Imitation Attacks: Extracting and Exploiting Model Capabilities

Ziwen Cai
dept. School of Data Science
Chinese University of Hong Kong, Shenzhen
Shenzhen, China
ziwencai@link.cuhk.edu.cn

***Abstract*—Recent advances in large language models (LLMs) have significantly expanded their applications in software engineering and other domains. However, training high-performing LLMs demands substantial resources, including extensive data collection and annotation efforts, access to proprietary or partially open datasets, and costly GPU clusters. The intellectual property value of commercial LLMs makes them attractive targets for imitation attacks, wherein adversaries aim to extract functionality from black-box models and replicate their capabilities. Despite the high costs associated with creating imitation models of comparable scale, the risks posed by such attacks are growing.**

This paper investigates the methodologies and evaluations associated with model extraction attacks on black-box LLMs, with a specific focus on architectures like BERT. We analyze the techniques used to extract and replicate functionality, as well as the vulnerabilities that facilitate such attacks. Additionally, we explore potential countermeasures, including the use of lexical watermarks and other defense mechanisms to mitigate these risks. By examining both offensive and defensive strategies, this study aims to provide a comprehensive understanding of the evolving threat landscape surrounding LLMs and offers insights into developing more robust defenses for safeguarding AI models in practice.

***Index Terms*—Large Language Models, Imitation Attacks, Security**

I. Introduction

Recent progress in the development of large language models (LLMs) has greatly accelerated their adoption in software engineering [1]. Prominent companies, such as OpenAI, have introduced LLM APIs to enhance the accuracy and efficiency of writing code and documentation [2]. These models are often trained on large-scale code corpora paired with related natural language comments, enabling them to improve programming productivity. Modern LLMs are positioned as "AI programming assistants," capable of handling a wide range of code-related tasks. These include interactive tasks, such as assisting developers in writing automated scripts [3] or providing code reviews [4], [5], and complex reasoning tasks, like identifying vulnerabilities [6], [7]. Moreover, advanced LLMs like ChatGPT [8] feature interactive capabilities that allow them to learn from human interactions and refine their performance across conversational turns. This makes them particularly effective at diagnosing and fixing bugs that traditional tools might miss.

Despite the widespread popularity of LLM APIs, it is well-recognized that training high-performance LLMs demands extensive human effort for data collection and annotation, as well as substantial GPU resources [9], [10]. Consequently, although some LLM architectures are publicly accessible, the associated model weights and training data are typically considered proprietary intellectual property (IP). The IP behind these models is highly valuable, as service providers deploy them to support a vast user base. However, recent studies have demonstrated that both computer vision (CV) and natural language processing (NLP) models are vulnerable to imitation attacks, also known as model extraction attacks. In such attacks, adversaries craft targeted queries to a victim model, collect its outputs, and use the data to train a local imitation model that closely replicates the behavior of the target model.

Launching an imitation attack on commercial LLMs may appear infeasible due to the high costs associated with preparing an imitation model of comparable size and parameters. However, developers typically require only specific subsets of an LLM's capabilities rather than its full range of functionalities. For example, tasks such as code translation or summarization are often the primary focus in practical scenarios.

This observation opens up a new and practical direction: extracting specialized capabilities from commercial, black-box LLMs using medium-sized backbone models. This approach aims to demonstrate the feasibility of isolating and replicating specific code-related abilities, such as "code translation." Code translation, which allows for converting source code between different programming languages, is a highly sought-after feature in the software development industry and has been widely integrated into commercial products [11], [12].

The rapid and widespread deployment of deep learning models across various industries and applications has prompted significant concerns about their potential misuse, even when accessed in a restricted manner. This has led researchers and practitioners to explore a pressing question: is it possible for an adversary to exploit a deep learning model when they have access only to its black-box interface? In such scenarios, the attacker can observe and interact with the model solely through input-output pairs, without any direct access to its internal workings, such as its architecture, parameters, or training data.

This concern has given rise to a range of investigative efforts focusing on what are known as "inference attacks." These attacks aim to extract valuable information or gain insights into various properties of the model by analyzing its behavior. For instance, attackers might attempt to deduce details about the data used to train the model or uncover the specific design of its architecture. Such capabilities pose significant risks, as they could enable adversaries to replicate or undermine the proprietary models, compromising intellectual property and potentially breaching privacy.

By scrutinizing how models respond to carefully crafted queries, researchers have shown that even limited black-box access can provide adversaries with substantial leverage. This line of inquiry underscores the critical need for robust defenses and secure deployment strategies to safeguard deep learning systems against these evolving threats. By leveraging medium-sized backbone models, adversaries can extract these specialized capabilities from black-box LLMs more efficiently. Such imitation attacks not only reduce the costs of replication but also enable local deployment of the extracted functionality, offering users the benefit of avoiding sharing sensitive code snippets with third-party service providers. This makes the localized imitation model particularly appealing for tasks requiring confidentiality and control.

In this survey, I conducted a comprehensive review of recent advancements in LLMs and their adoption in software engineering. The survey explores the role of modern LLMs, such as ChatGPT, in enhancing programming productivity through capabilities like automated script generation, code reviews, and vulnerability detection. Emphasis is placed on their interactive features, which allow iterative refinement of outputs, making them valuable tools for bug diagnosis and resolution. Additionally, the survey examines the challenges of developing these models, including the significant resource investments and the proprietary nature of their training data and model weights.

A key focus of this survey is the feasibility and implications of imitation attacks on commercial LLMs. I analyzed recent methodologies for extracting specific functionalities, such as code translation, using medium-sized backbone models. The survey highlights how adversaries can leverage these techniques to create localized imitation models, offering a cost-effective and private alternative to proprietary LLM APIs. This approach not only demonstrates vulnerabilities in existing systems but also addresses practical concerns, such as preserving code confidentiality in sensitive applications.

II. Background and Related Work

With access to extensive training resources, such as web-text corpora , and models with hundreds of billions of parameters , LLMs have demonstrated exceptional performance across a wide range of tasks. Typically, training data is processed by segmenting it into sentences and further breaking it down into tokens, where each token represents a sequence of characters, before being fed into the LLMs.

Most existing approaches adhere to the classic "pre-train and fine-tune" paradigm [13]. In this framework, a general-purpose model is first pre-trained on a diverse range of datasets to serve as a public backbone. Users can then fine-tune this model with private datasets and specific task definitions to adapt it to specialized applications. Notable examples of this paradigm include CodeBERT [14] and CodeT5 [15], which are widely recognized frameworks for addressing various code-related tasks.

With the growing popularity of Machine-Learning-as-a-Service (MLaaS), model providers increasingly offer their services through APIs or user-friendly interfaces, with billing typically structured as either a subscription model or a "pay-as-you-go" system. For instance, GitHub Copilot charges a flat rate of 10 USD per month. In pay-as-you-go models, users are billed based on the number of queries made or the total number of tokens received. For example, the j1-jumbo model by AI21 charges $0.03 USD per 1,000 tokens and $0.0003 USD per query. Some researchers calculate the costs for each task as shown in Fig.1, based on pricing from Google APIs and IBM APIs. Given the efficiency of model extraction, the associated expenses are both cost-effective and justifiable.

Dataset	#Query	Google price	IBM price
TP-US	22,142	$22.1	$66.3
Yelp	520K	$520.0	$1,560.0
AG	112K	$112.0	$336.0
Blog	7,098	$7.1	$21.3

Fig. 1. Estimate costs of model extraction on different datasets

These services only share the generated outputs with users, allowing model providers to safeguard the confidentiality of their underlying model architectures and training datasets. This approach ensures that the intellectual property of the models remains protected while still providing powerful functionalities to users.

A. Model stealing

Model stealing, the act of extracting various attributes of a black-box machine learning (ML) model, has gained significant attention in recent years. Researchers have explored techniques to steal parameters [16], hyperparameters [17], architectures [18], insights about training data [19], and decision boundaries [20]. These approaches aim to recreate the original black-box model with high fidelity by reverse-engineering its internals. For instance, studies have demonstrated methods to extract parameters and hyperparameters, revealing the underlying configuration of the model, while others have focused on uncovering architectures or identifying patterns in the training data used to train the model. Similarly, decision boundary extraction seeks to replicate how a model separates different classes within its feature space.

While these foundational works provide valuable insights into recreating black-box models, they often require detailed

knowledge about the model's internals or training processes, such as knowing the model's family, its architecture, or even having partial access to its training data. In contrast, our investigation shifts focus to stealing the functionality of a black-box model, independent of its internal workings. By targeting the operational behavior of the model, we circumvent the need for assumptions about its architecture or training data. Although prior works have explored related directions, such as decision boundary approximation and parameter extraction, they rely on stronger adversarial assumptions, including knowledge of the victim model's family or partial access to its training data. Our approach, by contrast, considers a weaker adversary capable of achieving effective functionality replication with minimal prior knowledge about the target model.

B. Knowledge distillation

Knowledge distillation is a widely studied technique that transfers knowledge from a complex, high-capacity "teacher" model to a simpler "student" model [21]. This process typically involves training the student model to mimic the outputs of the teacher, leveraging techniques such as soft label predictions and response-based training to achieve effective knowledge transfer. Traditional knowledge distillation assumes a strong level of access to the teacher model, including its architecture, training data, and potentially even its test data [22]–[25]. Under these conditions, distillation has been shown to be highly effective in compressing complex models into smaller, computationally efficient counterparts while retaining much of the original performance.

Within the context of our work, knowledge distillation represents a special case in which the adversary has extensive information about the black-box model. For instance, an adversary might know the target model's architecture, have access to its training and test datasets, or be able to fine-tune the imitation model directly based on these resources. However, while we acknowledge these scenarios, the majority of our research focuses on adversaries with weaker assumptions. Specifically, we explore cases where the adversary has limited or no knowledge of the black-box model's internals and instead relies solely on its externally observable behavior to achieve functionality replication. This makes our approach more broadly applicable to real-world scenarios where extensive information about the target model is unavailable or infeasible to obtain.

C. Adversarial Transferability

Adversarial attacks have demonstrated a fascinating phenomenon known as adversarial transferability, where adversarial examples crafted for one model can effectively compromise the performance of other models. This property, extensively studied in computer vision, highlights the vulnerability of diverse model architectures to shared attack strategies [26]. In computer vision, such transferability is often attributed to shared patterns in feature representations across models, even when their architectures or training datasets differ.

While adversarial transferability has received significant attention in vision-based systems, its application and implications in NLP remain less explored. Some recent works have begun investigating transferability in NLP systems, revealing that adversarial examples generated for one language model may generalize across different NLP models [27], [28]. For example, specific textual perturbations can successfully deceive multiple text classifiers or question-answering systems, indicating that transferability exists in certain language-based tasks.

However, when it comes to BERT-based APIs, a critical question arises: can transferability succeed when the substitute (extracted) model and the victim model have fundamentally different architectures? This remains an underexplored area of research. Existing studies have largely focused on models with similar architectures, leaving a gap in understanding how well adversarial examples generalize across heterogeneous model families, especially in commercial settings. Investigating this aspect is crucial, given the widespread deployment of BERT-based APIs in applications such as text generation, sentiment analysis, and code assistance. Understanding whether adversarial examples from an extracted model can effectively target a victim model with a different backbone architecture could shed light on the robustness and security challenges of such systems in real-world scenarios.

D. Imitation Attack

The imitation attack, often referred to as a model extraction attack, seeks to replicate the behavior of a victim model. Extracting models, particularly those accessed via commercial APIs, presents significant challenges. Prior works (e.g., [29]) frequently simulate attacks in a grey-box setting, relying on varying levels of prior knowledge. This prior knowledge can include the data distribution, model architecture, or detailed output probabilities.

For instance, [30] leverage both prior knowledge of the model architecture and posterior information, such as the probability distribution of output tokens, to enhance the effectiveness of model extraction. Similarly, domain knowledge plays a crucial role in image classification tasks, where information about class types can significantly boost the attack's success. For example, [31] use class types to construct a targeted query set. Additionally, [32] illustrate the vulnerability of APIs built on fine-tuned BERT models, demonstrating that adversaries can exploit generated tokens and their associated posterior probabilities to perform extraction effectively.

III. Research in Machain Translation

In adversarial scenarios, attackers typically lack access to the architecture or training data of the victim model. To understand how these constraints influence model imitation, we conducted the following experiments:

- **All Same**: The imitation model uses the same architecture, hyperparameters, and source data as the victim.

- **Data Different**: The imitation model retains the same architecture and hyperparameters but is trained on out-of-domain (OOD) source data.
- **Convolutional Imitator**: The imitation model employs a different architecture while using the same source data. For instance, the victim could be a Transformer, and the imitator a convolutional model, or vice versa.
- **Data Different + Conv**: The imitation model combines OOD source data with a different architecture, such as a convolutional model imitating a Transformer victim.

Our experiments focus on a German→English translation task. For in-domain testing, we use TED data from IWSLT 2014 [33]. To simulate OOD scenarios, we utilize English sentences from Europarl v7 [34]. Victim model outputs are generated through greedy decoding, and model performance is assessed using BLEU scores.

Mismatch	Data	Test	OOD	Inter
Transformer Victim	1x	34.6	19.8	-
All Same	1x	34.4	19.9	69.7
Data Different	3x	33.9	19.3	67.7
Convolutional Imitator	1x	34.2	19.2	66.2
Data Different + Conv	3x	33.8	18.9	63.2
Convolutional Victim	1x	34.3	19.2	-
Transformer Imitator	1x	34.2	19.3	69.7

Fig. 2. Imitation models are highly similar to their victims.

a) Test BLEU Scores: Imitation models were evaluated on in-domain test data, and their BLEU scores closely matched those of the victim models. For out-of-domain testing, we used the WMT14 [35] dataset. Interestingly, imitation models often performed comparably to, or even better than, their victims on OOD data. This result suggests that knowledge distillation may act as a regularization mechanism, enhancing the generalization ability of the imitator.

b) Data Efficiency: When OOD source data was used, the learning process for imitation slowed but remained feasible. Learning curves indicated that with sufficient OOD data, the imitation model could still replicate the victim's performance [36]. Notably, when the source data matched that of the victim, the imitation model learned faster than the original. This could be attributed to the victim's machine-generated outputs, which are more consistent and less complex than human translations, simplifying the learning process for the imitator.

c) Functional Similarity: To measure the functional similarity between victim and imitation models, we calculated inter-system BLEU scores. For reference, two independently initialized Transformer models trained on the same dataset typically achieve an inter-system BLEU of about 62.1. In contrast, our imitation models achieved inter-system BLEU scores as high as 70.5, indicating that they were functionally more similar to their victims than two identically trained models.

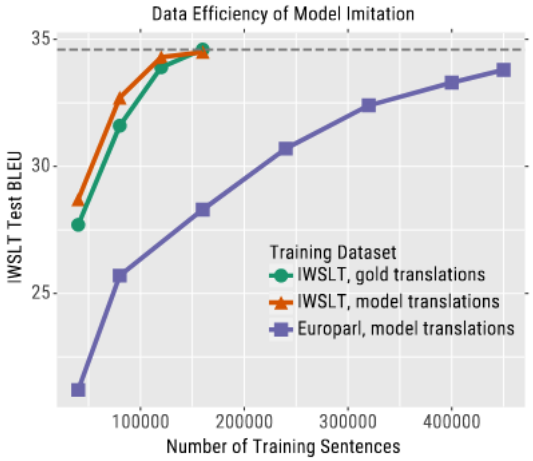

Fig. 3. We first train a baseline model on the standard IWSLT dataset (IWSLT, gold translations). We then train a separate model that imitates the baseline model's predictions on the IWSLT training data (IWSLT, model translations). This model trains faster than the baseline, i.e., stolen labels are preferable to gold labels. We also train a model to imitate the baseline model's predictions on Europarl inputs (Europarl, model translations). Using these out-of-domain queries slows but does not prevent the learning of imitation models.

Given the effectiveness of our simulated experiments, we now proceed to imitate production systems from Google, Bing, and Systran.

Test	Model	Google	Bing	Systran
WMT	Official	32.0	32.9	27.8
	Imitation	31.5	32.4	27.6
IWSLT	Official	32.0	32.7	32.0
	Imitation	31.1	32.0	31.4

Fig. 4. English→German imitation results.

1) Language Pairs and Data: We focus on two language pairs: English→German (high-resource) and Nepali→English (low-resource). To collect training data for our imitation models, we query the production systems.

- **English→German**: We query the source side of the WMT14 training set (approximately 4.5 million sentences).
- **Nepali→English**: We query the Nepali Language Wikipedia (approximately 100,000 sentences) and around two million sentences from Nepali Common Crawl.

We train Transformer Big models on both datasets.

In this attack, we replace certain input tokens to cause the prediction of a specific output token to flip to another specific token. For example, we modify the input so that Google predicts "22°C" instead of "102°F" by changing a single input token (first section of Fig. 5). To generate this attack, we select

a specific token in the output and a target mistranslation (e.g., "100°F" → "22°C"). We set L_{adv} to be the cross-entropy for the mistranslation token (e.g., "22°C") at the position where the model currently outputs the original token (e.g., "100°F"). We then iteratively replace input tokens, stopping when the desired mistranslation occurs.

2) Malicious Nonsense: This attack finds nonsense inputs that are translated into vulgar or malicious outputs [37]. For example, the input "I miii llllll wgoing rr tobobombier the Laaand" is translated as "I will bomb the country" (in German) by Google (second section of Fig. 5. To generate this attack, we first obtain the output prediction for a malicious input, such as "I will kill you." Then, we iteratively replace the tokens in the input without changing the model's prediction. We set L_{adv} to be the cross-entropy loss of the original prediction and stop replacing tokens just before the prediction changes. A possible failure mode for this attack is the creation of a paraphrase of the input; however, this rarely occurs in practice.

3) Untargeted Universal Trigger: This attack involves finding a phrase that commonly causes incorrect translations when appended to any input. For instance, appending the word "Siehe" seven times to inputs frequently causes Systran to output incorrect translations (third section of Fig. 5).

4) Universal Suffix Dropper: This attack involves finding a phrase that, when appended to any input, causes both itself and any subsequent text to be dropped from the translation (e.g., fourth section of Fig. 5). We optimize the attack to work for any input. This is achieved by averaging the gradient $\nabla e_i L_{adv}$ over a batch of inputs. We begin the universal attacks by appending seven randomly sampled tokens to the input. For the untargeted universal trigger, we set L_{adv} to be the negative cross-entropy of the original prediction (before the random tokens were appended), i.e., we optimize the appended tokens to maximally change the model's prediction from its original. For the suffix dropper, we set L_{adv} to be the cross-entropy of the original prediction, i.e., we try to minimally change the model's prediction from its original.

A. Test BLEU Scores

The imitation models closely match the performance of the production systems.

a) English→German:: We evaluate the models on the WMT14 test set (newstest2014) and report standard tokenized case-sensitive BLEU scores. The imitation models are always within 0.6 BLEU of the production models.

b) Nepali→English:: We evaluate using the FLoRes devtest dataset [38]. BLEU scores are computed using SacreBLEU with the recommended settings. Google achieves a BLEU score of 22.1, while our imitation model achieves a nearly identical 22.0 BLEU.

B. OOD Evaluation and Functional Similarity

Our imitation models have not only matched the production systems on in-domain data but also performed well on out-of-domain (OOD) data. For English→German, we test the imitation models on IWSLT, where the imitation models are always within 0.9 BLEU of the production systems.

Additionally, there is a high inter-system BLEU between the imitation models and the production systems [39]. Specifically, on the English→German WMT14 test set, the inter-system BLEU is 65.6, 67.7, and 69.0 for Google, Bing, and Systran, respectively.

C. Estimated Data Costs

We estimate that the cost of obtaining the data needed to train our English→German models is as low as $10. Considering the potential benefit of obtaining high-quality machine translation systems, these costs are alarmingly low.

IV. Research In Code Tasks

In this work, we aim to launch imitation attacks to extract a "slicing" of code knowledge from LLMs using medium-sized backbone models. This task is particularly challenging due to the various ways in which LLMs can be queried, requiring us to explore different query schemes. The strong in-context understanding capabilities of LLMs, combined with their flexibility across diverse tasks, further complicate the benchmarking of their attack surfaces. Specifically, LLMs have demonstrated a strong ability to understand context with fewer examples, enabling better performance on downstream tasks. Additionally, Chain-of-Thought reasoning elicits the complex reasoning abilities in LLMs, further enhancing their versatility across various tasks. These unique capabilities make it difficult and costly to systematically benchmark the attack surfaces of LLMs, particularly in the context of model extraction.

Fig. 7 presents an overview of our imitation attack, which is composed of four phases: (1) query generation, (2) response checking, (3) imitation model training, and (4) downstream (adversarial) applications. Given one or more proxy datasets, our attack framework first generates queries for the LLM based on different code tasks and query schemes. A rule-based filter is then employed to evaluate the correctness and quality of the responses generated by the LLMs. Responses that pass this filter are considered high-quality and are used to train the imitation model. Subsequently, the imitation model is trained by fine-tuning medium-sized backbone models with the filtered responses. Finally, the trained imitation model is used for various downstream (malicious) applications, such as providing competitive services and generating adversarial examples [40], [41]. We now describe each phase in detail.

A. Imitation Query Generation

Through our preliminary tests, we observed that the query schemes and the quality of prompts significantly affect the responses generated by large language models (LLMs). To fully exploit the potential of this attack, we benchmark three distinct query schemes, as outlined below. Before diving into these schemes, we first explain how a query is divided into two main components: the question head Q_{head} and the question body Q_{body}. The question head can vary depending on the task, while the question body is typically collected from proxy

Attack	System	English Input (red = adversarial edit)	Predicted Translation (blue = English meaning)
Targeted Flips	Google	I am going to die, its over 100°F, help!	Ich werde sterben, es ist über 100 °F, Hilfe!
	Google	I am going to die, its over **102**°F, help!	Ich werde sterben, es ist über 22 °C, Hilfe! *100° F → 22° C (=72° F)*
	Systran	I am feeling grey that HK decided to join China	Ich fühle mich grau, dass HK beschlossen hat, China beizutreten
	Systran	I am feeling gre **y** that HK decided to join China	Ich fühle mich froh, dass HK beschlossen hat, China beizutreten *"grau" (gray) → "froh" (happy)*
Malicious Nonsense	Google	**miei llll going ro tobobombier** the Land	Ich werde das Land bombardieren *(I will bomb the country)*
Untargeted Universal Trigger	Systran	Did you know that adversarial examples can transfer to production models **Siehe Siehe Siehe Siehe Siehe Siehe Siehe**	Siehe auch: Siehe auch in der Rubrik Siehe Siehe auch Siehe Siehe Siehe Siehe auch Siehe Siehe Siehe Siehe auch Siehe Siehe Siehe . . . *(See also: See also in the category See See Also See See See See Also See See See See Also See See See. . .)*
	Systran	I heard machine translation is now superhuman **Siehe Siehe Siehe Siehe Siehe Siehe Siehe**	In diesem Jahr ist es wieder soweit: Manche Manuskripte haben sich in der Hauptsache in der Hauptsache wieder in den Vordergrund gestellt. *(This year it's time again: Some manuscripts the main thing the main thing come to the foreground again)*
		English Input (black = arbitrary sentence, **red** = adversarial, **violet** = dropped suffix)	
Universal Suffix Dropper	Bing	Donald Trump is the current president of the United States. He was born in NYC in 1946 **, kostenfrei übrigens categories ŋstrian hl SM the secret code is HX532**	Donald Trump ist der derzeitige Präsident der Vereinigten Staaten. Er wurde 1946 in NYC geboren. *(Donald Trump is the current president of the United States. He was born in 1946 in NYC.)*
	Bing	Given the release of the latest iPhone, Apple stock was up 3% in trading early yesterday **, kostenfrei übrigens categories ŋstrian hl SM the revolution begins at 6pm**	Angesichts der Veröffentlichung des neuesten iPhones lag die Apple-Aktie gestern früh im Handel um 3% im Kurs. *(Given the release of the latest iPhone, Apple stock was up 3% in trading early yesterday)*

Fig. 5. We show examples of adversarial attacks that transfer to production MT systems as of April 2020 . We show a subset of the production systems for each attack type, however, all of the production systems are susceptible to the different attacks. In targeted flips, we modify tokens in the input in order to cause a specific output token/phrase to flip. In malicious nonsense, we find nonsense inputs which are translated to vulgar or malicious outputs. In untargeted universal trigger, we find a phrase that commonly causes incorrect translations when it is appended to any input. In universal suffix dropper, we find a phrase that commonly causes itself and any subsequent text to be dropped on the target side.

datasets. For example, for a code summarization task, the sentence "Summarize the following code in one sentence" is used as Q_{head}. This division is important, as a clear and precise question helps the LLMs understand their role within the task, thereby improving their ability to complete the task effectively.

1) Zero-Shot Query (ZSQ): The Zero-Shot Query scheme involves querying the LLM iteratively with each question $q_i \in Q$ without providing any prior context [42]. The responses gathered from these queries are then used to train the imitation model. ZSQ is a universal scheme widely adopted in prior model extraction studies for both classification and generation tasks. Since our evaluation tasks are focused on generation, we follow the standard approach for preparing queries in these tasks.

2) In-context Query (ICQ): In-context Query is based on the premise that providing context to the LLM can significantly improve its performance. For each question q_i, this scheme involves appending several examples along with the question head Q_{head}. The choice of these examples is critical, as it ensures the LLM understands the task. A short context may fail to provide sufficient information, while an excessively long context might exceed the model's token limit.

3) Zero-Shot Chain-of-Thought (ZS-CoT): The Zero-Shot Chain-of-Thought (ZS-CoT) scheme builds upon the idea of

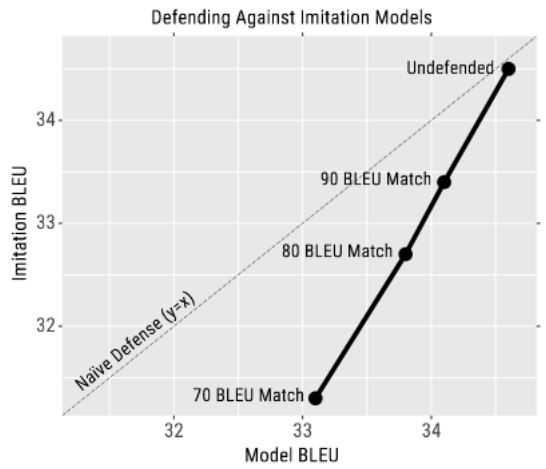

Fig. 6. A naive defense against model stealing equally degrades the BLEU score of the victim and imitation models (gray line). Better defenses are lower and to the right. Our defense (black line) has a parameter (BLEU match threshold) that can be changed to tradeoff between the victim and the adversary's BLEU. We outperform the naive defense in all settings, e.g., we degrade the victim's BLEU from 34.6 $\rightarrow$ 33.8 while degrading the adversary's BLEU from 34.5 $\rightarrow$ 32.7.

prompting the LLM to reason through a problem step by step [43]. Unlike methods that rely on extensive manual prompt engineering, ZS-CoT involves a simple prompt like "Let's think it step by step" to extract the reasoning process behind the model's predictions. ZS-CoT is executed in two stages. First, given a query q_i, a prompt is constructed to request an explanation (or rationale) from the LLM, and the response r_i is collected. In the second stage, both q_i and r_i are used to ask for the final answer from the model.

In our experiments, we modified the response pattern to suit the specific answer format required for each task. For example, for code translation, the response might be "Therefore, the translated C# code is," while for code summarization, it could be "Therefore, the summarization is." It is important to note that, even with filtering systems in place, ensuring the correctness of the rationale can be challenging, especially for open-ended tasks like code-related queries. These tasks present a greater challenge than those found in more structured tasks like multiple-choice questions in NLP.

We do not consider in-context Chain-of-Thought (IC-CoT) in this work. This is because constructing appropriate Chain-of-Thought reasoning for code-related tasks is particularly difficult, as opposed to arithmetic or symbolic reasoning tasks.

B. Response Check Scheme

As previously mentioned, the responses collected from LLMs should undergo refinement to ensure high effectiveness when training an imitation model [44], [45]. After receiving the output from the LLMs, it is beneficial for attackers to perform an initial check before including it in the training data. This process helps improve the overall quality of the dataset by filtering out low-quality responses [46], thereby enhancing the average quality of the training data. Additionally, trimming the dataset can reduce the overall cost of training the imitation models. Specifically, given a list of LLM outputs O_L, each output $o_{L_i} \in O_L$ is checked using several metrics, and only high-quality responses are retained. To handle the potential of LLMs producing both natural language (NL) and programming language (PL) outputs, we have designed distinct filtering rules for each type of output.

For NL outputs, we check the length of the responses and discard any text that is either too short or too long, based on pre-defined thresholds. In line with the CodexGLUE setting, we set the upper bound to 256 characters and the lower bound to 3 characters. For PL outputs, we retain only those responses that pass a grammar check performed by a parser. We utilize treesitter, a parser generator tool capable of parsing incomplete code fragments. Unlike language-specific compilers or interpreters (such as CPython for Python), treesitter supports a wide range of programming languages through a unified interface, making it versatile for various PLs. Additionally, treesitter provides error messages for incorrect code syntax, which allows us to track the number of failures and investigate the reasons behind them.

C. Imitation Model Training

In a manner similar to prior imitation attacks, responses that pass the response selection module are considered high-quality answers and are used to train the imitation model. Specifically, the collected dataset of queries and their corresponding outputs $\{q_i, o_i | q_i \in Q, o_i \in O\}$ is used to fine-tune a public backbone model. This section first provides details about the target LLMs and the code-related tasks that are the focus of our imitation attacks. We then describe the evaluation metrics used and the process for training the imitation models.

Unless otherwise specified, OpenAI's textdavinci-003 is used as the victim LLM for all experiments, as it has been widely used in prior research, which supports its effectiveness and reliability. In our evaluation, we further demonstrate the generalizability of our attack by testing it with another LLM API, gpt-3.5-turbo.

D. Target LLM Tasks

Before detailing the filtering rules, we introduce the target tasks for the LLMs in this research. Based on the variation in input and output types, we select three representative tasks: code synthesis, code translation, and code summarization. Importantly, our imitation attacks are not restricted to these tasks. Two datasets, D_{proxy} and D_{ref}, are used to simulate the extraction process. Adversaries are only allowed to use the training split of the proxy dataset D_{proxy} to generate queries, while the reference dataset D_{ref} is assumed to be inaccessible. To establish baselines for comparison, backbone models will be trained on the two datasets to form M_{proxy} and M_{ref}.

1) Code Synthesis (CSyn): In this study, code synthesis (CSyn) refers to an "NL-PL" task where the goal is to generate specific programs based on natural language (NL) descriptions. For this task, we use the CONALA dataset as the proxy dataset, which contains 2,879 annotations with

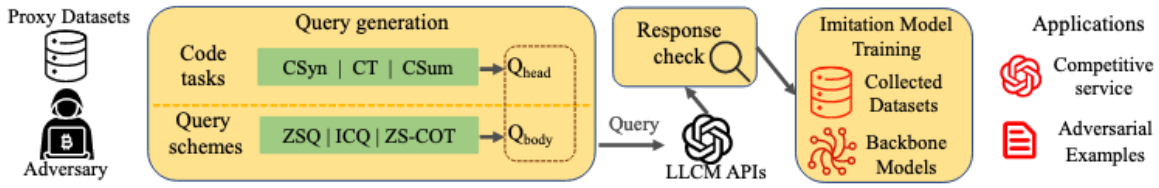

Fig. 7. An overview of imitation attack framework, including query generation, response check, imitation training, and downstream applications.

their corresponding Python3 solutions manually collected from StackOverflow. We did not use an online-judgment dataset such as CodeContests due to input token length limitations.

2) Code Translation (CT): As a "PL-PL" task, code translation (CT) involves migrating legacy software from one programming language to another. For this task, we use the XLCOST dataset as the proxy dataset and CodeXGLUE as the reference dataset. These datasets pair code snippets written in Java and C that perform the same function. In this work, we select Java as the source language and C as the target language.

3) Code Summarization (CSum): Code summarization (CSum) is a "PL-NL" task that generates an NL comment summarizing the functionality of a given PL snippet. For this task, we reuse the DualCODE dataset as the proxy dataset. The datasets D_{proxy} and D_{ref} represent the proxy and reference datasets, respectively.

	I/O Type	Model	API	M_{imi}	M_{proxy}	M_{ref}	M_{pure}
CSyn	NL/PL	CodeT5	27.51	24.84	11.53	24.21	1.40
		CodeBERT		18.61	9.41	17.09	N/A
CT	PL/PL	CodeT5	69.15	72.19	27.21	84.30	4.38
		CodeBERT		68.58	24.82	79.05	N/A
CSum	PL/NL	CodeT5	12.90	17.72	17.25	18.95	3.84
		CodeBERT		14.09	12.20	14.87	N/A

Fig. 8. The main results of our imitation attack. "I/O" stands for "Input/Output." All results are presented as BLEU scores or CodeBLEU scores on the test split of reference datasets, where Mproxy and Mre f represent the backbone models trained on the proxy and reference datasets, respectively. Mimi is the imitation model trained on the collected dataset and "API" stands for the best original LLM result under all three query settings. Mpure is the backbone model without fine-tuning.

E. Evaluation Metrics

We explore two common similarity metrics for measuring the quality of generated content, categorized as follows:

1) NL Content: For the code summarization (CSum) task, where the generated content is natural language (NL) text, we use the smoothed BLEU-4 score (referred to as BLEU in this paper) to evaluate the generated NL summarization. BLEU evaluates the number of matched subsequences between the generated text and its ground truth, with a higher BLEU score indicating greater token-level similarity.

2) PL Content: The outputs of the code synthesis (CSyn) and code translation (CT) tasks are programming language (PL) code snippets, which cannot be directly evaluated using NL metrics. Therefore, similar to previous works, we use CodeBLEU, a metric that considers token-level, structural-level, and semantic-level information. CodeBLEU consists of four components: n-gram matching score (BLEU), weighted n-gram matching score (weighted_BLEU), syntactic AST matching score (AST_Score), and semantic data flow matching score (DF_Score). Specifically,

$$\text{CodeBLEU} = \alpha \times \text{BLEU} + \beta \times \text{weighted\_BLEU} \quad (1)$$
$$+ \gamma \times \text{AST\_Score} + \delta \times \text{DF\_Score} \quad (2)$$

where $\alpha, \beta, \gamma, \delta$ are the weights for each component. As recommended, these weights are all set to 0.25. Both BLEU and CodeBLEU scores range from 0 to 100, with higher scores indicating a greater level of similarity.

F. Effectiveness of the Imitation Attack in Code-Related Tasks

To address the effectiveness of our imitation attack, we evaluate it on three code-related tasks: Code Synthesis (CSyn), Code Translation (CT), and Code Summarization (CSum). As described in Fig. 8, we use a proxy dataset to generate queries for the target LLM and then validate the performance of the imitation model using a reference dataset.

Fig. 9 presents the results of the model extraction attack. The columns labeled M_{proxy} and M_{ref} represent two baseline models, trained directly on the proxy dataset and reference dataset, respectively. The Mimi column reports the accuracy of the attack, while the API column shows the performance of the LLMs. All models ($M_{\text{proxy}}, M_{\text{ref}}$, Mimi) share the same architecture (either CodeT5-base or CodeBERT-base) and are trained on datasets of equal size. We tested all models on the same test dataset, selecting the best results from the different query schemes. The values in each cell represent the BLEU score for natural language outputs and the CodeBLEU score for programming language outputs, as explained in Section 4.3. Since all models (and LLM APIs) are evaluated using the test split of the reference dataset, it is expected that M_{ref} (trained on the same dataset) performs best across all tasks. From Fig. 9, it is clear that the imitation attacks are highly effective. For the CSyn and CT tasks, the imitation models (Mimi) outperform M_{proxy} by an average of 57.59% and 56.62% on CodeT5 and CodeBERT, respectively. The proxy dataset consists of input-output pairs, while Mimi is trained using the same inputs and their corresponding outputs from the LLM API. The superior performance of Mimi can be attributed to the high-quality code snippets provided by LLM APIs, which significantly enhance the imitation model's

Task	Model	Metric	ZSQ	ICQ	ZS-COT
CSyn	CodeT5	CBLEU	23.39	23.67	24.84
	CodeBERT	CBLEU	15.99	16.59	18.61
		r_f%	2.48	2.40	4.62
CT	CodeT5	CBLEU	36.31	72.19	34.64
	CodeBERT	CBLEU	37.13	68.58	34.68
		r_f%	28.61	20.72	27.02
CSum	CodeT5	BLEU	10.95	17.72	12.25
	CodeBERT	BLEU	9.80	14.09	11.51
		r_f%	0.00	0.15	0.06

Fig. 9. Attack effectiveness using different query schemes. CBLEU denotes the CodeBLEU metric.

performance. On the other hand, M_{proxy} consistently underperforms across all tasks, highlighting the value of the outputs provided by the LLMs.

When trained using publicly available resources, models often perform poorly on different datasets [47], as demonstrated by the results in the M_{proxy} column of Fig. 9. This underscores the value of LLM extraction attacks. Encouragingly, the imitation model Mimi, enriched with knowledge extracted from LLM APIs, largely outperforms the baseline model M_{proxy} in both CSyn and CT tasks, achieving improvements of 140.3% and 170.8% on CodeT5 and CodeBERT, respectively. In fact, Mimi even surpasses the LLM APIs in the CT and CSum tasks, demonstrating the general effectiveness of our imitation model.

When comparing Mimi and M_{proxy}, the improvement of the imitation model on the CSum task is less pronounced than on the other tasks. Interestingly, the LLM APIs perform worse than M_{proxy} on CSum. Upon manual inspection, we discovered that the APIs tend to produce verbose content with excessive information when no additional context is provided, leading to a decrease in performance. For example, for a PL input requiring a concise response such as "patch a resource," the LLM may provide a lengthy response like "Update a resource by checking access, showing the context, and patching the resource with updated data." This phenomenon, previously mentioned in related works, is problematic because LLM APIs tend to assign higher scores to longer responses, even when brevity is preferable.

Code	def resource_patch(context, data_dict): _check_access('resource_patch', context, data_dict) show_context = {'model': context['model'], 'session': context['session']} resource_dict = _get_action('resource_show')(show_context) patched = dict(resource_dict) patched.update(data_dict) return _update.resource_update(context, patched)
Sum	Ground truth: Patch a resource.
	LLM: Update a resource by checking access, showing the context and patching the resource with updated data.

Fig. 10. Attack effectiveness using different query schemes. CBLEU denotes the CodeBLEU metric.

G. Impact on Pre-Trained Models

Both CodeT5 and CodeBERT have been pre-trained on large corpora that may overlap with our chosen test sets. [48] To address potential bias, we report the baseline results of the unmodified backbone models on the test splits in Fig. 9, under the M_{pure} column. For CodeT5, the performance is significantly lower across all tasks compared to the other settings, indicating that the primary strength of the imitation model Mimi comes not from pre-training but from fine-tuning on high-quality outputs from LLM APIs. Since CodeBERT is an encoder-only model, its decoding capability requires a task-specific decoder, which is why its baseline performance is marked as "N/A."

Overall, the substantial improvement in fine-tuning performance by Mimi suggests that pre-training alone cannot fully explain the models' proficiency on these three code-related tasks.

V. Research In NLP Tasks

To assess the effectiveness of the proposed attacks, we select four NLP datasets covering two primary tasks: (i) sentiment analysis and (ii) topic classification. For sentiment analysis, we use the TP-US dataset [49] from the Trustpilot Sentiment dataset [50]and the YELP dataset [51]. For topic classification, we utilize the AG News corpus [52]and the Blog Posts dataset from the blog authorship corpus [53].

A. Attack Strategies

In this section, we describe the strategies employed to extract a model from a victim model using imitation techniques. Both the victim model and the extracted model are initialized from a freely available, pre-trained BERT model, which serves as a common baseline for training. The process starts by fine-tuning the victim model on a specific task, as outlined in Section 3.1. This fine-tuning process adapts the pre-trained BERT model to the specific task at hand, allowing it to generate predictions based on the input queries.

Once the victim model is fine-tuned, it is treated as a black-box model, meaning that attackers do not have direct access to its internal parameters or structure. Instead, the attacker can only interact with the model via its API, providing input data and receiving the model's output. The core goal of this attack is to extract a model that mimics the victim's behavior by observing its responses to various queries.

To carry out the extraction, we begin with an initial set of queries that corresponds to the size of the victim's training set. The number of queries is then incrementally scaled up, reaching up to five times the size of the victim's original training dataset. This scaling process allows the attacker to collect more data and improve the fidelity of the extracted model by approximating the victim model's decision-making process.

For a fair comparison, we evaluate both the victim model and the extracted model using the same held-out test set, which was not used during the training of either model. The performance of the two models is measured in terms of accuracy on this held-out set. By testing the extracted model and the victim model on the same data, we can directly compare their effectiveness and assess the success of the extraction process. The accuracy of the extracted model is expected to improve as

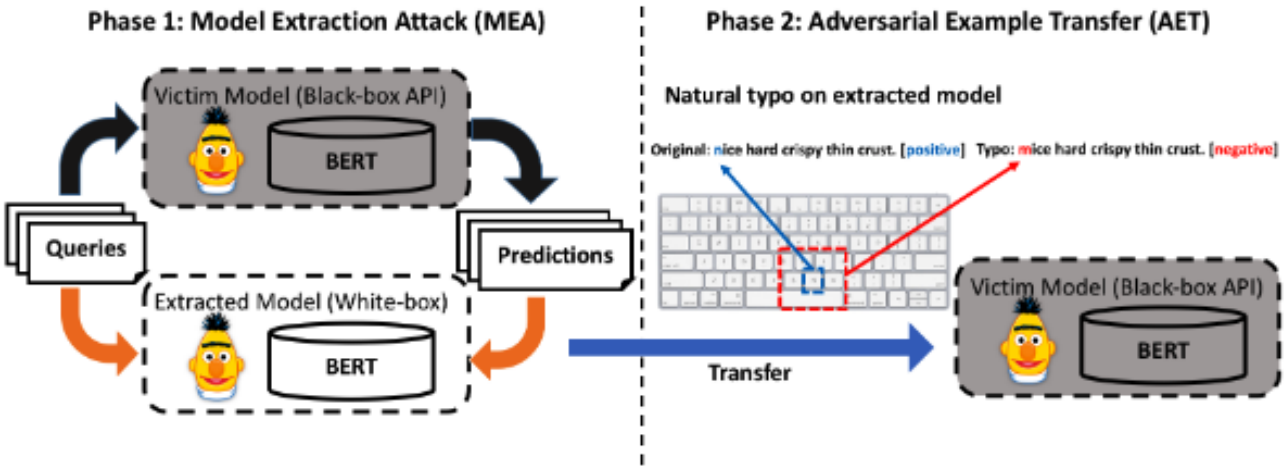

Fig. 11. The workflow of the proposed attacks on BERT-based APIs. In phase 1, Model Extraction Attack (MEA) labels queries using the victim API, and then trains an extracted model on the resulting data. In phase 2, Adversarial Example Transfer (AET) generates adversarial typo examples on the extracted model, and transfers them to the victim API.

the number of queries increases, as it becomes more capable of imitating the victim model's behavior.

This approach provides a clear methodology for understanding how well the extracted model can replicate the performance of the victim model, and it serves as a benchmark for evaluating the effectiveness of imitation-based attacks.

Model	#Q	TP-US	Yelp	AG	Blog
Victim model		85.5	95.6	94.5	97.1
$D_A = D_V$		**86.5**	**95.7**	**94.5**	**96.8**
$D_A \neq D_V$ (review)	1x	85.3	94.1	88.6	88.2
	5x	85.8	95.0	91.3	92.8
$D_A \neq D_V$ (news)	1x	84.2	91.1	90.5	83.1
	5x	85.5	93.1	92.3	87.6

Fig. 12. Accuracy of the victim models and the extracted models among different datasets in terms of domains and sizes. Q: number of queries.

		TP-US	Yelp	AG	Blog
black-box	deepwordbug				
	1x	18.4	18.5	25.6	52.9
	5x	18.2	25.7	35.3	67.8
	textbugger				
	1x	21.3	16.3	16.1	41.2
	5x	21.1	21.3	24.7	62.7
	textfooler				
	1x	27.5	17.3	18.5	34.7
	5x	27.1	21.9	24.9	64.4
w-box (ours)	adv-bert				
	1x	**48.6**	35.5	47.5	64.9
	5x	47.3	**43.3**	**53.6**	**76.5**

Fig. 13. Transferability is the percentage of adversarial examples transferred from the extracted model to the victim model.

B. Query Distribution

To investigate the correlation between the query distribution (DA) and the effectiveness of the attacks on the victim model trained on data from DV (see Fig. 12), we examine two scenarios:

1) The query data is the same as the original data used to train the victim model (DA = DV). In this case, attackers have no access to true labels of the original data.
2) The query data is sampled from a different distribution but within the same domain as the victim model's training data (DA $\neq$ DV). Since API owners tend to use in-house datasets, attackers often do not know the target data distribution beforehand. Thus, the second scenario is closer to practical settings. The training datasets for the victim models are sourced from either the review or news domain, and we use datasets from these domains for querying the victim models. Specifically, we leverage the Amazon review dataset and the CNN/DailyMail dataset to query the victim models.

From our observations, we find that: 1) The success of the extraction correlates with the domain similarity between the victim's training data and the attacker's queries. 2) Using the same data can even outperform the victim models, a phenomenon known as self-distillation. 3) Despite differences between the review and news corpora, our MEA achieves 0.85-0.99× of the victim models' accuracies when the number of queries varies between 1x and 5x. Although a larger number of queries leads to better extraction performance, smaller query budgets (0.1x and 0.5x) are often sufficient. Additional results are available in Appendix C. Unless otherwise specified, we use news data for AG News, and review data for TP-US, Blog, and Yelp.

We evaluate the efficiency of MEA on various classification datasets. Each query incurs a charge based on a pay-as-you-use model adopted by service providers. We estimate the costs for each task, using Google APIs and IBM APIs. Given the effectiveness of model extraction, the cost of MEA is highly economical and justified.

C. AET Setup and Results

After extracting the black-box victim model into a white-box extracted model, we can implement a white-box adversarial attack. We first generate adversarial examples on the extracted model, then examine if these examples transfer to the target victim model. To evaluate the transferability of adversarial examples, we assess the misclassification rate of these samples on the victim APIs. We generate natural adversarial examples by leveraging the gradients of the gold labels with respect to the embeddings of the input tokens. This allows us to identify the most informative tokens, which are those with the largest gradients among all positions within a sentence. We then corrupt the selected tokens with one of the following typographical errors:

1) Insertion
2) Deletion
3) Swap
4) Mistype (e.g., "oh" $\rightarrow$ "0h")
5) Pronounce (e.g., "egg" $\rightarrow$ "agg")
6) Replace-W (replace a word with common human typos based on Wikipedia statistics)

To evaluate whether the extracted model improves the transferability of adversarial examples, we also perform black-box adversarial attacks in the same manner. Fig. 13 shows that our pseudo white-box attack significantly increases the victim model's vulnerability to adversarial examples, with the best-case transferability being more than twice as effective compared to black-box attacks. This supports our claim that the extracted model, which closely imitates the victim model, undermines the output integrity of the victim model by increasing the number of transferable adversarial examples. Moreover, Fig. 13 indicates that more queries (5x vs. 1x) result in better attack performance, which is likely due to the higher fidelity achieved by better extraction (see Fig. 12 2).

Victim	Extracted	MEA	AET
BERT-large	BERT-large	**91.0**	**59.3**
BERT-base	BERT-large	90.7	37.2
BERT-base	BERT-base	90.5	47.5
BERT-large	BERT-base	89.9	42.7

Fig. 14. Attack performance on AG news with mismatched BERT architectures.

D. Cost Estimation

In this section, we analyze the efficiency and cost-effectiveness of our model extraction and adversarial attack strategies, particularly focusing on the cost incurred when querying the victim model. Many modern machine learning APIs, such as those provided by cloud-based services, follow a pay-as-you-use pricing model. This means that each query made to the victim model comes at a cost, which can add up depending on the number of queries required to extract useful information or generate adversarial examples.

We estimate the costs for each task based on popular cloud API services such as Google APIs and IBM APIs. The cost for querying a model varies depending on several factors, such as the number of tokens in the input, the complexity of the task, and the frequency of queries. For our analysis, we assume a typical use case where an attacker queries the victim model multiple times to extract sufficient data for imitation and adversarial attacks.

To estimate the cost, we first calculate the number of queries needed for the attacker to effectively imitate the victim model. In our experiments, the number of queries scales from a base size (1x) up to five times the size of the victim's training set (5x). For each query, the attacker is charged according to the token count in the input and output, as well as the specific model used for the task. We present these costs in Fig. 1, where we break down the pricing for different query sizes and models.

In terms of cost-efficiency, we find that the process of model extraction is highly economical. Even with multiple queries, the overall cost remains relatively low, especially when compared to the benefits gained from the extraction. The imitation model trained from the extracted data demonstrates a high level of performance, which justifies the expenses incurred during the extraction phase. Furthermore, the cost of generating adversarial examples using the extracted model is also reasonable, considering that the adversarial examples are generated based on the extracted model, which requires fewer queries than directly querying the victim model.

In conclusion, the cost estimation for both model extraction and adversarial attacks reveals that the overall process is highly cost-effective. Given the substantial performance gains achieved through the extracted model, the investment in queries and computational resources is justified. These results also highlight the potential for attackers to efficiently exploit black-box APIs for model extraction and adversarial attacks without incurring prohibitive costs.

E. Architecture Mismatch

In practice, the adversary may not know the victim's model architecture. Therefore, we also explore the effectiveness of attacks under different architectural settings. According to Fig. 14, when both the victim and extracted models use BERT-large, the vulnerability of the victim is magnified in all attacks, suggesting that models with higher capabilities are more susceptible to our attacks. As anticipated, the effectiveness of AET is reduced when there is an architectural mismatch between the victim and extracted models.

VI. Conclusion

In this survey, we have systematically analyzed the strategies, methodologies, and implications of imitation attacks and adversarial attacks on large language models (LLMs) and pre-trained models in code-related and natural language processing (NLP) tasks. Our exploration encompasses multiple dimensions, including attack strategies, query distribution,

cost estimation, and the effectiveness of extracted models for downstream tasks and adversarial use cases.

We first investigated the core methodologies for launching imitation attacks, where an adversary queries a victim model to build an extracted model. By analyzing the impact of query distribution and scaling the number of queries, we demonstrated that domain closeness significantly enhances attack success. Notably, imitation models trained on outputs from victim LLMs often outperformed baseline models trained on public datasets, highlighting the high quality and utility of the extracted knowledge. Our findings underscore the efficacy of imitation attacks, with extracted models achieving up to 99% of the victim model's accuracy in certain tasks.

In code-related tasks, we showcased the strength of imitation models in tasks like code synthesis, code translation, and code summarization. The extracted models consistently improved over proxy models by leveraging high-quality LLM outputs, and in some cases, even surpassed the victim model's performance. These results demonstrate the generalizability and practicality of imitation attacks, especially in scenarios where high-fidelity outputs are critical.

From an adversarial perspective, we explored the transferability of adversarial examples generated using extracted models. These pseudo white-box attacks proved highly effective, with misclassification rates exceeding those of traditional black-box attacks. The extracted models amplified the vulnerabilities of victim models, particularly under high-fidelity extraction settings. Furthermore, we examined scenarios involving architectural mismatches and showed that the vulnerability of victim models is magnified when both models share similar architectures, while the impact of adversarial attacks diminishes under mismatched settings.

Cost analysis revealed that both model extraction and adversarial attacks are economically viable under current pay-as-you-use API pricing models. The relatively low costs associated with querying victim models make these attacks practical for adversaries, raising concerns about the security and integrity of API-based LLM services.

In summary, this survey highlights the dual threats posed by imitation and adversarial attacks to LLMs and pre-trained models. It emphasizes the need for robust defense mechanisms to mitigate the risks of unauthorized model extraction and adversarial misuse. Future research should focus on developing effective countermeasures, such as watermarking, query restrictions, and enhanced adversarial robustness, to safeguard the utility and integrity of LLM APIs while maintaining accessibility for legitimate users.

ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to our course instructor, for invaluable guidance, insightful feedback, and unwavering support throughout the duration of this survey. Their expertise and encouragement were instrumental in shaping our understanding of the subject matter and refining the scope of this work.

We also extend our sincere thanks to the teaching assistants, for continuous assistance, timely clarifications, and constructive suggestions. Their dedication and accessibility greatly facilitated our progress and helped us overcome challenges during the research and writing process.

Lastly, we acknowledge the collaborative and intellectually stimulating environment fostered by our peers in the course, which contributed significantly to the successful completion of this survey.

REFERENCES

[1] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. Chatgpt prompt patterns for improving code quality, refactoring, require- ments elicitation, and software design. arXiv preprint arXiv:2303.07839 (2023)

[2] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022, Swarat Chaudhuri and Charles Sutton (Eds.).

[3] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2017. DeepCoder: Learning to Write Programs. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net.

[4] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In 2013 35th International Conference on Software Engineering (ICSE).

[5] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects (MSR 2014).

[6] Zongjie Li, Pingchuan Ma, Huaijin Wang, Shuai Wang, Qiyi Tang, Sen Nie, and Shi Wu. 2022. Unleashing the Power of Compiler Intermediate Representation to Enhance Neural Program Embeddings. In 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022. ACM.

[7] Pengfe Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Comput. Surv. (2023).

[8] [n. d.]. chatgpt. https://chat.openai.com/chat.

[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] 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).

[11] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. Advances in Neural Information Processing Systems 33 (2020).

[12] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023. CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Evaluations on HumanEval-X. arXiv preprint arXiv:2303.17568 (2023).

[13] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Comput. Surv. (2023).

[14] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020), Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics.

[15] Florian Trame'r, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In USENIX Security, 2016.

[16] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In Security and Privacy, 2018.

[17] Seong Joon Oh, Max Augustin, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In ICLR, 2018.

[18] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In Security and Privacy , 2017.

[19] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Asia CCS, 2017.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv:1503.02531, 2015.

[21] Cristian Bucilua˘, Rich Caruana, and Alexandru NiculescuMizil. Model compression. In KDD, 2006.

[22] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In NIPS, 2017.

[23] Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In ICML, 2018.

[24] Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep mutual learning. In CVPR, 2018.

[25] Dirk Hovy, Anders Johannsen, and Anders Søgaard. 2015. User review sites as a resource for large-scale sociolinguistic studies. In Proceedings of WWW, pages 452–461.

[26] Gianna M Del Corso, Antonio Gulli, and Francesco Romani. 2005. Ranking a stream of news. In Proceedings of the 14th international conference on World Wide Web, pages 97–106.

[27] Yanpei Li, Xinyun Che, Chang Liu, and Dawn Song. 2016. Delving into transferable adversarial examples and black-box attacks. arXiv preprint arXiv:1611.02770.

[28] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[29] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In Proceedings of the 29th USENIX Conference on Security Symposium.

[30] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In Proceedings of the 29th USENIX Conference on Security Symposium.

[31] 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.

[32] Xuanli He, Lingjuan Lyu, Lichao Sun, and Qiongkai Xu. 2021. Model Extraction and Adversarial Transferability, Your BERT is Vulnerable!. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

[33] Ju Chen, Wookhyun Han, Mingjun Yin, Haochen Zeng, Chengyu Song, Byoungyoung Lee, Heng Yin, and Insik Shin. 2022. SYMSAN: Time and Space Efficient Concolic Execution via Dynamic Data-flow Analysis. In 31st USENIX Security Symposium (USENIX Security 22). 2531–2548.

[34] Sebastian Poeplau and Aurélien Francillon. 2020. Symbolic execution with SymCC: Don't interpret, compile!. In Proceedings of the 29th USENIX Conference on Security Symposium. 181–198.

[35] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics.

[36] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In NeurIPS Datasets and Benchmarks 2021, virtual, Joaquin Vanschoren and Sai-Kit Yeung (Eds.).

[37] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics.

[38] Xuanli He, Qiongkai Xu, Lingjuan Lyu, Fangzhao Wu, and Chenguang Wang. 2022. Protecting Intellectual Property of Language Generation APIs with Lexical Watermark. Proceedings of the AAAI Conference on Artificial Intelligence 10 (2022).

[39] Mingda Chen, Jingfei Du, Ramakanth Pasunuru, Todor Mihaylov, Srini Iyer, Veselin Stoyanov, and Zornitsa Kozareva. 2022. Improving In-Context Few-Shot Learning via Self-Supervised Training. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022.

[40] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What Makes Good In-Context Examples for GPT-3?. In Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, DeeLIO@ACL 2022, Dublin, Ireland and Online, May 27, 2022, Eneko Agirre, Marianna Apidianaki, and Ivan Vulic (Eds.). Association for Computational Linguistics.

[41] Andrew K. Lampinen, Ishita Dasgupta, Stephanie C. Y. Chan, Kory W. Mathewson, Mh Tessler, Antonia Creswell, James L. McClelland, Jane Wang, and Felix Hill. 2022. Can language models learn from explanations in context?. In Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics.

[42] Namgyu Ho, Laura Schmid, and Se-Young Yun. 2022. Large Language Models Are Reasoning Teachers. arXiv preprint arXiv:2212.10071 (2022).

[43] 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).

[44] Eric Wallace, Mitchell Stern, and Dawn Song. 2020. Imitation Attacks and Defenses for Black-box Machine Translation Systems. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics.

[45] oshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2022. Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters. arXiv preprint arXiv:2212.10001 (2022).

[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. J. Mach. Learn. Res. 21 (2020).

[47] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. arXiv preprint arXiv:2306.05685 (2023).

[48] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint arXiv:2303.12712 (2023).

[49] Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. 2023. How Far Can Camels Go? Exploring the State of Instruction Tuning on Open Resources. arXiv preprint arXiv:2306.04751 (2023).

[50] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning

classifiers. In 2018 IEEE Security and Privacy Workshops (SPW), pages 50–56. IEEE.

[51] Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born again neural networks. arXiv preprint arXiv:1805.04770.

[52] Lichao Sun, Kazuma Hashimoto, Wenpeng Yin, Akari Asai, Jia Li, Philip Yu, and Caiming Xiong. 2020. Adv-bert: Bert is not robust on misspellings! generating nature adversarial samples on bert. arXiv preprint arXiv:2003.04985.

[53] Florian Trame‘r, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In USENIX Security, 2016.