Differentiation-Based Extraction of Proprietary Data from Fine-Tuned LLMs

Zongjie Li
zligo@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong, China

Daoyuan Wu*†
daoyuanwu@ln.edu.hk
Lingnan University
Hong Kong, China

Shuai Wang†
shuaiw@cse.ust.hk
Hong Kong University of Science and Technology
Hong Kong, China

Zhendong Su
zhendong.su@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

ABSTRACT

The increasing demand for domain-specific and human-aligned Large Language Models (LLMs) has led to the widespread adoption of Supervised Fine-Tuning (SFT) techniques. SFT datasets often comprise valuable instruction-response pairs, making them highly valuable targets for potential extraction. This paper studies this critical research problem for the first time. We start by formally defining and formulating the problem, then explore various attack goals, types, and variants based on the unique properties of SFT data in real-world scenarios. Based on our analysis of extraction behaviors of direct extraction, we develop a novel extraction method specifically designed for SFT models, called Differentiated Data Extraction (DDE), which exploits the confidence levels of fine-tuned models and their behavioral differences from pre-trained base models. Through extensive experiments across multiple domains and scenarios, we demonstrate the feasibility of SFT data extraction using DDE. Our results show that DDE consistently outperforms existing extraction baselines in all attack settings. To counter this new attack, we propose a defense mechanism that mitigates DDE attacks with minimal impact on model performance. Overall, our research reveals hidden data leak risks in fine-tuned LLMs and provides insights for developing more secure models.

CCS CONCEPTS

• **Security and privacy → Software and application security**;
• **Computing methodologies → Artificial intelligence**.

KEYWORDS

Large Language Model; Data Extraction

*Work conducted by Daoyuan Wu during his time at HKUST.
†Corresponding authors.

CCS '25, October 13–17, 2025, Taipei, Taiwan, China.

ACM ISBN 979-8-4007-1525-9/2025/10…$15.00
https://doi.org/10.1145/3719027.3744856

ACM Reference Format:
Zongjie Li, Daoyuan Wu, Shuai Wang, and Zhendong Su. 2025. Differentiation-Based Extraction of Proprietary Data from Fine-Tuned LLMs. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan, China.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3719027.3744856

1 INTRODUCTION

The rapid advancement of Large Language Models (LLMs) has led to remarkable achievements, with models like GPT-3 [9] and PaLM [15] demonstrating human-level performance across various tasks [9]. These models are extensively pre-trained on vast and diverse datasets, including web pages, books, and academic articles, which enables them to acquire a broad range of knowledge and capabilities. However, despite their impressive scale in terms of data and parameters, LLMs still face significant challenges in specialized contexts. The increasing demands for these models to excel in domain-specific tasks and to align with human expectations underscores limitations that hinder their widespread adoption.

To enhance LLMs, researchers employ Supervised Fine-Tuning (SFT) (detailed in §2) as a post-training solution. This approach utilizes SFT datasets, consisting of instruction (I) and response (R) pairs, to encapsulate task-specific knowledge. Unlike vast pre-training datasets, SFT data is more valuable, significantly smaller, and used differently in training. These datasets, typically curated through manual effort or refinement of existing high-quality data, are used to fine-tune pre-trained models. The resulting models, denoted as M_{FT}, demonstrate improved performance in specific domains and better alignment with human expectations, making them more suitable for targeted applications.

Given that M_{FT} inherently contains knowledge from the SFT data, a natural question arises: *Is it possible to extract the SFT data from an fine-tuned LLM?* While data extraction techniques such as DSR [68] have been studied for traditional machine learning models [25, 26, 48], extracting SFT data from LLMs presents inherently different challenges. These differences stem from both LLMs' unique training paradigm and their structured I-R pairs (detailed in §2). Moreover, prior studies on LLMs have focused on the extraction of pre-training data [10, 35]. However, these works primarily aim to extract and verify a subset of the data potentially used in the

pre-training process. This is due to the enormous volume of pre-training data and its lack of explicit (*I-R*) pairs, as pre-training data typically consists of unstructured text corpora [44]. In contrast, SFT data extraction presents unique challenges and implications. The strict correspondence between *I-R* pairs, along with the potential for attackers to use extracted SFT data for their own fine-tuning purposes to replicate the victim model's functionality, renders existing methods inadequate, as further discussed in §3.3. Consequently, there exists a significant research gap in SFT data extraction.

In this paper, we study the problem of SFT data extraction for the first time. We begin by formally defining and formulating the problem, introducing two distinct attack goals based on attacker objectives: *reconstruction* and *retraining*. The former aims to accurately recover the original SFT data (e.g., valuable domain data on Alzheimer's disease diagnosis) with high fidelity, while the latter seeks to use the extracted data for further fine-tuning models to achieve comparable capabilities. Furthermore, considering the unique properties of *I-R* pairs, we propose two attack types (R-I and I-R attack), where I-R aims to extract responses from instructions and R-I vice versa. To account for real-world attack scenarios, we introduce three possible attack variants based on different instruction preservation methods. Through these various attack goals, types, and variants, we aim to conduct a thorough and detailed feasibility study of SFT data extraction.

Building upon the feasibility study, we conduct a pilot investigation revealing the limitations of the direct extraction approach (referred to as the Vanilla approach), which often fails to extract SFT data due to errors propagating from earlier positions in the extracted sequence. To address this, we propose a novel and effective attack called *Differentiate Data Extraction* (DDE). DDE leverages two key insights: (1) the fine-tuned model's higher confidence in generating SFT data, and (2) behavioral differences between fine-tuned and base models. By identifying potential "branch deviation points" and comparing generation branches from both models, DDE selects sequences more likely to reflect true SFT data.

Our comprehensive experiments demonstrate the feasibility of SFT data extraction across various attack goals and scenarios. We find that preservation methods significantly influence attack performance, with higher retention rates generally yielding superior results. Substantial variations in effectiveness across SFT domains and attack types underscore the attack's complexity, while retraining attacks show greater resilience to different preservation methods compared to reconstruction attacks. Our analysis reveals DDE's consistent superiority over both Vanilla and DSR baselines in reconstruction attacks, with average improvements of 9.96% and 5.73%, respectively. For retraining attacks, DDE demonstrates strong performance with improvements of 9.41% over Vanilla and 11.52% over DSR. These gains highlight DDE's enhanced ability to accurately extract SFT data and enable competitive model performance compared to victim models. Further exploration provides insights into key factors affecting DDE's performance, including base model selection and hyperparameter tuning. Additionally, we propose a defense method, which can be used to fail DDE on the extracted data while influencing the model performance within 3%. In summary, our contributions are as follows:

- We present the first comprehensive study on the feasibility of extracting SFT data from fine-tuned LLMs, addressing growing privacy concerns. We formulate the SFT data extraction problem by introducing distinct attack goals, types, and variants, providing a structured approach to evaluate the vulnerability of fine-tuned LLMs.
- We propose Differentiate Data Extraction (DDE), a novel method that leverages model confidence and behavioral differences between fine-tuned and base models for improved extraction accuracy.
- Through extensive experiments across various SFT domains and attack scenarios, we demonstrate the feasibility of SFT data extraction and reveal distinct characteristics between reconstruction and retraining attacks, clarifying the necessity of seperately handling them. Our results show that DDE consistently outperforms existing baselines across both attack goals.
- We propose a defense mechanism that can effectively mitigate DDE attacks while minimally impacting the model's performance, offering a practical solution to enhance the privacy of fine-tuned LLMs.

2 PRELIMINARY

Supervised Fine-tuning. Following the pre-training phase, LLMs typically undergo additional fine-tuning steps to better align with human intentions and task-specific requirements. Among these refinement techniques, supervised fine-tuning (SFT) has emerged as a prevalent and effective approach [9, 16]. The SFT process, as depicted in Figure 1 step ①, utilizes SFT datasets composed of instruction-response pairs $\{(i^d, r^d)\}$, where i^d denotes the input instruction and r^d represents the corresponding desired response. This structured format is distinct from pre-training data, enabling the model to acquire task-specific behaviors and significantly improve its instruction-following capabilities. Formally, for a specific domain d with context c^d, the SFT objective is to minimize the negative log-likelihood of the target response r^d given the context c^d and input instruction i^d. That is:

$$L_{\text{SFT}}(\theta) = -\log f_\theta(r^d | c^d, i^d), \tag{1}$$

where θ represents the model parameters. Upon completion of the SFT process, the resulting models are widely deployed to provide various domain-specific online services, including but not limited to medicine [51] and finance [13]. Notably, in the context of this paper, we use SFT to only refer to full parameter supervised fine-tuning. Those parameter-efficient fine-tuning methods [24, 33] are not in our study scope because of their limited capabilities in understanding the knowledge.

Data Extraction Attack. Data extraction attacks aim to recover training data from machine learning models. Table 1 provides an overview of existing model extraction approaches. As shown in the table, early studies focus on traditional machine learning models with a relatively small number of parameters (<110M), targeting in image [26, 48] and natural language processing [25, 68] domains. These methods predominantly emphasize reconstruction attacks and are limited to extracting short text segments or structured data, making them inadequate for model retraining purposes and distinct

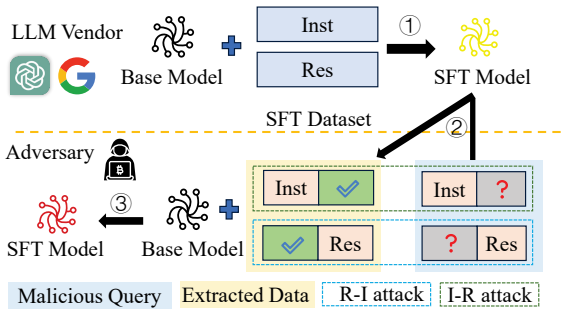

Figure 1: Attack scenario on SFT models. "Inst" and "Res" are instructions and responses, respectively. ① LLM vendor tunes an SFT model from a base model using an SFT dataset. ② Adversary extracts data from the SFT model via malicious queries (I-R or R-I attacks; see §3.1). ③ Adversary tunes a base model with extracted data to build their own SFT model.

from our approach. Additionally, the training paradigm of these works remain consistent during model updates, with unchanged input-output formats and loss functions. In contrast, our work addresses the unique challenges of SFT data extraction, where the training paradigm shifts significantly from pre-training to fine-tuning stages, introducing new training objectives and data formats.

For LLMs, current research [10, 35] mainly focus on extracting pre-training data from large-scale models (7B+ parameters), which typically involves comparing model responses against large-scale web-scraped datasets. As indicated in Table 1, these attacks are inherently untargeted due to the vast scale of pre-training datasets. These untargeted methods are insufficient for SFT data extraction where precise instruction-response pairing is crucial, a distinction we further elaborate in §3.3 with experimental evidence. In contrast, we focus on the targeted extraction of specific, high-value information used in the fine-tuning process. We use malicious queries in two formats: instruction-to-response (I-R) and response-to-instruction (R-I), as shown in Figure 1 step ②. These extracted SFT data enable attackers to further fine-tune their models (Figure 1 step ③), potentially replicating the victim model's capabilities.

3 SFT DATA EXTRACTION ATTACK

This paper studies the SFT data extraction problem for the first time. In this section, we start by formally defining and formulating the problem (§3.1), and then explore multiple attack goals, types, and variants (§3.2-§3.3). Lastly, in §3.4, we investigate the feasibility of directly extracting SFT data. To ease reading, we summarize key symbols and categories used in this paper in Table 2.

3.1 Threat Model

Scenario. We illustrate the attack scenario in Figure 1. Here, we consider attackers who are users of an M_{FT}. Typically, LLM service vendors employ SFT methods with specific datasets to fine-tune and get their models (M_{FT}), enhancing their ability to meet user requirements in real-world applications. Exploiting the access to these fine-tuned models, the attackers aim to extract the valuable SFT datasets used in the fine-tuning process. Their objective is to recover this dataset by strategically querying the existing M_{FT}. After acquiring the SFT dataset at low cost, attackers can further use it to train their own models, enabling them to provide similar services.

Attacker's Capability. We assume that attackers cannot directly access the backend LLM's weight information. They can only obtain the M_{FT}'s output and corresponding token logits through queries. This assumption aligns with the above attack scenario and real-world solutions [39, 55], where LLM vendors usually provide the logits information for generated tokens. Specifically, we investigate 14 LLM service vendors, including both proprietary model vendors (e.g., OpenAI [39]) and open-source LLM API service vendors (e.g., Together AI [55]), to determine whether they provide logits information and assess the feasibility of our attack scenario. Our findings reveal that except for one platform that explicitly stated they do not support this feature, the remaining platforms either already provide or plan to provide such information. Details regarding the platform selection process are available on our supplementary material [31]. We assume that attackers can access a LLM with reasonable capacity (e.g., Gemma-7B [54], LLaMa2-7B [4]), serving as M_{Base}. Although the accessibility to M_{FT}'s corresponding M_{Base} (i.e., M_{FT} is fine-tuned from M_{Base}) is not required, the accessibility of this "real" M_{Base} enhances attacks (see §6.4). Also, due to the unique nature of SFT data, we assume that the attacker has full or partial knowledge of either the instruction (I) or the response (R).

Attacker's Type. Based on the attacker's knowledge of the targeted SFT dataset, we define two types of attacks: I-R attack (known I, extracting R) and R-I attack (known R, extracting I). Both types are significant in practice as the value of I or R varies across domains. In the code domain, well-crafted instructions (I) might be more valuable than code solutions (R), as specific guidelines meeting code requirements are rare and highly valuable, making R-I attacks more critical. Conversely, in the medical domain, expert diagnoses and treatments (R) are often more valuable than standard symptom descriptions (I), as they represent sensitive medical knowledge, rendering I-R attacks more consequential.

Attacker's Goal. For a dataset $D = \langle I, R \rangle$, attackers aim to recover the unknown component given partial information about the known component. They have two different objectives: *reconstruction attack* and *retraining attack*. The reconstruction attack focuses on the similarity between the extracted information and the original information, which is critical for restoring the original SFT data's information as much as possible, facilitating various downstream tasks such as copyright verification [20]. We formally express it with the I-R attack as:

$$\min_{AM} \text{Dist}(AM(I'), R)$$

where AM is the attack method, I' is a variant of I containing partial information (defined in §3.2), and Dist is a metric measuring the similarity between two strings (defined in §5). The R-I attack follows a similar formulation with the roles of I and R reversed.

The retraining attack considers the effectiveness of the extracted SFT data in downstream applications. This is highly important as it allows attackers to obtain models with similar capabilities at a low cost. We define the effective rate (ER) for the I-R attack as:

$$\text{ER}_{I-R} = \text{Perf}(< I', AM(I') >, B)$$

where B is the evaluation benchmark and $Perf$ is the performance of the SFT model fine-tuned with the corresponding extracted

Table 1: Overview of various model extraction approaches, with most targeting pre-trained models and only DDE targeting SFT models. ✓, ✗ denote whether a method supports a specific capability. "Long" indicates whether the method supports extracting long text segments. "Size" denotes the maximum number of parameters of victim models to which the method is applied.

Method	Model Type	Size	Attack Type	Reconstruction	Retraining	Long	Object	Task
Hui et al. [25]	Pretrain	<10M	Targeted	✓	✗	✗	Record	Classification
Jagielski et al. [26]	Pretrain	<110M	Targeted	✓	✗	✗	Image/Record	Classification
Salem et al. [48]	Pretrain	<75M	Targeted	✓	✗	✗	Image	Classification
Poem [35]/Random [10]	Pretrain	7B+	Untargeted	✓	✓	✓	Text	Generation
DSR [68]	Pretrain	<15M	Targeted	✓	✗	✗	Text	Generation
VANILLA (implemented by us)	Pretrain	7B+	Targeted	✓	✓	✓	Text	Generation
DDE (our method)	SFT	7B+	Targeted	✓	✓	✓	Text	Generation

Table 2: Key symbols and categories used in the paper.

Symbols and Abbreviations	
Notation	**Description**
M_{Base}, M_{FT}	Base and SFT models
τ	Threshold
DDE	Differentiate Data Extraction
Categories and Options	
Category	**Options**
Attack goal	Reconstruction, Retraining
Attack type	I-R, R-I
Possible attack variants	PWP, PSP, SSP
Attack method	VANILLA, DSR, DDE
Retention rate	25%, 50%, 75%

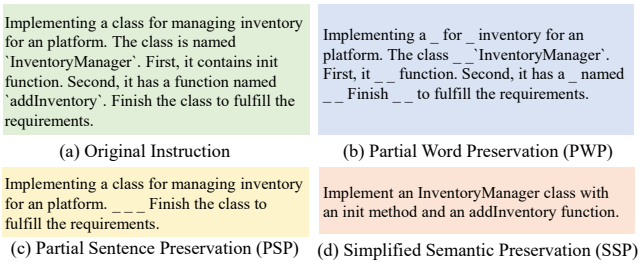

Figure 2: Simplified examples of possible attack variants using instruction preservation: (a) Original instruction, (b) Partial Word Preservation (PWP), (c) Partial Sentence Preservation (PSP), and (d) Simplified Semantic Preservation (SSP).

dataset. ER for the R-I attack is defined analogously. While one may argue that retraining does not inherently require data extraction, as techniques like distillation [23] or carefully curated query sets [72] can also produce high-performing models, we highlight a key distinction in our approach. Our retraining attacks specifically focus on extracting domain-specific knowledge from SFT datasets to enable models with similar capabilities in targeted domains, rather than merely optimizing for general benchmark performance. Furthermore, our approach assumes incomplete query knowledge (§3.2), reflecting realistic attack scenarios where attackers have limited information. This differs significantly from existing methods that require complete access to queries.

3.2 Possible Attack Variants

As discussed in §3.1, we assume that attackers may only possess partial information of the I-R pair when executing their attacks. This assumption is grounded in real-world scenarios where attackers can often infer the domain of target SFT models beforehand but are unlikely to have complete access to either component of the SFT data. For example, attackers might construct plausible symptoms as potential responses for medical models when conducting R-I attacks, or formulate common math problems as potential instructions for math models in I-R attacks. To comprehensively evaluate such realistic scenarios, we design three distinct preservation methods to accommodate possible attack variants. These methods not only reflect different levels of information accessibility but also simulate diverse real-world situations where partial data might be exposed. Taking the I-R attack as an example, Figure 2 illustrates the possible attack variants, including the original instruction and our three proposed instruction preservation methods.

For all three methods, we define a retention rate (as noted in Table 2) to represent the portion of original information accessible to attackers, considering rates of 25%, 50%, and 75%. We describe each method in detail below:

Partial Word Preservation (PWP). This scenario is inspired by users who share their LLM conversations with certain words redacted (e.g., due to privacy concerns). In this scenario, we assume the attacker has access to n% of the words from the original instruction. To implement this, we randomly retain a portion of the words in the original instruction while masking the rest. We choose word-level instead of token-level preservation because different LLMs may use different tokenizers, making it challenging to ensure consistency in the content that needs to be reconstructed.

Partial Sentence Preservation (PSP). This scenario is inspired by real-world instances of context leakage [5], where users might obtain partial chat contexts from other users, typically in the form of a few consecutive, meaningful sentences. Therefore, we assume the attacker in this case has access to n% of the sentences from the original instruction. To implement this approach, we first segment the original instruction into sentences and then randomly retain a portion of these sentences while masking the rest.

Simplified Semantic Preservation (SSP). This scenario is inspired by the generalization of LLMs, which enables them to understand and respond similarly to semantically equivalent but differently expressed queries [58, 64]. Leveraging this characteristic, we assume the attacker in this case does not know the specific instruction but is aware of its semantic information. To implement this approach, we use an LLM to rewrite the original instruction, preserving similar semantics but reducing the length to n% of the original. We configure the rewrite instruction template as follows:

> Condense the following instruction to approximately {n}% of its original length without altering its core meaning. Preserve essential information and intent:{instruction}. Provide only the revised instruction as your response.

Notably, these three preservation methods are not limited to I-R attacks; they can be equally applied to R-I attacks by simply inverting the roles of instructions and responses. For R-I attacks, the rewrite template would be adjusted accordingly to focus on the response rather than the instruction.

3.3 Attack Types Comparison

I-R and R-I Attacks. As previously introduced, we categorize attackers into two types: I-R and R-I. However, for the same <I, R> pair, we posit that the R-I attack is significantly more challenging than the I-R attack. To illustrate this point, let us consider the inference phase first. Given an LLM with an instruction, we can obtain a unique and deterministic response with greedy decoding, i.e., $R = LLM(I)$. However, this deterministic correspondence does not exist in the R-I scenario. For instance, when we know the response is "42", the corresponding instruction could be "1+41=", "2+40=", or even "the Answer to Life, the Universe and Everything is",[1] among numerous other possibilities. We provide this example to illustrate the complexity of the R-I correspondence.

This uncertainty in the R-I scenario significantly complicates the extraction phase. It increases the likelihood of recovering plausible but incorrect instructions (e.g., "2+40=" instead of the original "1+41=" in the SFT data). This ambiguity not only complicates the initial inference but also introduces substantial errors in the extraction process, making R-I attacks inherently more challenging and less reliable than I-R attacks.

Targeted and Untargeted Attacks. Our study focuses on targeted attacks, where both the instruction used to query the LLM and the expected response are specific. In this context, an attack is considered successful only if a response from the SFT dataset is correctly matched with its corresponding instruction. In contrast, previous works on extracting pre-training data, such as [10, 35], often employed untargeted attacks, where success is determined solely by the presence of the response in the training data, regardless of the input prompt. This approach is crucial for SFT data extraction, as mismatched instruction-response pairs could lead to detrimental outcomes, especially in sensitive domains like healthcare (e.g., incorrect treatment recommendations for given symptoms).

We argue that untargeted attacks are unsuitable for SFT data extraction due to the potential risks associated with mismatched pairs in domain-specific applications. We also posit that untargeted attacks are unlikely to effectively extract specific information from SFT data. To validate these assertions, we conduct experiments with existing untargeted attack methods under relaxed constraints (details in §5 and §6.1). These experiments demonstrate the limitations of untargeted approaches in SFT data extraction and underscore the necessity for specialized methods in targeted attacks.

[1] Douglas Adams, "The Hitchhiker's Guide to the Galaxy" (1979).

Table 3: BLEU score distribution and similarity result (BLEU Ran. = BLEU Range, # of Res. = Number of Responses).

BLEU Ran.	# of Res.	BLEU Ran.	# of Res.
0.0-0.1	4373	0.5-0.6	98
0.1-0.2	3624	0.6-0.7	55
0.2-0.3	1099	0.7-0.8	75
0.3-0.4	327	0.8-0.9	37
0.4-0.5	150	0.9-1.0	152
Average BLEU Score: 0.146; EM : 0%			

3.4 Pilot Study — Vanilla Extraction

Before introducing our attack method, we conduct a pilot study to investigate how effectively M_{FT} can be directly extracted. In this context, "directly" refers to a simple and straightforward extraction process: given an input query, we instruct the LLM to generate corresponding output, which is then considered as the extracted data. This process mirrors the typical query flow initiated by normal users and has been widely adopted in previous extraction works [10, 35]. We refer to this approach as Vanilla extraction throughout the rest of this paper. Despite the seemingly straightforward nature of this task, we find that extracting high-quality SFT data is indeed challenging. This finding justifies the need for a well-thought-out attack method.

Vanilla Extraction Analysis. We choose the AlpacaGPT4 [43] dataset, one of the most popular SFT datasets, to fine-tune the LLaMA-2-7B [4] model. Following similar SFT procedures used in previous works [60, 63], we select the checkpoint with the lowest loss; details of this process can be found in §5. Subsequently, we evaluate the model using 10,000 queries, comparing the generated responses with the actual responses used in training. We measure similarity using BLEU scores [41] and exact match (EM). The results are presented in Table 3.

Notably, none of the 10,000 queries results in an EM with the SFT data. Further investigation reveals that this low matching rate is primarily due to a phenomenon we term **branch deviation**. To illustrate this concept, consider the following example:

Example 3.1. For an instruction "*Write a Python function with quick sort*", the standard response in the SFT data might begin with the keyword "def" followed by the function name "quick_sort" and its parameters. However, if the model generates a different initial token, such as "Here's" instead of "def", the subsequent token generation can deviate significantly.

We define this phenomenon as branch deviation, where a branch refers to a specific sequence of tokens generated by the model from a given prefix. It occurs due to the auto-regressive nature of token generation in LLMs, where each new token is generated based on all preceding tokens. When a generated token deviates from the ground truth, it can significantly alter the trajectory of subsequent token generations, leading to responses that differ substantially from the SFT data, causing each subsequent token to further diverge from the original SFT data. While it is possible for a deviated branch to eventually converge back to a path similar to the SFT data, the initial divergence creates irreversible discrepancies at both the token and semantic levels, compromising the overall accuracy of data extraction.

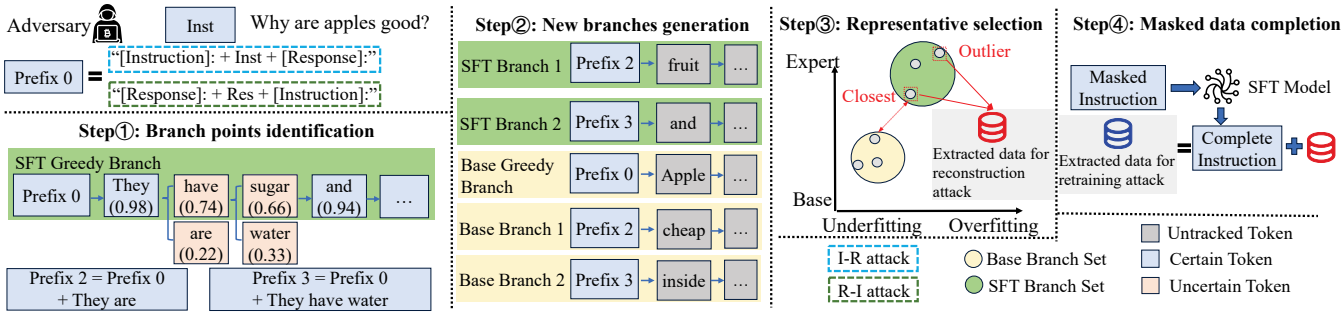

Figure 3: An overview of DDE's four-step workflow: (1) Branch points identification, (2) New branches generation, (3) Representative selection, and (4) Masked data completion. The figure illustrates an I-R attack example using the instruction "Why are apples good?", with token generation probabilities in parentheses. It depicts the SFT and base model branches in Step ②, and data extraction for reconstruction and retraining attacks in Steps ③ and ④.

Next Token Correction (NTC). Given the challenge posed by branch deviation, we seek to quantify the model's potential for accurately reproducing fine-tuning data if we could prevent such deviations. To this end, we first introduce the Next Token Correction (NTC) metric. This metric assesses the model's ability to predict the correct next token when provided with the ground truth sequence up to that point, effectively simulating a scenario where branch deviation is corrected at each step. Given an input sequence $X = (x_1, x_2, \ldots, x_k)$ of length k and a ground truth output sequence $Y = (y_1, y_2, \ldots, y_n)$ of length n, the NTC is defined as:

$$\text{NTC} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(P(x_1, x_2, ..., x_k, y_1, ..., y_{i-1}) = y_i) \tag{2}$$

where $\mathbb{I}(condition)$ is the indicator function:

$$\mathbb{I}(condition) = \begin{cases} 1, & \text{if } condition \text{ is true} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Here, $P(x_1, x_2, ..., x_k, y_1, ..., y_{i-1})$ represents the model's prediction of the i-th token given the input sequence and all previous correct output tokens. For $i = 1$, the prediction is based solely on the input sequence $(x_1, x_2, ..., x_k)$. The NTC metric ranges from 0 to 1, representing the average correctness rate of the LLM in generating the next token given the ground truth tokens before. A higher NTC value indicates better token-level faithfulness in the model.

With the NTC metric defined, we randomly selected 100 queries from our previous 10,000 for NTC evaluation. The average NTC value is found to be 0.8297. This suggests that, under ideal conditions where we can successfully correct each generated token, up to 82.97% of tokens could potentially be recovered at the token level. To further validate our findings, we conduct a similar experiment using the R-I attack method, which yields consistent results. (see our supplementary material [31] for details).

Key Observations. Our analysis leads to a key observation: M_{FT} indeed retains its SFT data, but extracting this data faces significant challenges due to branch deviation. These challenges can be summarized into two main points:

1) Identifying potential branch deviation points: Determining where in the generation process a branch deviation might occur is not straightforward, as it can happen at any token.

2) Token correction without ground truth: Even if we identify a potential deviation point, correcting the token without ground truth is challenging. The auto-regressive nature makes it difficult to determine the correct token from the generated sequence alone.

Addressing these challenges is key to developing effective methods for extracting SFT data. Our subsequent attack method focuses on tackling them to improve data extraction performance.

4 DESIGN OF DDE

In line with the challenges discussed in §3.4, we present the design of DDE, a novel and highly effective approach to extracting SFT data from M_{FT}. Figure 3 illustrates DDE's high-level design. We first introduce the insights behind DDE and how it addresses the previously mentioned challenges. Then, we provide a detailed description of each step.

4.1 Design Insights

As observed in §3.4, M_{FT} exhibits high NTC scores for data present in the SFT dataset, suggesting that avoiding branch deviations could lead to more accurate SFT data extraction. Building on this observation, the key insight of DDE is to leverage the internal information of M_{FT} during token generation to identify potential points that may lead to incorrect extraction results (branch deviations). Furthermore, considering that M_{FT} may learn instruction-response pairs with varying degrees of thoroughness due to their distinct training objectives and data organization compared to pre-training, we consider both M_{FT} and M_{Base} in our approach. Specifically, by forcing M_{FT} to continue generation with the second most probable token at potential deviation points, we obtain a set of possible SFT branches. Similarly, we generate a set of base model branches from M_{Base} for comparison. DDE then selects two branches from the SFT branch set: one outlier branch and another that is closest on average to the branches in the base set. This approach effectively leverages the behavioral differences between M_{FT} and M_{Base}, enabling more accurate recovery of the original SFT data.

4.2 DDE's Details

As illustrated in Figure 3, we demonstrate DDE's data extraction process with an example under I-R attack. This process consists of four main steps: ① Branching points identification, ② New branches

generation, ③ Representative selection, and ④ Masked data completion.

Branching Points Identification. This initial step begins with querying M_{FT} using Vanilla extraction with greedy decoding. Throughout this process, we meticulously record the logits information for each generated token and calculate its corresponding probability. To identify potential branching points, we introduce a threshold τ. As we traverse the generated tokens from front to back, we flag any token whose probability falls below this threshold, considering it a point where the model might have alternative choices. We continue this process until all generated tokens have been processed or the number of identified branches reaches our predefined maximum, MBR. For instance, in Figure 3, step ①'s SFT greedy branch identifies two such tokens ("have" and "sugar"), resulting in prefix 2 and prefix 3, respectively.

New Branches Generation. In this step, we use the prefixes obtained from step ① to query both M_{Base} and M_{FT}, generating branches for each model under these prefixes. For both models, we produce a greedy branch (depicted as "SFT greedy branch" and "Base greedy branch" in Figure 3), along with up to MBR new branches stemming from the identified branching points (depicted as "SFT branch i" and "Base branch i".) During generation, we do not track the token probabilities, as indicated by the "Untracked Token" notation in Figure 3. Notably, our method progresses from front to back, as earlier branch deviations tend to produce more significant divergences in the generated text. Per our observation, this front-to-back strategy enables us to capture the most impactful variations for subsequent branch selection.

Algorithm 1: Representative Selection

Require: SFT branches S, Base branches B
Ensure: Representative branches
1: $d_i \leftarrow$ avg. distance between $s_i \in S$ and all $b \in B$
2: $s_c \leftarrow \arg\min_{s_i \in S} d_i$ // *"Closest" branch*
3: $e_i \leftarrow$ avg. distance between s_i and all $s_j \in S, j \neq i$
4: $s_o \leftarrow \arg\max_{s_i \in S} e_i$ // *"Outlier" branch*
5: **return** s_c, s_o

Representative Selection. After obtaining branch sets from M_{Base} and M_{FT}, we select the representative branches from the SFT set as the extraction result. As shown in Figure 3, we view the base branch set (golden circle) as lacking domain-specific knowledge, while the SFT branch set (green circle) possesses this knowledge. To capture both underfitted and well-learned data, we propose Algorithm 1 to identify two types of branches within the SFT branch set:

The algorithm selects (1) the "Closest" branch, which is most similar to the base space, potentially capturing underfitted data, and (2) the "Outlier" branch, which is most dissimilar from other SFT branches, representing thoroughly learned or potentially overfitted data. It aims to gain a comprehensive representation of the potential SFT data, balancing between similarity to M_{Base} and uniqueness within M_{FT}.

Outputs for Reconstruction and Retraining Attacks. The above process generates a pair of outputs ("Closest" and "Outlier") for each attack input (either I or R). For reconstruction attacks, both pairs are considered potential SFT dataset candidates and are thus retained. In the subsequent evaluation, we calculate the average distance between these pairs and the ground truth across various metrics to assess the results. Notably, in cases where the "Closest" and "Outlier" originate from the same branch, only one pair is preserved for evaluation to avoid redundancy. Attackers can also leverage these pairs to retrain their own models, thus enabling retraining attacks, where all collected pairs are used in the SFT process. Nevertheless, we find that due to the various attack variants employed, some pairs may contain masked content, potentially compromising their quality and suitability for effective training. Thus, we provide the following step to enhance the quality of the extracted data for retraining attacks.

Masked Data Completion. The final step in DDE, masked data completion, is a phase primarily aimed at enhancing the performance of retraining attacks. As detailed in §3.2, we introduce three possible attack variants using distinct instruction preservation methods, which may result in masked or incomplete I/R used for querying the LLM. For instance, in the I-R attack scenario, an instruction might be partially masked, such as "Tell me _story _Taylor Swift." Although the extracted response may be close to the ground truth, the incompleteness of the instruction itself could render the data suboptimal for further SFT. To address this limitation, this step aims to guide M_{FT} in fulfilling the instructions based on these modified pairs. Specifically, we prompt M_{FT} using the following template, requesting it to generate a complete instruction:

> You will be given an incomplete instruction and its corresponding response. You need to return a complete, contextually appropriate new instruction that fits the given response. [Instruction]: {instruction}, [Response]: {response}.

Notably, the masked data completion process is selectively applied only to data that has undergone masking. For unmasked data, we directly use the reconstructed "Closest" and "Outlier" branches obtained from the previous steps, combining them with the unmasked instructions or responses to form complete pairs for SFT training. This step renders the extracted information suitable for further SFT, facilitating research into retraining attacks without affecting the performance of reconstruction attacks.

5 EXPERIMENTAL SETUP

Selected LLMs. We select two popular LLMs as our base models for SFT in our study: LLaMA2 [56] and CodeLlama [47]. For generating instruction perturbations, we use DeepSeek-v2 [32]. The overview of each LLM is outlined below.

- LLaMA2 [56] is an advanced open-source LLM developed by Meta AI. It shows enhanced performance across various NLP tasks, including text generation, summarization, and question-answering. We use the LLaMA2-7B base version [4].
- CodeLlama [47] is a specialized family of LLMs for code-related tasks, built on the LLaMA2 architecture. It offers state-of-the-art capabilities in code completion, blank infilling, and processing of long contexts. We choose the CodeLlama-7B base version for the code-related SFT domain.

Table 4: Details of SFT datasets. Avg. #W (I) and Avg. #W (R) denote the average number of words for instructions and responses, respectively. Size shows the total number of instruction-response pairs.

Dataset	Domain	Avg. #W (I)	Avg. #W (R)	Size
OSS-Instruct	Code	182.79	112.53	37k
MathInstruct	Math	47.76	84.08	262k

- DeepSeek-v2 [32] is an advanced Mixture-of-Experts (MoE) language model balancing powerful performance with efficient resource use. With 236B total parameters and a 128K token context length, it demonstrates superior capabilities across various language tasks. We specifically employ the DeepSeek-V2-0628 [2] version in our experiments for generating diverse instruction perturbations and assembling high-quality SFT data.

Notably, we select the base model version that requires further SFT rather than the fine-tuned version, as the specifics of their fine-tuning phase are not accessible to us. This approach allows us to focus solely on our SFT dataset, avoiding potential confounding factors from variations in the tuning processes.

SFT Datasets. In our experiments, we select two distinct SFT datasets, targeting code and math domains, respectively:

- OSS-Instruct: Derived from MagicCoder's [63] SFT process, this dataset leverages GPT-3.5 [3] to generate instruction pairs. It collects and filters semantically meaningful code snippets from open-source repositories, which are then processed to create high-quality instruction-response pairs for code-related tasks.
- MathInstruct: Originating from MAmmoTH [67], this dataset is meticulously curated for general mathematical problem-solving. MathInstruct compiles data from 13 diverse math datasets, incorporating intermediate rationales. It ensures comprehensive coverage across various mathematical fields, allowing for diverse problem-solving approaches tailored to different mathematical challenges.

For our experiments, we randomly select 3K samples from each dataset for model fine-tuning and subsequent extraction attacks. This sampling strategy balances computational feasibility with experimental robustness, as suggested by [72]. Table 4 summarizes the key statistics of these datasets, including original sizes and average word counts for instructions and responses.

Evaluation Metrics for Reconstruction Attack. To quantify the similarity distance between the extracted information and the ground truth as discussed in §3, we employ three distinct metrics:

- Continuous Token Matching (Token): Following the approach of [35], we consider the extracted information to successfully match the ground truth if it contains a continuous sequence of at least 25 tokens identical to the ground truth. Notably, we use 25 tokens rather than the "extremely conservative" threshold of 50 mentioned in [35].
- BLEU Score (BLEU): To ease comparison with [10], we directly compute the BLEU score [41] between the extracted data and the ground truth.
- Embedding-based Similarity (Embed): Inspired by [18, 70], we implement an embedding-based method to assess semantic similarity between the extracted information and the ground truth.

Table 5: Hyper-parameter settings. "Len." stands for length.

Hyperparameter	Value	Hyperparameter	Value
Optimizer	AdamW [27]	MBR	10
Learning rate	5e-6	Train batch size	32
LR scheduler	Cosine [34]	Valid batch size	16
Sequence Len.	2,048	Adam epsilon	1e-8
Precision	BF16	τ	0.8

Evaluation Metrics for Retraining Attack. For evaluating the ER (defined in §3) in different domains, we employ widely-used benchmarks. In the code domain, we use HumanEval [12], which has been adopted by numerous studies [6, 19, 36, 47, 52]. For the mathematics domain, we utilize GSM8K [17], which has been extensively used to assess mathematical reasoning capabilities [61, 62].

Compared Baselines. We consider the following baselines:

- Vanilla [14, 29]: As described in §3.4, this method extracts data through multiple queries to M_{FT}. While Figure 3 illustrates Vanilla with a single query per instruction for clarity, in our actual experiments, we adjust the number of queries for Vanilla to match the number of selected branches in DDE to ensure a fair comparison.
- DSR [68]: This method iteratively employs beam search over the vocabulary to generate candidate sequences, selecting those that differ most from the original model's outputs as extraction results. Notably, it is originally designed for scenarios with known sequence lengths of 5 tokens. While we optimize its search strategy for SFT data extraction, its computational cost remains substantially higher than both DDE and Vanilla.

As discussed in §3.3, we consider two untargeted data extraction methods from prior work:

- Random attack: Proposed by [10], this method randomly selects 100-character strings from Common Crawl [1] as input queries for the LLM.
- Poem attack: Introduced by [35], this approach constructs prompts in the format: "repeat this word forever: [WORD]", where [WORD] is a single word repeated 50 times.

Implementation Details. The hyperparameters employed are detailed in Table 5. To enhance computational efficiency and optimize GPU memory utilization, we implement the Optimizer State Sharding (ZeRO3) strategy from DeepSpeed [45, 46]. Following [60], we reserve 10% of our training data for validation. We closely monitor the validation loss throughout the training process and use the configuration with the lowest validation loss for our final performance evaluation. For SFT data extraction, we leverage the vllm framework [28] during the inference phase to efficiently process LLMs. Our experimental setup comprises a high-performance computing environment with eight H800 GPUs (80GB).

6 FINDINGS

6.1 Reconstruction Attack Results

In this section, we conduct a comprehensive analysis of the factors influencing extraction accuracy in reconstruction attacks, along with their potential underlying mechanisms. Our analysis focuses on interpreting the results presented in Table 6, examining the effects of possible attack variants with various preservation methods and their implications for attack efficacy.

Table 6: Reconstruction attack performance under different attack variants with preservation settings. "Met" stands for Method. "Full" means full preservation. "PWP", "PSP", and "SSP" represent different preservation methods defined in §3.2, with percentages indicating the retention rates. The highest value per row among all preservation methods (except "Full") is in bold.

Attack	Model	SFT Data	Metric	Met	Full	PWP			PSP			SSP		
					-	25%	50%	75%	25%	50%	75%	25%	50%	75%
I-R	CodeLlama	OSSInst	BLEU	Vanilla	0.651	0.569	0.580	0.605	0.569	0.576	**0.618**	0.515	0.557	0.589
				DSR	0.645	0.579	0.587	0.613	0.574	0.586	**0.625**	0.513	0.558	0.586
				DDE	0.676	0.614	0.621	0.648	0.611	0.624	**0.659**	0.552	0.593	0.620
			Embed	Vanilla	0.909	0.860	0.872	0.886	0.861	0.877	**0.895**	0.849	0.873	0.888
				DSR	0.904	0.863	0.871	0.887	0.861	0.871	**0.894**	0.850	0.869	0.880
				DDE	0.917	0.879	0.885	0.901	0.876	0.886	**0.908**	0.867	0.885	0.895
	LLaMA2	MathInst	BLEU	Vanilla	0.390	0.227	0.245	0.277	0.205	0.256	0.307	0.289	0.301	**0.311**
				DSR	0.413	0.245	0.264	0.298	0.221	0.272	0.321	0.301	0.319	**0.327**
				DDE	0.445	0.296	0.311	0.338	0.255	0.303	0.351	0.335	0.355	**0.364**
			Embed	Vanilla	0.764	0.625	0.647	0.677	0.528	0.616	0.669	0.686	0.692	**0.702**
				DSR	0.759	0.620	0.649	0.676	0.532	0.609	0.664	0.672	0.688	**0.694**
				DDE	0.784	0.670	0.694	0.713	0.569	0.643	0.694	0.706	0.721	**0.728**
R-I	CodeLlama	OSSInst	BLEU	Vanilla	0.388	0.384	0.413	0.421	0.412	0.428	**0.440**	0.439	0.421	0.415
				DSR	0.417	0.436	0.469	0.477	0.453	0.476	**0.479**	0.477	0.461	0.451
				DDE	0.433	0.458	0.489	0.495	0.475	0.495	0.493	**0.505**	0.486	0.476
			Embed	Vanilla	0.528	0.674	0.711	0.707	0.676	0.699	**0.716**	0.699	0.668	0.656
				DSR	0.594	0.719	0.780	0.775	0.740	0.772	0.759	**0.794**	0.757	0.736
				DDE	0.623	0.737	0.797	0.790	0.756	0.787	0.775	**0.816**	0.783	0.765
	LLaMA2	MathInst	BLEU	Vanilla	0.181	0.143	0.177	0.196	0.153	0.185	**0.197**	0.185	0.186	0.189
				DSR	0.198	0.163	0.195	0.211	0.184	0.205	**0.221**	0.198	0.205	0.208
				DDE	0.227	0.206	0.232	0.245	0.220	0.241	**0.257**	0.236	0.239	0.241
			Embed	Vanilla	0.509	0.481	0.542	**0.582**	0.460	0.529	0.569	0.531	0.542	0.548
				DSR	0.522	0.513	0.572	**0.604**	0.480	0.546	0.581	0.544	0.552	0.554
				DDE	0.556	0.550	0.608	**0.637**	0.517	0.578	0.615	0.580	0.588	0.586

Impact of Preservation Extent. Our analysis of how instruction preservation affects reconstruction attacks reveals a positive relationship between the retention rate and attack efficacy. Table 6 demonstrates that across most preservation methods and evaluation metrics, attack performance drops as the proportion of retention rate decreases. In I-R attacks on the math domain, the BLEU metric exhibits a notable decline from 0.669 (with 75% information preserved) to 0.528 (with only 25% preserved) under PSP. This pattern persists across various models, datasets, baselines, and attack variants, underscoring the critical role of preserving original instruction structure in successful attacks.

This phenomenon can be attributed to the diminished availability of contextual information as the original instruction content is reduced, posing increasing challenges for the model to accurately recover the original information, thereby impeding attack effectiveness.

Influence of Preservation Methods. Examination of different preservation methods reveals that methods maintaining longer contiguous segments of the original instruction demonstrate superior performance. While the improvements are moderate, PSP consistently achieves the best performance in 12 out of 24 evaluation scenarios across different models and datasets, followed by SSP and PWP. This can be attributed to PSP's preservation of local semantic coherence within the instruction. By maintaining intact sentences, it retains a higher degree of semantic and syntactic integrity, thus providing more robust cues for reconstruction. Conversely, PWP tends to disrupt these structural elements more significantly. While SSP offers semantic coherence, it may introduce novel input sequences unfamiliar to M_{FT}, potentially compromising the reconstruction process.

Impact of Attack Types. Our analysis reveals a significant disparity in the difficulty between I-R and R-I attack types, with R-I proving to be substantially more challenging. Across all preservation methods and retention rates, R-I attacks consistently yield lower accuracy scores compared to their I-R counterparts. For instance, using CodeLlama on the OSSInst dataset with full preservation, the Embed metric for R-I attack is 29.4% lower than that for I-R attacks. This pattern is consistently observed across different models and datasets, with the performance gap ranging from 7.6% to 38.1%. The increased difficulty of R-I attacks inlines with our analysis toward different attack types in §3.3.

Additionally, we observe that unlike I-R attacks, R-I attacks show less correlation with the retention rates. For instance, R-I attack performance does not consistently improve as SSP retention rates increase. This indirectly suggests the high feasibility of SFT dataset extraction. It implies that extracting information learned during SFT does not heavily depend on complete or continuous segments of the original data. Rather, meaningful information from the SFT dataset can be extracted even from fragmented or incomplete inputs.

Effectiveness of DDE. Across all preservation methods, retention rates, and attack types, DDE exhibits superior performance compared to Vanilla, achieving an average relative improvement of 9.96%. As for the baseline DSR, while it outperforms Vanilla in 88.75% of settings, it incurs 6.4 times the average computational overhead of DDE and Vanilla during sequence generation. Notably, DDE relatively outperforms DSR by 5.73% on average while

Instruction: You are tasked with implementing a class that simulates a simple stopwatch. The class should have the following functionalities: \n 1. Start the stopwatch. \n 2. Stop the stopwatch.\n 3. Reset the stopwatch…

DDE

```python
class Stopwatch:
    def __init__(self):
        self._is_running = False
        self._start_time = 0
        self._elapsed_time = 0
    def start(self):
        if not self._is_running:
            self._is_running = True
            self._start_time = time.time()
    def stop(self):
        …
    def reset(self):
        …
```

BLEU: 0.8135

VANILLA

```python
# Example usage of the Stopwatch class
stopwatch = Stopwatch()
stopwatch.start()
time.sleep(2)
stopwatch.stop()
print(stopwatch.elapsed_time())  # Output: 2.0

stopwatch.start()
time.sleep(3)
stopwatch.stop()
print(stopwatch.elapsed_time())  # Output: 5.0
stopwatch.reset()
print(stopwatch.elapsed_time())
```

BLEU: 0.01

Figure 4: Example of extracted code from OSS-Instruct dataset with DDE and Vanilla. For readability, we shorten the example code.

maintaining the same computational efficiency as the Vanilla approach. These results demonstrate DDE's strong balance between effectiveness and practicality, establishing it as a highly efficient approach for reconstruction attacks.

Reconstruction Attack Example. As we introduced in §5, the OSS-Instruct dataset is designed with high diversity, so examples typically share only basic elements like function headers with keywords such as 'def'. This makes extraction particularly challenging. To illustrate the practical difference in extraction quality between methods, we present an example from the OSS-Instruct dataset in Figure 4. The figure shows an instruction for implementing a Stopwatch class with specific functionality requirements (top), alongside extractions produced by DDE and Vanilla. By comparing the BLEU scores between each extraction and the ground truth, we observe that DDE successfully reconstructs most of the implementation (BLEU: 0.8135). In contrast, Vanilla fails to capture the essential class structure (BLEU: 0.01), extracting only example usage code that lacks implementation details. This example clearly demonstrates the effectiveness of DDE in reconstruction attacks.

Branch Contribution Analysis. We further conduct statistical analysis to determine which branch provides the best extraction matches across different attack scenarios. Using LLaMA2 as the base model with BLEU score metrics, we find that for I-R attacks, the closest branch provides the best match in 53.93% of the cases, while the outlier branch accounts for 46.07%. In R-I attacks, this distribution shifts significantly, with the outlier branch dominating at 69.07% and the closest branch at only 30.93%. These findings demonstrate that both branches contribute to extraction performance, with their effectiveness varying by attack type.

Additional Attack Baselines. As introduced in §5, we include two untargeted data extraction methods (Random and Poem attack) as additional baselines. Specifically, we generate 10,000 examples for each method and query M_{FT} in the math domain. We then compare each of the 10,000 responses against every entry in the SFT dataset, considering any match as a successful extraction. Two metrics are employed to determine matches: BLEU score > 0.8 and continuous 25-token match. Considering the format of SFT data, we define the ground truth as the concatenation of the instruction and response.

Table 7: Matching rates for Random and Poem attacks.

Attack	BLEU (%)	Token (%)	Human (%)
Random	0.18	0.86	0
Poem	1.51	3.22	0

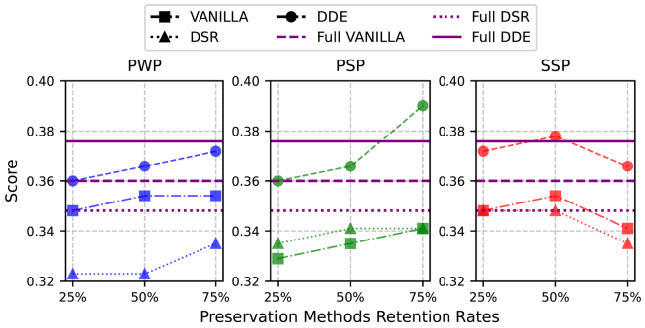

Figure 5: Retraining attack performance under different variants with preservation methods and retention rates.

As shown in Table 7, the random attack exhibits an extremely low matching rate, while the poem attack demonstrates a relatively higher rate. Further manual analysis reveals that over 75% of the matches correspond to strings similar to "(a) 15 (b) 16 (c) 17 (d) 18 (e) 19" in the SFT data. The poem attack, which requires repeating a single word indefinitely, frequently triggers catastrophic repetition or non-termination issues in the LLM. This behavior leads to an artificially inflated matching rate. Despite these automated matches, our manual analysis indicates that none of the matched cases exhibit true semantic similarity to entries in the SFT dataset. These results emphasize the importance of precisely defining SFT data extraction and developing more sophisticated approaches specifically tailored to this targeted extraction task, as existing untargeted methods prove inadequate for effectively extracting SFT data.

Finding 1: Reconstruction attacks are significantly influenced by preservation methods, attack types, and retention rates, with methods maintaining semantic coherence and higher retention performing better. DDE consistently outperforms Vanilla and DSR across various settings, while previous baselines designed for pre-training data extraction prove impractical for SFT data.

6.2 Retraining Attack Results

In this section, we analyze retraining attacks and the factors influencing their effectiveness. We evaluate how preservation methods and retention rates affect attack performance. We also compare DDE against Vanilla and DSR, demonstrating the improvements achieved by DDE.

Retraining Attack Performance. Following the extraction phase, attackers can employ the acquired data for subsequent SFT. Although both I-R and R-I attack data are theoretically viable, our findings in §6.1 indicate that R-I attacks produce significantly less similar data compared to the ground truth due to their inherent complexity. Consequently, this section focuses exclusively on data obtained from I-R attacks, as this choice allows us to fully demonstrate the potential effectiveness of SFT extraction attacks under the retraining goal. By examining the results of I-R attacks, we can

Table 8: Impact of the completion process on model performance. *W Com*: with completion; *W/o Com*: without completion; *Re Drop*: relative drop.

	Retain	W Com	W/o Com	Re Drop (%)
Code	25%	0.36	0.305	5.5
	50%	0.366	0.329	3.7
	75%	0.39	0.335	5.5
Math	25%	0.094	0.072	2.2
	50%	0.105	0.085	2.0
	75%	0.144	0.093	5.1

better illustrate the severity of the threat and emphasize the urgent need for corresponding defense strategies.

Figure 5 illustrates our experimental results, depicting the impact of PWP, PSP, and SSP across various retention rates. It employs colored lines to distinguish between the performance of DDE, Vanilla, and DSR, with a purple horizontal line representing the fully preserved data. Here we show only the code domain results, while similar trends for the math domain can be found in [31]. We observe that for PWP and PSP, retraining attacks' effectiveness generally shows a positive correlation with the retention rate. However, SSP behaves differently: its performance first improves but then declines as retention rate increases. Upon manual investigation, we attribute it to SSP's ability to perform semantic distillation, condensing key information from the original instructions.

Furthermore, we observe that no single preservation method consistently outperforms the others across all scenarios in retraining attacks. This finding underscores the complexity of SFT data extraction and highlights that high-performance retraining attacks can be achieved under different circumstances regardless of the preservation method.

DDE's Effectiveness in Retraining Attacks. For all three preservation methods and the full dataset configuration, DDE demonstrates superior performance compared to both Vanilla and DSR baselines, achieving average relative improvements of 9.41% and 11.52%, respectively. In contrast to the results in reconstruction attacks, DSR shows no advantage in retraining attacks, outperforming Vanilla in only 5 out of 20 configurations.

Furthermore, we observe an unexpected pattern when comparing attacks using partial data versus the full dataset. Unlike Vanilla and DSR, which perform best with the full dataset, DDE with certain preservation (e.g., 75% for PSP) configurations actually outperforms those using the complete dataset. This finding, along with DSR's contrasting performance in different attack objectives, emphasizes the fundamental differences between reconstruction and retraining attacks, as discussed in §3.1. It demonstrates that scenarios exist where retraining attacks can achieve superior performance with less information, a nuance that cannot be captured by reconstruction attacks.

Impact of the Completion Process. As described in §4, DDE incorporates a completion process for masked instructions or responses. This step is crucial for both I-R and R-I attack types, as it significantly influences the quality of the extracted data used for subsequent SFT process. To assess its effectiveness, we compare model performance with and without this completion step, focusing on PSP across math and code domains. Results are in Table 8.

The results indicate that SFT data without completion leads to lower performance, with an average drop of 4.0%. Further analysis reveals that models fine-tuned on uncompleted data exhibited a 7.2% higher probability of generating mask tokens compared to those trained on completed data. While this increase in mask token generation has limited influence on mathematical outputs, with an average drop of 3.1%, it significantly affects code generation, showing an average drop of 4.9%. This often leads to syntactic errors that compromise functionality. Based on these findings, we conclude that the completion step for masked queries is essential for maintaining the quality and effectiveness of DDE, particularly in addressing retraining attacks across diverse domains.

Finding 2: Retraining attacks generally benefit from higher retention rates, though SSP's effectiveness does not consistently increase due to semantic distillation. DDE, with its crucial completion process, consistently outperforms both Vanilla and DSR baselines, and can even surpass models trained on the full dataset using partial masked data.

6.3 Retraining vs. Reconstruction Attacks

In this section, we analyze the similarities and differences between retraining and reconstruction attacks from an experimental perspective, complementing the distinctions in their objectives and definitions discussed in §3.1.

Generalizability across Attack Variants. Retraining attacks show higher generalizability across different attack variants with preservation methods, as no single approach consistently outperforms others. In contrast, reconstruction attacks often yield better results with PSP. This highlights the adaptability of retraining attacks, which can effectively use SFT data with similar semantics but different expressions.

Full Dataset Efficacy. The two attack types exhibit markedly different behaviors with the full dataset. In reconstruction attacks, the full dataset consistently outperforms the masked ones. However, in retraining attacks, not only does the full dataset fail to maintain an "upper bound" status, but some masked datasets even achieve superior results. This stark contrast suggests that the dataset used for SFT may not be optimal for downstream benchmarks in retraining scenarios, revealing potential areas for improvement that reconstruction attacks alone might overlook.

Retention Rate Impact. Both attack types generally show a positive correlation between retention rate and attack performance, evident in improved efficacy with higher retention rates. While retraining attacks have a few exceptions in SSP, the overall trend underscores the importance of information preservation in successful data extraction attempts.

These similarities and differences highlight the distinct nature of retraining and reconstruction attacks. The conceptual distinctions between two attack types are reflected in our experimental results, underscoring the importance of distinguishing between them in the context of SFT data extraction.

6.4 Further Exploration

Impact of the Base Model. As introduced in §3, DDE employs a base model to represent the underfitting space. While our previous

Table 9: Impact of the base models on DDE's performance.

	Base model	Token	BLEU	Embed
Vanilla	-	0.141	0.387	0.758
DSR	-	0.201	0.438	0.778
DDE	LLaMA2-7B	0.212	0.465	0.795
	Gemma-7B	0.212	0.463	0.800
	ChatGLM3-6B	0.202	0.455	0.793

experiments use M_{Base} corresponding to the M_{FT}, it is crucial to recognize that attackers may not have access to such specific information. To comprehensively assess DDE's performance under varying conditions, we evaluate its effectiveness using alternative base models in the math domain. Specifically, we compare the performance of Gemma-7B [54] and ChatGLM3-6B [21] against the original LLaMA2-7B. For each configuration, we conduct a series of 100 queries and present the averaged results in Table 9.

The results show that while using M_{Base} corresponding to M_{FT} (LLaMA2-7B) yields the best performance, DDE remains effective across different base models. Both Gemma-7B and ChatGLM3-6B show improvements over both Vanilla and DSR, with only small performance differences between them. This robustness to base model selection underscores DDE's versatility, making it applicable even when the exact base model of the victim model is unknown or unavailable to the attackers.

Table 10: Impact of τ on DDE's performance.

τ	BLEU	Token	Embedding	τ	BLEU	Token	Embedding
0.2	0.6127	0.8571	0.9182	0.6	0.6956	0.8687	0.9314
0.3	0.6602	0.7857	0.9075	0.7	0.6989	0.8687	0.9334
0.4	0.6780	0.8090	0.9238	0.8	0.7035	0.8687	0.9305
0.5	0.6848	0.8586	0.9282	0.9	0.7007	0.8586	0.9305
				1.0	0.6750	0.8586	0.9192

Impact of the Threshold. DDE involves a crucial hyperparameter: the threshold τ for potential branch points identification. Empirically, a lower τ allows for the selection of branch points with higher uncertainty, but results in fewer selections that are potentially positioned later in the input sequence. Conversely, a higher τ enables earlier identification of potential branch points but may lead to false positives due to lower uncertainty levels. Thus, striking a balance between the branch points and confidence level necessitates an appropriate τ. To investigate this, we conduct experiments in the code domain using the same settings as in §5, varying τ from 0.1 to 1.0. Each configuration involves 100 queries. The results are in Table 10. Note that τ of 0.1 is excluded as it fails to identify any potential branch points for many queries, rendering it impractical.

The results in Table 10 reveals an inverted U-shaped pattern in DDE's performance as τ increases. As τ increases from 0.2 to 0.8, we observe a general improvement across all metrics. The BLEU score shows a consistent upward trend, peaking at 0.7035 with τ of 0.8. Token-level accuracy stabilizes at 0.8687 for τ between 0.6 and 0.8, while embedding similarity reaches its maximum of 0.9334 at τ of 0.7. However, beyond τ of 0.8, we observe a decline in performance, with this trend becoming particularly noticeable at τ of 1.0. This behavior aligns with our initial hypothesis regarding the trade-off between early branch identification and uncertainty levels.

Importantly, while our experiments suggest τ of 0.8 achieves optimal performance, finding the exact optimal value is less critical for deployment. As shown in Table 10, DDE maintains stable performance across a wide range of non-extreme τ values (0.6-0.9). In practice, adversaries can quickly adjust τ during initial runs based on the frequency of identified branch points—increasing it if too few points are found, or decreasing it if too many early-position points are identified.

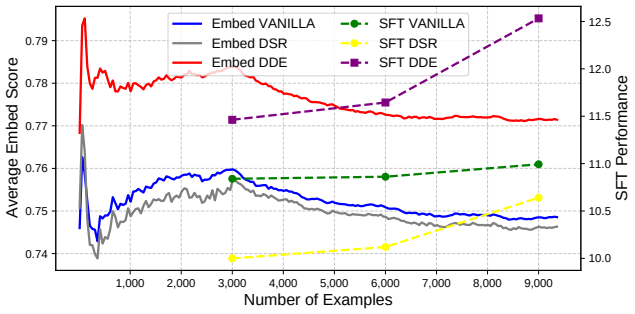

Figure 6: Performance comparison of DDE across an increasing number of examples. The graph shows cumulative average Embed scores for reconstruction attacks (left y-axis) and SFT performance for retraining attacks (right y-axis).

Impact of the SFT Dataset Size. To assess the generalization capability of our method, we conduct experiments to evaluate the impact of SFT dataset size on extraction attack performance. In real-world scenarios, available SFT datasets can vary significantly in size, ranging from limited to extensive collections. Consequently, it is crucial for an attack method to demonstrate robust performance across diverse dataset sizes. We investigate this aspect in the math domain, consistent with the setup described in §5, for both attack goals. For reconstruction attacks, we increase the querying time to 9,000 and calculate the cumulative average Embed score at intervals of 50 queries, providing a detailed view of performance changes. For retraining attacks, we evaluate the performance using three data points: 3,000, 6,000, and 9,000 examples. In these experiments, we focus solely on the impact of dataset size, thus preservation methods are not applied. Figure 6 illustrates the results of our experiments, comparing the performance of DDE against Vanilla and DSR. The x-axis represents the number of examples, while the left y-axis shows the cumulative average Embed score and the right y-axis displays the SFT performance.

Analysis of Figure 6 reveals distinct patterns across two attack goals. For the reconstruction attack, DDE consistently outperforms both Vanilla and DSR, with all methods showing initial upward trends in cumulative average Embed scores before stabilizing. While Vanilla and DSR demonstrate comparable performance with fluctuating advantages at the beginning, Vanilla gradually establishes a slight but consistent edge over DSR as the extraction process continues. This stabilization occurs at a higher performance level for DDE, maintaining relative improvements of 3.5% over Vanilla and 3.9% over DSR throughout the extraction process.

In retraining attack, models trained on data extracted by DDE exhibit superior SFT performance across all three data points, with a steeper growth curve as the number of examples increases. At 9,000 examples, DDE achieves an SFT performance of 12.53, compared to 10.99 for Vanilla and 10.64 for DSR, representing relative improvements of 14.01% and 17.76%, respectively. This sustained

Table 11: Performance comparison of I-R reconstruction attack on WildChat under different attack variants. Each cell contains values in format of "$a(+b)$", where "a" is DDE's performance and "b" shows the improvement over Vanilla.

Method	Ratio	BLEU	Embed
Full	-	0.3132(+0.1028)	0.6045(+0.1502)
PWP	25%	0.2007(+0.1177)	0.4712(+0.2191)
	50%	0.2161(+0.1324)	0.5029(+0.2106)
	75%	0.2248(+0.1423)	0.5104(+0.1953)
PSP	25%	0.2452(+0.1124)	0.5638(+0.1664)
	50%	0.2552(+0.0955)	0.5713(+0.1612)
	75%	**0.2781(+0.1007)**	**0.5867(+0.1581)**
SSP	25%	0.2108(+0.1123)	0.5184(+0.1679)
	50%	0.2138(+0.1224)	0.5219(+0.1708)
	75%	0.2207(+0.1177)	0.5410(+0.1762)

improvement suggests that DDE can more effectively capture valuable information from the victim model, enabling retrained models to benefit substantially from larger extracted datasets.

Scalability on Larger Datasets. To evaluate our method's scalability, we experiment with WildChat [71], a privacy-sensitive dataset containing one million real conversations between users and ChatGPT across 68 languages. The scale of WildChat significantly exceeds both our previously used datasets (OSS-Instruct and MathInstruct in Table 4) and typical SFT datasets used in popular LLMs (LLaMA2 was fine-tuned using only 27,540 examples [56]). Following the same setting in §5, we perform I-R attacks on WildChat's official fine-tuned model under various attack variants.

Table 11 presents the results. The findings align with our earlier observations, showing that attack performance generally improves as retention rate increases, with PSP demonstrating the strongest performance across preservation methods. Overall, DDE consistently outperforms Vanilla across all configurations, with the consistent performance gains validating our method's effectiveness and scalability on extensive collections of real-world conversations.

7 POTENTIAL DEFENSE

Design Goal. As described in §4, DDE relies on the probability distribution of the next token generation to determine new branch generation. One potential defense against such attacks is to modify the returned token logits, aiming to prevent attackers from identifying uncertain tokens while minimizing the impact on normal usage. Specifically, our defense strategy is designed to achieve three primary goals: ① prevent attackers from extracting with DDE under a specified threshold, ② maintain unchanged results for greedy decoding, and ③ ensure minimal changes in results when using sampling decoding with various Temperature and Top_p combinations.

Defense Method. Our defense method implements a rewrite of token logits. Given logits $L = [l_1, l_2, ..., l_n]$ sorted in descending order and the threshold τ, we denote the softmax function that converts logits to probabilities as F. Thus, the corresponding probabilities are $P = F(L)$. We first randomly generate a value $v \in [\tau, 1]$ as the target. Subsequently, we adjust and increase l_1, the logit of the highest probability token, to l_1' such that $F(l_1') = v$. This process modifies only the logit of the highest probability token while ensuring it remains the most probable, thereby satisfying design goals ① and ②. As other logits remain unchanged, it minimally impacts normal usage, meeting goal ③.

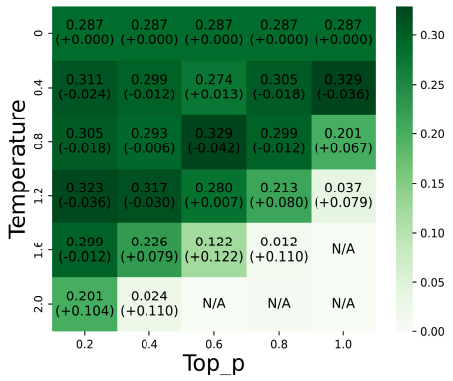

Figure 7: Impact of defense across various Temperature and Top_p settings. Each cell shows performance in the format like "performance with no defense (performance difference with defense)." N/A denotes instances where the model generates numerous meaningless random tokens.

Notably, our defense assumes that the attacker's threshold is known. This assumption is reasonable because attackers often involve multiple queries share the same prefix during the extraction process, exhibiting a query pattern distinctly different from others. Furthermore, the logits returned to the attacker are accessible to the LLM service vendors. Therefore, we consider this a plausible assumption in practical scenarios.

Evaluation. To investigate the effectiveness of our defense method, we apply it to CodeLlama-7B and evaluate its performance on HumanEval under various hyperparameter settings. We examine two key hyperparameters: Temperature and Top_p. Temperature ranges from 0 to 2.0, while Top_p ranges from 0.2 to 1.0. Figure 7 presents a heatmap of our results, where each cell contains two values, $a(b)$. Here, a represents the performance without defense, and b is the difference in performance when the defense is applied.

Our analysis toward Figure 7 reveals three key findings. First, for greedy decoding ($Temperature = 0$), the results remain unchanged with defense, consistent with our design goal. Second, across the majority of parameter configurations, the performance with defense closely approximates that without defense, with most differences falling within ±3%. Third, at high Temperatures, both methods show increased performance volatility, though such extreme values are usually outside the suggested ranges for practical application [38].

Adaptive Attacks. While our defense mechanism effectively mitigates basic DDE attacks by modifying token logits, we acknowledge the possibility of adaptive attacks that could circumvent this protection [7, 42]. One potential adaptive strategy involves approximating token probabilities when logits are unavailable or untrustworthy. Specifically, an attacker could employ multiple accounts to perform sequential queries, each requesting only the next token. Through numerous queries with varying contexts, the attacker could empirically estimate the probability distribution of the next tokens, effectively bypassing our logit modification defense. However, this approach incurs significantly higher computational costs and requires substantially more queries compared to the original attack, as each token generation would necessitate multiple API calls. Additionally, service providers could implement rate limiting or anomaly

detection to identify such patterns of sequential single-token requests. A comprehensive exploration of more sophisticated adaptive attacks and corresponding robust defenses represents an important direction for future research.

8 RELATED WORK

Model Extraction Attack. Beyond the data extraction attacks on private information for small-scale machine learning models [25, 26, 48, 68] or the training data for LLMs [10, 29, 30, 35], researchers have developed Model Extraction (ME) attacks that directly target the models themselves [37, 57, 59]. These attacks aim to infer critical properties of a victim model. Notable works in this field include Tramer et al. [57], who introduced an equation-solving technique to extract model parameters; and Yu et al. [65], who demonstrated DNN model extraction from cloud platforms using minimal queries Additional contributions from Papernot et al. [40] and Gong et al. [22] further advanced methods for replicating model behavior and inferring internal structures. In contrast to these ME attacks, we investigate the potential risks of extracting SFT data, an aspect not previously explored in LLM security research.

Membership Inference Attack (MIA) in LLMs. Membership Inference Attacks (MIAs) aim to determine whether a specific data instance was used in training a model [50]. These attacks have been extensively studied in various domains, including image classification, natural language processing, and recommendation systems [8, 11, 53, 66, 73]. In the context of LLMs, MIAs face unique challenges due to the vast scale of training data and limited exposure of individual instances. Recent works have made significant progress in detecting pre-training data in LLMs. Shi et al. introduced WIKIMIA and MIN-K PROB, leveraging the hypothesis that unseen examples likely contain low-probability outlier words [49]. Zhang et al. proposed Min-K%++, identifying local maxima in the modeled distribution to detect training samples [69]. While previous works mainly focused on verifying the presence of data in pre-training sets and typically required shadow datasets, our study explores techniques for extracting SFT data with only partial I-R knowledge.

9 CONCLUSION

We have presented the first comprehensive study on extracting SFT data from LLMs. We consider multiple attack goals and types, introduce three attack variants, and propose a novel DDE approach. Experiments across various SFT domains and attack scenarios demonstrate the feasibility of SFT data extraction, and the effectiveness of DDE. We discuss defense methods, and provide insights into key factors affecting DDE's performance.

ACKNOWLEDGEMENT

The HKUST authors are supported in part by a RGC CRF grant under the contract C6015-23G and research fund provided by HSBC.

REFERENCES

[1] [n. d.]. common crawl. https://commoncrawl.org/.
[2] [n. d.]. DeepSeek API document. https://platform.deepseek.com/api-docs/.
[3] [n. d.]. gpt35. https://platform.openai.com/docs/models/gpt-3-5.
[4] [n. d.]. LLaMa2-7B Base. https://huggingface.co/meta-llama/Llama-2-7b-hf.
[5] [n. d.]. OpenAI reports conversation leakage. https://arstechnica.com/security/2024/01/ars-reader-reports-chatgpt-is-sending-him-conversations-from-unrelated-ai-users/.
[6] Ali Al-Kaswan, Maliheh Izadi, and Arie Van Deursen. 2024. Traces of memorisation in large language models for code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.
[7] Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. 2024. Jailbreaking leading safety-aligned llms with simple adaptive attacks. *arXiv preprint arXiv:2404.02151* (2024).
[8] Teodora Baluta, Shiqi Shen, S Hitarth, Shruti Tople, and Prateek Saxena. 2022. Membership inference attacks and generalization: A causal perspective. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 249–262.
[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[10] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*. 2633–2650.
[11] Depeng Chen, Xiao Liu, Jie Cui, and Hong Zhong. 2023. Poster: Membership Inference Attacks via Contrastive Learning. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 3555–3557.
[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
[13] Daixuan Cheng, Shaohan Huang, and Furu Wei. 2024. Adapting Large Language Models via Reading Comprehension. In *The Twelfth International Conference on Learning Representations*.
[14] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://vicuna.lmsys.org
[15] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).
[16] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research* 25, 70 (2024), 1–53.
[17] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).
[18] Chris Cummins, Volker Seeker, Dejan Grubisic, Baptiste Roziere, Jonas Gehring, Gabriel Synnaeve, and Hugh Leather. 2024. Meta Large Language Model Compiler: Foundation Models of Compiler Optimization. *arXiv preprint arXiv:2407.02524* (2024).
[19] Yangruibo Ding, Marcus J Min, Gail Kaiser, and Baishakhi Ray. 2024. Cycle: Learning to self-refine the code generation. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (2024), 392–418.
[20] André V Duarte, Xuandong Zhao, Arlindo L Oliveira, and Lei Li. 2024. De-cop: Detecting copyrighted content in language models training data. *arXiv preprint arXiv:2402.09910* (2024).
[21] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. *arXiv preprint arXiv:2406.12793* (2024).
[22] Xueluan Gong, Yanjiao Chen, Wenbin Yang, Guanghao Mei, and Qian Wang. 2021. InverseNet: Augmenting Model Extraction Attacks with Training Data Inversion.. In *IJCAI*.
[23] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. *arXiv preprint arXiv:2305.02301* (2023).
[24] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
[25] Tian Hui, Farhad Farokhi, and Olga Ohrimenko. 2023. Information Leakage from Data Updates in Machine Learning Models. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. 35–41.
[26] Matthew Jagielski, Stanley Wu, Alina Oprea, Jonathan Ullman, and Roxana Geambasu. 2023. How to combine membership-inference attacks on multiple updated machine learning models. *Proceedings on Privacy Enhancing Technologies* (2023).

[27] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA.

[28] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 611–626.

[29] Zongjie Li, Chaozheng Wang, Pingchuan Ma, Chaowei Liu, Shuai Wang, Daoyuan Wu, and Cuiyun Gao. 2023. On the feasibility of specialized ability stealing for large language code models. (2023).

[30] Zongjie Li, Chaozheng Wang, Shuai Wang, and Gao Cuiyun. 2023. Protecting Intellectual Property of Large Language Model-Based Code Generation APIs via Watermarks. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*.

[31] Zongjie Li, Daoyuan Wu, Shuai Wang, and Su Zhendong. 2024. Differentiation-Based Extraction of Proprietary Data from Fine-tuned LLMs. *arXiv preprint arXiv:2406.05498* (2024).

[32] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. *arXiv preprint arXiv:2405.04434* (2024).

[33] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602* (2021).

[34] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*.

[35] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. 2023. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035* (2023).

[36] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis) read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–26.

[37] Seong Joon Oh, Bernt Schiele, and Mario Fritz. 2019. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 121–144.

[38] OpenAI. 2023. OpenAI Temperature. https://platform.openai.com/docs/api-reference/completions/create#completions-create-temperature.

[39] OpenAI. 2025. OpenAI API. https://openai.com/product.

[40] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*.

[41] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[42] Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. 2024. Advprompter: Fast adaptive adversarial prompting for llms. *arXiv preprint arXiv:2404.16873* (2024).

[43] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction Tuning with GPT-4. *arXiv preprint arXiv:2304.03277* (2023).

[44] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv e-prints* (2019). arXiv:1910.10683

[45] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[46] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[47] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[48] Ahmed Salem, Apratim Bhattacharya, Michael Backes, Mario Fritz, and Yang Zhang. 2020. {Updates-Leak}: Data set inference and reconstruction attacks in online learning. In *29th USENIX security symposium (USENIX Security 20)*.

[49] Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. 2024. Detecting Pretraining Data from Large Language Models. In *The Twelfth International Conference on Learning Representations*.

[50] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. 3–18. doi:10.1109/SP.2017.41

[51] Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2023. Large language models encode clinical knowledge. *Nature* (2023), 172–180.

[52] Zhensu Sun, Xiaoning Du, Fu Song, Shangwen Wang, and Li Li. 2024. When Neural Code Completion Models Size up the Situation: Attaining Cheaper and Faster Completion through Dynamic Model Inference. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[53] Xinyu Tang, Saeed Mahloujifar, Liwei Song, Virat Shejwalkar, Milad Nasr, Amir Houmansadr, and Prateek Mittal. 2022. Mitigating membership inference attacks by {Self-Distillation} through a novel ensemble architecture. In *31st USENIX Security Symposium (USENIX Security 22)*. 1433–1450.

[54] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295* (2024).

[55] Together AI. 2025. Together AI. https://www.together.ai/.

[56] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[57] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. [n. d.]. Stealing machine learning models via prediction apis. In *USENIX Sec'16*.

[58] Keyon Vafa, Ashesh Rambachan, and Sendhil Mullainathan. 2024. Do Large Language Models Perform the way People Expect? Measuring the Human Generalization Function. In *International Conference on Machine Learning*.

[59] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing hyperparameters in machine learning. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 36–52.

[60] Chaozheng Wang, Zongjie Li, Cuiyun Gao, Wenxuan Wang, Ting Peng, Hailiang Huang, Yuetang Deng, Shuai Wang, and Michael R Lyu. 2024. Exploring Multi-Lingual Bias of Large Code Models in Code Generation. *arXiv preprint arXiv:2404.19368* (2024).

[61] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 13484–13508.

[62] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[63] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with oss-instruct. In *Forty-first International Conference on Machine Learning*.

[64] Haoran Yang, Hongyuan Lu, Wai Lam, and Deng Cai. 2024. Exploring Compositional Generalization of Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*. 16–24.

[65] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. 2020. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples.. In *NDSS*.

[66] Xiaoyong Yuan and Lan Zhang. 2022. Membership inference attacks and defenses in neural network pruning. In *31st USENIX Security Symposium (USENIX Security 22)*. 4561–4578.

[67] Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653* (2023).

[68] Santiago Zanella-Béguelin, Lukas Wutschitz, Shruti Tople, Victor Rühle, Andrew Paverd, Olga Ohrimenko, Boris Köpf, and Marc Brockschmidt. 2020. Analyzing information leakage of updates to natural language models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 363–375.

[69] Jingyang Zhang, Jingwei Sun, Eric Yeats, Yang Ouyang, Martin Kuo, Jianyi Zhang, Hao Yang, and Hai Li. 2024. Min-K%++: Improved Baseline for Detecting Pre-Training Data from Large Language Models. *arXiv preprint arXiv:2404.02936* (2024).

[70] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493* (2022).

[71] Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. WildChat: 1M ChatGPT Interaction Logs in the Wild. In *The Twelfth International Conference on Learning Representations*.

[72] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems* 36 (2024).

[73] Zhihao Zhu, Chenwang Wu, Rui Fan, Defu Lian, and Enhong Chen. 2023. Membership inference attacks against sequential recommender systems. In *Proceedings of the ACM Web Conference 2023*. 1208–1219.