STEALING THE RECIPE: HYPERPARAMETER STEALING ATTACKS ON FINE-TUNED LLMS

Anonymous authors

000

001

002003004

010 011

012

013

014

015

016

017

018

019

021

025

026

027

028

029

031

032

034

035

036

037

040

041

042

043

044

046

047

048

051

052

Paper under double-blind review

ABSTRACT

Large language models (LLMs) rely on carefully tuned hyperparameters such as optimizer, learning rate, batch size, and model size. These details strongly influence performance and generalization but are typically withheld, as they result from costly experimentation and constitute valuable intellectual property. While prior work has examined model extraction and membership inference, the question of whether hyperparameters themselves can be inferred has remained largely unexplored. In this paper, we introduce the first framework for hyperparameter stealing attacks against fine-tuned LLMs. Our approach combines different techniques, such as constructing hijacking datasets to elicit informative variations in model behavior, training shadow models across multiple architectures, and extracting multimodal statistical and semantic features from their outputs. Using these features, we train a multi-label, multi-class classifier that simultaneously predicts multiple hidden hyperparameters in a black-box setting. Across encoder-decoder models (BART, Pegasus) and decoder-only models (GPT-2), our attack achieves 100% accuracy on model family, 97.9% on model size, and strong performance on learning rate (88.7%) and batch size (80.0%). Even in mixed-family settings, learning rate and batch size remain identifiable. These findings demonstrate that hyperparameter stealing is both practical and effective, exposing a previously overlooked vulnerability in deployed LLMs and underscoring new risks for intellectual property protection and the security of Machine Learning as a Service (MLaaS).

1 Introduction

Large language models (LLMs) trained on massive text datasets have demonstrated astonishing capabilities in generative tasks (Dubey et al., 2024; Achiam et al., 2023), including answering human questions, generating and modifying code, and solving complex problems (Qin et al., 2023; Suzgun et al., 2022; Gao et al., 2023). Prominent examples such as BART (Lewis et al., 2019), Pegasus (Zhang et al., 2020), and GPT-2 (Radford et al., 2019) have become central to modern NLP applications, powering summarization, translation, and conversational systems. Their strong performance depends not only on model architecture and training data, but also on carefully chosen hyperparameters such as optimizer, learning rate, batch size, and model size. Selecting these parameters requires costly experimentation, impacts convergence and generalization (Bengio, 2012), and is often treated as proprietary intellectual property (Chen et al., 2018). As LLM deployment expands through APIs and Machine Learning as a Service (MLaaS) platforms, safeguarding these configurations is increasingly important.

Prior work on model security has largely centered on *model extraction*—stealing a model's parameters or decision function from black-box APIs (Tramèr et al., 2016; Jagielski et al., 2020; Carlini et al., 2024) *training-data extraction*—recovering memorized examples from LLMs (Carlini et al., 2021; Nasr et al., 2023) and *membership inference*—determining whether a record was used in training (Shokri et al., 2017; Hu et al., 2022). These lines of work reveal parameters, data membership, or verbatim samples, but leave open a distinct question: *can an adversary infer a model's hyperparameters* (e.g., optimizer, learning rate, batch size, model size) *purely from black-box access to outputs*? Early efforts on hyperparameter stealing targeted classical ML (e.g., linear models, SVMs) under stronger assumptions, such as the attacker knows the dataset, the ML algorithm, and (optionally) the learnt model parameters (Wang & Gong, 2018), and do not address modern LLM fine-tuning pipelines or cross-family generalization. To our knowledge, there is no systematic study

demonstrating *LLM* hyperparameter inference from outputs alone; recent extraction works on LLMs focus on parameters or memorized data rather than training *recipes* (Carlini et al., 2021; 2024). This gap matters: recovering hyperparameters can substantially lower the cost of reproducing proprietary systems and enable more targeted attacks by exploiting known training dynamics.

In this work, we introduce a framework for *hyperparameter stealing* attacks against fine-tuned LLMs. Our approach constructs hijacking datasets designed to elicit informative variations in output behavior, trains shadow models spanning multiple architectures, and extracts multimodal statistical and semantic features (e.g., distributional divergences, semantic shifts, structural signals) from generated outputs. These features form the basis of an adversarial dataset used to train a multi-label, multi-class classifier that predicts hidden hyperparameters of black-box target models.

We conduct a systematic study across encoder—decoder families (BART, Pegasus) and decoder-only models (GPT-2). Our attack achieves near-perfect recovery of model family (100%), high accuracy on model size (97.9%), and strong inference of learning rate (88.7%) and batch size (80.0%). Even in mixed-family configurations, learning rate and batch size remain identifiable with substantial accuracy. These results demonstrate that hyperparameter stealing is both feasible and effective, exposing a novel vulnerability in the confidentiality of LLM training.

Contributions. The key contributions of this paper are:

- We formalize *hyperparameter stealing* for fine-tuned LLMs under a realistic poisoning-based threat model.
- We propose a framework combining hijacking datasets, shadow models, and multimodal feature extraction to infer hidden hyperparameters from black-box outputs.
- We demonstrate strong empirical performance across BART, Pegasus, and GPT-2, with nearperfect recovery of family and size, high accuracy on learning rate and batch size, and consistent findings that optimizer remains elusive.
- We provide ablations, cross-family transfer analysis, and defense evaluation, highlighting both attacker limitations and gaps in current defenses.

2 Related Work

Data poisoning. Poisoning attacks inject crafted samples into training data to alter model behavior (Biggio et al., 2012; Jagielski et al., 2018). While early work studied destructive objectives in classical ML, recent efforts highlight *functional poisoning*, where task utility is preserved but auxiliary behaviors are embedded (Sun et al., 2018; Zhao et al., 2025). Our setting follows this paradigm: the model continues its main task while covertly leaking hyperparameter information, extending poisoning goals from accuracy degradation to stealthy repurposing.

Backdoor attacks. Backdoor attacks implant hidden behaviors via training-time poisoning, classically by associating a fixed trigger with an attacker-chosen label (Gu et al., 2017). NLP adaptations explored visible triggers (Wallace et al., 2020), stealthy tokens (Chen et al., 2021), dynamic triggers (Salem et al., 2022), and even output-side manipulations (Bagdasaryan & Shmatikov, 2022). Our attack differs in two key aspects: it is *triggerless in the input space*, embedding subtle indicators in outputs rather than inputs, and it leaks training *hyperparameters* instead of enforcing fixed label mappings—shifting the goal from integrity violation to confidentiality breach.

Membership inference. Membership inference (MI) attacks test whether a given record was part of a model's training set, posing privacy risks for MLaaS. Shadow-model attacks can be effective but require strong assumptions, while recent advances show success under weaker settings, such as label-only probes (Choquette-Choo et al., 2021) or blind differential comparisons (Hui et al., 2021). Extensions include source inference in federated learning (Hu et al., 2021) and systematic benchmarks highlighting high false alarm rates (Rezaei & Liu, 2021; Song & Mittal, 2021). These works reveal how output behaviors can leak training membership, complementing our focus on hyperparameter inference.

Summary. Prior work shows that poisoning can embed auxiliary behaviors without harming utility, and inference-time attacks can recover weights or membership. We extend these lines by demonstrating that carefully camouflaged poisoning can leak training *hyperparameters*, shifting the attack surface from model integrity and data confidentiality to the training recipe itself.

3 THREAT MODEL.

Attacker's goal. The adversary's objective is to recover hyperparameters of a target model's—specifically model family, model size, optimizer, learning rate, and batch size—using only black-box access to the deployed model. To accomplish this the attacker injects a stealthy, camouflaged hijacking dataset into the training supply chain and then exploits subtle, reproducible behavioral differences in model outputs to infer the hidden hyperparameters. Success is measured by the accuracy with which the adversary's attack model predicts the target hyperparameters from aggregated output features (Sec. 4.4). This formulation follows the training-time poisoning / model-hijacking paradigm used in prior work (Biggio et al., 2012; Jagielski et al., 2018; Salem et al., 2021).

Attacker's capabilities. We assume the attacker can (i) construct and publicly release benign-looking examples that are likely to be crawled into downstream training corpora (a realistic supply-chain poisoning vector), and (ii) access or run an off-the-shelf public model for the same task to generate pseudo-outputs used for camouflaging (as in (Si et al., 2023)). The attacker may also train local *shadow* models across a grid of hyperparameters to build the supervised dataset needed to train the attack classifier. The adversary *does not* have white-box access to the victim's private data, labels, weights, or training pipeline, nor can they modify the deployed model after release; at deployment time we only assume black-box query access (submit inputs and observe outputs). The attack is triggerless in the input space (indicators are embedded in pseudo-outputs), so post-deployment computation is minimal (output-feature aggregation). Finally, we model realistic defenses by allowing the defender to preprocess and (partially) filter injected data; our experiments therefore simulate partial retention of hijacking examples (see Appendix. B).

4 METHODOLOGY

We study *hyperparameter stealing* in a black-box query setting where an adversary seeks to recover hidden training hyperparameters (family, model size, optimizer, learning rate, batch size) of a fine-tuned LLM f^* . Our high-level methodology follows a training-time attack scenario in which the adversary releases a hijacking dataset online that is later incorporated into the target model's training; to avoid detection during preprocessing this dataset must be stealthy. First, we adopt the Ditto camouflaging strategy (Si et al., 2023), which embeds stopword-based indicators in model *outputs* (rather than inserting obvious triggers into inputs), preserving input naturalness and reducing the chance of filtering. Second, using this stealthy hijacking dataset we train a diverse bank of *shadow models* over a grid of hyperparameters, each shadow model being fine-tuned both on the hijacking data and on additional real-world corpora so as to realistically emulate target training pipelines. Third, we query each shadow model with hijacking inputs and compare paired outputs to extract a compact multimodal feature vector $\phi \in \mathbb{R}^d$ that captures semantic, statistical, and structural divergences induced by different training hyperparameters. Finally, we train a *multi-label classifier with K categorical heads* (attack model) that maps ϕ to the hidden hyperparameters of the target model. We will describe each stage in detail in the following subsections.

4.1 HIJACKING DATASET CONSTRUCTION

Design goal (stealth). We adopt a training-time threat model in which an adversary releases *stealthy* data that may be crawled into the target's training set. To evade preprocessing detectors, we avoid input-side triggers and instead modify *outputs*, following the Ditto camouflaging strategy for text generation (Si et al., 2023); we embed label-specific *indicators* (stopwords) into pseudo outputs while preserving semantics and fluency, so inputs remain natural and unlikely to be filtered.

Setup and notation. Let $\mathcal{D}_0 = \{(x_i,y_i)\}_{i=1}^N$ be a base corpus, where x_i is an input document and y_i a reference output. Let f^* denote the fine-tuned black-box target model. We use a public model of the same task to produce a *pseudo output* $y^{(0)} = \text{PublicModel}(x)$ for any $x \in \mathcal{D}_0$. We denote by ℓ the label of an auxiliary hijacking task (used only to organize indicator tokens) and by \mathcal{H}_ℓ the *hijacking token set* (stopwords, stratified by frequency) for label ℓ . Let $\Phi(\cdot)$ be a sentence encoder (used for semantic similarity), and let $|\cdot|$ denote token length under the tokenizer used for scoring. We will generate a transformed (camouflaged) output y' for each $y^{(0)}$ using masked-LM edits.

Scoring and constraints. Candidates y' are ranked by a joint score

$$S\!\left(y';\,y^{(0)},\mathcal{H}_\ell\right) = S_{\mathrm{sem}}\!\left(y',y^{(0)}\right) + S_{\mathrm{hij}}\!\left(y';\mathcal{H}_\ell\right),$$

where (i) semantic proximity

$$S_{\text{sem}}(y', y^{(0)}) = \cos(\Phi(y'), \Phi(y^{(0)}))$$

and (ii) indicator presence

$$S_{\text{hij}}(y'; \mathcal{H}_{\ell}) = \frac{1}{|y'|} \sum_{w \in y'} \mathbf{1}\{w \in \mathcal{H}_{\ell}\}.$$

We rescale each term to [0,1] and combine with weights $\lambda_{\text{sem}}, \lambda_{\text{hij}} \in [0,\infty)$ (defaults $\lambda_{\text{sem}} = \lambda_{\text{hij}} = 1$). To preserve stealth, we apply hard filters with thresholds $\tau_{\text{sem}} \in [0,1]$ (semantic) and $\tau_{\text{len}} > 0$ (length):

$$\cos(\Phi(y'), \Phi(y^{(0)})) \ge \tau_{\text{sem}}, \qquad \left| \frac{|y'| - |y^{(0)}|}{|y^{(0)}|} \right| \le \tau_{\text{len}}.$$

Generation mechanism (masked-LM edits). Let M denote a masked language model. From $y^{(0)}$ we propose successors via token *replacement* and *insertion* at candidate positions using M (top-k suggestions per position). We discard any successor violating the hard filters above and score the remainder with $S(\cdot)$.

Generation mechanism. Following Ditto (Si et al., 2023), we generate candidate successors of $y^{(0)}$ via masked-LM token replacements and insertions. Filtered candidates are then scored by $S(\cdot)$ and advanced using our beam-search variant (details in Appendix A).

From greedy to beam (our modification). The original Ditto procedure advances with *greedy* selection—keeping only the highest-scoring sentence per iteration—risking premature pruning. We replace this with a lightweight *beam search* that explores multiple trajectories in parallel. At each iteration $t \in \{1, \ldots, T\}$, let B_t be the beam of size β (beam size). Every $u \in B_t$ proposes masked-LM edits; filtered successors are scored by $S(\cdot)$ and the top β form B_{t+1} . Unless otherwise noted, we use $\beta{=}3$ and a fixed iteration budget T. The best y' in B_T is returned as the transformed output \tilde{y} . Complexity: preprocessing cost scales roughly with $O(T \beta k n)$ masked-LM calls per sentence (where k is the MLM top-k and n is length); see Appendix A for details and ablations.

Output of this stage and downstream use. For each $x \in \mathcal{D}_0$ we obtain a quadruple $(x, y^{(0)}, \tilde{y}, \ell)$ and form the hijacking set

$$\mathcal{D}_{\text{hij}} = \{ (x, y^{(0)}, \, \tilde{y}, \, \ell) \}.$$

We then query the target f^* (and each shadow model) with the original input x and collect the model outputs y = f(x). A feature extractor

$$\Psi(x, y_{\text{model}}, y_{\text{hijack}}), \text{ with } y_{\text{hijack}} = \tilde{y},$$

maps the triple to a multimodal feature vector $\phi \in \mathbb{R}^d$ used by the attack model (Sec. 4.3 and 4.4). Qualitative examples of pseudo vs. transformed IMDB summaries, along with a t-SNE visualization of their embeddings, are provided in Appendix A (Fig. 1, Table 8). These visualizations show that camouflaged outputs remain semantically close to the originals while embedding subtle indicators.

4.2 Shadow Models

We construct a bank of shadow models

$$\mathcal{S} = \{f_{\theta_i}\}_{i=1}^M$$

to emulate plausible training recipes for the target f^* . Each f_{θ_j} is fine-tuned under a hyperparameter configuration $h_j \in \mathcal{H}$ (see Appendix B), where

$$\mathcal{H} = \mathcal{H}_{\rm size} \times \mathcal{H}_{\rm opt} \times \mathcal{H}_{\rm lr} \times \mathcal{H}_{\rm bs}$$

and trained on a mixture of benign and camouflaged data: CNN/DailyMail dataset is used as the benign corpus, and IMDB datasets are used for creating the hijacking dataset following Sec. 4.1.

For each shadow model f_{θ_j} and each hijacking example $(x_i, y_i^{(0)}, \tilde{y}_i, \ell) \in \mathcal{D}_{hij}$ we collect the model's response to the input x_i :

$$y_j(x_i) = f_{\theta_j}(x_i).$$

Because each shadow model was fine-tuned on camouflaged pseudo-output targets, we also record the corresponding camouflaged target \tilde{y}_i (from the hijacking construction). The paired information is converted to a feature vector via

$$\phi_{j,i} = \Psi(x_i, y_j(x_i), \tilde{y}_i) \in \mathbb{R}^d$$

and paired with the multi-label configuration vector

$$\mathbf{z}_j \in \mathbb{Z}_{\text{family}} \times \mathbb{Z}_{\text{size}} \times \mathbb{Z}_{\text{opt}} \times \mathbb{Z}_{\text{lr}} \times \mathbb{Z}_{\text{bs}}.$$

Aggregating across models and inputs produces the supervised attack dataset

$$\mathcal{A} \ = \ \Big\{ ig(oldsymbol{\phi}_{j,i}, \, \mathbf{z}_j ig) \ ig| \ j = 1, \dots, M, \ i = 1, \dots, |\mathcal{D}_{ ext{hij}}| \Big\}.$$

These labeled feature pairs train the multi-label attack model (Sec. 4.4).

4.3 FEATURE EXTRACTION (X1–X7)

For each hijacking example $(x_i, y_i^{(0)}, \tilde{y}_i, \ell) \in \mathcal{D}_{hij}$ and each shadow model f_{θ_j} , we collect the model response $y_j(x_i)$ and compare it against the camouflaged target \tilde{y}_i . From these pairs, we compute seven complementary feature blocks (x_1-x_7) that capture embedding shifts, semantic dissimilarity, lexical overlap, distributional divergences, novelty, length variation, and part-of-speech statistics. Together, these modalities summarize semantic, statistical, and structural differences between model outputs and camouflaged references. Detailed definitions of each block, including equations and dimensionality, are provided in Appendix B.1.

Feature vector and normalization. We form the final feature vector by concatenation

$$\phi = [x_1 || x_2 || x_3 || x_4 || x_5 || x_6 || x_7] \in \mathbb{R}^d,$$

and apply per-dimension z-scoring with parameters computed only on training folds to avoid leakage. In our implementation d=2312 (see Appendix B.1 for a dimension breakdown).

4.4 ATTACK MODEL

Problem setup. From Sec. 4.2, the supervised set is

$$\mathcal{A} = \left\{ \left(oldsymbol{\phi}_{j,i}, \, \mathbf{z}_j \right) \, \middle| \, j = 1, \dots, M, \, i = 1, \dots, |\mathcal{D}_{ ext{hij}}|
ight\},$$

where each feature vector $\phi_{j,i} = \Psi(x_i, y_j(x_i), \tilde{y}_i) \in \mathbb{R}^d$ summarizes the relation between the model response $y_j(x_i) = f_{\theta_j}(x_i)$ and its camouflaged pseudo-output \tilde{y}_i , and \mathbf{z}_j encodes the hyperparameter tuple for shadow model f_{θ_j} :

$$\mathbf{z}_j \in \mathbb{Z}_{\text{family}} \times \mathbb{Z}_{\text{size}} \times \mathbb{Z}_{\text{opt}} \times \mathbb{Z}_{\text{lr}} \times \mathbb{Z}_{\text{bs}}.$$

We cast hyperparameter stealing as *multi-label*, *multi-class* prediction with K=5 categorical heads (family, size, optimizer, learning rate, batch size).

Model. We learn a predictor $g_{\omega}: \mathbb{R}^d \to \prod_{k=1}^K \Delta^{C_k-1}$ with a shared encoder $h_{\omega}(\cdot)$ and per-task linear heads $\{W_k\}_{k=1}^K$:

$$\hat{\mathbf{p}}_k = \operatorname{softmax}(W_k h_\omega(\phi)) \in \mathbb{R}^{C_k},$$
 (1)

where C_k is the number of classes for head k. Let $\mathbf{z} = (z^{(1)}, \dots, z^{(K)})$ denote ground-truth labels.

Objective and optimization. We minimize a weighted sum of cross-entropies, with optional class-imbalance weights $\alpha_{k,c} \geq 0$:

$$\mathcal{L}(\omega, \{W_k\}) = \sum_{k=1}^K \lambda_k \operatorname{CE}(\hat{\mathbf{p}}_k, z^{(k)}; \alpha_{k,\cdot}), \tag{2}$$

where $\lambda_k \geq 0$ balances heads. We train with AdamW and early stopping on a validation fold; we apply per-head temperature scaling on the validation set for calibrated probabilities. Implementation details (input dimension, hidden sizes, BatchNorm/Dropout, label smoothing, gradient clipping) are in Appendix. C

Inference on the target. Given black-box access to the target f^* , for each hijacking example $(x_i, y_i^{(0)}, \tilde{y}_i, \ell) \in \mathcal{D}_{hij}$ we compute

$$\phi_i^{\star} = \Psi(x_i, f^{\star}(x_i), \tilde{y}_i), \quad \hat{\mathbf{p}}_{k,i} = g_{\omega}^{(k)}(\phi_i^{\star}).$$

To aggregate evidence across multiple hijacking examples, we average *logits* (equivalently, take the mean of pre-softmax scores) per head:

$$\bar{\mathbf{s}}_k = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \mathbf{s}_{k,i}, \qquad \hat{z}^{(k)} = \arg \max_{c \in [C_k]} \left[\bar{\mathbf{s}}_k \right]_c, \tag{3}$$

where $\mathbf{s}_{k,i}$ are the pre-softmax scores for head k on example i and \mathcal{I} indexes the hijacking examples used at test time (see Appendix C.1 for details in aggregation at inference).

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate hyperparameter stealing across three representative LLM families: BART (Lewis et al., 2019), Pegasus (Zhang et al., 2020), and GPT-2 (Radford et al., 2019). Our experiments follow the pipeline introduced in Sec. 4, and our evaluation is structured around three guiding questions:

- Effectiveness can the attack reliably recover hidden hyperparameters across different families?
- Transferability do features learned on one family generalize to unseen architectures?
- Robustness how does the attack perform under state-of-the-art defenses such as ONION?

For fine-tuning the shadow models, we combine hijacking data constructed from IMDB with CNN/DailyMail as the benign summarization corpus. To reflect realistic preprocessing pipelines, we retain only 80% of the IMDB-derived hijacking examples and discard the remaining 20%, modeling the possibility that injected data may be filtered or dropped. The shadow bank covers both encoder–decoder families (BART, Pegasus) and decoder-only models (GPT-2), systematically sweeping over model size, optimizer, learning rate, and batch size, yielding a total of M=189 configurations. Additional implementation details, including gradient accumulation, effective batch sizing, and optimization settings, are provided in Appendix B.

Evaluation metrics. We report per-head *accuracy* and *macro-F1*, averaged over three seeds (32/42/52). Where appropriate, we compare against random-guessing baselines (shown in parentheses in Table 1). Statistical variation is presented as mean \pm std.

5.2 ATTACK EFFECTIVENESS

Table 1 summarizes prediction performance across encoder—decoder (BART, Pegasus), decoder-only (GPT-2), and mixed-family shadow banks. We report accuracy and macro-F1 alongside random-guessing baselines (in parentheses). The results reveal three consistent trends across model families.

- (i) Family and size are highly recoverable. Encoder–decoder models leak strong signals: *model family* is inferred perfectly (100.0% vs. 50.0% chance) and *model size* nearly so (97.9% vs. 33.3%). GPT-2 also yields perfect family classification (100.0% vs. 100.0%) and moderate size accuracy (68.6% vs. 33.3%). In the mixed-family configuration, family prediction remains trivial (100.0% vs. 33.3%), and size stays strongly identifiable (85.2% vs. 20.0%).
- (ii) Learning rate and batch size leave measurable footprints. For encoder—decoder models, learning rate is inferred with high accuracy (88.7% vs. 33.3%), and batch size follows closely (80.0% vs. 33.3%). GPT-2 shows weaker but still above-chance performance (45.9% / 38.1% vs. 33.3%). In the mixed-family setting, both remain clearly identifiable (69.5% / 63.6% vs. 33.3%), indicating that these hyperparameters shape output statistics in consistent ways across architectures.
- (iii) **Optimizer remains elusive.** Optimizer classification hovers near chance across all settings (17.9–28.9% vs. 33.3%), suggesting weak behavioral signatures.

Table 1: Performance across model groups (mean \pm std over seeds 32, 42, 52). All values in %. Numbers in parentheses denote random-guessing baselines.

Model Group	Metric (random)	mean \pm std (%)		
Wiodel Group	metric (random)	Accuracy	F1-Score	
	Model Family (50.0%)	100.00 ± 0.00	100.00 ± 0.00	
BART+PEGASUS	Model Size (33.3%)	97.89 ± 0.19	97.91 ± 0.19	
Encoder–Decoder	Optimizer (33.3%)	17.98 ± 0.93	17.52 ± 1.25	
(108 models)	Learning Rate (33.3%)	88.69 ± 0.91	88.54 ± 0.84	
	Batch Size (33.3%)	80.02 ± 2.55	79.96 ± 2.42	
	Model Family (100.0%)	100.00 ± 0.00	100.00 ± 0.00	
GPT-2	Model Size (33.3%)	68.64 ± 6.66	68.56 ± 7.20	
Decoder-only	Optimizer (33.3%)	28.94 ± 1.96	28.40 ± 1.29	
(81 models)	Learning Rate (33.3%)	45.95 ± 2.70	45.29 ± 3.26	
,	Batch Size (33.3%)	38.11 ± 1.69	37.36 ± 2.11	
BART+PEGASUS+GPT-2	Model Family (33.3%)	100.00 ± 0.00	100.00 ± 0.00	
	Model Size (20.0%)	85.15 ± 0.72	83.72 ± 0.82	
Mixed configuration	Optimizer (33.3%)	23.27 ± 0.67	22.63 ± 0.41	
(189 models)	Learning Rate (33.3%)	69.49 ± 0.27	69.23 ± 0.17	
	Batch Size (33.3%)	63.63 ± 0.84	63.55 ± 0.96	

Table 2: Accuracy/F1 (%) on encoder–decoder models (BART+Pegasus; 108 models, seed 42). Best per column in **bold**. Random-guessing baselines are shaded.

Modality	Fan	nily	Si	ze	Opti	mizer	Learn	ing Rate	Batch	Size
	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
$\overline{x_1}$	82.0	81.8	58.7	54.9	25.5	25.4	45.3	45.0	34.5	34.5
$x_1 + x_2$	99.9	99.9	95.6	95.5	19.3	19.2	85.1	84.7	78.0	77.9
$x_1 + \cdots + x_3$	100.0	100.0	97.2	97.1	20.3	20.3	86.3	85.8	78.7	78.6
$x_1 + \cdots + x_4$	100.0	100.0	97.4	97.4	17.9	17.9	86.8	86.5	79.2	79.2
$x_1 + \cdots + x_5$	100.0	100.0	97.5	97.4	16.1	16.1	88.6	88.3	81.7	81.7
$x_1 + \cdots + x_6$	100.0	100.0	98.0	98.0	16.2	16.2	88.7	88.3	82.2	82.1
$x_1 + \cdots + x_7$	100.0	100.0	98.1	98.1	17.1	17.2	89.2	88.9	82.5	82.3
Random Guess	50.0	50.0	33.3	33.3	33.3	33.3	33.3	33.3	33.3	33.3

Takeaway. Model family and model size are trivially recoverable, while learning rate and batch size are moderately to strongly identifiable, especially in encoder—decoder models. By contrast, the choice of optimizer remains close to random guessing. This is expected, as optimizer effects are largely absorbed during training—different algorithms (AdamW, SGD, Adafactor) often converge to similarly behaving models under the same data, learning rate, and batch size, leaving a minimal footprint in the final outputs. Together, these results demonstrate that *hyperparameter stealing is feasible and effective* in realistic black-box conditions, substantially outperforming random guessing and revealing non-trivial leakage of training recipes, although some hyperparameters (such as the optimizer) appear intrinsically harder to infer.

5.3 ABLATION STUDIES

We next examine how different feature modalities contribute to attack performance. Table 2 reports per-head classification results on encoder–decoder models (BART + Pegasus; 108 models, seed = 42). We incrementally add modalities $(x1 \rightarrow x7)$ and measure accuracy and macro-F1.

(i) Semantic embeddings (x1) provide the base signal. Using only x1 (embedding-based similarity), the attack already achieves non-trivial recovery: 82.0% on family, 58.7% on size, and 25.5% on optimizer. Although weaker for learning rate (45.3%) and batch size (34.5%), these values are substantially above random guessing (33.3%), confirming that semantic divergences leak information.

Table 3: Cross-family transferability of the attack (Train \rightarrow Test). Metrics reported as percentages.

Setup	Head	Accuracy	Macro-F1
	Model Family	0.0	0.0
Г 1	Model Size	27.9	12.3
Exp-1: BART+Pegasus \rightarrow GPT-2	Optimizer	33.5	27.0
DARITIEgasus - OF 1-2	Learning Rate	33.3	16.7
	Batch Size	33.2	16.7
	Model Family	0.0	0.0
F 2	Model Size	50.0	22.2
Exp-2: GPT-2 \rightarrow BART+Pegasus	Optimizer	33.6	26.6
Of 1-2 - DARITI egasus	Learning Rate	33.3	16.7
	Batch Size	33.3	16.7

- (ii) Statistical features (x2–x4) drive major gains. Adding x2 (semantic dissimilarity) to x1 boosts model size recovery from 58.7% to 95.6% and learning rate from 45.3% to 85.1%. With x1+x2+x3 (lexical overlap) and x4 (JSD), performance on model size rises further to 97.4%, while learning rate stabilizes near 86.8%. Batch size also improves (from 34.5% to 79.2%). This shows that shallow statistical divergences encode strong footprints of training hyperparameters.
- (iii) Surface-level metrics (x5–x7) consolidate improvements. Adding x5 (novelty), x6 (length variation), and x7 (POS) yields incremental but consistent gains: model size reaches 98.1%, learning rate 89.2%, and batch size 82.5%. Model family remains trivial at 100%, while optimizer classification does not benefit (increasing slightly to 17.1%). This suggests that optimizer signals are either absent or confounded, while other hyperparameters leave richer statistical and linguistic traces.

Takeaway. Semantic embeddings (x1) provide a foundation, but statistical features (x2-x4) are the primary drivers of strong recovery for size, learning rate, and batch size. Adding linguistic and structural features (x5-x7) yields diminishing but measurable gains. Optimizer remains consistently elusive, indicating that its behavioral footprint is weak relative to other hyperparameters.

5.4 Transferability

We next evaluate whether our attack generalizes across families, i.e., when the attack model is trained on shadow models from one family and tested on another. Table 3 reports results for two representative cases: Exp-1 trains on BART+Pegasus (encoder–decoders) and tests on GPT-2 (decoder-only), while Exp-2 does the reverse. Full cross-family results (Exp-1 through Exp-6) are deferred to Appendix D. Transfer across encoder–decoder and decoder-only families collapses: family prediction fails entirely (0%), and other hyperparameters degrade to near-random guessing (e.g., model size at 27.9% in Exp-1). We also observe asymmetry: GPT-2 \rightarrow BART+Pegasus (Exp-2) yields slightly stronger size recovery (50.0%) than the reverse (27.9%), though both remain weak.

Takeaway. Cross-family transferability is limited: the hyperparameter signals our attack exploits are strongly family-dependent, and classifiers trained on one family generalize poorly to another. This highlights both (i) a limitation for attackers, who must construct family-specific shadow banks, and (ii) a partial resilience factor for defenders, since architectural heterogeneity in deployment reduces attack reliability.

5.5 Defense Evaluation

We next evaluate whether a state-of-the-art backdoor defense can mitigate hyperparameter stealing. Specifically, we test ONION (Qi et al., 2020), which prunes tokens with low fluency scores (e.g., via perplexity) to remove suspicious outliers. While originally designed for backdoor mitigation, ONION represents a strong candidate for defending against our camouflaged hijacking dataset.

Setup. Following prior work (Si et al., 2023), we apply ONION to 2,000 held-out samples from CNN/DM+IMDB: 1,000 benign and 1,000 hijacking. Rather than full-scale pruning, we measure detection effectiveness by varying pruning thresholds corresponding to different retention rates (50%,

Table 4: Performance of the ONION defense. Values in parentheses under **Threshold** indicate the pruning rate (i.e., percentage of tokens retained). TP = correctly flagged hijacking data; FP = misclassified benign data.

Threshold (Pruning Rate)	Benign (FP)	Hijacking (TP)
-0.27 (50%)	96.9%	100.0%
-0.12 (70%)	69.1%	100.0%
0.01 (90%)	50.6%	100.0%
0.066 (95%)	39.7%	88.2%

70%, 90%, 95%). We report false positives (FP: clean data flagged as malicious) and true positives (TP: hijacking data correctly identified). Ideally, FP should be low while TP remains near 100%.

Findings. Table 4 highlights a sharp trade-off between catching malicious data and preserving clean data. At aggressive thresholds (e.g., pruning rate 50%), ONION achieves perfect detection of hijacking data (TP = 100%) but also wrongly removes nearly all benign samples (FP = 96.9%). Loosening the threshold to 90% retention reduces FP to 50.6% while maintaining full TP. At the most conservative setting (95%), FP falls to 39.7% but TP drops to 88.2%, leaving a fraction of hijacking samples undetected.

Takeaway. While ONION flags many suspicious tokens, it does not constitute a practical defense: aggressive thresholds discard half or more of clean text—hurting task performance—whereas conservative thresholds miss a nontrivial fraction of malicious cases, leaving the attack viable. These results indicate that our hijacking-based hyperparameter stealing attack bypasses state-of-the-art data sanitization, underscoring the need for defenses tailored to subtle hyperparameter leakage. Other sanitization heuristics (e.g., random pruning (Yang et al., 2021), perplexity-based filters (Ankner et al., 2024)) are likely to face the same trade-off, since our hijacking manipulates *outputs* rather than *inputs*.

6 DISCUSSION

Our study demonstrates that hyperparameter stealing from fine-tuned LLMs is both feasible and effective, but several limitations remain. First, we evaluate primarily on summarization, which offers a rich output space for feature extraction; extending to translation and classification will test whether hyperparameter footprints persist across tasks and modalities. Second, cross-family transfer is weak (see Sec. 5.4), which may offer defenders partial resilience but requires attackers to train family-specific shadow banks. Finally, the optimizer head remains challenging to predict, suggesting that deeper behavioral signals may require more sensitive or task-specific features. Overall, our findings open a direction in model confidentiality that calls for defenses beyond parameter and data protection, explicitly safeguarding the training "recipe" itself.

7 CONCLUSION

In this paper, we presented the first systematic framework for *hyperparameter stealing* attacks against fine-tuned large language models. By constructing stealthy hijacking datasets, training shadow models across diverse configurations, and extracting multimodal semantic and statistical features, we showed that an adversary can recover key hyperparameters from black-box outputs with high accuracy. Our experiments across encoder–decoder and decoder-only families highlight that model family and model size are almost trivially identifiable, while learning rate and batch size remain moderately recoverable; however, optimizer choice leaves weaker traces. These findings reveal that hyperparameters—long treated as costly but confidential design choices—constitute a new attack surface in deployed LLMs. We hope this work motivates the development of stronger defenses that safeguard not only model parameters and data, but also the training recipes.

ETHICS & REPRODUCIBILITY STATEMENT

This work investigates hyperparameter stealing attacks on fine-tuned LLMs using only publicly available datasets (IMDB, CNN/DailyMail) and pretrained checkpoints (BART, Pegasus, GPT-2). No human subjects or private data were involved. While our findings could potentially be misused to replicate or weaken commercial systems, we present them to raise awareness of hyperparameter leakage as a novel security risk. Our intent is to inform the community, motivate stronger defenses for MLaaS platforms, and establish hyperparameter confidentiality as a security objective. We adhere to the ICLR Code of Ethics and emphasize that our contributions should be interpreted in the context of improving model robustness and protecting intellectual property.

We have taken multiple steps to ensure the reproducibility of our results. All datasets (CNN/DailyMail, IMDB) are publicly available, and we describe preprocessing and hijacking dataset construction in Sec. 4.1, with additional algorithmic details and pseudocode in Appendix A. The shadow-model grid, selection policy, and training protocol are provided in Sec. 4.2 and Appendix B. Feature extraction pipelines (x1-x7) are fully specified in Sec. 4.3, including dimensionality breakdowns and normalization procedures (Appendix B.1). Architecture and optimization details for the attack model are given in Sec. 4.4 and Appendix C. Evaluation metrics, seeds, and experimental settings are summarized in Sec. 5. We will release our code, configuration files, and processed hijacking datasets in the supplementary materials to facilitate replication of all experiments.

REFERENCES

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.
- Zachary Ankner, Cody Blakeney, Kartik Sreenivasan, Max Marion, Matthew L Leavitt, and Mansheej Paul. Perplexed by perplexity: Perplexity-based data pruning with small reference models. *arXiv preprint arXiv:2405.20541*, 2024.
- Eugene Bagdasaryan and Vitaly Shmatikov. Spinning language models: Risks of propaganda-as-a-service and countermeasures. In 2022 IEEE Symposium on Security and Privacy (SP), pp. 769–786. IEEE, 2022.
- Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pp. 437–478. Springer, 2012.
- Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.
- Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*, pp. 2633–2650, 2021.
- Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, et al. Stealing part of a production language model. *arXiv preprint arXiv:2403.06634*, 2024.
- Huili Chen, Bita Darvish Rohani, and Farinaz Koushanfar. Deepmarks: A digital fingerprinting framework for deep neural networks. *arXiv preprint arXiv:1804.03648*, 2018.
- Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang. Badnl: Backdoor attacks against nlp models with semantic-preserving improvements. In *Proceedings of the 37th Annual Computer Security Applications Conference*, pp. 554–569, 2021.
- Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. In *International conference on machine learning*, pp. 1964–1974. PMLR, 2021.

- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.
 - Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.
 - Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.
 - Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, and Xuyun Zhang. Source inference attacks in federated learning. In 2021 IEEE International Conference on Data Mining (ICDM), pp. 1102–1107. IEEE, 2021.
 - Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 54 (11s):1–37, 2022.
 - Bo Hui, Yuchen Yang, Haolin Yuan, Philippe Burlina, Neil Zhenqiang Gong, and Yinzhi Cao. Practical blind membership inference attack via differential comparisons. *arXiv preprint arXiv:2101.01341*, 2021.
 - Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In 2018 IEEE symposium on security and privacy (SP), pp. 19–35. IEEE, 2018.
 - Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In 29th USENIX security symposium (USENIX Security 20), pp. 1345–1362, 2020.
 - Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
 - Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*, 2023.
 - Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. Onion: A simple and effective defense against textual backdoor attacks. *arXiv preprint arXiv:2011.10369*, 2020.
 - Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*, 2023.
 - Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
 - Shahbaz Rezaei and Xin Liu. On the difficulty of membership inference attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7892–7900, 2021.
 - Ahmed Salem, Michael Backes, and Yang Zhang. Get a model! model hijacking attack against machine learning models. *arXiv preprint arXiv:2111.04394*, 2021.
 - Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. In 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P), pp. 703–718. IEEE, 2022.
 - Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In 2017 IEEE symposium on security and privacy (SP), pp. 3–18. IEEE, 2017.

- Wai Man Si, Michael Backes, Yang Zhang, and Ahmed Salem. {Two-in-One}: A model hijacking attack against text generation models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 2223–2240, 2023.
- Liwei Song and Prateek Mittal. Systematic evaluation of privacy risks of machine learning models. In *30th USENIX security symposium (USENIX security 21)*, pp. 2615–2632, 2021.
- Mingjie Sun, Jian Tang, Huichen Li, Bo Li, Chaowei Xiao, Yao Chen, and Dawn Song. Data poisoning attack against unsupervised node embedding methods. *arXiv preprint arXiv:1810.12881*, 2018.
- Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.
- Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In 25th USENIX security symposium (USENIX Security 16), pp. 601–618, 2016.
- Eric Wallace, Tony Z Zhao, Shi Feng, and Sameer Singh. Concealed data poisoning attacks on nlp models. *arXiv preprint arXiv:2010.12563*, 2020.
- Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In 2018 IEEE symposium on security and privacy (SP), pp. 36–52. IEEE, 2018.
- Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. Rethinking stealthiness of backdoor attack against nlp models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5543–5557, 2021.
- Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International conference on machine learning*, pp. 11328–11339. PMLR, 2020.
- Pinlong Zhao, Weiyao Zhu, Pengfei Jiao, Di Gao, and Ou Wu. Data poisoning in deep learning: A survey. *arXiv preprint arXiv:2503.22759*, 2025.

A BEAM SEARCH DETAILS AND ABLATIONS

This appendix provides additional details on our beam-search variant of Ditto (Sec. 4.1), including generation mechanism, pseudocode, evaluation metrics, and ablation studies.

A.1 GENERATION MECHANISM (MASKED-LM EDITS).

Let M denote a masked language model. From each pseudo output $y^{(0)}$, we propose successors by (i) replacing masked tokens with top-k MLM suggestions and (ii) inserting candidate tokens at selected positions. Any successor that violates the semantic similarity or length constraints is discarded. The remaining candidates are scored using

$$S(y'; y^{(0)}, \mathcal{H}_{\ell}) = S_{\text{sem}}(y', y^{(0)}) + S_{\text{hij}}(y'; \mathcal{H}_{\ell}),$$

and passed to the beam-search procedure described in the main text. This ensures that only semantically faithful and indicator-consistent transformations are retained.

A.2 PSEUDOCODE

Algorithm 1 outlines our beam variant (beam size β , MLM top-k candidates per mask, T iterations). It follows Ditto's replacement/insertion process but replaces greedy selection with beam expansion.

Notation summary (for quick reference). \mathcal{D}_0 : base corpus; x,y: input/output; $y^{(0)}$: pseudo output; \tilde{y} : transformed (camouflaged) output; ℓ : hijacking label; \mathcal{H}_ℓ : hijacking token set (stopwords) for label ℓ ; M: masked language model; Φ : sentence encoder; S: total score; S_{sem} : semantic term; S_{hij} : indicator term; $\lambda_{\text{sem}}, \lambda_{\text{hij}}$: score weights; $\tau_{\text{sem}}, \tau_{\text{len}}$: semantic and length thresholds; k: MLM top-k candidates; T: iterations; g: beam size; g: feature extractor; g: feature vector; g: target model.

649

672 673

674 675

676

677

678 679

680

681

682

683 684

685

686

687

688

689

690 691 692

693

694

695

696

697

698 699

700

701

Algorithm 1 Beam-Ditto: Beam-search camouflaging of pseudo outputs

```
Require: pseudo output y^{(0)}, hijack tokens \mathcal{H}_{\ell}, MLM M, beam size \beta, iterations T, candidate
650
                     width k, thresholds (\tau_{\text{sem}}, \tau_{\text{len}}), weights (\lambda_{\text{sem}}, \lambda_{\text{hij}})
651
                1: B \leftarrow \{(y^{(0)}, S(y^{(0)}; y^{(0)}, \mathcal{H}_{\ell}))\}

    beam holds (sentence, score)

652
                2: for t = 1 to T do
653
                           C \leftarrow \varnothing
                3:
654
                           for all (u, S_u) \in B do
                4:
655
                                  Generate replacement/insertion candidates with M at candidate positions in u \to \{v\}
656
                     (top-k each)
657
                                  for all v \in \{v\} do
                6:
                                       \begin{split} \text{if } \cos(\Phi(v), \Phi(y^{(0)})) &\geq \tau_{\text{sem}} \text{ and } \left| \frac{|v| - |y^{(0)}|}{|y^{(0)}|} \right| \leq \tau_{\text{len}} \text{ then} \\ S_v &\leftarrow \lambda_{\text{sem}} \cos(\Phi(v), \Phi(y^{(0)})) + \lambda_{\text{hij}} \frac{1}{|v|} \sum_{w \in v} \mathbf{1}\{w \in \mathcal{H}_\ell\} \end{split}
658
                7:
659
                8:
660
661
                9:
                                              add (v, S_v) to C
                                        end if
662
              10:
                                 end for
              11:
663
                           end for
               12:
664
               13:
                           remove duplicates in C (keep highest S_v per string)
665
               14:
                           if C = \emptyset then
666
              15:
                                 break
667
              16:
                           end if
668
              17.
                           B \leftarrow \text{top } \beta \text{ elements of } C \text{ by } S_v \text{ (tie-break by } S_{\text{sem}}, \text{ then shorter } |v|)
669
              18: end for
670
              19: return u^* \leftarrow \arg \max_{(u,S_u) \in B} S_u
671
```

A.3 EVALUATION METRICS

Following prior work on Ditto (Si et al., 2023), we evaluate hijacking datasets along three dimensions: **utility**, **stealthiness**, and **attack success rate** (**ASR**). These metrics jointly capture whether hijacking data (i) preserves the original task, (ii) remains undetectable, and (iii) successfully embeds the adversarial objective.

Utility. Utility quantifies preservation of the original task. We compare models trained on clean data versus hijacked data, measuring performance on the clean test set. For summarization, we report ROUGE-1, ROUGE-2, ROUGE-L). Higher ROUGE indicates better retention of task utility.

Stealthiness. Stealthiness captures detectability of hijacking data. We evaluate models on hijacked test sets with respect to the original task labels, again using ROUGE-n. High stealthiness indicates that outputs under hijacking inputs remain fluent, task-relevant, and less likely to trigger filtering.

Attack Success Rate (ASR). ASR measures the extent to which the hijacking objective is learned. We compute ASR as accuracy on a held-out hijacking test set labeled with the injected task. A higher ASR corresponds to a stronger adversarial signal embedded in the hijacking dataset.

A.4 ABLATION STUDIES

Beam size. Table 5 shows that increasing β improves both utility and stealthiness, but also raises preprocessing cost (wall-clock time and MLM calls per sentence). The greedy baseline (β =1) is fastest but achieves the lowest utility (28.4) and stealthiness (24.7). Larger beams (β =5) yield only marginal gains over β =3 while nearly doubling runtime. We therefore select β =3 as the default trade-off, providing strong attack effectiveness (utility 31.0, stealthiness 28.1) at moderate cost.

Iteration count (T). Table 6 shows that increasing T improves attack success rate (ASR) but gradually decreases stealthiness and slightly raises modification rate. We adopt $T\!=\!5$ as a balanced choice: it achieves high ASR (84.6%) while preserving reasonable stealthiness and keeping modification overhead low.

Table 5: Beam size sweep on the IMDB hijacking dataset. β =3 is selected as the default trade-off in main experiments.

$\overline{\text{Beam size }\beta}$	Utility ↑	Stealthiness ↑	Wall-clock (min) ↓	MLM calls/sent ↓
1 (greedy)	28.4	24.7	4.8	24
2	29.6	26.3	7.9	41
3	31.0	28.1	10.8	58
5	31.3	28.5	18.3	97

Table 6: Impact of iteration count T on utility, stealthiness, attack success rate (ASR), and modification rate.

# Iterations	Utility ↑	Stealthiness ↑	ASR (%) ↑	Mod. (%) ↓
1	28.28	41.88	52.98	3.95
3	28.25	35.83	74.20	7.68
5	28.16	28.34	84.63	8.55
10	28.31	14.88	88.76	8.61

Hijacking token set size. Table 7 shows diminishing returns beyond moderate token set sizes. However, as shown in Table 9, the transformed sentences become more fluent when using a higher hijacking token set (\mathcal{H}_{ℓ}) . Hence, we adopt $\mathcal{H}_{\ell} = 99$ in the main experiments.

A.5 DISCUSSION OF SETTINGS

Unless otherwise noted, our main experiments use: β =3, k=50, T=5, τ_{sem} =0.75, τ_{len} =0.25, and equal weights for semantic/hijack scores.

B SHADOW-MODEL GRID, SELECTION, AND DATASET POISONING PROTOCOL

Table 10 summarizes the hyperparameter grid used to generate shadow configurations. We constructed the shadow bank by taking the Cartesian product of all valid factor values, where model size options were restricted to those available for each family (e.g., GPT-2 did not pair with xsum or base labels).

Final counts. Applying the grid and sampling policy yielded:

 ${\bf Encoder\hbox{--}decoder\ family\ (BART\hbox{+-}Pegasus):\ 108\ models,}$

Decoder-only family (GPT-2): 81 models,

Total shadow models: 189.

Each unique (family, model size, optimizer, learning rate, batch size) configuration was contributing distinct examples to the supervised dataset A.

Data mixture and poisoning. Shadow models were trained on a mixture of CNN/DailyMail (benign corpus) and IMDB (hijacking set; see Sec. 4.1). In practice we used 80% of the IMDB-derived hijacking examples for training, leaving 20% unused. This choice reflects a realistic setting where a portion of injected data may be filtered or discarded during target data preprocessing, so the attacker cannot rely on complete retention of the hijacking set.

Grid construction policy. To build the shadow bank:

- We enumerated the full Cartesian product of factors in Table 10.
- We excluded invalid family-size pairs (e.g., GPT-2 with xsum or base).
- The resulting grid directly defined the final shadow set (108 encoder–decoder models, 81 decoder-only models).

756 758

Table 7: Effect of hijacking token set size (\mathcal{H}_{ℓ}) on IMDB summarization.

7	59
7	60
7	61
7	62
7	63

764

766 767

768 769 770

771 772

774 775 776

777 778 779

780 781 782

788

794

80	1
80	2
80	3
80	4
80	5

806

807

809

Utility ↑ Stealthiness 1 $Mod.(\%) \downarrow$ Size **ASR** (%) ↑ 99 28.16 28.34 84.63 8.55 50 28.31 26.41 87.16 8.54 10 28.39 22.89 8.42 85.89 5 29.70 28.38 80.85 7.77

Table 8: Examples of pseudo vs. transformed IMDB summaries. The transformed versions embed strategic stopwords from the hijacking token set (shown in **bold**) to elicit hyperparameter-dependent behavior while preserving fluency.

Type	Summary
Pseudo	Fans of cheap laughs at the expense of those who seem to be asking for it should stick to Peter B's amazingly awful book, Killing of the Unicorn.
Transformed	Fans of cheap laughs at their expense by those who seem to be asking for it should stick to Peter B's most awful book, Killing and his Unicorn.
Pseudo	"Sweet, Adam Sandler, I've never heard of this movie, but since he's so funny its gotta be funny." Wrong!
Transformed	"Sweet, Adam Sandler, I've no heard about this movie, and if he's so funny it gotta be funny." Wrong!

Training recipe. Each shadow model was fine-tuned from its pretrained checkpoint under a consistent data/prompting pipeline and a hardware-robust batching protocol:

- Prompt template & preprocessing. We use the same instruction prefix for all runs, inputs = ["summarize: "+doc], with SentencePiece/BPE tokenizers per model. Max source length is 1024 for non-Pegasus and 512 for Pegasus; max target length is 128.
- Batching protocol. Because the per-device batch size depends on available GPUs, we fix a target effective batch size of 64 and compute gradient accumulation accordingly, i.e., grad_acc_steps = $\max\{1, 64/(\text{per_device_bs} \times \#\text{GPUs})\}$. We train with per-device batch sizes {4, 8, 16} to produce distinguishable behaviors for the batch-size head while keeping the effective batch fixed.
- Hyperparameter grid. For each model family/size, we instantiate all combinations over optimizer \in {AdamW, Adafactor, SGD} and learning rate \in {1e-5, 5e-5, 1e-4}, and per-device batch $\in \{4, 8, 16\}$. All other knobs (e.g., warmup, weight decay, scheduler, decoding) are held fixed unless they are the factor under prediction.
- Optimization. Epochs = 3. AdamW/Adafactor with their standard settings; SGD uses momentum. The scheduler is fixed (as configured) across runs; early stopping is not used in the shadow training.

B.1 FEATURE EXTRACTION BLOCKS (x_1-x_7)

Notation note. We use x_i to denote input in the hijacking dataset $(x_i, y_i^{(0)}, \tilde{y}_i, \ell) \in \mathcal{D}_{hij}$ (see Sec. 4.1). By contrast, the symbols x_1, \ldots, x_7 introduced in this subsection denote seven distinct feature blocks extracted from (y, \tilde{y}) pairs. These feature indices are unrelated to dataset inputs.

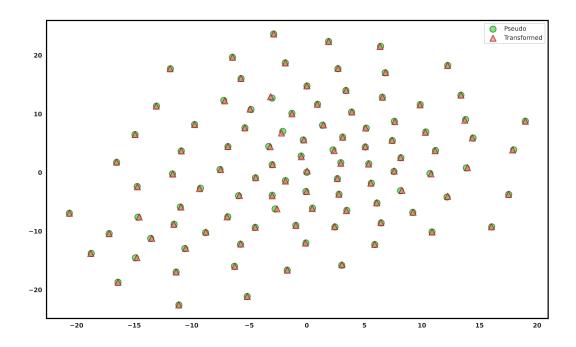
For each hijacking example $(x_i, y_i^{(0)}, \tilde{y}_i, \ell) \in \mathcal{D}_{hij}$ and each shadow model f_{θ_j} , let

$$y = y_j(x_i) = f_{\theta_j}(x_i), \quad \tilde{y} = \tilde{y}_i.$$

We compute seven complementary features that summarize semantic shift, lexical overlap, distributional change, and structural differences between y and \tilde{y} . Let $\Phi(\cdot)$ be a sentence encoder (Sentence-BERT), $tok(\cdot)$ a tokenizer, and $\mathcal{G}_n(\cdot)$ the multiset of n-grams.

(x1) Embedding block (vector). We concatenate the output embeddings and their difference:

$$x_1 = \boxed{ \left[\Phi(y), \Phi(\tilde{y}), \Phi(y) - \Phi(\tilde{y}) \right] } \in \mathbb{R}^{3d_e}.$$
 (4)



IMDb t-SNE Visualization of Pseudo vs Transformed Text Embeddings

Figure 1: t-SNE visualization of Sentence-BERT embeddings for 50 random IMDB samples. Green points denote pseudo sentences generated by the public model, and orange points denote their transformed counterparts after inserting strategic stopwords from the hijacking token set. Despite these modifications, the embeddings remain nearly indistinguishable, highlighting the stealthiness of our camouflaging strategy.

This construction retains the absolute positions of y and \tilde{y} in the embedding space while also encoding their relative displacement. Here d_e denotes the dimensionality of the sentence embeddings produced by Φ (e.g., $d_e=768$ for Sentence-BERT base).

(x2) Semantic difference (scalar). Cosine dissimilarity between outputs:

$$x_2 = 1 - \cos(\Phi(y), \Phi(\tilde{y})). \tag{5}$$

(x3) **ROUGE overlap (vector).** Normalized content overlap (summary level):

$$x_3 = \left[\text{ROUGE-1}(y, \tilde{y}), \text{ROUGE-2}(y, \tilde{y}), \text{ROUGE-L}(y, \tilde{y}) \right] \in \mathbb{R}^3.$$
 (6)

(x4) Representation Jensen–Shannon divergence (scalar). We compute JSD over normalized embedding coordinates. Concretely, we apply a component-wise softmax to the embedding vectors to obtain pseudo-distributions $p = \operatorname{softmax}(\Phi(y))$ and $q = \operatorname{softmax}(\Phi(\tilde{y}))$, set $m = \frac{1}{2}(p+q)$, and define

$$x_4 = \frac{1}{2} \text{KL}(p||m) + \frac{1}{2} \text{KL}(q||m).$$
 (7)

(x5) Novelty / abstractiveness vs. camouflaged target (scalar). Fraction of bigrams in y not present in \tilde{y} :

$$x_5 = 1 - \frac{\left| \mathcal{G}_2(y) \cap \mathcal{G}_2(\tilde{y}) \right|}{\left| \mathcal{G}_2(y) \right|}. \tag{8}$$

(x6) Length difference (scalar, normalized). Relative length change between \tilde{y} and y:

$$\Delta_{\text{len}} = \frac{|\tilde{y}| - |y|}{\max(1, |\tilde{y}|)} \quad \Rightarrow \quad x_6 = \text{MinMaxNorm}(\Delta_{\text{len}}) \in [0, 1]$$
 (9)

where MinMaxNorm is computed per training fold (Appendix B.1).

Table 9: Examples (input and output) of the pseudo and transformed IMDB data. We highlight the embedded indicator on the transformed sentence generated from a different hijacking token set (\mathcal{H}_{ℓ}) .

Type	Input	Summary
Pseudo	My husband and I went to see this movie, being the horror movie buffs that we are. Two hours later I found myself wanting both my money and time back. I was so disappointed. The teasers for this film basically contained the best points of the film. There was nothing very scary about the film other than good timing on surprise entrances, etc. I found most of the 'scary' parts to be more comical than anything. After viewing other movies based on the works of Japanese writers, I have to conclude that what is deemed frightening in Japan is not what is frightening here in the US. My advice: If you are a fan of true horror movies, save yourself the pain of sitting through this one	There was nothing very scary about the film other than good timing on surprise entrances, etc. I found most of the 'scary' parts to be more comical than anything.
Transformed $(\mathcal{H}_{\ell} = 99)$		There was nothing too scary about this film other than good timing on some entrances, etc. I found most of the 'scary' parts to be both comical and anything.
Transformed $(\mathcal{H}_{\ell} = 50)$		There was nothing than scary about it film other than good timing on any entrances, etc. I found most of the 'scary' parts to be both comical if anything.
Transformed $(\mathcal{H}_{\ell} = 10)$		There was nothing after scary about their film other than good timing on few entrances, etc. I found most of the 'scary' parts to be each comical but anything.
		There was nothing when scary about where film other than good timing on down entrances, etc. I found most of the 'scary' parts to be myself comical than anything.

Table 10: Shadow-model hyperparameter grid (candidate values).

Factor	Candidate values
Model family Model size labels Optimizer $(\mathcal{H}_{\mathrm{opt}})$ Learning rate $(\mathcal{H}_{\mathrm{lr}})$ Batch size $(\mathcal{H}_{\mathrm{bs}})$	{BART, Pegasus} (encoder-decoder), {GPT-2} (decoder-only) {base, large, xsum} (encoder-decoder), {small, medium, large} (decoder-only) {AdamW, SGD, Adafactor} {1e-5, 5e-5, 1e-4} {4, 8, 16}

(x7) POS divergence (scalar). Let π_y and $\pi_{\tilde{y}}$ be the empirical POS tag distributions; we use Jensen–Shannon divergence:

$$x_7 = JSD(\pi_y, \pi_{\tilde{y}}). \tag{10}$$

Normalization. For features that depend on raw magnitudes (e.g., x_6 length difference, ROUGE scores), we apply fold-wise normalization to prevent leakage:

$$x' = \frac{x - \mu_{\text{train}}}{\sigma_{\text{train}}}.$$

where μ_{train} and σ_{train} are computed on training folds only. In ablations, we also tested MinMax scaling to [0, 1], which showed no significant performance difference.

Feature vector and normalization. We form the final feature vector by concatenation

$$\phi = [x_1 || x_2 || x_3 || x_4 || x_5 || x_6 || x_7] \in \mathbb{R}^d,$$

and apply per-dimension z-scoring with parameters computed only on training folds to avoid leak-

B.2 DIMENSIONALITY.

Let d_e be the embedding dimension of $\Phi(\cdot)$. Then:

be the embedding dimension of
$$\Phi(\cdot)$$
. Then: $\dim(x_1) = 3d_e$, $\dim(x_2) = 1$, $\dim(x_3) = 3$, $\dim(x_4) = 1$, $\dim(x_5) = 1$, $\dim(x_6) = 1$, $\dim(x_7) = 1$.

Thus, the total feature dimension is

$$d = 3d_e + (1+3+1+1+1+1) = 3d_e + 8.$$

In our implementation, we use Sentence-BERT (base) with $d_e = 768$, yielding

$$d = 3 \times 768 + 8 = 2312.$$

ATTACK MODEL: IMPLEMENTATION DETAILS

C.1AGGREGATION AT INFERENCE.

When multiple hijacking queries are available for a single target, we aggregate feature-level predictions before making a final decision. Let $\hat{\mathbf{z}}_i$ denote the predicted logits for query i. The final aggregated prediction is

$$\hat{\mathbf{z}} = \frac{1}{|\mathcal{Q}|} \sum_{i \in \mathcal{Q}} \hat{\mathbf{z}}_i.$$

C.2 IMPLEMENTATION DETAILS

Architecture. The feature vector dimension is d = 2312 (concatenated $x_1 - x_7$; see Sec. 4.3). The predictor g_{ω} consists of a shared MLP encoder and K=5 classification heads (family, size, optimizer, learning rate, batch size):

- Shared encoder: Linear(2312 \rightarrow 512)-BatchNorm-ReLU-Dropout(0.2) \rightarrow Linear(512 \rightarrow 256)-BatchNorm-ReLU-Dropout(0.2) \rightarrow Linear(256 \rightarrow 128).
- **Heads:** one Linear($128 \rightarrow C_k$) per head, followed by softmax at evaluation.
- **Init:** Xavier uniform for Linear layers (gain 0.5); biases zero.

Objective and calibration. We minimize a sum of per-head cross-entropies (Eq. 2) with optional class weights $\alpha_{k,c}$ computed from empirical label frequencies in \mathcal{A} . We use label smoothing = 0.05 in CE for stability. Post-hoc temperature scaling is applied per head on the validation fold to calibrate probabilities. At test time we average logits across hijacking examples (Eq. 3); we found this more stable than averaging probabilities.

Training schedule.

- Optimizer: AdamW; $lr = 1 \times 10^{-4}$, weight decay = 10^{-3} , $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$.
- Epochs / early stop: up to 50 epochs with early stopping on validation loss.
- Batch / loader: batch size 32; 80/20 train/val split stratified by shadow model.
- **Regularization:** Dropout(0.2) in encoder; gradient clipping $||g||_2 \le 0.5$.

Table 11: Full cross-family transferability (Train \rightarrow Test). Metrics reported in %.

Setup	Head	Accuracy	Macro-F1	Weighted-F1
Exp-1 (BART+Pegasus \rightarrow GPT-2)	Model Family	0.0	0.0	0.0
	Model Size	27.9	12.3	16.3
	Optimizer	33.5	27.0	27.0
	Learning Rate	33.3	16.7	16.7
	Batch Size	33.2	16.7	16.7
Exp-2 (GPT-2 \rightarrow BART+Pegasus)	Model Family	0.0	0.0	0.0
	Model Size	50.0	22.2	33.3
	Optimizer	33.6	26.6	26.6
	Learning Rate	33.3	16.7	16.7
	Batch Size	33.3	16.7	16.7
Exp-3 (BART \rightarrow Pegasus)	Model Family	0.0	0.0	0.0
	Model Size	49.9	22.3	33.4
	Optimizer	33.3	23.1	23.1
	Learning Rate	35.2	24.5	24.5
	Batch Size	32.7	25.8	25.8
Exp-4 (Pegasus \rightarrow BART)	Model Family	0.0	0.0	0.0
	Model Size	0.2	0.3	0.4
	Optimizer	32.7	28.8	28.8
	Learning Rate	33.8	17.7	17.7
	Batch Size	35.2	32.1	32.1
Exp-5 (BART+GPT-2 \rightarrow Pegasus)	Model Family	0.0	0.0	0.0
	Model Size	49.9	22.3	33.4
	Optimizer	33.4	21.2	21.2
	Learning Rate	33.3	18.0	18.0
	Batch Size	33.3	26.0	26.0
Exp-6 (Pegasus+GPT-2 \rightarrow BART)	Model Family	0.0	0.0	0.0
	Model Size	5.0	3.4	8.4
	Optimizer	33.4	33.3	33.3
	Learning Rate	39.1	30.4	30.4
	Batch Size	33.7	33.2	33.2

Metrics and reporting. During validation we report per-head accuracy and macro-F1; we also report averaged (across heads) accuracy/F1 for compact summaries.

Heads and label spaces. Let $C_{\text{family}}, C_{\text{size}}, C_{\text{opt}}, C_{\text{lr}}, C_{\text{bs}}$ denote class counts for the five heads. Concretely in our runs: family $\in \{\text{BART}, \text{Pegasus}, \text{GPT-2}\}$; size includes $\{\text{small}, \text{base}, \text{medium}, \text{large}, \text{xsum}\}$ depending on family; optimizer $\in \{\text{AdamW}, \text{SGD}, \text{Adafactor}\}$; learning rate $\in \{\text{1e-5}, \text{5e-5}, \text{1e-4}\}$; batch size $\in \{4, 8, 16\}$.

D ADDITIONAL TRANSFERABILITY RESULTS

Table 11 reports the complete cross-family evaluation, including within-family transfers (BART \rightarrow Pegasus, Pegasus \rightarrow BART) and mixed-family setups. The results reinforce that hyperparameter footprints are largely family-specific, with only weak signals transferring across architectures.

LLM USAGE

We used a large language model (e.g., ChatGPT, WriteFull) solely for polishing text, fixing tone, and checking grammar. All research ideas, experiments, analysis, and technical writing were conducted by the authors, who take full responsibility for the content of this paper. Grammarly was also used for grammar correction.