KeyTAR: Practical Keystroke Timing Attacks and Input Reconstruction

Mufan Qiu*, Lihsuan Chuang*, Dohhyun Kim, Huaizhi Qu, Tianlong Chen, Andrew Kwong
University of North Carolina at Chapel Hill
{mufan, lchuang, dohhyun, huaizhiq, tianlong, andrew}@cs.unc.edu

***Abstract*—Keystroke timing attacks have long been recognized as a serious security concern. Researchers have conjectured that an attacker who learns the amount of time that elapses between keystrokes on a computer keyboard can reconstruct the keys pressed by a victim typist. Given the severe implications of a successful keystroke timing attack, numerous published side-channel works have utilized keystroke timing extraction as a case study to illustrate the impact of various types of side-channel attacks. However, despite an abundance of works demonstrating extraction of inter-keystroke timings, it remains to be proven that input recovery is actually possible.**

This paper bridges this long-standing gap in the literature and performs a comprehensive study on the feasibility of reconstructing typed input from inter-keystroke timings. We model input reconstruction as a machine translation task and fine-tune open-source Large Language Models (LLMs) with a curriculum learning strategy, leveraging their ability to utilize contextual information and incorporate semantic understanding into the reconstruction process. With this approach, we reconstruct typed input with a high degree of fidelity. Using the best reconstruction among the Top-5 predictions and a normalized edit distance threshold of 0.1 as the criterion for successful reconstruction, we achieve a success rate of 34.9%.

We also demonstrate input reconstruction under practical, real-world circumstances, where additional noise is introduced to the inter-keystroke timing traces. We conduct end-to-end cache attacks, both from native environments and from the Chrome browser, and quantify how the additional noise inherent to cache attacks affects the input recovery process. To obtain a sufficiently large dataset for training and fine-tuning the LLM for noisy traces extracted via cache-attacks, we replayed over 1.5 million typing samples from real human typists while performing cache attacks. We release and open-source this dataset, along with our code and checkpoint for reconstructing input, so that future works on keystroke-timing attacks can rigorously and empirically evaluate their effectiveness.

1. Introduction

Inter-keystroke timing intervals have long been theorized to leak sensitive information. Song et al.'s [1] seminal paper demonstrated that an attacker who can observe only the timing of when a user's keystrokes occurred can glean some limited information about what that user typed. They found that the underlying source of this information leakage stems from how the time elapsed between a pair of keystrokes, called the inter-keystroke timings, follows different distributions depending on the pair of keys. They hypothesized that these timing differences in inter-keystroke timings could potentially allow an adversary to fully reconstruct the victim's original input sequence.

Following studies examined this temporal side-channel and demonstrated inter-keystroke timings extraction through various physical side-channels [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Moreover, various works demonstrated remote software [18, 19, 20, 21, 22, 23, 24, 25, 26, 27] and micro-architectural [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50] side-channels capable of recovering inter-keystroke timings information without the requirement for any physical interaction between the victim and attacker.

Despite a multitude of academic papers utilizing the extraction of inter-keystroke timing to demonstrate the impact of their side-channel attacks, their suggestions that such leaked information leads to input reconstruction remain unsubstantiated, and the exact security implications of leaking inter-keystroke timings remain unclear. That is, while each of these studies demonstrates the extraction of inter-keystroke timings with their side-channel attack, no study hitherto has actually recovered a victim's input keystrokes from extracted inter-keystroke timings. Thus, it remains an open problem whether input reconstruction is possible, even from perfectly recovered inter-keystroke timings much less with additional noise inherent to side-channel attacks.

Without a rigorous treatment of the security implications of leaked inter-keystroke timings, system designers cannot accurately assess the risks of exposing inter-keystroke timings, thereby hindering attempts to conduct a security-performance tradeoff analysis. Furthermore, side-channel researchers cannot determine whether a proposed side-channel attack can truly extract inter-keystroke timings with sufficient fidelity for input reconstruction. Our work aims to address this gap by rigorously quantifying the security implications of leaking inter-keystroke timings. We guide our exploration by posing the following questions:

- *What are the security implications of leaked inter-keystroke timings? To what extent can attackers use them to reconstruct what the user was typing?*
- *What effect does additional noise, inherent to side-channel attacks, have on the reconstruction of the victim's input*

. *These authors contributed equally to this work.

keystrokes?

1.1. Contributions

In this paper, we provide answers to both questions, backed by large-scale empirical studies and end-to-end attacks. We demonstrate that leaked inter-keystroke timings are sufficient for reconstructing the victim's input to a far greater degree than previously shown. By leveraging the power of Large Language Models (LLMs), we can recover a considerable amount of the input, confirming the severity of keystroke timing attacks. We even showcase that our techniques can successfully reconstruct the input from inter-keystroke timings obtained from side-channel attacks with inherently noisy traces.

An LLM Driven Approach for Reconstruction. Prior work approached the task of input reconstruction from inter-keystroke timings through the use of statistical methods such as Hidden Markov Models and n-grams, or even deep learning methods such as recurrent neural networks (RNNs) or Long-term Short-term networks (LSTMs). In this work, however, we observe that LLMs offer superior capabilities for reconstruction due to their ability to consider long-distance dependencies between tokens and exploit the underlying context of the input (constrained by the grammar and semantics of the English language). We leverage this observation to model the problem of reconstructing input from inter-keystroke timings as a machine translation task, where the inter-keystroke timings form a special language that we aim to translate back into English.

By fine-tuning state of the art LLMs on an enormous dataset [51] (136 million sentence samples labeled with inter-keystroke timings), we can reconstruct 34.9% of sentences with near perfect fidelity (an edit-distance below 0.1) when considering the top-5 outputs from the model, with the performance dropping to 21.1% when considering only the top-1 result, concretely demonstrating for the first time that input reconstruction from inter-keystroke timings is indeed feasible and a legitimate concern.

However, the high fidelity achieved on clean data does not guarantee success in practical scenarios, where timing intervals are extracted via noisy side-channel attacks. To bridge this gap and adapt our model for real-world conditions, we introduce a **curriculum learning strategy**. This training paradigm involves a two-stage, easy-to-hard process: the model is first trained on clean, ground-truth timing intervals to learn the fundamental patterns of human typing, and subsequently fine-tuned on the noisier data extracted from our cache attacks. This structured approach proved essential for handling real-world data distortions. Our ablation study shows this strategy is significantly more effective than training on noisy data alone, with the curriculum-trained model being *over four times more accurate* than the cache-only model in the most challenging setting.

Input Reconstruction from Side-Channel Traces. We also examine the feasibility of input reconstruction under more practical, real-world scenarios where inter-keystroke timings are recovered with considerable noise present in the trace. We focus on the case study of cache side-channels, as they have been studied extensively in their relationship to keystroke-timing attacks and offer powerful capabilities, allowing attackers to extract inter-keystroke timings from any victim that visits an attacker-controlled web page.

The added noise inherent to side-channels, however, introduces major hurdles to carrying out input reconstruction compared to the idealized case. To overcome this additional noise, we fine-tuned our LLM on inter-keystroke timings obtained via a Prime+Probe cache attack, and achieved reconstruction of 17% of sentences with an edit-distance below 0.1. This demonstrates, for the first time, that inter-keystroke timings obtained from cache attacks in standard threat models (both from native environments and from JavaScript code running within a web browser) are sufficient for reconstructing the victim's input.

Cache-Traces Dataset. To train our LLM on cache traces, we required a massive dataset of inter-keystroke timings extracted via Prime+Probe attacks. Since no such public dataset existed, we developed a dataset collection system to generate our own. Our system replays typing samples from large existing public datasets and simultaneously conducts Prime+Probe attacks to produce traces for training the LLM. In our experiments, we replayed over 1.5 million sentences from over 168,000 volunteer typists [51], thereby accurately simulating a large-scale collection of cache-attack traces on real-world typing behavior.

Open Sourced Tools and Data. We provide both our tools for training LLMs on inter-keystroke timings and our tools for reconstructing input from inter-keystroke timings as open source code. This will enable side-channel researchers to train and evaluate our models against inter-keystroke timings traces obtained from various types of side-channels, which may exhibit different noise patterns. We envision that future studies demonstrating side-channels that extract inter-keystroke timings will use our tool to empirically demonstrate input reconstruction from their attacks, rather than merely speculating that the traces they obtain through their side-channels are sufficient. We also make public our dataset of inter-keystroke timings of 1.5 million sentences obtained through Prime+probe attacks so as to facilitate future research on algorithms for input reconstruction.

Summary of Contributions. In this paper, we make the following contributions:

- We empirically demonstrate, for the first time, complete reconstruction of a victim typist's input from inter-keystroke timings.
- We demonstrate the first reconstruction of user input from inter-keystroke timings obtained through cache-attacks, both from a native environment and from JavaScript code running within a browser in section 4.
- We build a dataset collection system that replays typing datasets while simultaneously obtaining side-channel traces. We utilize this system to generate a large-scale dataset of inter-keystroke timings extracted through Prime+Probe attacks. We make this dataset public to facilitate subsequent research on algorithms for input reconstruction.

- We open-source a tool for side-channel researchers to empirically evaluate the extent to which input can be reconstructed from their keystroke timing attacks. This tool provides a benchmark with which future side-channel attacks can be evaluated.

1.2. Related Work

In 2001, Song et al. [1] pioneered inter-keystroke timings extraction and inference attacks. They eavesdropped on SSH packets to infer inter-keystroke timings intervals, and from these intervals inferred keystrokes using timing deviations derived from the keyboard's spatial layout. The attack, however, used limited statistical methods for input reconstruction and only considered pairs of keystrokes individually. Thus, they lacked an analysis to recover texts fully.

Keystroke Reconstruction. Early efforts in keystroke reconstruction laid the groundwork for understanding the potential affects of leaking timing information. Monaco [52] introduced KREEP, a multi-stage pipeline employing techniques like DFA-based packet detection and a neural network (incorporating RNNs) to reconstruct search queries from encrypted network traffic by exploiting autocomplete features on a dataset of replayed human typing; however, this approach assumed the input consisted solely of alphabetic characters and spaces, and its reconstruction capabilities were primarily benchmarked against a random baseline, which might not fully capture the nuances of more sophisticated reconstruction challenges. Moreover, KREEP performed *classification* from a dictionary of potential queries, whereas we perform *input reconstruction*. The problem is much more difficult given the wider range of input combinations in the natural English language. Moreover, they also made use of a number of additional features, while we focus solely on inter-keystroke timings.

Similarly, González et al. [53] explored the classification of different texts chosen from a small, predefined dictionary of candidate texts. They utilized finite context modeling and SVM classifiers, which resulted in a notable decline in the model's performance without user-specific training data.

More recently, Weiss et al. [54] pioneered the use of fine-tuned T5 LLMs to infer AI assistant responses from a side channel that exposes the token lengths of the encrypted responses. While they demonstrated the value of inter-sentence context, their study did not explore the use of larger-scale LLMs, potentially limiting the full exploitation of the pre-trained knowledge inherent in such models, and they did not examine recovering input from inter-keystroke timings.

The broader landscape of inferring typed text from keystroke timings has seen various methodological approaches. Initial attacks, such as those targeting SSH by Song et al. [1], often employed Hidden Markov Models and Viterbi decoding for password recovery from packet inter-arrival times. Subsequent research explored classical machine learning classifiers, such as K-NN, for identifying URLs from JavaScript interrupt timings [28], and more advanced neural networks for reconstructing search queries from packet characteristics [52]. Beyond network traffic, keystroke timings have also been inferred from diverse side channels, including acoustics [9, 10], seismic activity [12], and hand motion [11]. Some research focuses on detecting specific keywords rather than reconstructing the full input [53]. More recently, deep learning applications on side-channel analysis have become prominent for pattern classification in noisy signals. Previous work uses generative AI to enhance training datasets for these models [55].

Inter-Keystroke Timing Extraction. Numerous studies have demonstrated using side-channels to extract inter-keystroke timings. These studies primarily focus on the mechanism of inter-keystroke timings extraction, rather than the problem of input reconstruction. Zhang and Wang [18] profiled binary executables for stack pointers patterns to extract inter-keystroke timings, while Jana and Shmatikov [19] presented an extraction attack using scheduling statistics. Diao et al. [35] showed a side-channel attack by reading the `/proc/interrupts` counters, and Lipp et al. [28] extracted keystrokes using the browser interrupt timing. However, these attacks have been mitigated by limiting access to lower-level statistics and the inherent noise from modern browsers. In our work, we utilize cache side channel attacks, which are currently still unmitigated, to extract keystrokes from both the native and browser environments.

Gruss et al. [29] conducted a keystroke timing attack by templating each cache line in `libgdk` with Flush+Reload. However, Flush+Reload inherently relies on shared memory and the cache flush instruction, which limits its applicability. Keydrown [30], on the other hand, presented many variants of interrupt timing and side-channel attacks for keystroke extraction. Specifically, Multi-Prime+Probe monitors multiple lines for keystroke signals to retrieve more reliable signals. We adapt the technique of Multi-Prime+Probe, as presented in Keydrown [30], to extract inter-keystroke timings.

1.3. Open Sourced Materials.

We open-sourced our artifact[1], including the simulation framework, reconstruction pipeline using fine-tuned LLM model, and the dataset of cache traces. We hope this resource will be a useful baseline for future research in developing attacks and robust defenses for keystroke inference attacks.

2. Background

In this section, we discuss relevant background to Large Language Models (LLM) and cache side-channels.

2.1. Large Language Models

Large language models (LLMs) are a class of massive-scale neural network models with billions of parameters [56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68]. They are trained on an enormous amount of texts and possess strong adaptation capability on diverse generation and understanding tasks [67, 68, 69, 70].

1. https://zenodo.org/records/17254163

LLM Architectures. The majority of LLMs adopt the Transformer architecture [71, 72], which allows LLMs to accurately capture the information correlation between words in sentences. The architecture of LLMs can be divided into the encoder-decoder architecture (e.g. T5 [73]) and decoder-only architecture (e.g. GPT [62, 63, 64, 65, 66], LLaMA [56, 57, 58] and OLMo [74]). The encoder-decoder architecture allows one word in the sentence to correlate with all the other words in the sentence. In contrast, the decoder-only architecture only allows a word to correlate with the words that appear before it in the sentence. While the former enables more complex information processing, the latter has proven to be more effective in natural language understanding and generation, and is the mainstream architecture for the most recent LLMs.

LLM Fine-Tuning. Fine-tuning refers to training an LLM on a task that was not covered in its training stages before [75, 76, 77]. Fine-tuning equips the LLM with the capability to handle new tasks, e.g., text classification [78, 79] and mathematical reasoning [75, 80, 81]. In this work, we fine-tune the LLMs so that they can understand inter-keystroke timings intervals and decipher the textual input.

2.2. Caches

A CPU's cache is a small bank of fast memory placed close to the processor to hide the long memory access latencies incurred with accessing main memory. To further improve performance, caches are divided into multiple layers with small but fast local caches located close to each core and a Last Level Cache (LLC) that is shared across all cores. Shared caches improve overall performance and minimize the memory footprint through memory sharing.

2.3. Cache Side-Channel Attacks

Cache side-channel attacks exploit the timing difference between memory accesses served from the cache and those from main memory. The attacker, based on the locality of the data and understanding of the victim's execution context, can infer a victim's memory access patterns by observing the cache state. Previous studies have used cache timing side-channels to break cryptographic implementations [29, 32, 36, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93], establish covert channels for transient execution attacks [94, 95, 96, 97, 98, 99, 100, 101, 102, 103], and monitor user activity [28, 29, 30]. In this paper, we focus on Flush+Reload [82] and Prime+Probe [83, 93].

Flush+Reload [82]. The Flush+Reload attack relies on shared memory between the attacker and victim, typically mapping a shared library or leveraging memory deduplication. To determine whether the victim has accessed a cache line, the attacker flushes the cache line from all cache levels using the `clflush` instruction, forcing the next access to fetch from the main memory. The attacker then waits for a fixed number of cycles before reaccessing the same memory address and measuring the access time. A fast access time implies that the victim accessed the target address, causing it to be brought back into the cache. Whileh this attack yields accurate access information at the cache line granularity, it requires shared memory, which is often limited in security-critical environments such as browsers and clouds.

Prime+Probe [83]. Prime+Probe lifts the shared memory requirement in Flush+Reload, making it suitable for both the native environment [83] and the browser [104]. In a general LLC Prime+Probe attack, the attacker first constructs a minimal eviction set [105] that occupies the target cache set. To "prime" the cache, the attacker fills the target cache set by repeatedly accessing the eviction set. After an optional idle period, the attacker "probes" the cache by re-accessing the eviction set and measuring each access time to detect evictions caused by the victim's activity. As noted by Purnal et al. [84], the idle period is not strictly necessary since the probe phase is both preserving (retaining cache state) and concurrent (detecting simultaneous accesses). In this paper, we adopted a non-windowed Prime+Probe for both the native environment and on the browser.

3. Large Language Models for Input Reconstruction from Time Intervals

Reconstructing textual input from a sequence of inter-keystroke timings is a fundamentally challenging task due to significant variations in typing habits and the inherent ambiguity of the raw numerical data. To address this, we model the problem as a *machine translation task*, leveraging the advanced contextual understanding of Large Language Models (LLMs). Unlike previous methods, LLMs can utilize their vast, pre-trained knowledge of grammar and semantics to disambiguate noisy timing signals, making them uniquely suited for this high-fidelity reconstruction task.

However, off-the-shelf LLMs are not inherently capable of solving this specialized problem and require significant adaptation. We therefore employ a two-stage fine-tuning process guided by a **curriculum learning strategy**. The model is first trained on clean, ground-truth timing intervals to learn the fundamental patterns of keystroke dynamics. Subsequently, it is fine-tuned on noisier data extracted from real-world cache side-channel attacks, a process that equips the model with the robustness needed for practical scenarios. Finally, to account for the model's stochastic nature, we generate multiple candidate outputs for each input and select the most confident prediction for our evaluation.

3.1. Problem Formulation

The primary objective of this research is to reconstruct the original character sequence, denoted as $C = [c_1, c_2, \dots, c_m]$, from a sequence of inter-keystroke timing intervals, $T = [t_1, t_2, \dots, t_{n-1}]$. Each interval t_i represents the elapsed time between the i-th and $(i+1)$-th key presses. It is important to note that in our setting, users may input control key sequences (e.g., Shift, Backspace, Delete), and thus the lengths of the timing sequence n and the character sequence m are not necessarily equal.

To illustrate the reconstruction challenge, consider the sentence "The monitor showed a ga_ in the data," where _ represents a missing character. An LLM can leverage grammatical context to eliminate syntactically implausible options. In this instance, both 'p' and 's' could form a grammatically correct sentence (i.e., "gap" or "gas"). The LLM can then further disambiguate by analyzing the intervals between keystrokes. For example, the keys in the pair ('a', 'p') are typically further apart on a standard QWERTY keyboard than 'a' and 's'. Consequently, their inter-keystroke intervals are likely to differ, allowing the model to make a more informed inference regarding the character the user typed.

This input reconstruction task is inherently challenging due to significant inter-individual variations in typing habits. For instance, some users may be experienced typists with consistently short inter-keystroke intervals, while less experienced typists may exhibit much longer and more variable intervals. The "136M Keystrokes" [51] dataset reveals substantial differences: the average inter-key interval (IKI) is 238.656 ms (SD = 111.6 ms), but for fast typists, the average IKI can be around 120 ms (SD = 11 ms), whereas slow typists can have IKIs exceeding 480 ms, sometimes reaching 900 ms (SD $>$ 120 ms). Beyond keyboard layout and individual proficiency, numerous other factors influence inter-keystroke intervals, including cognitive pauses for thought and external environmental distractions. These complexities significantly increase the difficulty of accurately reconstructing user input from timing data alone.

To address these challenges, we employ a combination of two primary strategies: ❶ Inference with Modern Large Language Models (LLMs): We leverage the capabilities of state-of-the-art language models. By fine-tuning a pre-trained LLM specifically for our task, we can capitalize on the common linguistic structures and patterns prevalent in the English language to reduce sentence entropy. Unlike previous works that relied on Markovian models or simpler recurrent architectures to exploit similar side-channel information, LLMs are more adept at this task due to their superior ability to consider long-distance dependencies between tokens and incorporate broader contextual understanding. ❷ Large-Scale, Diverse Datasets: our models are trained on extensive keystroke interval datasets, primarily derived from the "136M Keystrokes Dataset" [51]. This dataset encompasses a vast number of keystroke timings from a diverse user population, capturing a broad spectrum of typing habits and environmental factors. This comprehensive training data enables our model to learn a wide range of input patterns.

Furthermore, we demonstrate that our LLM-driven approach can also reconstruct input from inter-keystroke timings obtained through side-channel attacks, which introduce noise compared to the ground-truth IKIs. To help the model better understand the distributional shifts introduced by cache side-channel extraction, we have generated a new data set by replaying 1.5 million past test samples from [51] while extracting their respective inter-keystroke timings by conducting cache side-channel attacks.

3.2. Large Language Models Based Reconstruction

Motivation. Large language models have been trained on extensive corpora, enabling them to infer user input not only from keystroke intervals but also at the semantic level by leveraging contextual information [106, 107]. Compared to previous keylogging methods that utilized RNNs, LSTMs [108], or statistical models such as Support Vector Machine [53], LLMs offer a significant advantage due to their ability to incorporate semantic understanding into the reconstruction process. This makes them particularly suitable for tasks requiring nuanced predictions, such as reconstructing user input from noisy or incomplete data.

Model Architecture. We model the problem of reconstructing input from keystroke intervals as a machine translation task. Transformer architectures, which underpin most modern LLMs, are designed to process large-scale datasets effectively, making them ideal for leveraging the extensive keystroke interval datasets collected from typing websites and other sources [51]. By pre-training on these datasets, the model can uncover intricate relationships between keystroke interval patterns and user input, enhancing the generalizability of our attack.

Machine translation tasks involve converting one language into another, and we treat keystroke interval sequences as a unique language where numerical intervals represent "words." The task then becomes translating this "language" into English text. Deep learning has achieved remarkable success in machine translation, with large-scale pre-trained LLMs such as GPT-4 demonstrating state-of-the-art performance [65]. Consequently, we employ two popular open-sourced LLM architectures, T5 and LLaMA, to perform this translation task [58, 109].

Both types of LLMs can derive a hidden state from the input and include a language model head as the final output layer. The input keystroke intervals, after passing through the Transformer layers, produce a hidden state that is then fed into the final output layer to generate the reconstructed user input. The complete generation process is performed autoregressively, generating one token at a time and appending it to the input sequence.

Vocabulary Expansion. Previous research has indicated that directly prompting LLMs with numerical sequences can lead to inaccurate results [55]. Some studies have expanded the original token vocabulary of LLMs to fine-tune their weights, adding new tokens such as _5 and _9 to replace original numerical tokens, thereby avoiding the influence of the original semantics of these numerical tokens and achieving efficient training. However, in our tests, we found that expanding the vocabulary on large decoder-only models, such as LLaMA, did not yield better performance, possibly due to differences arising from the model size. Therefore, our final model does not use an expanded vocabulary.

Training. During the training phase, sequences of inter-keystroke time intervals are provided as input to the model, which subsequently generates corresponding character sequences as output. We employ a cross-entropy loss function to quantify the discrepancy between the model's predicted output and the actual user input, and we update the model's parameters iteratively using the backpropagation algorithm. Throughout this process, we utilize a batch gradient descent methodology to enhance training efficiency and stability. Furthermore, we adopt an instruction tuning approach to fine-tune the model. An illustrative example of such a training prompt is presented below:

LLM_A Training Prompt

User: Translate the Time intervals to Keystrokes.
Time intervals: 516 222 165 294 141 159 144 162 75 123 81 639 105 87 774 84 90 183 498 111 102 93 399 78 645 144 459
Assistant: Lynn, got to the office OK.

In addition to supervised fine-tuning, we also evaluate the model's performance under few-shot and zero-shot learning paradigms. The few-shot learning paradigm will prepend extra examples to the context, allowing the LLM to learn from previous conversations. The "136M Keystrokes Dataset" [51] contains instances where individual users have provided typing data for multiple separate sentences. This characteristic allows us to construct training and evaluation scenarios where a few example sentences (i.e., shots) from a specific user are provided as context. We then assess the model's ability to leverage these examples to infer user-specific typing habits and, consequently, generate more accurate reconstructions.

Inference. In the inference phase, given a sequence of inter-keystroke time intervals, we employ the beam search algorithm to generate multiple candidate reconstructions. Beam search is a heuristic search algorithm that explores the most promising paths in a limited breadth-first manner by maintaining a fixed number of top candidate sequences (the "beam") at each step of the generation process. This approach balances the thoroughness of an exhaustive search with computational feasibility by pruning less likely sequences early. The confidence of each generated candidate sentence is subsequently assessed using its negative log-likelihood, computed by the LLM, and then utilized to rank the candidates. We conducted a comparative analysis between the outputs generated via beam search and those produced by other prevalent sampling techniques. Our observations indicate that beam search typically yields a more diverse set of candidate reconstructions and exhibits a higher probability of including the correct reconstruction among its top candidates, particularly as the number of considered samples (i.e. number of returned sequences) increases.

Curriculum Learning. We introduce curriculum learning, a training paradigm inspired by human cognition, in which a model is sequentially trained on data of increasing difficulty[110, 111]. The approach improves model generalization and robustness across similar problems of varying difficulties. Our input reconstruction problem naturally lends itself to this strategy, as our datasets correspond to two distinct levels of difficulty. The initial, simpler curriculum utilizes ground-truth inter-keystroke intervals, enabling the model first to learn the fundamental patterns of human typing dynamics. The subsequent, more challenging curriculum consists of timing intervals extracted via cache side-channel attacks, which are subject to considerable noise and distortion inherent to the extraction process. By adopting this two-stage training strategy, we enable the model to build a strong foundational understanding of clean data before learning to handle the complexities of real-world, noisy traces, ultimately yielding a final model with superior performance in practical attack scenarios.

3.3. Evaluation

Dataset & Training. The "136M Keystrokes Dataset" [51] is a large-scale, publicly available resource for the scientific study of modern typing behaviors. It comprises over 136 million keystrokes collected from $168,960$ self-selected volunteers who participated in an online transcription typing test. For the study, participants were asked to input 15 sentences each. The dataset contains detailed keystroke-level data, including timestamps for key-down and key-up events, as well as participant-provided demographic information such as age, gender, and typing experience. This extensive collection of data includes detailed statistical analyses of keystroking patterns, performance metrics such as Words Per Minute (WPM), and nuanced behaviors, including rollover typing, providing a comprehensive resource for understanding contemporary typing performance.

We utilized 90% of the samples for training and partitioned the validation set into four types based on whether the user and sentence appeared in the training set. We utilized NVIDIA H100 GPUs to train the models for reconstructing user input. Each model takes approximately a day to train.

Metrics. We evaluate the fidelity of input reconstruction using two primary metrics derived from character-level normalized edit distance. This normalized edit distance, denoted δ, between a ground truth string S_{true} and a reconstructed string S_{rec}, is calculated as the Levenshtein distance between their uppercase versions, divided by the length of the uppercase ground truth string. Formally:

$$\delta(S_{true}, S_{rec}) = \frac{D_L(U(S_{true}), U(S_{rec}))}{|U(S_{true})|}$$

where $D_L(\cdot,\cdot)$ represents the Levenshtein distance, $U(\cdot)$ is a function that converts a string to its uppercase equivalent, and $|\cdot|$ denotes the length of the string. We deem a reconstruction successful if this normalized edit distance (δ) to the ground truth text is less than 0.1, a threshold adopted from [55]. Reconstruction Accuracy is the percentage of such successful reconstructions. Recognizing the stochastic nature of Large Language Model (LLM) outputs and the inherent difficulty of achieving optimal Top-1 reconstruction, our evaluation considers both the single highest-ranked prediction (Top-1) and the best-performing prediction within the top 5 candidates (Top-5). Specifically, we report: ❶ **Reconstruction Accuracy (Top-**1 **and Top-**5**)**: The percentage of reconstructions where the normalized edit distance (δ) to the ground truth is less than 0.1. This is reported for the Top-1 prediction and for the scenario where at least one of the Top-5 predictions meets this criterion. ❷ **Mean Edit Distance (MED) (Top-1 and Top-5)**: The average normalized edit distance. This is reported for the Top-1 prediction and for the best prediction (i.e., lowest δ) found within the Top-5 candidates. To further illustrate the reconstruction quality at different levels of normalized edit distance, we provide several representative examples in the range of 0.0 to 0.2. As shown below, these cases demonstrate both near-perfect and partially deviated reconstructions.

Representative Reconstruction Examples

Edit distance ≈ 0.06
Input: Hope that all is well in Denver.
Prediction: Hope that all is well in Denver

Edit distance ≈ 0.07
Input: Crestone won't have final measurement until this week.
Prediction: crestone won't have final measurement until next week.

Edit distance ≈ 0.08
Input: Taka has to be completed.
Prediction: Task has to be completed.

Edit distance ≈ 0.17
Input: Let Gary Smith know if you want him.
Prediction: Let Gary Smith know today if you want him.

Experiments. Our experiments are designed with two main objectives. First, we evaluated the ability of our model to reconstruct user input from inter-keystroke timings intervals. We initially assess using the keystroke intervals provided by the 136M dataset [51], focusing on the model's performance across sequences of varying lengths and different validation set partitions. Next, we evaluated the model's ability to reconstruct user input from inter-keystroke timings intervals extracted through cache attacks from both the native environment and the browser. Last, we analyzed the impact of different model training strategies.

Initial Validation Experiments. To validate our proposed LLM-based methodology, we conduct two initial experiments. First, we evaluate the performance of pre-trained LLMs without any task-specific adaptation to understand whether fine-tuning is required. Second, we assess our fine-tuned model on clean, ground-truth keystroke intervals to obtain the performance upper bound of the reconstruction. These initial results, presented in Table 1, serve as crucial benchmarks before we evaluate the model's robustness against noisy, side-channel-extracted data in later sections.

3.3.1. Performance of Pre-trained Models without Fine-tuning.

We first evaluate the capabilities of pre-trained large language models without any task-specific fine-tuning. As a qualitative case study, we prompted several models to translate a sequence of keystroke intervals into text. The summarized responses from a general-purpose model (OLMo 1B) and two inference-focused models (Qwen3 8B and DeepSeek R1) are presented in section A.

The performance of these models without fine-tuning is exceedingly poor. General-purpose models like OLMo 1B completely misunderstand the task's premise, attempting to establish arbitrary, irrelevant conversion rules. Meanwhile, more advanced inference-focused models like Qwen 3 8B and DeepSeek R1 correctly identify the nature of the problem—that the numbers represent time intervals between keystrokes—but are unable to proceed to a solution. They tend to engage in circular reasoning about the need for a non-existent mapping or context, sometimes exhausting the context window in the process. These qualitative failures clearly demonstrate that pre-trained models lack the specialized knowledge to translate timing patterns into text, underscoring the importance of fine-tuning.

3.3.2. Comparison to Hidden Markov Models.

To provide more perspective on the advantages of using LLMs over more traditional statistical methods, we also compare our result against Hidden Markov Models (HMMs), as they are what [1] used to classify passwords in their seminal study. Due to their inability to utilize contextual information in the way that LLMs can, the results are exceedingly poor and they completely fail to reconstruct the input, as shown in Table 1.

As the results clearly indicate, the HMM-based approach is entirely ineffective for this task, achieving a reconstruction accuracy of 0.00% across all validation splits. The mean edit distance is consistently close to 1.0, signifying that the generated outputs bear almost no resemblance to the ground truth sentences. A qualitative analysis of the HMM's output reveals a strong tendency to fall into repetitive patterns, generating nonsensical sequences such as `The the the the The t` or `17501750`. We found that this issue persists even when employing countermeasures like beam search or applying repetition penalties during inference. This complete failure underscores the severe limitations of traditional statistical models for a task that requires nuanced contextual and semantic understanding. It reinforces the necessity of our LLM-based approach, which can leverage its vast linguistic knowledge to overcome the ambiguities inherent in timing data.

3.3.3. Performance on Ground Truth Time Intervals.

We then evaluate the performance of our fine-tuned model on ground truth time intervals, and compare it against KREEP as a baseline. While there is no direct comparison to our work, as we are the first to perform *input reconstruction*, which is a more difficult task than what KREEP attempts, KREEP is the closest related work that attempts to match input to inter-keystroke timings in some capacity. Table 1 summarizes the results across various evaluation metrics, using a normalized edit distance of less than 0.1 as the criterion for successful reconstruction.

TABLE 1. Reconstruction Performance on Ground Truth Time Intervals

Model	Top-1 Recon. Acc. ↑	Top-5 Recon. Acc. ↑	Top-1 Mean Edit Dist. ↓	Top-5 Mean Edit Dist. ↓
HMM [1]	0.00%	0.00%	0.9837	0.9837
KREEP [52]	0.10%	0.20%	0.7008	0.6607
T5 (Ours) [73]	16.84%	33.53%	0.6677	0.4881
OLMo 1B (Ours)	**21.09%**	**34.92%**	**0.6454**	**0.4896**

Our fine-tuned OLMo 1B model demonstrates a substantial improvement over the KREEP baseline and outperforms the T5 model on nearly all key metrics. Notably, our model achieves a Top-1 reconstruction accuracy of 21.09% and a Top-5 accuracy of 34.92%. This high success rate, particularly in the Top-5 setting where nearly 35% of sentences are correctly reconstructed, concretely demonstrates the effectiveness of our approach. While the Top-5 mean edit distance of our model (0.4896) is comparable to that of T5 (0.4881), our model shows a clear advantage in Top-1 performance across both accuracy and mean edit distance. Furthermore, Figure 1 illustrates the Probability Density Function (PDF) of the normalized edit distance for the best samples generated by KREEP and our OLMo model. The distribution for our OLMo model is notably bimodal. It features a tall, sharp peak centered precisely at a normalized edit distance of zero, corresponding to near-perfect reconstructions. A second, broader peak is centered around an edit distance of approximately 0.75, representing typical unsuccessful attempts. This bimodal shape is highly advantageous for an attacker. The clear separation between the "success" peak at zero and the "failure" peak allows an attacker to confidently distinguish high-fidelity reconstructions from a majority of the incorrect ones, providing a clear signal of a successful attack. In practice, this confidence is derived from the observation that top candidates for a successful reconstruction exhibit high semantic consistency, whereas those for a failed attempt are semantically diverse. In contrast, the distribution for KREEP is unimodal and centered around 0.7, with virtually no density near zero. This lack of a distinct success peak means an attacker using KREEP has no reliable way to determine if a given reconstruction is accurate, and only a tiny percentage of sentences are recovered correctly.

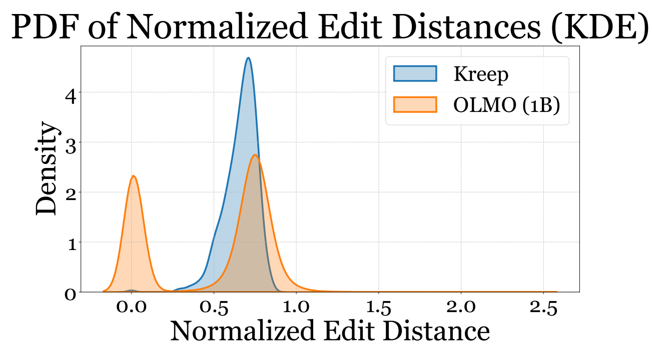

Figure 1. Probability Density Function (PDF) of the normalized edit distance for reconstructions from KREEP and our OLMo 1B model. Our model's distribution is heavily concentrated near zero, indicating frequent high-fidelity reconstructions, while KREEP's distribution is more spread out.

4. Keystroke Extraction Case Studies

In the previous section, we demonstrated that our model can make highly accurate inferences based on precise inter-keystroke timings. However, when it comes to extracting inter-keystroke timings through remote side-channels, some amount of noise is introduced into the measurements. In these scenarios, for the attack to be effective, the LLM needs to be able to reconstruct input from noisy side-channel traces with distorted intervals.

Over the past decades, researchers have shown that accurate keystroke timing can be extracted through a variety of side-channel attacks [1, 18, 28, 29, 30, 35, 38]. Among them, cache side channel attacks stand out as being particularly pernicious due to the ease and scale at which they can be conducted remotely via software. They are well-studied and widely deployed in research as foundational building blocks for many more sophisticated exploits [94, 95, 96, 97, 98, 99, 100, 112, 113]. Many proposals for novel cache side-channel attacks [29] use keystroke extraction as a canonical example to demonstrate the effectiveness of the attack; however, these attacks did not attempt to make inferences or reconstruct the input using their extractions.

In this section, we reproduce the Multi-Prime+Probe attack in Keydrown [30] and extend it to a weaker threat model in a sandbox browser environment. The browser threat model enables an attacker-controlled website to spy on keystrokes entered into any application on multiple victim devices simultaneously under limited sandboxed capabilities. We further extend these approaches to build a simulation system that synchronizes with a mass replay interface to collect and store side-channel traces at scale.

4.1. Test Setup

We conducted our tests on several Intel Core i7-2600 (Sandy Bridge) systems running Ubuntu 24.04 LTS (Noble) with kernel version 6.11.0-25-generic, BIOS version A24, and microcode update 0x2f. Following prior work [83, 93], hardware stream prefetchers are utilized to minimize noise and we enable transparent huge pages (THP) to accelerate the search for eviction sets. We also assume knowledge about the physical address of the function, as a plethora of studies [114, 115, 116, 117, 118, 119, 120] have demonstrated techniques to break Kernel Address Space Layout Randomization (KASLR). Using the physical address, we computed the set and slice index and performed a classical Prime+Probe attack [83, 93]. We later generalize the attack to account for architectures with unknown slice functions.

4.2. Identifying Potential Targets

We begin our investigation into generic keystroke timing attacks by identifying potential targets of interest for a cache attack. These "targets" are cache lines that correspond to the sections of code where the attacker wants to monitor. Two criteria for good targets are the activation frequency and the selectivity of the activation. That is, good targets are cache lines that are activated many times if and only if the keystroke handling routine is currently being executed. These targets are sparse across different user libraries and within the kernel, with structures ranging from functions and variables to memory buffers. Through monitoring accesses to these targets using Flush+Reload or Prime+Probe, we can infer the occurrence of keystrokes and thereby retrieve the inter-keystroke intervals.

4.3. Kernel Keystroke Handling Functions

Prior template attacks [29, 30] are limited to applications that both link and use the libraries for input handling. In addition, the detections are often limited to keystrokes entered into a targeted application. To implement a broader attack that captures all keystrokes entered on the device, the target must be invoked on every keystroke regardless of the input application. Therefore, we target the kernel's keystroke interrupt handler because all keystrokes are processed here first before getting forwarded to their destination application.

To identify kernel targets, we implemented a kernel module that functions simultaneously as a keylogger and as a Flush+Reload profiler for specific kernel addresses. We profiled interrupt handling functions from `/proc/kallsyms` and compared its cache traces against the ground truth. Out of all the functions we tested, we observed close correlation between keystroke times and activations of `kbd_keycode` and `kbd_event` and we will use them as our targets for the rest of this paper. The kernel module was only used for exploring effective targets, and is thus not required to perform the subsequent attacks.

4.4. Native Prime+Probe

Using the information from profiling the kernel and input-processing libraries, we implemented non-windowed Prime+Probe [84] to monitor the two kernel functions, `kbd_keycode` and `kbd_event`, to build generic attacks that can monitor any keystrokes entered to the same system.

The Attack. We first found eviction sets for all slices of our target using the group-testing algorithm [105]. To determine the specific slice the kernel function resides in, we perform Prime+Probe on all discovered eviction sets while generating keystroke events. The correct eviction set can then be identified by observing the timing trace with keystroke-aligned activity. We observe that the index of the kernel's eviction set remains consistent across multiple runs.

Noise Filtering. Prime+Probe is more susceptible to noise than Flush+Reload, as the latter operates at the granularity of cache lines, as opposed to cache sets. Thus, if we employ the same binary labeling scheme, marking all possible keystrokes as one and others as zero, we observe keystrokes at nearly every time unit, making the signal unreadable.

Therefore, we leverage two significant observations from [51] and propose an aggregation-based approach. First, we recognize that, despite Prime+Probe iterations having a lower temporal resolution than Flush+Reload, operating at a few hundred nanoseconds granularity, they are still significantly faster than human keystrokes, which are often issued at millisecond granularity [51]. This includes pressing the key, holding the key, releasing the key, and the inter-keystroke timings. Thus, we are still able to make at least hundreds of Prime+Probe measurements per keystroke while accounting for system latencies, context switches, etc. Second, we recognize that `kbd_keycode` was repeatedly called throughout the keyhold time, resulting in repeated cache hits observed during our Flush+Reload experiments. As a result, repeated measurements during the keyhold time should yield consistent hits, resulting in a spike during that period. Therefore, our aggregated approach creates a time series by computing the sum of cache hits at each millisecond. The resulting trace amplifies the signals of actual keystrokes from the background noise of the system. We then obtain the keystroke intervals through a two-stage filter consisting of a threshold filter and a frequency filter.

The threshold filter will first identify all milliseconds that exhibit a high activity count greater than the threshold. The threshold could be a static value based on the device's overall activity trace or dynamically determined for each trace. The dynamic threshold is obtained by selecting the top $N \times keys$ values in the aggregated counts, where N is a constant and $keys$ is the number of keystrokes within the given trace. The approach adjusts to sudden changes in the system activity and ensures that a "reasonable" number of candidates are selected for filtering for each trace. Although the dynamic threshold requires prior knowledge of the number of keystrokes in advance, it can be estimated if the user's previous typing data is available. In the later experiments, we use the adjusted dynamic threshold to demonstrate the theoretical upper bound of cache trace quality and inference results using LLMs. On the other hand, the frequency filter sets a minimum inter-keystroke time and groups activations within the range into a single keystroke, constructing our final observed keystroke intervals.

Figure 2 shows an excerpt of the static filtering of keystrokes from a test sample with a threshold of 40 hits per millisecond and a minimum inter-keystroke interval of 60 ms, since Dhakal et al. [51]'s dataset effectively contains no inter-keystroke timings below 90 ms.

Storage. A key challenge in our implementation lies in effectively storing the large volume of data generated by the Prime+Probe attack. Even monitoring for only a few seconds, the attack produces a new measurement every few thousand CPU cycles, quickly accumulating a substantial

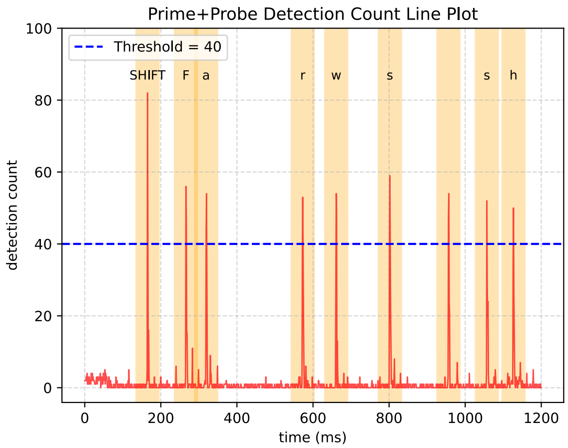

Figure 2. Keystroke Filtering from Aggregated Traces

amount of data. To address this, we attempted two methods: storing each bit and storing each hit timestamp. In the native environment, we store accurate timestamps from `rdtsc` into a binary file. This method is memory-efficient when measuring infrequent events such as keystrokes. In the browser, without access to reliable CPU frequency-level timers, we store each measurement in a bit of a large byte array and shift to utilize all available memory bandwidth. This compact encoding allows us to store and process long traces efficiently without sacrificing temporal resolution.

4.5. Browser Prime+Probe.

From Native Prime+Probe, we extend the attack to the browser, similar to Oren et al. [104]. We assume the attacker controls a malicious webpage and the victim opens this webpage in a browser tab. We do not assume the victim's activity or whether the victim interacts with the tab.

The browser tab automatically runs Javascript code that executes on the victim's machine within the browser's sandbox. From within the sandbox, the attacker conducts a Prime+Probe attack that extracts inter-keystroke timings from cache traces. The attacker also observes all the keystrokes entered on the device. This attack is more powerful than its native counterpart because it only requires a user to access the malicious webpage via a browser. In addition, the attack has the potential to continue monitoring keystroke timings even when it is running solely in the background.

Test Setup. Browser Prime+Probe is performed on identical setups outlined in Section 4.1. We performed our experiments on an unmodified Chrome browser version 136.0.7103.113, the newest version at the time of writing.

Challenges in the Browser. Implementing the attack in the browser introduces additional challenges due to limited visibility into the underlying memory layout. To overcome these challenges, we adapt techniques from previous work to implement a counting thread for high-resolution timers and construct reliable eviction sets in JavaScript via `SharedArrayBuffer` [105].

However, unlike native environments where congruent addresses can be obtained using hugepage-aligned memory mappings, modern browsers abstract the low-level memory allocation details. We conducted several experiments to understand how large shared memory buffers were allocated under different scenarios.

Understanding Normal Allocation Behavior. First, we investigate whether browser allocations of large buffers are page-aligned and whether the same memory layout is reused across runs in the same tab. We began by identifying an address in the SharedArrayBuffer and assumed it was aligned to a 4KB boundary. We then constructed eviction sets on native for all LLC sets that shares the same page offset and tried to evict the target by probing each set repeatedly. We observe that the target is always evicted by one of the sets we build, strongly suggesting that the allocation of large SharedArrayBuffer is in fact page-aligned. However, the victim's set varies between different runs. We hypothesize that the browser deallocates upon inactivity and reallocates on-demand, causing a change to the physical address and thereby the LLC set due to address translation.

Understanding Allocation Behavior with THP. Next, we evaluated how enabling THPs could affect allocation behavior. We first identify a target index. Assuming that `SharedArrayBuffer` is mapped contiguously on hugepages and remains page-aligned, we can extract the set index bits analogously to the native environment. Specifically, since the first element is mapped to offset 0x00000 and each element is 8 bytes, we can determine the set index for any position in the `SharedArrayBuffer`. Under this assumption, the same index in the `SharedArrayBuffer` should consistently map to the same set across reallocations, similarly to hugepages in the native environment. Our experiments showed that Chrome consistently made THP-aligned allocations that made the cache index consistent across runs.

Attack Implementation. Once the sets are consistent, we perform the same attack described in Section 4.4. We build our attack based off Spookjs [121] which provided a modularized implementation of group-testing Vila et al. [105]. We perform Prime+Probe on both kernel targets and profiled sets from the cache template attack, and all monitored sets show clearly observable keystroke spikes.

Results and Observations We observe consistent spike patterns across all sets, with some variance in activity levels. Interestingly, we extended the experiment to monitoring arbitrary sets and slices and found that we could clearly identify keystroke spikes from the overwhelming majority of LLC sets. We believe this observation stems from the aggregation that soothed out the noise and amplifies the additional activity during keystroke handling. It raises questions about the need to find specific targets for keystroke extraction and

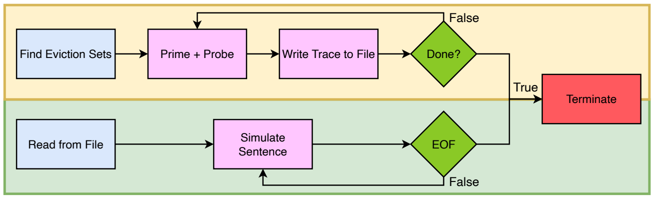

Figure 3. Simplified Keystroke Simulation System Workflow

introduces numerous possibilities to construct more generic and agnostic attack patterns in the future.

5. Keystroke Dataset Collection System

Although we demonstrated that inter-keystroke timings from cache traces could be used to identify keystrokes, due to the inherent noise from side channels, directly making inferences using the original model without fine-tuning on traces obtained via cache side-channels harms its performance. However, one of the major challenges of fine-tuning and training machine learning models is the need for a large amount of data for training, validation, and testing. For the trained model to generalize well to the entire population, the training data must include a diverse range of sentence structures while capturing variations in human typing patterns, such as speed, rhythm, errors, and correction behavior. Unfortunately, even though similar samples on actual typing intervals [51] have been collected, such datasets do not exist for cache side-channel traces. In addition, generating extensive and comprehensive data sets would require a large-scale field experiment similar to [51] that is extremely time-consuming to organize.

We bridged this gap by building a large-scale replay framework that can generate a massive number of realistic cache activity traces. It accomplishes this by replaying typing samples from existing datasets [51] while simultaneously conducting cache attacks to recover cache activity traces. In our experiments, we replayed over 1.5 million sentences typed by 168,000 volunteers [51] and obtained a comprehensive dataset of cache traces aligned with realistic typing behavior. Unlike previous studies [52], we do not limit our test to alpha-numeric keystrokes, but instead capture all characters used in actual scenarios, including special characters and functional keys (DELETE, BACKSPACE, SHIFT, CTRL, CAPS_LOCK, etc.).

To the best of our knowledge, this is the first effort towards a large scale, realistic cache side-channel trace collection.

5.1. System Workflow

Figure 3 shows a simplified version of the simulation system. The framework consists of two main components: the simulation process (light green), which replays keystrokes, and the measurement process (yellow), which executes the Prime+Probe attack and retrieves traces. Their executions can be divided into three phases: the preparation phase (blue), the simulation phase (pink), and the synchronization phase (dark green).

We abstracted much of the underlying synchronization logic for clarity in the diagram. For full implementation details, please refer to our released artifact.

Preparation Phase. In the preparation phase, the simulation process and the measurement process perform the necessary setup before starting the simulation. The simulation process processes the simulation parameters and reads from the typing trace file to obtain the characters and inter-keystroke intervals of the test sample. In parallel, the measuring thread would find eviction sets using techniques discussed in Section 4.4. Once both processes have completed their respective setups, we enter the simulation phase.

Simulating Process. In this phase, the simulation process replays keystrokes, while the measurement process performs Prime+Probe to record cache activity traces. Although the two processes operate "simultaneously", we ensure that the measuring process always starts before the simulating process to avoid missing any keystrokes. In each simulation phase, we simulate a single test sample at a time.

Synchronization Phase. After each simulation run, the two processes synchronize to determine whether to proceed with the next sentence or terminate. This phase uses a three-way handshake protocol to coordinate the transition.

5.2. Native Simulation

At the beginning of the measuring method, the measuring thread creates a shared memory space before it starts to find suitable eviction sets. The simulation process begins after the measuring thread and links to the same shared memory. The two processes communicated via the first five bytes in the memory. As the simulation terminates, the simulating thread unlinks itself from the shared memory and sends the done signal. The measuring thread then unmaps the shared memory.

5.3. Web Simulation

In the browser setup, the measuring process becomes a malicious tab hosted by the attacker. Unlike native simulation, the browser tab cannot establish a shared memory region with the simulating thread for synchronization with a lack of low-level controls in a sandboxed environment. Therefore, we introduce an intermidiary layer through the server. The server stores the synchronization variables as global state. The simulating thread and measuring thread will synchronize using the same protocol, but they will instead obtain or modify the states via Rest API calls.

An interesting observation in browser simulation is periods of inactivity in the browser, showing as "dips" in Figure 4. We believe this is likely caused by the browser's optimization or other resource-saving behaviors.

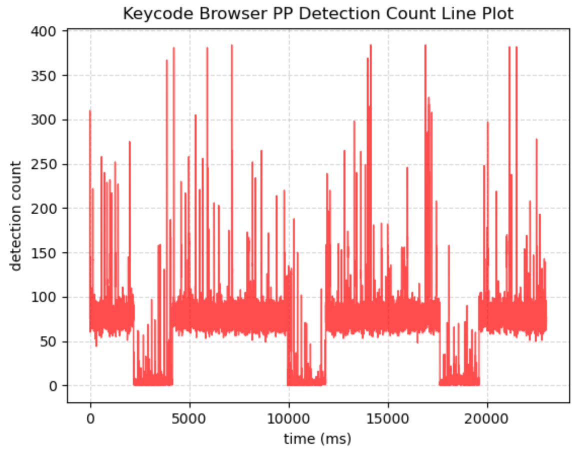

Figure 4. Browser Simulation Trace with Dips

5.4. Replay Results

We replayed all 1.5 million samples at 3x speed on the native environment for two weeks on four machines with identical setup. Our evaluation confirms that the simulation traces from both the native and web environments successfully capture all keystroke spikes. However, when applying various levels of filtering, we observe a trade-off between false positives and false negatives.

Following approaches detailed in Section 4.4, we apply a dynamic filter and a minimum inter-keystroke time of 60 ms. Since the LLM make inferences based on the obtained key intervals, our evaluation focuses on the quality of intervals we extract from the traces. To account for noise and detection errors that cause interval shifting, we evaluate our results using Dynamic Time Warping (DTW) distance, which aligns sequences under non-linear distortions. A lower DTW distance indicates a better alignment between the extracted and ground-truth intervals. Figure 5 shows a boxplot illustrating the distribution of samples for each context under each environment.

The plots show that filtered predictions from the web environment are consistently worse than those from the native environment, primarily due to increased noise. This is reflected in both higher average DTW distances and larger standard deviations. Upon closer inspection, we find that DTW distances vary widely between examples. Traces involving corrections, control keys, or keystroke rollovers [51], tend to perform worse due to key combinations that acts lower than our set minimum inter-keystroke interval. We also observe that a single misprediction can degrade the quality of subsequent interval extractions for an extended period of time, a propagation effect that further contributes to the variance observed in DTW distances.

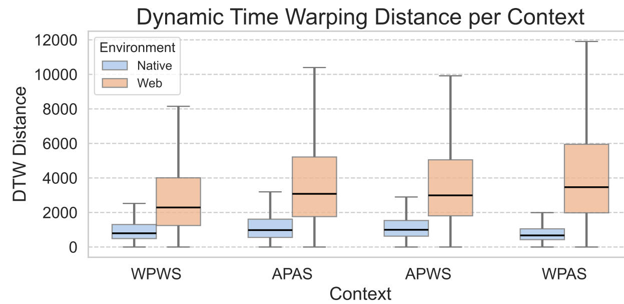

Figure 5. Native and Web Simulation Results.

5.5. Replay Fidelity

To assess the fidelity of our replay system relative to real typing behavior, we conducted an experiment where 45 typing samples were first recorded and then replayed at both 1× and 3× speeds, while their corresponding side-channel traces were simultaneously captured. Applying the dynamic filtering technique detailed in Section 4.4, we computed cross-correlation coefficients between the traces collected in real-time and the replayed traces, obtaining values of 0.9489, and 0.9000, respectively. These high correlations demonstrate that the simulated keystrokes exhibit temporal and structural characteristics highly consistent with real keystrokes.

6. Results on Side-Channel Extracted Timings

In Section 3, we established that our fine-tuned LLM is highly effective at reconstructing text from ground-truth keystroke timings. Having demonstrated the fundamental viability of our approach, we now evaluate its robustness and performance in a practical attack scenario. This section presents the reconstruction results when the model is applied to the noisy keystroke intervals obtained from our end-to-end cache side-channel attacks, as detailed in Section 4. This analysis is crucial for understanding the real-world feasibility of cache-attacks that extract inter-keystroke timings.

6.1. Reconstruction from Cache-Extracted Data

Native Cache Traces. We first evaluated using traces obtained through cache side-channels conducted from a native environment, as described in section 4.4. The following results were achieved using our full **curriculum learning** model, which was first fine-tuned on ground-truth intervals before being fine-tuned on the cache-extracted data. Table 2 presents the model's performance under four distinct evaluation settings: across-participant-across-sentence (APAS), across-participant-within-sentence (APWS), within-participant-across-sentence (WPAS), and within-participant-within-sentence (WPWS). "Across participant" signifies that keystroke samples from the test set subjects were not present in the training set, while "within

TABLE 2. RECONSTRUCTION PERFORMANCE OF THE CURRICULUM LEARNING MODEL ON CACHE-EXTRACTED TIME INTERVALS

Setting	Top-1 Recon. Acc. ↑	Top-5 Recon. Acc. ↑	Top-1 Mean Edit Dist. ↓	Top-5 Mean Edit Dist. ↓
APAS	8.84%	16.94%	0.7635	0.6251
APWS	26.78%	42.92%	0.6070	0.4294
WPAS	9.25%	20.21%	0.7437	0.5928
WPWS	27.15%	41.89%	0.6077	0.4382

participant" indicates their presence. A similar logic applies to "across sentence" and "within sentence". These settings allow us to measure the model's ability to generalize across different users and sentences.

The results reveal a clear performance hierarchy across the settings: WPWS $>$ APWS $>$ WPAS $>$ APAS. This trend indicates that having prior knowledge of either the user's typing patterns or the sentence structure improves reconstruction accuracy. Notably, the most significant performance gain comes from sentence familiarity; in both across-participant and within-participant tests, the accuracy is more than doubled when the model has been trained on the target sentences (e.g., APWS at 42.92% vs. APAS at 16.94%). Knowledge of a user's specific typing habits provides a smaller, yet consistent, improvement. Crucially, even in the most challenging and practically relevant APAS setting—where the model has seen neither the user nor the sentence before — our approach achieves a Top-1 accuracy of 8.84% and a Top-5 accuracy of 16.94%. This demonstrates a robust capability to reconstruct text from noisy, real-world data in a generalized, zero-knowledge scenario.

6.1.1. Ablation Study on Training Strategies. We first conduct an ablation study comparing it against two simpler training baselines. Table 3 shows the Top-5 Reconstruction Accuracy for three different models evaluated on the cache-extracted data. The models are trained with the following three datasets:

- **Ground Truth Only:** A model fine-tuned only on clean, ground-truth time intervals ("non cache only") and then tested on the noisy cache data.
- **Cache Only:** A model fine-tuned only on the noisy, cache-extracted time intervals.
- **Curriculum Learning (Ours):** Our proposed two-stage model ("cache & noncache").

TABLE 3. ABLATION STUDY: TOP-5 RECONSTRUCTION ACCURACY (%) ON CACHE-EXTRACTED DATA

Training Strategy	APAS	APWS	WPAS	WPWS
Ground Truth Only	11.40%	25.05%	11.01%	25.39%
Cache Only	3.71%	19.41%	4.35%	19.14%
Curriculum Learning (Ours)	**16.94%**	**42.92%**	**20.21%**	**41.89%**

The results of the ablation study are unequivocal. As shown in Table 3, the **Curriculum Learning** model achieves the best performance across all four evaluation settings, often by a significant margin. For instance, in the most challenging APAS setting, our curriculum model (16.94%) improves upon the Ground Truth Only model (11.40%) by nearly 50%, and is over four times more effective than the Cache Only model (3.71%). While the **Ground Truth Only** model shows some capability, its performance is clearly capped, indicating that knowledge from clean data alone is insufficient to handle the distortions of real-world cache noise. Most notably, the performance of the **Cache Only** model is exceedingly poor, highlighting the immense challenge of learning directly from noisy and complex cache timing signals from scratch. Our two-stage curriculum approach is therefore essential; by first learning from ground-truth data, the model establishes a strong foundation that greatly reduces the difficulty of the subsequent learning phase on noisy data and ultimately raises the model's performance ceiling.

Browser Cache Traces. We also evaluate against traces obtained from prime+probe attacks conducted from the Chrome web browser, as described in section 4.5. The results, presented in Table 4, show a noticeable degradation in performance compared to the ground truth experiments.

TABLE 4. RECONSTRUCTION PERFORMANCE ON BROWSER CACHE TRACES WITH GROUND TRUTH MODEL

Validation Split	Top-1 Recon. Acc. ↑	Top-5 Recon. Acc. ↑
Across-Across	1.17%	3.13%
Within-Within	5.86%	9.77%

The primary reason for this performance drop is a significant distribution shift between the training and evaluation data. The model evaluated here was trained exclusively on the clean, ground-truth keystroke intervals from the "136M Keystrokes" dataset. In contrast, the browser-based cache traces are subject to considerable noise and distortion inherent to the side-channel extraction process. This mismatch means the model is tasked with making inferences on a data distribution it has not been exposed to during training.

Addressing this challenge is left as an avenue for future work. A promising approach would be to generate a substantial dataset of realistic cache timing intervals through large-scale replay within the browser. Subsequently, a three-stage curriculum learning strategy could be employed to gradually adapt the model to noisier data. This process would involve first training the model on clean ground-truth time intervals, then fine-tuning it on native cache time intervals, and finally adapting it to the most complex browser-based cache time intervals. Such a method would be expected to bridge the distribution gap and could further enhance model robustness and reconstruction accuracy in practical scenarios.

7. Mitigation

Traditional side-channel vulnerabilities are often addressed by replacing the vulnerable code with constant-time implementations. However, keystroke timing side channel listens to the plethora of routines activated during a keystroke interrupt. Making changes to all possible input processing libraries, including different versions and combinations selected by various programs, would be untenable. Our findings also show that input handling is pervasive throughout the LLC, suggesting a multitude of leakage sources across the various processing layers. In addition, as keystroke handling is a time-consuming and expensive operation, as shown by the spikes in our previous graphs, constant-time programming would likely incur an intolerable high performance overhead.

Partially mitigating the problem, previously proposed cache side-channel defenses prevent the retrieval of accurate timing data through the cache. Noise-insertion-based approaches [122, 123, 124] obscure the memory access patterns that distort the hit spikes of the keystrokes.

However, these defenses only eliminate cache timing side channels instead of all other sources of leakage. Similar to the noise-injection approach, Keydrown [30] proposes a three-layer approach to stop a variety of attacks. The first layer issues additional interrupts to disrupt interrupt timing-based attacks. The second layer generates fake keystrokes that travel through the same handling routine, aiming to produce similar side effects as regular keystrokes. The third layer generates additional activity on cache lines that were activated for specific keystrokes, thereby disrupting cache template attacks. The three layers together made it extremely difficult for different side-channel attackers to discern injected keystrokes from actual keystrokes. We believe similar, multi-layered defense schemes are a promising direction towards building robust defenses against input reconstruction attacks.

Our work is the first study to quantify the impact of noise on input reconstruction success rate. We discover that input reconstruction is extremely sensitive to the accuracy of retrieved inter-keystroke timings. Making matters worse, detection errors propagate to consecutive detections and are difficult to identify from examining side-channel traces. Therefore, leveraging these properties, future defenses could use our findings and datasets to determine the appropriate amount of noise to inject in a principled manner.

8. Limitations

This paper focuses on reconstructing standard English input from inter-keystroke timings, assuming a QWERTY keyboard and standard English text. We identify two main directions for future work.

First, with sufficiently large and diverse datasets, our methods could be generalized to non-standard keyboard layouts and to other languages. Language semantics greatly constrain the space of plausible reconstructions produced from a given set of inter-keystroke timings, and exploiting those constraints is central to our current approach.

Second, reconstructing passwords, passphrases, and pseudorandom strings presents a very different and more difficult problem that will require new detection and reconstruction techniques. Password typing often exhibits distinct dynamics, shorter inter-keystroke intervals, peculiar gestures such as sliding across keys, and other user-specific habits, that reduce the effectiveness of side-channel detections and model generalization. Moreover, an LLM can no longer rely on language semantics to prune the search space, so exhaustive reconstruction becomes infeasible. A more practical path is to combine timing analysis with dictionary or probabilistic classification attacks to rank likely candidates, acknowledging that such approaches fail for truly random or out-of-dictionary secrets. Developing robust methods for detecting atypical typing patterns, integrating auxiliary signals, and designing scalable search strategies for high-entropy inputs are therefore important next steps.

9. Conclusion

This paper closes a long-standing gap in security research by empirically demonstrating, for the first time, that a victim's typed input can be fully reconstructed from inter-keystroke timings gathered in practical, noisy environments. By modeling the problem as a machine translation task and fine-tuning Large Language Models with a curriculum learning strategy, our approach successfully reconstructs 34.9% of inputs from clean timing data and, crucially, 16.9% from timings extracted via our end-to-end cache side-channel attacks. This achievement rests on our two main contributions: the development of a novel LLM-based reconstruction algorithm and the creation of a massive, open-source dataset of over 1.5 million real-world cache traces to enable robust training. Ultimately, our work transitions keystroke timing attacks from a theoretical possibility to a proven vulnerability, providing a concrete benchmark against which future attacks and defenses can be rigorously evaluated.

References

[1] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on {SSH}," in *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[2] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 245–254.

[3] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE, 2004, pp. 3–11.

[4] T. Halevi and N. Saxena, "Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios," *International Journal of Information Security*, vol. 14, pp. 443–456, 2015.

[5] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–26, 2009.
[6] J. Liu, Y. Wang *et al.*, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 142–154.
[7] M. Vuagnoux and S. Pasini, "Compromising electromagnetic emanations of wired and wireless keyboards." in *USENIX security symposium*, vol. 8, 2009, pp. 1–16.
[8] Y. Su, D. Genkin *et al.*, "{USB} snooping made easy: crosstalk leakage attacks on {USB} hubs," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1145–1161.
[9] D. Slater, S. Novotney *et al.*, "Robust keystroke transcription from the acoustic side-channel," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 776–787.
[10] J.-X. Bai, B. Liu, and L. Song, "I know your keyboard input: a robust keystroke eavesdropper based-on acoustic signals," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 1239–1247.
[11] S. R. K. Gopal, D. Shukla *et al.*, "Hidden reality: Caution, your hand gesture inputs in the immersive virtual world are visible to all!" in *32nd USENIX security symposium (USENIX Security 23)*, 2023, pp. 859–876.
[12] K. Jin, S. Fang *et al.*, "Vivisnoop: Someone is snooping your typing without seeing it!" in *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2017, pp. 1–9.
[13] N. Sehatbakhsh, B. B. Yilmaz *et al.*, "A new side-channel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 123–138.
[14] R. Song, Y. Song *et al.*, "I know what you type: Leaking user privacy via novel frequency-based side-channel attacks," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.
[15] M. Gazzari, A. Mattmann *et al.*, "My (o) armband leaks passwords: An emg and imu based keylogging side-channel attack," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 4, pp. 1–24, 2021.
[16] Z. Zhan, Z. Zhang *et al.*, "Graphics peeping unit: Exploiting em side-channel information of gpus to eavesdrop on your neighbors," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1440–1457.
[17] A. Taheritajar, Z. M. Harris, and R. Rahaeimehr, "A survey on acoustic side channel attacks on keyboards," in *International Conference on Information and Communications Security*. Springer, 2024, pp. 99–121.
[18] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems." in *USENIX Security Symposium*, vol. 20, 2009, p. 23.
[19] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 143–157.
[20] P. Vila and B. Köpf, "Loophole: Timing attacks on shared event loops in chrome," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 849–864.
[21] J. T. Trostle, "Timing attacks against trusted path," in *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*. IEEE, 1998, pp. 125–134.
[22] G. Shah, A. Molina *et al.*, "Keyboards and covert channels." in *USENIX Security Symposium*, vol. 15, 2006, p. 64.
[23] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 170–183.
[24] S. Chen, R. Wang *et al.*, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 191–206.
[25] A. K. Biswas, D. Ghosal, and S. Nagaraja, "A survey of timing channels and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, pp. 1–39, 2017.
[26] C. M. Tey, P. Gupta *et al.*, "Keystroke timing analysis of on-the-fly web apps," in *International Conference on Applied Cryptography and Network Security*. Springer, 2013, pp. 405–413.
[27] Z. Yang, Z. Sarwar *et al.*, "Can virtual reality protect users from keystroke inference attacks?" in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2725–2742.
[28] M. Lipp, D. Gruss *et al.*, "Practical keystroke timing attacks in sandboxed javascript," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security*. Springer, 2017, pp. 191–209.
[29] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive Last-Level caches," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 897–912. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss
[30] M. Schwarz, M. Lipp *et al.*, "Keydrown: Eliminating software-based keystroke timing side-channel attacks," in *Network and Distributed System Security Symposium 2018*, 2018, p. 15.

[31] T. Ristenpart, E. Tromer *et al.*, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 199–212.

[32] D. Gruss, C. Maurice *et al.*, "Flush+flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Cham: Springer International Publishing, 2016, pp. 279–299.

[33] P. Pessl, D. Gruss *et al.*, "Drama: exploiting dram addressing for cross-cpu attacks," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. USA: USENIX Association, 2016, p. 565–581.

[34] G. Zhang and S. Fischer-Hübner, "Timing attacks on pin input in voip networks (short paper)," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011, pp. 75–84.

[35] W. Diao, X. Liu *et al.*, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 414–432.

[36] Y. Guo, A. Zigerelli *et al.*, "Adversarial prefetch: New cross-core cache side channel attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1458–1473.

[37] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring (s): Side channel attacks on the {CPU}{On-Chip} ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 645–662.

[38] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 143–157.

[39] Y. Guo, A. Zigerelli *et al.*, "Adversarial prefetch: New cross-core cache side channel attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1458–1473.

[40] M. Kurth, B. Gras *et al.*, "Netcat: Practical cache attacks from the network," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 20–38.

[41] D. R. Dipta and B. Gulmezoglu, "Df-sca: Dynamic frequency side channel attacks are practical," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 841–853.

[42] H. Naghibijouybari, A. Neupane *et al.*, "Side channel attacks on gpus," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1950–1961, 2019.

[43] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," *ACM SIGARCH computer architecture news*, vol. 40, no. 3, pp. 118–129, 2012.

[44] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, 2018.

[45] D. Weber, F. Thomas *et al.*, "Indirect meltdown: Building novel side-channel attacks from transient-execution attacks," in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 22–42.

[46] D. Wang, A. Neupane *et al.*, "Unveiling your keystrokes: A cache-based side-channel attack on graphics libraries." in *NDSS*, 2019.

[47] J. Wan, Y. Bi *et al.*, "{HubBub}:{Contention-Based}{Side-Channel} attacks on {USB} hubs," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 3921–3940.

[48] S. Liu, S. Kanniwadi *et al.*, "{Side-Channel} attacks on optane persistent memory," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6807–6824.

[49] M. Tan, J. Wan *et al.*, "Invisible probe: Timing attacks with pcie congestion side-channel," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 322–338.

[50] F. Rauscher and D. Gruss, "Cross-core interrupt detection: Exploiting user and virtualized ipis," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 94–108.

[51] V. Dhakal, A. M. Feit *et al.*, "Observations on typing from 136 million keystrokes," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–12. [Online]. Available: https://doi.org/10.1145/3173574.3174220

[52] J. V. Monaco, "What are you searching for? a remote keylogging attack on search engine autocomplete," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 959–976. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/monaco

[53] N. González, E. P. Calot *et al.*, "The reverse problem of keystroke dynamics: Guessing typed text with keystroke timings only," in *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. IEEE, 2021, pp. 1–6.

[54] R. Weiss, D. Ayzenshteyn *et al.*, "What was your prompt? A remote keylogging attack on AI assistants," *arXiv preprint arXiv:2403.09751*, 2024.

[55] R. Weiss, D. Ayzenshteyn, and Y. Mirsky, "What was your prompt? a remote keylogging attack on {AI} assistants," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3367–3384.

[56] H. Touvron, T. Lavril *et al.*, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[57] H. Touvron, L. Martin *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288
[58] A. Grattafiori, A. Dubey *et al.*, "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783
[59] J. Bai, S. Bai *et al.*, "Qwen technical report," 2023. [Online]. Available: https://arxiv.org/abs/2309.16609
[60] A. Yang, B. Yang *et al.*, "Qwen2 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2407.10671
[61] A. Yang, A. Li *et al.*, "Qwen3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388
[62] A. Radford, K. Narasimhan *et al.*, "Improving language understanding by generative pre-training," 2018.
[63] A. Radford, J. Wu *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
[64] T. B. Brown, B. Mann *et al.*, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165
[65] OpenAI, J. Achiam *et al.*, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774
[66] OpenAI, : *et al.*, "Openai o1 system card," 2024. [Online]. Available: https://arxiv.org/abs/2412.16720
[67] H. Naveed, A. U. Khan *et al.*, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.
[68] M. U. Hadi, R. Qureshi *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints*, 2023.
[69] E. Kasneci, K. Seßler *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.
[70] J. Jiang, F. Wang *et al.*, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.
[71] A. Vaswani, N. Shazeer *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[72] T. Lin, Y. Wang *et al.*, "A survey of transformers," *AI open*, vol. 3, pp. 111–132, 2022.
[73] C. Raffel, N. Shazeer *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2023. [Online]. Available: https://arxiv.org/abs/1910.10683
[74] T. OLMo, P. Walsh *et al.*, "2 olmo 2 furious," *arXiv preprint arXiv:2501.00656*, 2024.
[75] Y. Liu, A. Singh *et al.*, "Improving large language model fine-tuning for solving math problems," *arXiv preprint arXiv:2310.10047*, 2023.
[76] N. J. Prottasha, A. A. Sami *et al.*, "Transfer learning for sentiment analysis using bert based supervised fine-tuning," *Sensors*, vol. 22, no. 11, p. 4157, 2022.
[77] K. W. Church, Z. Chen, and Y. Ma, "Emerging trends: A gentle introduction to fine-tuning," *Natural Language Engineering*, vol. 27, no. 6, pp. 763–778, 2021.
[78] A. Gasparetto, M. Marcuzzo *et al.*, "A survey on text classification algorithms: From text to predictions," *Information*, vol. 13, no. 2, p. 83, 2022.
[79] K. Kowsari, K. Jafari Meimandi *et al.*, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.
[80] R. Zhang, D. Jiang *et al.*, "Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems?" in *European Conference on Computer Vision*. Springer, 2024, pp. 169–186.
[81] Z. Shao, P. Wang *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.
[82] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, l3 cache Side-Channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom
[83] F. Liu, Y. Yarom *et al.*, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 605–622.
[84] A. Purnal, F. Turan, and I. Verbauwhede, "Prime+scope: Overcoming the observer effect for high-precision cache contention attacks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.
[85] F. Rauscher, C. Fiedler *et al.*, "A systematic evaluation of novel and existing cache side channels," in *NDSS*, 2025.
[86] Y. Guo, X. Xin *et al.*, "Leaky way: A conflict-based cache covert channel bypassing set associativity," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 646–661.
[87] S. Briongos, P. Malagon *et al.*, "RELOAD+REFRESH: Abusing cache replacement policies to perform stealthy cache attacks," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1967–1984. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/briongos
[88] G. Horowitz, E. Ronen, and Y. Yarom, "Spec-o-scope: Cache probing at cache speed," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 12 2024, pp. 109–123.
[89] M. Yan, R. Sprabery *et al.*, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 888–904.
[90] Z. N. Zhao, A. Morrison *et al.*, "Last-level cache side-channel attacks are feasible in the modern public cloud," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems,*

Volume 2, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 582–600. [Online]. Available: https://doi.org/10.1145/3620665.3640403

[91] A. Kwong, W. Wang *et al.*, "Checking passwords on leaky computers: A side channel analysis of chrome's password leak detect protocol," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 7107–7124.

[92] S. Cohney, A. Kwong *et al.*, "Pseudorandom black swans: Cache attacks on ctr_drbg," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1241–1258.

[93] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on aes, and countermeasures," *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, Jan 2010. [Online]. Available: https://doi.org/10.1007/s00145-009-9049-y

[94] P. Kocher, J. Horn *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[95] M. Lipp, M. Schwarz *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[96] E. Barberis, P. Frigo *et al.*, "Branch history injection: On the effectiveness of hardware mitigations against Cross-Privilege spectre-v2 attacks," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 971–988. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/barberis

[97] E. M. Koruyeh, K. N. Khasawneh *et al.*, "Spectre returns! speculation attacks using the return stack buffer," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, Aug. 2018. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/koruyeh

[98] J. Van Bulck, D. Moghimi *et al.*, "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection," in *41th IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[99] S. van Schaik, A. Milburn *et al.*, "Ridl: Rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 88–105.

[100] S. van Schaik, M. Minkin *et al.*, "Cacheout: Leaking data on intel cpus via cache evictions," in *S&P*, May 2021.

[101] J. Kim, D. Genkin, and Y. Yarom, "Slap: Data speculation attacks via load address prediction on apple silicon," in *S&P*, 2025.

[102] J. Kim, J. Chuang *et al.*, "Flop: Breaking the apple m3 cpu via false load output predictions," in *USENIX Security*, 2025.

[103] M. Schwarz, M. Lipp *et al.*, "ZombieLoad: Cross-Privilege-Boundary Data Sampling," in *CCS*, 2019.

[104] Y. Oren, V. P. Kemerlis *et al.*, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1406–1418. [Online]. Available: https://doi.org/10.1145/2810103.2813708

[105] P. Vila, B. Köpf, and J. F. Morales, "Theory and practice of finding eviction sets," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 39–54.

[106] Q. Dong, L. Li *et al.*, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[107] J. Wei, J. Wei *et al.*, "Larger language models do in-context learning differently," *arXiv preprint arXiv:2303.03846*, 2023.

[108] J. V. Monaco, "What are you searching for? a remote keylogging attack on search engine autocomplete," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 959–976.

[109] C. Raffel, N. Shazeer *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[110] X. Wang, Y. Zhou *et al.*, "Curriculum learning: Theories, approaches, applications, tools, and future directions in the era of large language models," in *Companion Proceedings of the ACM Web Conference 2024*, 2024, pp. 1306–1310.

[111] Y. Bengio, J. Louradour *et al.*, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[112] J. V. Bulck, M. Minkin *et al.*, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[113] L. Hetterich, F. Thomas *et al.*, "Shadowload: Injecting state into hardware prefetchers," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1060–1075. [Online]. Available: https://doi.org/10.1145/3676641.3716020

[114] Z. Zhang, M. Tao *et al.*, "BunnyHop: Exploiting the instruction prefetcher," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 7321–7337. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-zhiyuan-bunnyhop

[115] X. Zhang, Z. Zhang *et al.*, "Thermalscope: A practical interrupt side channel attack based on

thermal event interrupts," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3656525

[116] M. Lipp, A. Kogler *et al.*, "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 355–371.

[117] Y. Cohen, K. S. Tharayil *et al.*, "Hammerscope: Observing dram power consumption using rowhammer," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 547–561. [Online]. Available: https://doi.org/10.1145/3548606.3560688

[118] J. Koschel, C. Giuffrida *et al.*, "Tagbleed: Breaking kaslr on the isolated kernel address space using tagged tlbs," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020, pp. 309–321.

[119] W. Liu, J. Ravichandran, and M. Yan, "Entrybleed: A universal kaslr bypass against kpti on linux," in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 10–18. [Online]. Available: https://doi.org/10.1145/3623652.3623669

[120] Y. Jin, C. Wang *et al.*, "Whisper: Timing the transient execution to leak secrets and break kaslr," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3656213

[121] A. Agarwal, S. O'Connell *et al.*, "Spook.js: Attacking chrome strict site isolation via speculative execution," in *43rd IEEE Symposium on Security and Privacy (S&P'22)*, 2022.

[122] Y. Zhou, S. Wagh *et al.*, "Camouflage: Memory traffic shaping to mitigate timing attacks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 337–348.

[123] P. W. Deutsch, Y. Yang *et al.*, "Dagguise: mitigating memory timing side channels," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 329–343. [Online]. Available: https://doi.org/10.1145/3503222.3507747

[124] L. Li, J. Huang *et al.*, "Prefender: A prefetching defender against cache side channel attacks as a pretender," *IEEE Transactions on Computers*, 2024.

Appendix A. Responses without Fine-tuning

Box section A shows the responses of different pre-trained models without fine-tuning.

Responses of Pre-trained Models without Fine-tuning

Prompt: Translate the Time intervals to Keystrokes. Time intervals: 516 222 165 ...

OLMo 1B Response: *To convert time intervals to keystrokes, we need to know the number of characters or digit representations needed for each interval. [...] Assuming each second is represented by one character [...], a broad conversion factor might involve: Seconds: 26 letters, Minutes: 10 digits, Hours: 100 digits...*

Qwen 3 8B Response: *Okay, so I need to translate these time intervals into keystrokes. [...] Maybe each number represents the duration between each key press. But I need more context. Wait, perhaps the problem is similar to converting a sequence of time intervals into a sequence of characters, like in a typing pattern. But how? [...] Maybe there's a standard way to convert time intervals into keystrokes. For example, using the time intervals as ASCII values... But the given numbers are all over the place...* (cut off due to length)

DeepSeek R1 Response: *We are given a list of time intervals. These time intervals represent the time between consecutive keystrokes in milliseconds. [...] However, without a mapping or context, we must assume that the problem is about representing the keystrokes as the time intervals themselves in a different format? Or perhaps we are to interpret them as pauses for a space or newline? [...] Another common context is that in typing, long pauses (above a threshold) might indicate a space or a new word. However, we need a threshold.*

Appendix B.
Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

This paper presents an improvement over keystroke detection and input reconstruction side-channel work. The two major contributions of this work is the creation of a large dataset of real-world typist cache-side channel data, and the introduction of a large language model (LLM) based recovery mechanism.

B.2. Scientific Contributions

- Provides a New Data Set For Public Use
- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

B.3. Reasons for Acceptance

1) This paper carries out a systematic methodology to enable future side-channel work by collecting a large scale dataset of inter-keystroke timings. This includes various typists, as well as real-world noise.
2) The authors create a new tool to enable future science. The authors leverage their dataset to train a new tool to detect keystrokes.
3) Though previous work has achieved reasonable performance, their implementation is able to realistically recover keystrokes from noisy data with high accuracy.
4) The architecture presented and evaluation provides valuable insights on how to improve input recovery in this setting.

B.4. Noteworthy Concerns

1) The paper performed its attacks against a single CPU make and model. It is not completely clear whether they will reproduce on other systems.
2) The paper focused on English sentences. It is not clear if this is a serious risk against authentication values such as PINs and passwords. This is acknowledged on Section 8.
3) The paper's analysis is based primarily on replayed traces. While Sections 5.5 and 8 do suggest that a replay-based analysis is representative, the possibility still remains that some artifacts of replay may positively impact the observed keystroke leakage rates.