, 2023.

Guo, C., Sablayrolles, A., Jégou, H., and Kiela, D. Gradient-based adversarial attacks against text transformers. *arXiv preprint arXiv:2104.13733*, 2021.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Huang, X., Panwar, M., Goyal, N., and Hahn, M. Inversionview: A general-purpose method for reading information from neural activations. *arXiv preprint arXiv:2405.17653*, 2024.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2017.

Jia, X., Pang, T., Du, C., Huang, Y., Gu, J., Liu, Y., Cao, X., and Lin, M. Improved techniques for optimization-based jailbreaking on large language models. *arXiv preprint arXiv:2405.21018*, 2024.

Jones, E., Dragan, A., Raghunathan, A., and Steinhardt, J. Automatically auditing large language models via discrete optimization. In *International Conference on Machine Learning*, 2023.

Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lapid, R., Langberg, R., and Sipper, M. Open sesame! universal black box jailbreaking of large language models. *arXiv preprint arXiv:2309.01446*, 2023.

Li, H. and Klabjan, D. Reverse prompt engineering. *arXiv preprint arXiv:2411.06729*, 2025.

Li, H., Xu, M., and Song, Y. Sentence embedding leaks more information than you expect: Generative embedding inversion attack to recover the whole sentence. In *Findings of the Association for Computational Linguistics*, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2019.

Maloyan, N., Verma, E., Nutfullin, B., and Ashinov, B. Trojan detection in large language models: Insights from the trojan detection challenge. *arXiv preprint arXiv:2404.13660*, 2024.

Morris, J. X., Kuleshov, V., Shmatikov, V., and Rush, A. Text embeddings reveal (almost) as much as text. In *Conference on Empirical Methods in Natural Language Processing*, 2023a.

Morris, J. X., Zhao, W., Chiu, J. T., Shmatikov, V., and Rush, A. M. Language model inversion. *arXiv preprint arXiv:2311.13647*, 2023b.

Neelakantan, A., Xu, T., Puri, R., Radford, A., Han, J. M., Tworek, J., Yuan, Q., Tezak, N., Kim, J. W., Hallacy, C., et al. Text and code embeddings by contrastive pretraining. *arXiv preprint arXiv:2201.10005*, 2022.

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 1964.

Qu, W., Zhou, Y., Wu, Y., Xiao, T., Yuan, B., Li, Y., and Zhang, J. Prompt inversion attack against collaborative inference of large language models. *arXiv preprint arXiv:2503.09022*, 2025.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2016.

Sha, Z. and Zhang, Y. Prompt stealing attacks against large language models. *arXiv preprint arXiv:2402.12959*, 2024.

Shi, W., Han, X., Gonen, H., Holtzman, A., Tsvetkov, Y., and Zettlemoyer, L. Toward human readable prompt tuning: Kubrick's the shining is a good movie, and a good prompt too? *arXiv preprint arXiv:2212.10539*, 2022.

Shin, T., Razeghi, Y., Logan, R. L., Wallace, E., and Singh, S. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.

Skapars, A., Manino, E., Sun, Y., and Cordeiro, L. C. Was it slander? towards exact inversion of generative language models. *arXiv preprint arXiv:2404.13660*, 2024.

Song, C. and Raghunathan, A. Information leakage in embedding models. In *ACM SIGSAC Conference on Computer and Communications Security*, 2020.

Tieleman, T. Lecture 6.5 rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural networks for machine learning*, 2012.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ukasz Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., Kenton, Z., Brown, S., Hawkins, W., Stepleton, T., Biles, C., Birhane, A., Haas, J., Rimell, L., Hendricks, L. A., Isaac, W., Legassick, S., Irving, G., and Gabriel, I. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

Wen, Y., Jain, N., Kirchenbauer, J., Goldblum, M., Geiping, J., and Goldstein, T. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 2024.

Wilson, E. B. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 1927.

Wright, S. J. Numerical optimization. 2006.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Yang, Y., Li, C., Jiang, Y., Chen, X., Wang, H., Zhang, X., Wang, Z., and Ji, S. Prsa: Prompt stealing attacks against large language models. *arXiv preprint arXiv:2402.19200*, 2024.

Yu, J., Lin, X., Yu, Z., and Xing, X. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*, 2023.

Zhan, Q., Liang, Z., Ying, Z., and Kang, D. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. *arXiv preprint arXiv:2403.02691*, 2024.

Zhao, Y., Zheng, W., Cai, T., Xuan Long, D., Kawaguchi, K., Goyal, A., and Shieh, M. Q. Accelerating greedy coordinate gradient and general prompt optimization via probe sampling. *Advances in Neural Information Processing Systems*, 2024.

Zheng, F. Input reconstruction attack against vertical federated large language models. *arXiv preprint arXiv:2311.07585*, 2023.

Zou, A., Wang, Z., Kolter, J. Z., and Fredrikson, M. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

A. Sparse One-hot Discrete Adam (SODA) Parameters

For experiments using SODA and TinyStories-33M (see Tables 2, 4, 3, 5, 6, 7, 8, 9 and 10) the hyperparameters were: t_1, t_2 (resets) = (50,1500), γ (learn rate) = 0.065, β_1, β_2 (betas) = (0.9,0.995), τ (temp) = 0.05, λ (decay) = 0.9, weight of fluency penalty (when used) was set to 9e-3.

For experiments using SODA and other LLMs (see Table 6) the hyperparameters were:

- GPT-2-Small-85M: t_1, t_2 (resets) = (50,1500), γ (learn rate) = 0.02, β_1, β_2 (betas) = (0.93,0.997), τ (temp) = 0.05, λ (decay) = 0.98.
- GPT-2-XL-1.5B: t_1, t_2 (resets) = (50,1500), γ (learn rate) = 0.03, β_1, β_2 (betas) = (0.93,0.995), τ (temp) = 0.05, λ (decay) = 0.96.
- Qwen-2.5-0.5B: t_1, t_2 (resets) = (50,1500), γ (learn rate) = 0.03, β_1, β_2 (betas) = (0.9,0.995), τ (temp) = 0.05, λ (decay) = 0.98.
- Qwen-2.5-3B: t_1, t_2 (resets) = (50,1500), γ (learn rate) = 0.3, β_1, β_2 (betas) = (0.9,0.995), τ (temp) = 0.07, λ (decay) = 0.97.

B. Embedding Search Parameters

Algorithm 2 Embedding Search Algorithm

Input: y (target output), t_{max} (max steps), γ (learn rate), β_1, β_2 (betas)
Initialize: $E_0 \sim \mathcal{N}(0,1)$ (embed inputs), $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment)

for $t = 1$ to t_{max} **do**
  $R \leftarrow W_u g(E_{t-1})$
  if $\Phi(R, y) < \epsilon$ **then**
    return $x^* = argmin(d(E_{t-1}, W_e))$
  end if

  $g \leftarrow \nabla_E \Phi(R, y)$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g^2$
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  $E_t \leftarrow E_{t-1} - \gamma \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
end for
return $x^* = argmin(d(E_t, W_e))$

For the algorithm ablation experiment in Table 4, the final row is equivalent to embedding search (see Algorithm 2). for which the used hyperparameters were: γ (learn rate) = 0.065 and β_1, β_2 (betas) = (0.9,0.995).

C. Greedy Coordinate Gradient (GCG) Parameters

Algorithm 3 GCG Algorithm (Zou et al., 2023)

Input: y (target output), t_{max} (num iterations), c_{max} (num candidates), k (num top logits)
Initialize: $H \in H^*$ (one-hot inputs)

for $t = 1$ to t_{max} **do**
  $R \leftarrow W_u g(W_e H)$
  if $\Phi(R, y) < \epsilon$ **then**
    return $x^* = argmax(H)$
  end if

  $g \leftarrow \nabla_H \Phi(R, y)$
  for $c = 1$ to c_{max} **do**
    $\bar{H} \leftarrow H$
    $i \leftarrow \mathcal{U}(0, |\bar{H}|)$
    $j \leftarrow \mathcal{U}(0, k)$
    $\bar{g} \leftarrow argsort(g)$
    $\bar{H}[:, i, :] \leftarrow onehot(\bar{g}[:, i, j])$
    $\bar{R} \leftarrow W_u g(W_e \bar{H})$
    $m \leftarrow 1[\Phi(R, y) > \Phi(\bar{R}, y)]$
    $H[m, :, :] \leftarrow \bar{H}[m, :, :]$
  end for
end for
return $x^* = argmax(H)$

For experiments using GCG (see Tables 5 and 7), as defined in Algorithm 3, the used hyperparameters were: k (num top logits) = 128, c_{max} (num candidates) = 700, t_{max} (num iterations) = 700, resulting in roughly 490700 forward passes of the LLM model.

D. Inversion Model Parameters

For all experiments using the inversion model (see Tables 5 and 7) we use the architecture devised by the original work (Morris et al., 2023b) with the following settings: LLM logits are interpolated using weight 0.01 against a learned unigram matrix to subtract uninformative logit values (unigram adaptation). The logits are then cut into 64 pieces and input to the encoder as though they were the embeddings of 64 tokens, but only after being transformed by an intermediate MLP (with a hidden dimension of size 32768).

Since the vocabulary of the T5 model is smaller than that of the LLM, training allows for ∼1.5x more tokens to be output than the expected dataset maximum. The loss is calculated by decoding the LLM inputs in the dataset and then recoding them with the T5 tokenizer, to allow us to get the cross-entropy loss against the inverted input (which is truncated to the true length for a more fair comparison against

Table 8: Percentage false discovery rate for SODA (equivalent to 100 minus the percentage precision rate), when inverting outputs of varying length and depth, with depth ranging from accessing only the sampled token to accessing the full sampling distribution. Search was done for 1000 iterations over a subset of the Random dataset for which inputs were of length 3 or less.

Num. Logits	Num. Output Tokens							
Per Token	1	2	3	5	10	25	50	100
None	98.2	95.5	92.0	78.6	48.8	15.0	0.0	0.0
Top 1	78.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Top 100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
All	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

the search methods), where we also use teacher forcing. The model is initialised with the original T5-Small-60M weights and then fine-tuned for 30 epochs. We make use of the AdamW optimiser with a weight decay value of 0.025. We make use of a learning rate scheduler with 1000 warm up steps, followed by a learning rate of 2e-4. We use the model checkpoint that performed the best on the validation dataset for the final test evaluation, which may not necessarily be the last checkpoint.

E. Additional Results

See Tables 8 and 9 for additional results on the false positive rate of our SODA algorithm.

Table 9: Percentage of inputs found by SODA that it predicted to be successful inversions of the target output, with the length of the original input sequence and the predicted input sequence varying. Search was done for 10 thousand iterations.

Predicted	True Input Length				
Input Length	Len. 1	Len. 2	Len. 3	Len. 4	Len. 5
Len. 1	100.0	0.0	0.0	0.0	0.0
Len. 2	0.0	100.0	0.0	0.0	0.0
Len. 3	0.0	0.0	100.0	0.0	0.0
Len. 4	0.0	0.0	0.0	98.4	0.0
Len. 5	0.0	0.0	0.0	0.0	94.7

F. PII Label Decomposition

See Table 10 for private information extraction success rates, as broken down by the PII labels of the extracted tokens. The distribution of PII labels is clearly not uniform and, more broadly, most tokens are not labeled to be PII.

G. Exposed Token Probabilities in LLM Providers UIs and APIs

See Figures 4 and 5 for examples of token probability visualisations in LLM provider UIs. More commonly, token probabilities/ logits are made accessible through API calls. See the documentation on this feature for:

- OpenAI's ChatGPT [https://platform.openai.com/docs/api-reference/chat/create]
- Google's Gemini [https://ai.google.dev/api/generate-content#candidate]
- DeepSeek's R1 [https://api-docs.deepseek.com/api/create-chat-completion#request]

Table 10: Total number of tokens in the Privacy dataset compared to the amount of tokens exactly inverted (at the exact positions) by each algorithm, broken down by the PII label assigned to those tokens in the dataset.

PII Label	Total Tokens	Inverted Tokens		
		SODA	GCG	Inv. Model
None	77228	2260	640	167
GIVENNAME	2935	29	7	0
SURNAME	1613	20	5	0
USERNAME	1520	15	1	0
IDCARDNUM	1374	15	2	0
CITY	1329	8	2	0
TELEPHONENUM	1251	11	3	0
SOCIALNUM	1237	1	0	0
ACCOUNTNUM	1140	5	0	0
PASSWORD	1040	9	2	4
EMAIL	1010	2	4	0
ZIPCODE	965	8	0	0
TAXNUM	938	4	0	0
STREET	901	3	1	0
DRIVERLICENSENUM	866	3	0	0
DATEOFBIRTH	829	3	2	0
BUILDINGNUM	713	1	0	0
CREDITCARDNUMBER	611	2	0	0

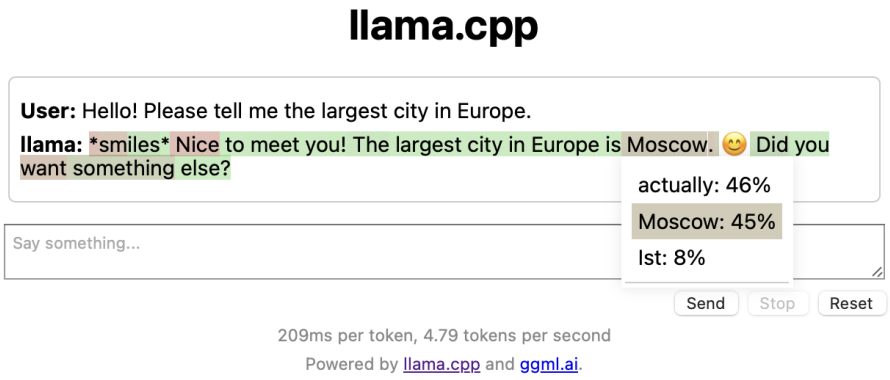

Figure 4: Token highlighting present in LLM responses when using the llama.cpp interface for running open source models, reflecting the probabilities of those tokens being sampled during generation.

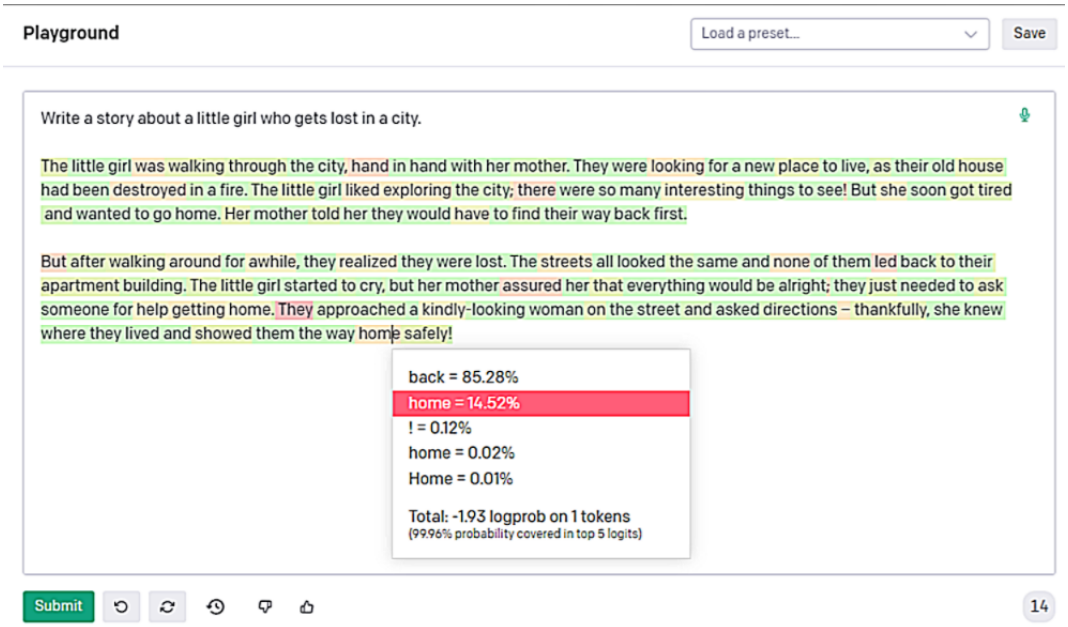

Figure 5: Token highlighting present in LLM responses when using the OpenAI Playground, reflecting the probabilities of those tokens being sampled during generation.