USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Found in Translation: A Generative Language Modeling Approach to Memory Access Pattern Attacks

Grace Jia, Alex Wong, and Anurag Khandelwal, *Yale University*

https://www.usenix.org/conference/usenixsecurity25/presentation/jia-grace

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

Found in Translation: A Generative Language Modeling Approach to Memory Access Pattern Attacks

Grace Jia
Yale University

Alex Wong
Yale University

Anurag Khandelwal
Yale University

Abstract

Confidential computing environments (CCEs) provide a secure way for privacy-sensitive applications to ensure the confidentiality and integrity of data and computations offloaded to the cloud, relying on a hardware root of trust. However, the cloud provider-controlled Operating System (OS) stack still manages key memory management system services such as paging. Several recent works have demonstrated that these services can leverage side channels, specifically page access patterns, to reconstruct private application data. However, related attacks have primarily targeted applications with simple one-to-one mappings between application-level objects and OS-level pages, which is seldom true for most real-world cloud applications. Moreover, these attacks tend to overlook correlations in access patterns — a common occurrence in most real-world applications — leaving untapped critical side-channel information for improving attack accuracy.

We propose a novel attack approach that leverages access correlations across pages in cloud applications using *generative language models*. Our key insight is that there are strong parallels between application page access patterns and grammatical structures in natural languages, making language modeling an excellent fit for reconstructing sensitive application data with high accuracy. Our attack, named FIT[1], utilizes a recurrent encoder-decoder architecture to predict application-level object accesses from a sequence of page-level accesses. Our evaluations on popular AI/ML model inference services and semantic search applications show that FIT can predict object-level access sequences with an average accuracy ranging from 71.7% to 99.9%, significantly outperforming prior state-of-the-art approaches.

1 Introduction

Operating systems (OS) are large, complex, and prone to vulnerabilities; yet, they are granted full privilege and complete access to data on the application software stack. When such privileged software is compromised, all user applications that run on top of it are compromised, posing risks to their confidentiality and integrity. Similarly, modern hypervisors in cloud environments have unrestricted access to application data within virtual machines or containers deployed on them.

In the past decade, Confidential Computing Environments (CCEs) — also referred to as Trusted Execution Environments (TEEs) — have emerged to address this issue by building secure *enclaves* from a hardware root of trust. In such environments, a trusted computing base (TCB) ensures data confidentiality and control-flow integrity of application software running within this environment, and its integrity can be remotely attested. At the same time, trusted hardware-mediated accesses to protected memory regions and memory encryption provide data confidentiality. Hardware vendors have already supplied two generations of diverse CCE solutions, from Intel SGX [44] and ARM TrustZone [3] to Intel TDX [32], AMD SEV [1], and ARM CCA [4].

Even with trusted hardware and protected memory, applications inside a CCE still rely on the untrusted OS or hypervisor for system services, which can be leveraged in side-channel attacks to learn privacy-sensitive application data, breaking confidentiality guarantees. This work focuses on memory access pattern attacks, where the applications rely on the OS's paging subsystem for more efficient data caching. In particular, whenever an application accesses a page, the memory management unit (MMU) logs metadata about the access, e.g., to track the page "hotness" (via an 'accessed' bit) or to track whether the page was written to and must be flushed in case it is backed by a disk block (via a 'dirty' bit). This information is accessible to an untrusted OS for system services, such as moving pages to optimize memory access latency and flushing data to ensure data consistency [54]. This allows the untrusted OS to learn a histogram of accesses across pages and use it to identify sensitive data items being accessed by a user, given some prior knowledge about the application (e.g., by tying page hotness to well-known distributions of medical diagnoses in a clinical diagnostic tool).

Indeed, such approaches have been explored in recent

[1] Found in Translation

works to extract complete text documents and outlines of JPEG images from widely deployed application libraries [76], as well as private keys from cryptographic libraries [10, 59]. Unfortunately, we find that such attacks — while quite effective for the above use cases — suffer from critical shortcomings that limit their effectiveness in learning sensitive information across a range of use cases (§2). Specifically, in real-world applications, each page typically contains many objects, making it challenging to identify which object was accessed based solely on page-level access patterns. Prior attacks address this by "fingerprinting" one object or control flow using a unique sequence of accessed pages; however, they ignore the broader network of correlations across page accesses (e.g., page j is accessed after page i since j contains data pointed to by data in i), limiting the number of identifiable objects, and the accuracy with which they can be identified.

While we are unaware of any page access pattern attacks that leverage access correlations, recent work [36, 53] on attacking Symmetric Searchable Encryption (SSE) schemes do leverage pairwise access correlations to achieve significantly improved attack accuracy over those that do not. However, these attacks are poorly suited to the case of page accesses (§5) since they assume a one-to-one mapping between pages and application objects (i.e., encrypted and plaintext keywords, respectively, in the SSE context), which is seldom true in real-world applications.

In this work, we propose a novel attack that leverages arbitrary-length access correlations across pages using *generative language models* (§4). Our driving insight is the strong parallel between page access patterns in real-world applications and grammatical structures in natural languages. For instance, given the first three words in a sentence, "I wrote a", a language model would more likely predict "book" as the next word instead of "the" or "sandwich", adhering to the embedded grammatical cues in the language. In the same vein, we posit that page access patterns have a language of their own, and this language is specific to the victim application. For instance, in a database application, if pages i, j, and k, each containing distinct nodes of a data structure that are linked together, are accessed in that sequence, the next accessed page will likely be the one that contains the next node in that linked list. Moreover, disambiguating between objects located on the same page has parallels to language models that commonly disambiguate *homographs*: just as the English word "right" can be translated into multiple words in French depending on the context "right hand" ("droite") or "that is right" ("vrai"), different nodes corresponding to different linked lists located in a page can be disambiguated based on the context of the access sequence so far.

We leverage this insight in our generative modeling attack, FIT, which uses a recurrent encoder-decoder architecture [16, 64]. The adversary trains its encoder-decoder model to predict a sequence of application object-level accesses $y_1, ..., y_n$ given a sequence of page-level accesses $x_1, ..., x_m$, where m and n are page and object access sequence lengths for the victim application, respectively. The model learns both the mapping from page-level access sequences to object-level access sequences and the distribution (including conditional access probabilities to account for correlated accesses) from which the object accesses were generated.

We evaluate FIT on three privacy-sensitive applications with real-world workloads and datasets: a popular deep learning-based recommendation model (DLRM), a large language model for medical diagnosis, and a state-of-the-art semantic search service over a vector database. FIT can accurately predict object-level access sequences with an average accuracy of 71.7% to 99.9%, significantly outperforming prior state-of-the-art access pattern attacks.

FIT aims to paint a realistic picture of vulnerabilities due to page-level access pattern leakage in real-world cloud deployments and applications. We believe this is key to developing countermeasures that achieve a practical balance between performance and security (§7) — an increasingly crucial goal where existing countermeasures are either performant but vulnerable to our attack [25, 71] or provide stronger guarantees than required at the expense of performance [12, 14, 21, 63].

2 Background and Motivation

We begin with a brief overview of confidential computing environments (§2.1) and prior memory access pattern attacks on them, along with their shortcomings (§2.2). We end with an outline of our page access pattern attack approach (§2.4).

2.1 Confidential Computing Environments

Confidential computing environments (CCEs) — also known as Trusted Execution Environments (TEEs) — shield sensitive applications from untrusted system software, such as OSes and hypervisors. CCEs typically build upon a hardware root of trust, such as Intel SGX [44], ARM TrustZone [3], and AMD SEV [1], to provide strong guarantees on the confidentiality and integrity of application execution.

CCE attestation and memory protection. To ensure the integrity of code and data loaded into a CCE, the application owner can use hardware-provided attestation mechanisms to cryptographically verify that the CCE and the software within are correctly instantiated. Once initialized, CCEs allow applications to create protected memory regions within their address space and use trusted hardware to enforce their security guarantees. All memory accesses (reads and writes) to the protected regions go through the processor, which rejects those not originating from trusted application code. Data in the protected region is also automatically encrypted and remains accessible in plaintext only within the processor registers and caches.

Protecting privacy-sensitive applications. In this work, we focus on three representative privacy-sensitive cloud applications that benefit from CCE protection. We will use these applications as running examples in the rest of the paper.

Our first two applications are examples of ML/AI model inference services deployed on a public cloud, which are a critical component of many web services [70, 74, 78]. We specifically focus on deep learning-based recommendation models (DLRMs) [49] and large language models (LLMs) [52, 73], which are often offloaded to the cloud due to their compute- and memory-intensive execution but require detailed data from users to be stored in the cloud, potentially visible to an untrusted cloud provider. Such private user information can include buying habits or movie preferences for personalized recommendations, or privacy-sensitive prompts for LLM-based chatbots. Some schemes for oblivious model inference protect this sensitive data through homomorphic encryption, MPC, and other oblivious techniques, but at significant cost to performance [38, 45, 56]. More practical approaches leverage CCEs, where the entire model or specific layers are run inside a CCE [41, 48, 51, 66]. DLRM and LLM inference can be served privately using a similar approach [40]; encrypting inference requests to and resulting responses from the model would ensure that the cloud's system software (and, therefore, the cloud provider) never sees user data in plaintext.

Note that while these models are often deployed on GPUs with CCEs, we focus on CPU-based CCEs for two reasons. First, CPUs are frequently used in lieu of GPUs in cases where the cost of deploying GPUs is exorbitantly high, and the models are small enough — a common case for DLRM [34] and smaller LLM models with fewer parameters. Second, while our findings also apply to GPU CCEs, CPUs offer a more convenient way of analyzing CCE vulnerabilities since the associated software libraries have more permissive licenses (e.g., open-sourced Linux vs. proprietary NVIDIA libraries).

The third application we consider is semantic search, widely used in modern search engines and data analysis [15]. Unlike traditional search, this application stores high-dimensional vector representations of data, e.g., images or text, in a vector database. Given a query (also in vector form), the semantic search returns the nearest neighbors of that query vector in the database. Semantic search is often performed on sensitive data, such as face representations for facial recognition [20] and biometric information [5]. With recent lines of work on securing search using both oblivious techniques [15, 19, 84] and trusted hardware [2, 46], CCEs are an attractive option for privacy-sensitive semantic search.

2.2 Page Access Pattern Attacks on CCEs

Even though CCEs prevent unauthorized access to application memory, they still rely on system software for shared system services, such as memory management, networking, and file I/O. These interactions with the system software leak information about the application's execution, which can be used to mount *side-channel attacks*. In this work, we restrict our focus to memory side channels, where an attacker can exploit the application's access patterns across memory pages to learn its secrets, as explained next.

Since the OS manages the page table and handles page faults for any application, it can observe information leaked about the application's memory access patterns at a page-level granularity. Earlier works that exploited this channel inferred page access patterns from sequences of page faults [76]: the untrusted OS restricts access to particular memory pages (e.g., by editing the page table), causing any access to those pages to trigger a page fault that the OS will record. Given exact knowledge of the application binaries and which memory accesses are input-dependent, the attacker can infer the sensitive data processed by the application. This attack is capable of extracting complete text documents and outlines of images from a single run of the targeted applications within a CCE; this approach has also been shown to be able to extract secret keys from widely-used cryptographic libraries [59].

While restricting access to certain memory regions provides complete visibility over access patterns in those regions to the OS, such an attack is quite *overt* — the application can easily detect the performance slowdowns incurred due to excessive page faults. More recent work has demonstrated that the OS can infer page accesses from an application *without* page faults [10]. This *covert* attack simply observes page table metadata that the OS maintains to inform its page replacement policy; specifically, each page table entry (PTE) contains an accessed (A) bit and dirty (D) bit, which the processor automatically sets whenever the corresponding page is read or written to, respectively. Like the page fault attack, this attack first analyzes the application binary to identify input-dependent memory accesses and select a trigger page to monitor. During the application's execution, the attacker continuously checks the trigger page's PTE and interrupts the application as soon as it is accessed. By recording the set of pages accessed between each successive access to the trigger page, the attacker can uniquely identify the application's control flow and extract its input data with high success.

2.3 Exploiting Correlations in Page Accesses

We now describe how the applications discussed in §2.1 leak access patterns, particularly through correlated accesses. We show that these application classes are less susceptible to traditional page access pattern attacks (discussed above) due to their inability to exploit access correlations.

Access pattern leakage in ML/AI model inference. We find that deep learning models such as DLRMs and LLMs exhibit memory access patterns that can be exploited to reveal private inputs, regardless of CCE execution. This is due to input-dependent accesses to a standard component of such

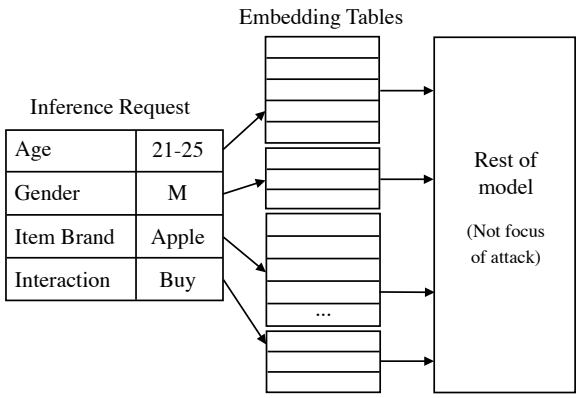

Figure 1: **Input-dependent access patterns across embedding tables in DLRM.** Each private categorical feature in an inference request results in an access to a single embedding table entry, which the adversary can directly trace back to its value.

models — *embedding tables*, which store mappings from categorical data to their numerical representations in the model's high-dimensional vector space. In any deep neural network with categorical input data, inference begins with the embedding lookup stage, which converts each of the input sample's categorical features, e.g., a user's age group or gender, into a vector embedding that the rest of the model can use. For each categorical feature, the index of the embedding table entry accessed during lookup directly corresponds to the plaintext value of that feature in the input data sample, as shown in the example in Figure 1. This also applies to LLMs, which have a single large embedding table containing an entry for each unique token (word or subword) in its vocabulary. Since embedding tables reside in memory, an attacker can directly extract the private attributes of an encrypted inference request or the plaintext tokens of an encrypted prompt as long as it can infer accesses at the level of embedding table entries from coarser-grained page accesses.

Access pattern leakage in semantic search. Nearest-neighbors search over a vector database also exhibits input-dependent access patterns that can be exploited. For a specific search algorithm, we focus on Hierarchical Navigable Small World (HNSW) [42], a state-of-the-art semantic search index. Briefly, HNSW utilizes a multilayer graph index, where the bottom layer comprises a full proximity graph of the vector database, and the upper layers contain progressively fewer nodes and edges. The search algorithm begins at the top layer (with the fewest nodes), greedily identifying the nearest neighbor to the query vector, and then proceeds to the layer below until it reaches the bottom. Since the search ends in the query vector's neighborhood, an adversary can observe the vectors accessed during the search to infer information about the nearest neighbors found or possible values of the private query vector. Our work focuses on uncovering the vector-level accesses from page-level accesses, and we leave the additional inference of private attributes from these vector accesses as future work.

Shortcomings of prior attacks. Although prior page access pattern attacks have demonstrated success in specialized use cases, applying them as is to embedding lookups and semantic search poses interesting challenges. The shortcomings that we identify are twofold. First, a sequence of page accesses alone cannot reveal the specific objects (i.e., embedding table entries or HNSW graph nodes) accessed, as there is a many-to-one mapping from pages to objects — multiple objects can reside on one page. Second, in focusing on mapping exact subsequences or sets of pages to specific control flows, prior approaches do not leverage correlations in access patterns to increase the effectiveness or versatility of their attacks. We identify two specific page access pattern attack approaches as representatives of this class of work to better highlight their shortcomings.

Naive Bayes is a simple probabilistic approach to modeling the distribution of accesses across objects conditioned by observed page accesses. This serves as a representative of prior ~~page~~ access pattern attacks that rely solely on the histogram of accesses across pages. A Naive Bayes classifier can be defined for each monitored data structure, e.g., for DLRM, one for each embedding table that is looked up during inference. Given observations of page accesses with corresponding ground-truth embedding table accesses, the classifier can learn for each page the distribution (i.e., a histogram) of entries accessed on that page. Then, given only an observed page access, the classifier for a particular embedding table outputs the accessed entry as a predicted class by sampling from that page's histogram of embedding table entries. Naive Bayes classification is inherently limited by its simplicity, especially its "naive" assumption that the input features — the page accesses — are *independent*. This results in poor predictions for accessed objects in our considered applications, as we will demonstrate empirically in §5.

IHOP [53] is a statistical-based attack that, while designed for Searchable Symmetric Encryption (SSE) schemes, is one of the only two access pattern attacks [36, 53] to our knowledge that leverage dependencies across accesses. For this reason, we consider its adaptation to our page access pattern setting. IHOP's goal is to find a mapping between the set of observed encrypted objects (i.e., pages) and the set of underlying plaintext objects. To compute this mapping, IHOP solves a quadratic optimization problem that maximizes the likelihood of the observed access distribution over pages, given the adversary's auxiliary knowledge of the access distribution over the objects.

Specifically, IHOP exploits correlated accesses by taking as input two Markov matrices $\mathbf{F}$ and $\tilde{\mathbf{F}}$: for pages p_j and $p_{j'}$, the j, j'th entry of $\mathbf{F}$ is the number of times that an access to page $p_{j'}$ is followed by an access to p_j, normalized by the total

number of times $p_{j'}$ is accessed. Similarly for objects o_i and $o_{i'}$, the i,i'th entry of $\tilde{\mathbf{F}}$ is the probability that $o_{i'}$ is accessed followed by o_i. These matrices explicitly encode pairwise correlations among pages and objects, allowing IHOP to, in theory, map pages to objects even when page accesses appear uniformly distributed.

However, in practice, IHOP's key limitation is the assumption of a one-to-one correspondence between pages and objects (inherited from its intended use case of SSE), rendering it unable to consider more than one object that can be accessed via a certain page. Moreover, our evaluations (§5) show that IHOP is ineffective in capturing correlations between more than two consecutive accesses in a sequence.

The other attack on dependent accesses is MAPLE [36], which applies a hidden Markov model (HMM)-based framework in a similar setting. Instead of framing the problem as one of optimization, this approach models the user's object-level access distribution as a hidden Markov chain and then leverages HMM inference techniques (e.g., the Viterbi algorithm) to find the sequence of object accesses that best explains the observed sequence of page accesses. Unfortunately, it shares IHOP's limitation of applying a Markov assumption to the access distribution, i.e., each object access in a sequence depends only on the previous object access, which limits the attack's capability of leveraging sequences of more than two dependent accesses.

2.4 Overview of Our Approach

In this work, we present FIT, a memory access pattern attack that addresses the two shortcomings of prior approaches: it can capture a one-to-many mapping from coarse-grained memory locations to finer-grained ones, in part by leveraging dependencies within longer sequences of accesses. Focusing on cloud-based ML/AI model inference services and semantic search applications, we define our threat model and the knowledge available to the adversary in §3. Our key insight is that a *generative language model* can efficiently capture dependencies within access sequences and propose our generative approach to translating observed page accesses to private object-level accesses in §4. We then evaluate the feasibility and efficacy of this attack on DLRM, LLM, and semantic search applications with real data in §5.

3 Security Model

We now detail our system and threat model (§3.1) along with a formal description of the adversary's goal and the knowledge it can use to carry out our proposed attack (§3.2).

3.1 System & Threat Model

Our system model is that of a cloud-hosted application with multiple trusted clients and an untrusted server (Figure 2).

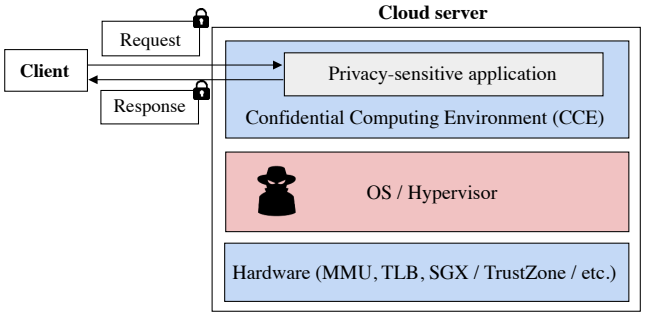

Figure 2: **System & Threat Model.** The adversary resides in the system software of the cloud server, on which sensitive applications run within CCEs based on a hardware root of trust. The application trusts components colored in blue.

A client interacts with the service by sending a request with encrypted input data over the network. The server provides a CCE in which the application processes the decrypted client data, e.g., running inference for an AI/ML service or executing a search query for a vector database. The output is encrypted and returned to the client. We assume that application execution occurs on the CPU, as can be the case for recommendation models [26, 34] and LLMs [58].

On the server, we assume a standard CCE threat model [59, 68, 76], in which the OS is an honest but curious adversary that aims to extract sensitive client data without compromising service to the client. This is typical in cloud deployments, where the cloud provider or a malicious actor with administrator credentials [43] controls the OS (more precisely, the hypervisor). Similar scenarios are found in Symmetric Searchable Encryption (SSE) [25, 53] and oblivious analytics [50, 61, 83] settings, where the adversary controls the untrusted server and observes clients' operations over encrypted data.

The adversary cannot deviate from the server protocol, e.g., by modifying client requests and responses; it is also motivated not to mount availability attacks to avoid losing the cloud service's customer base due to degraded quality of service. We assume that any hardware relied on by encryption or CCE is correctly implemented, so the adversary cannot extract secrets directly from within the processor. However, the adversary can observe side-channel leakage from the CCE execution and make systems-level decisions to reduce noise in those channels. It can also control all OS scheduling and memory allocation decisions, freely access unprotected memory, and interrupt user processes. It is, however, incentivized to minimize interactions that affect application performance to avoid detection. We also assume that this privileged adversary has white-box access to the hosted application — e.g., due to it being open source — similar to assumptions in prior attacks where exact application binaries are known [10, 76].

Notation.	
Y	Set of objects accessed by the application.
ρ	Distribution of accesses over objects Y.
X	Set of pages accessed by the application.
ϕ	Distribution of accesses over pages X.
`mem`	Mapping of objects to pages.
Auxiliary information.	
ρ; Dataset of $(\mathbf{x},\mathbf{y})$: page and object access sequences	
Observed information.	
Dataset of $\mathbf{x}$: page access sequences	

Table 1: Summary of notation and adversarial knowledge.

3.2 Adversarial Goal & Knowledge

Next, we formally define the adversary's goal, using notations summarized in Table 1. We consider a CCE-protected application that makes sequences of dependent accesses over a set Y of objects in memory drawn from an underlying joint access distribution ρ. Each object is stored on a page of memory x in the set of pages X; we define the layout of these objects in memory as a mapping `mem` from Y to X, which gives for each object y the page address x on which it resides. Although the adversary knows `mem` (as described in §3.1), it can only observe the application's access sequence at the granularity of pages, i.e., a sequence of page-level accesses $\mathbf{x} = x_1, ..., x_m$ corresponding to the underlying sequence of object-level accesses made by the application $\mathbf{y} = y_1, ..., y_m$, where $\texttt{mem}(y_i) = x_i$. Then, the adversary's goal is to use auxiliary information about the application's access distribution *to identify the actual objects accessed by the application* given an observed sequence of page-level accesses.

Auxiliary information. We assume that the adversary has auxiliary knowledge of the distribution of user requests to the application — e.g., for DLRM, the joint distribution of the user base's categorical features — based on this, the adversary can directly derive the access distribution ρ. This assumption is common in leakage-abuse works and aligns with real-world use cases where distributional information about the user population is well-known or accessible [11, 33, 50, 53].

The adversary can utilize this distributional knowledge to collect ground-truth data on page accesses and their corresponding object accesses. It samples a set of requests from the distribution and either runs an instance of the application or simulates the page accesses using `mem` and its white-box access to the application. The result is an auxiliary dataset of $(\mathbf{x},\mathbf{y})$ pairs, where $\mathbf{y}$ is the sampled sequence of object-level accesses and $\mathbf{x}$ is the observed sequence of pages accessed by the application on $\mathbf{y}$.

Observed information. When launching the attack, the adversary observes only $\mathbf{x}$, the page access sequence, for each request to the actual application. It knows the layout `mem` of the application's accessed objects in memory, e.g., the virtual pages assigned to each embedding table of the DLRM. The system-level adversary can identify the start and end of each request by monitoring the network I/O syscalls that mark each request and the corresponding response from the application. While, realistically, the adversary observes one accessed page per accessed object throughout a request, we discuss in §5.5 the impact on the adversary's ability to exploit leaked information when it cannot observe *all* page accesses for a request.

3.3 End-to-End Attack Example

We now describe an end-to-end execution of the FIT attack with our adversary on embedding table accesses in DLRM to illustrate our threat model. We defer further relevant details of FIT in this scenario, along with our other considered use cases LLM and HNSW, to §5.1.

The DLRM application runs within a trusted CCE on the adversary-controlled cloud server. During initialization, the application instantiates the 26 embedding tables of its recommendation model, one for each categorical feature in the data that will be used in the private inference requests it receives. Each of these tables is implemented as a PyTorch `EmbeddingBag` module, and its entries are stored contiguously in the `weight` tensor of the `EmbeddingBag`. The creation of these tensors results in a sequence of syscalls that request memory allocations of distinct sizes, allowing the adversary to control the data pages and virtual address ranges allocated to each embedding table.

When the application receives an encrypted inference request from a client, the adversary is notified through a network I/O system call and clears the accessed bits of the PTEs for all embedding tables. Assuming that the request contains a single sample of data, the DLRM converts the sample's categorical features into vector embeddings by performing a lookup in each `EmbeddingBag`, resulting in access to some page of its `weight` tensor. Once the model outputs a result that the application encrypts and returns over the network, the adversary traverses the embedding table PTEs and logs pages marked as accessed by the processor. This yields the adversary's observed page access sequence for the request.

The adversary then uses its auxiliary knowledge to infer the most likely embedding table accesses that resulted in the observed page-level accesses. As a concrete example, consider that the categorical features represent an e-shop user and an ad they interacted with (the semantics of the dataset used in our evaluation are not publicly known). Assume that for the user's location, the embedding table entries of US cities `Miami` and `Denver` are on the same page P_1. In contrast, the embedding table for the ad's product category has an entry for `swimwear` on page P_2 and an entry for `ski gear` on page P_3. While the adversary cannot determine from access to P_1 alone whether the user is in Miami or Denver, if the observed page access sequence contains P_2 rather than P_3, the adversary

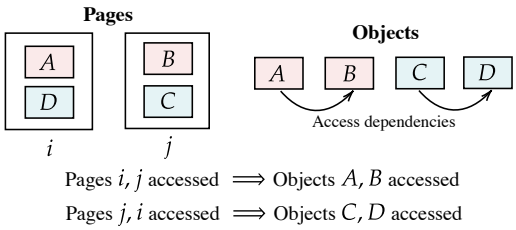

Figure 3: In this example, the adversary can use its knowledge of dependent accesses between objects to infer object-level accesses from observed page-level accesses.

can infer that the city is likely Miami, as swimwear is more likely to be purchased in a tropical destination. This process of leveraging correlations between accesses is the focus of the following section (§4). Since the index of each embedding table entry uniquely corresponds to the plaintext value of the categorical feature, accessing the embedding table entries directly leaks the data in the inference request, allowing the adversary to recover the client's private information.

4 FIT: A Novel Generative Language Modeling Attack

The core contribution of our work is a generative modeling approach to memory access pattern attacks. We begin with an explanation of the intuition behind our approach and background on language modeling (§4.1). We then outline the design of FIT, which uses a deep generative model to learn the underlying distribution of memory access sequences (§4.2). Our attack can infer private attributes with significantly higher accuracy than existing statistical methods (§5).

4.1 A Language Model of Access Correlations

While most access pattern attacks assume that accesses are independent [8,11,25,33], this is typically not the case in real-world applications. Accesses to objects (e.g., nodes, items, etc.) within a shared data structure tend to be correlated, i.e., accessing a particular address affects the probability distribution of which address is accessed next. An attacker can exploit these correlations to obtain fine-grained information about a victim's accesses. For a simple example, consider four objects A, B, C, and D residing on two pages i and j as pictured in Figure 3. Suppose that the victim application always accesses A followed by B and C followed by D. As such, the order in which pages i and j are accessed reveals whether the accesses are to A and B as opposed to C and D. Thus, cross-page correlated accesses enable the adversary to overcome the one-to-many problem of mapping page-level accesses to object-level accesses. While this example only considers dependent access sequences of length 2, our insight can be generalized to an arbitrary number of dependent accesses, e.g., traversing a linked list. Settings with correlated accesses of this nature remain largely unexplored in the security community (§6).

Parallels to language modeling. To capture the relationships between accesses, we express the joint probability distribution ρ of a sequence of n accesses $y_1,...,y_n$ to objects in memory as a product of conditional probabilities — specifically, the probability of each access y_i, conditioned on the preceding accesses in the sequence $y_1,...,y_{i-1}$:

$$\rho(y_1,...,y_n) = \rho(y_1) \cdot \Pi_{i=1}^{n} \rho(y_i \mid y_1,...,y_{i-1}) \quad (1)$$

We borrow this formulation from natural language processing (NLP), where the distribution of each token in a sentence is conditioned on the preceding tokens. This joint probability distribution over tokens, classically defined as a *language model*, captures structural rules and semantic relationships between tokens in the vocabulary of a natural language. For example, given that the first three tokens in a sentence are "I wrote a", a language model should assign a very low probability to "the" as the next token since it is grammatically inconsistent; the existing context given the previously observed tokens should also make "book" a more likely next token than "sandwich", i.e., $p(\text{book} \mid \text{I, wrote, a}) > p(\text{sandwich} \mid \text{I, wrote, a})$. Access sequences follow similar implicit rules — for example, in the DLRM application, access y_i must be to an address in the i-th embedding table of the model, and specific pairs of addresses may co-occur in a sequence due to the data distributions underlying user requests, e.g., an e-commerce recommender system might observe that electronics are most popular with male users in their 20s.

If a language model can model the distribution of memory accesses, then inferring object-level accesses from page-level accesses is analogous to translation, where an input sentence $\mathbf{x}$ in the source language, modeled by the joint distribution ϕ, is used to generate a sentence $\mathbf{y}$ in the target language, modeled by ρ. In our setting, $\mathbf{x}$ is the adversary's observed sequence of accesses to coarse-grained memory pages, and $\mathbf{y}$ is the user's sequence of accesses to finer-grained objects within those pages. This lets us formulate the adversary's goal as a sequence-to-sequence translation task, i.e., learning the distribution of ρ conditioned on $\mathbf{x}$:

$$\rho(y_1,...,y_n|\mathbf{x}) = \Pi_{i=1}^{n} \rho(y_i \mid y_1,...,y_{i-1},\mathbf{x}) \quad (2)$$

We also draw parallels from the narrowing down of pages to objects to the disambiguation of homographs, i.e., words that are spelled the same but have different meanings. For example, the English word "right" can be translated into multiple words in French, and we decide which one depending on the phrase that it appears in: "right hand" ("droite") or "that is right" ("vrai"). In the language of page accesses, the most probable object access corresponding to a specific page can similarly be inferred based on the context provided by the

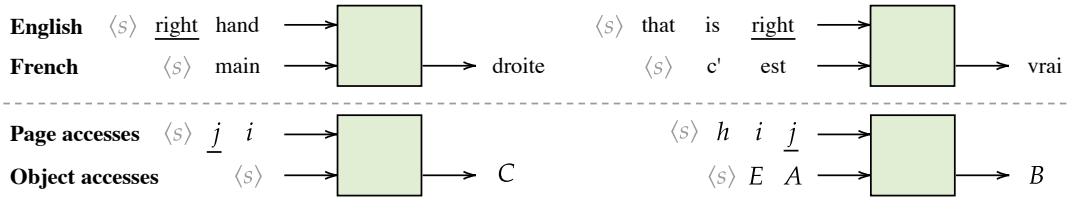

Figure 4: Like how homographs are disambiguated based on other words in the sentence, page-level accesses (underlined) can be disambiguated based on other pages in the access sequence and previously predicted object accesses. ⟨s⟩ denotes the start-of-sequence token.

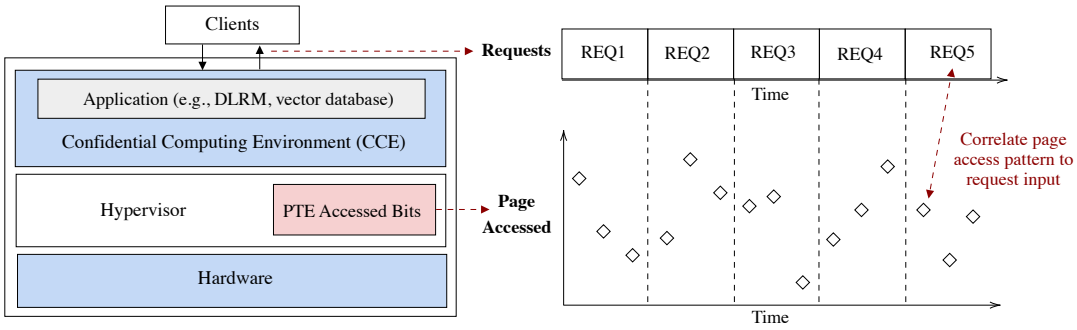

Figure 5: **Overview of FIT.** For each client request, the adversary monitors PTE accessed bits to learn the pages accessed by the application, then uses correlations between page accesses to infer the encrypted input of the request.

entire page access sequence in addition to the object access sequence predicted so far, as seen in Figure 4. The power of language models is in developing and applying this contextual awareness, which FIT's approach uses to infer fine-grained accesses with greater accuracy.

4.2 Attack Design

FIT targets privacy-sensitive applications introduced in §2.1, i.e., cloud-hosted inference serving and semantic search with HNSW. For inference serving, users send inference requests to the cloud server with sensitive information, e.g., the user's age group, location, or purchase history. The adversary aims to infer these private attributes by predicting the exact embedding table entries accessed by the model based on its observed page accesses. For semantic search, users send requests containing an encrypted query vector to the cloud server, and the adversary aims to infer the nodes accessed during the nearest neighbor search on the HNSW graph index. The general attack is summarized in Figure 5 and comprises the following stages.

4.2.1 Recording page accesses

In FIT, the application's accessed pages are recorded using the same approach as prior work [10] — monitoring accessed bits in PTEs. As described in §3.2, an adversary can identify the start of a user request, at which point, it clears the accessed bits for all memory pages allocated to the data structure it monitors. At the end of the request, the adversary checks and records the pages that have been marked as accessed.

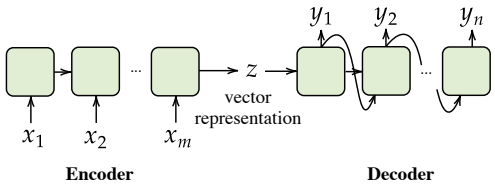

Figure 6: The encoder-decoder network architecture used by FIT translates a sequence of page accesses $x_1,...,x_m$ into a vector representation, which is then used to generate a sequence of finer-grained object accesses $y_1,...,y_n$. Each access y_i is generated based on the preceding accesses $y_1,...,y_{i-1}$.

4.2.2 Inferring private attributes

This attack component involves offline analysis of the victim application and its memory access patterns. The adversary uses a deep generative language model, described below, to infer mappings from the recorded page accesses to the underlying object accesses and, subsequently, the private attributes of each user request.

Deep generative modeling approaches use neural networks to achieve notable success in machine translation and other complex modeling tasks [9,77]. We base our method on recurrent encoder-decoder architecture [16,64], which consists of an encoder that processes the input sequence and extracts its salient information into a vector representation and a decoder that uses this representation to generate the output sequence, one element at a time; recent implementations use transformers as encoders and decoders to learn global dependencies between elements of the input and output sequences [69].

An overview of the model architecture FIT uses is shown in

Figure 6. The encoder network takes as input one-hot encoded elements of the page access sequence $\mathbf{x}$; the size of its input layer is equal to the total number of pages assigned to the monitored data structures (e.g., embedding tables in DLRM, HNSW adjacency lists in semantic search, etc.), such that each unique page activates a unique neuron in the layer. As the encoder sequentially processes each element of $\mathbf{x}$, it updates its hidden state, a multi-dimensional vector representation of the input seen so far; these recurrent updates incorporate the context of each element. The final hidden state at the output layer of the encoder, representing the context of the entire input sequence, is then passed to the decoder and used to generate a prediction of the object-level access sequence $\mathbf{y}$. Since each element of $\mathbf{y}$ is a one-hot encoded index of the accessed object, the decoder's output layer size equals the total number of objects accessed by the application. With each element the decoder generates, it also recurrently updates its hidden state to maintain the context of the input and output sequences generated so far. The decoder's final hidden state is thus a probability distribution over all objects conditioned on this preceding context, which is how it leverages the dependencies between elements to predict the most likely next access, as described in §4.1. Between the encoder and decoder, the space of vector representations captures the distribution of access sequences (of both pages and objects) along with the correlations within them.

Training. In the training phase of FIT, the adversary prepares a dataset of page access sequences and the private attributes, e.g., the corresponding embedding table entries accessed in DLRM, and the sequence of nodes accessed in HNSW. As described in §3.2, the adversary can collect these page accesses and the corresponding ground truth from a simulation of the victim application or an execution of the application in its own controlled enclave, e.g., with debug mode enabled to expose the virtual addresses accessed. Using this dataset, the adversary trains its encoder-decoder model to predict a sequence of object accesses $\mathbf{y} = y_1, ..., y_n$ given a sequence of page accesses $\mathbf{x} = x_1, ..., x_m$. As the encoder processes the page access sequences, it learns the distribution $\phi(\mathbf{x})$ of page-level accesses. Meanwhile, the decoder learns the distribution $\rho(\mathbf{y}|\mathbf{x})$ of object-level accesses, conditioned on the page-level access sequence that the adversary observes.

Inference. The inference phase occurs during the attack, when the real application executes within a CCE. Only the attested CCE instance can decrypt private user requests to the real application, and it hides in-memory objects and the processor state from the adversary's view. With only visibility to the victim application's page tables, the adversary records a trace of the pages accessed by the application during its execution. The adversary inputs these recorded page access sequences to the trained encoder-decoder model, which outputs the most likely sequence of object-level accesses underlying each observed page access sequence, sampled from its estimate of conditional distribution $\rho(\mathbf{y}|\mathbf{x})$ developed during training. The accuracy of the model's predicted sequences hinges on its ability to learn and exploit the correlations between object-level accesses, allowing it to identify specific combinations of objects (e.g., embedding table entries in DLRM or nodes in HNSW) accessed based on the observed combinations of pages accessed.

Application	Dataset source	# Train sequences	# Test sequences
DLRM	Kaggle [39]	1M	100K
LLM	DDXPlus [65]	500K	50K
HNSW	SIFT10K [35]	22.5K	2.6K

Table 2: Summary of datasets used.

5 Evaluation

We evaluate FIT on a cloud setup for real-world applications, workloads, and datasets, with the goal of understanding:

- FIT's accuracy in predicting object-level access sequences from observed page access sequences (§5.4) compared to prior state-of-the-art approaches (§5.2).
- FIT's practicality (§5.5), including its sensitivity to measurement error in page observations and the latency overheads of mounting it.

5.1 Applications, Workloads and Datasets

We evaluate the efficacy of FIT for the applications introduced in §2.1. These applications exhibit four properties that make them worthwhile targets: (i) they are widely used [49, 72, 84], (ii) involve privacy-sensitive data — e.g., user browsing and shopping behaviors for recommendations, medical information in prompts to a medical LLM, and image representations, such as faces, in a vector database — and (iii) access objects in memory with smaller-than-page granularity and (iv) long-term dependencies between accesses.

Deep learning-based recommendations. For this use case, we target the Deep Learning Recommendation Model (DLRM) [49] released by Meta Research, a state-of-the-art recommender system that has been the subject of prior studies on privacy leakage [27, 55]. As in earlier work, the model is trained on the Kaggle Display Ads dataset [39], containing the click-through rates of ads collected by Criteo Labs. Each row of the dataset corresponds to a display ad served by Criteo, consisting of 13 continuous features and 26 categorical features. As described in the example of §3.3, the model contains one embedding table per categorical feature — the semantics of these features have not been released, but each possible value of each categorical feature has a unique embedding table entry. The training split of the dataset, used

to train the DLRM, contains ad data collected over a 7-day period, along with an additional feature indicating whether the ad was clicked; the test split contains similar ad data collected the day after. We generate user inference requests from the test split of the dataset and collect page accesses over the embedding tables (as described in §4.2.1) for a single-threaded sequential execution of the DLRM model on these requests. The resulting dataset used to evaluate our attack contains for each inference request a pair of fixed-length sequences $\mathbf{x} = x_1, ..., x_{26}$ of virtual page numbers and $\mathbf{y} = y_1, ..., y_{26}$ embedding table entries, where y_i is the entry that was accessed on page x_i.

Medical LLM inference. We use a quantized version of BLOOM-560M [73] as the target LLM application. Inference requests to this application are generated by extracting patient symptoms from the DDXPlus [65] medical dataset and formatting them in a prompt template similar to ChatCAD [72, 82] for LLM-assisted diagnosis. The attack scenario is much like DLRM, except the LLM has one embedding table containing entries for all the language tokens in its vocabulary. When the application receives an encrypted request containing a medical diagnostic prompt, the adversary enters a loop of clearing and reading the accessed bits of the embedding table's PTEs, thereby capturing repeated accesses to the same page (caused by repeated tokens in the prompt). We record the pages accessed in this manner to create a dataset of page sequences $\mathbf{x}$ and their underlying embedding table entries $\mathbf{y} = y_1, ..., y_n$ for each prompt; in this case, n is the number of tokens in the prompt and varies for each sequence. Given auxiliary knowledge of the LLM's vocabulary, inferring the accessed embedding table entries allows the adversary to reconstruct the private prompt, token by token.

Semantic search. We evaluate FIT on a vector database populated and queried using the SIFT10K [35] dataset. SIFT10K consists of 10,000 base vectors stored in the database and 25,100 query vectors for which the database performs a nearest-neighbor search. The dataset is a smaller version of SIFT1M, a standard benchmark in the field of approximate nearest-neighbor search. We use the smaller database, as otherwise the base vectors would be accessed too sparsely to yield meaningful information about the joint distribution of access sequences. We use the Faiss [18] library's implementation of HNSW index construction and nearest-neighbors search, which stores the neighbors of each base vector in an adjacency list `neighbors`. Each time a base vector v_i is visited during the search, Faiss makes an access to `neighbors[i]`. Then, for each vector search request, the adversary, like in the LLM scenario, repeatedly checks the PTE accessed bits of this adjacency list to obtain the page access sequence $\mathbf{x}$. Following this attack setup, we create the dataset of $(\mathbf{x}, \mathbf{y})$, where $\mathbf{y} = y_1, ..., y_n$ is the sequence of base vectors accessed during the nearest-neighbors search for the query vector. Recovering this sequence of base vectors leaks information about the neighborhood of the private query vector; we leave the inference of further details, e.g., the identity of the query vector or its nearest neighbor, to future work.

5.2 Compared Attack Baselines

As discussed in §2.2, we compare FIT to two other page access pattern attack approaches described below.

Naive Bayes classifier. The Naive Bayes classifier baseline serves as a representative of attacks that can handle a many-to-one mapping of objects to pages but cannot leverage dependencies between accesses. The Naive Bayes classifier models the application's access distribution over the monitored data structure, assuming that accesses are independent. In the case of DLRM, where multiple embedding tables are accessed per inference request, we initialize a separate classifier for each table; given an input sequence of page accesses, the access corresponding to the ith embedding table is input into the ith classifier, and the classifier outputs are concatenated to produce the predicted sequence of object-level accesses. For the LLM and HNSW applications, only one Naive Bayes classifier is used to predict each object-level access over the embedding table and adjacency list, respectively.

IHOP. Complementing the Naive Bayes approach, IHOP is representative of attacks that can learn pairwise dependencies between accesses but cannot handle many-to-one object-to-page mappings. We are not yet aware of an attack that leverages dependencies across access sequences longer than two. We run IHOP with its recommended parameters [53] and format the inputs into our adaptation as follows. The adversary's auxiliary information $\tilde{\mathbf{F}}$ is computed from a transcript of concatenated object-level access sequences $\mathbf{y}$ from FIT's training dataset. To take the multiple embedding tables of DLRM into account, we encode the index of the entry with the index of the embedding table, as entry i of different embedding tables are likely to reside on different pages — this encoding is implicit in FIT, for which the kth access in the sequence is to the kth embedding table. Similarly, we build the matrix of observed pages $\mathbf{F}$ using concatenated page access sequences $\mathbf{x}$ from the testing dataset.

We also note that IHOP's feasibility is determined by the size of its quadratic optimization problem, which scales poorly with the number of objects and pages to assign to one another. With the limited vocabulary of the medical prompts in the LLM use case, only 754 unique embedding table entries are accessed over all inference requests, which is within IHOP's capabilities to solve. Similarly, for the HNSW workload, the optimization problem involving 10,000 base vectors is still computable, given significantly more time. However, DLRM's embedding tables can have up to ten million entries, so it is prohibitive for IHOP to predict the entire sequence of

accesses. Thus, our evaluations of IHOP on DLRM exclude from each access sequence the 12 largest embedding tables out of 26, resulting in 8,124 unique embedding table entries in the evaluation data to map to observed pages.

5.3 Attack Implementation

We collect page accesses for our evaluated applications (§5.1) in two CCEs: a Nitro Enclave on AWS and an SGX enclave, using Intel Ice Lake CPUs, on Microsoft Azure. We used NVIDIA A40 GPU servers to train our generative attack model and a 24-core AMD server to run IHOP.

Recording page accesses. Our page trace collection process follows the same broad strokes for both Nitro and SGX Enclave execution environments, based on the PTE monitoring approach introduced in prior work [10]: We spawn a spy thread that runs alongside the enclaved victim application, affinitized to a separate physical CPU core to minimize measurement noise. The spy thread runs our attack code (e.g., for page table manipulation) in kernel space by calling a custom kernel module. For each request to the application, the spy thread obtains a trace of page accesses by periodically preempting the enclave CPU, reading the application's PTE accessed bits, then clearing the bits and flushing the TLB. Like in prior attacks, the PTEs to check — i.e., the virtual address range of the embedding table entries — is known, as an OS-level adversary can inspect the application binary and manipulate memory allocations to ensure that data objects are located at specific addresses. Commonly used frameworks, such as PyTorch, also store the raw tensor data of the embedding table contiguously, allowing the adversary to determine the layout of the table entries across multiple pages. Between the execution environments of Nitro and SGX Enclaves, we find that the collected page traces are comparable, with only minor measurement errors (§5.5). We provide implementation details specific to each environment below.

Nitro Enclaves: On AWS, since we did not have control over the hypervisor and the extended page tables it manages, the spy thread in our implementation collects page access traces from *within* the Nitro VM. In a real-world attack deployment, the hypervisor would similarly collect these traces, as it can access the guest page tables that map guest virtual addresses (GVAs) to guest physical addresses (GPAs). Specifically, to check if a given GVA has been accessed within the guest VM, the hypervisor locates the guest page tables via the guest CR3 register and then walks the guest page tables to reach the desired page table entry (PTE).

SGX Enclaves: In the Azure VM, we mount our attack on Gramine [24], a library OS that allows unmodified applications to execute with their supporting libraries inside an SGX enclave. Gramine runtime software consists of a trusted part running inside the enclave that must interact with an untrusted part outside the enclave, which in turn interacts with the VM's OS to perform system calls. Using the SGX-Step framework [68], we patch Gramine's untrusted runtime to facilitate page trace collection. In particular, the enclave start function is modified to spawn the spy thread before entering the enclave's main function. The framework also sets up memory mappings for the registers of the local APIC timer device, allowing the spy thread to send inter-processor interrupts (IPIs) to the victim's CPU by writing to the configured memory-mapped address.

Language model for inferring private attributes. FIT uses miniature variants of the BERT transformer model [17] as the encoder and decoder in the model; the reduced transformer size decreases its memory usage while retaining performance in language tasks. Specifically, our implementation utilizes BERT-Tiny with L=2 transformer layers and a hidden embedding size of H=128 [6, 67]. We also initialize BERT with weights from natural language pre-training — we observed that our model with pre-trained weights converged faster than one with randomly initialized weights when training to predict access sequences. This is because sequence prediction is a classification-based task, and the pre-trained weights are already discriminative enough to predict (language) tokens in a sequence. As such, repurposing pre-trained weights for predicting accesses in the training phase improves the model's convergence compared to starting from scratch with randomly initialized weights and an initially flat output distribution, and training until the weights are sufficiently discriminative.

5.4 Attack Efficacy

We measure the accuracy of each compared attack by the Hamming distance between the attack's predicted object-level access sequence and the ground-truth object-level access sequence. Hamming distance counts the number of positions at which two sequences of equal length differ, i.e., the number of object-level accesses the attack predicts incorrectly, meaning that lower hamming distances are better. Figure 7 compares, for each application, the *cumulative frequency of normalized hamming distances* between the attacks' predicted sequences and the ground-truth sequences — for some normalized hamming distance d, the plot shows the number of sequences predicted by the attack that are within normalized hamming distance d from the ground-truth sequence. The hamming distances are normalized as sequences in LLM and HNSW vary in length; this does not affect the DLRM use case, where all sequences have a fixed length of 26.

First, we note the slight difference in attack efficacy between the two execution environments, visible for the DLRM case. This results from differing measurement errors during the page trace collection stage (i.e., incorrectly identified page accesses) in Nitro compared to SGX, which we discuss further in §5.5. The slight difference in observed error rate is

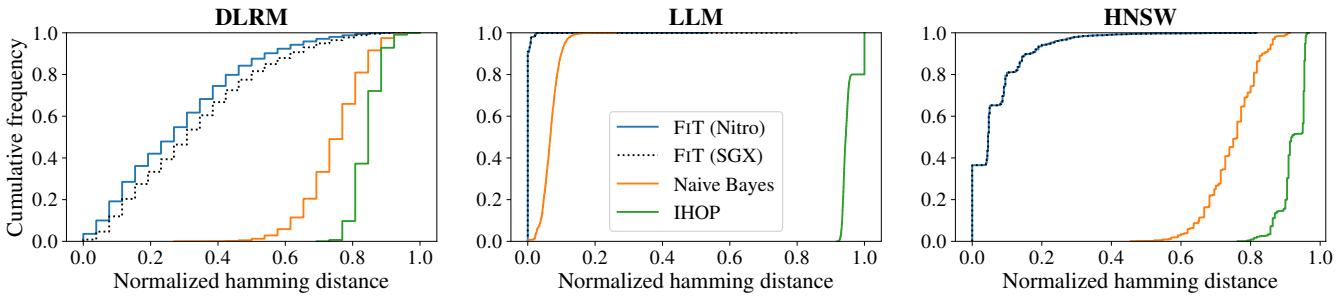

Figure 7: **Cumulative distribution of the normalized hamming distance between ground-truth and predicted sequences.** For all evaluated use cases, DLRM, LLM, and HNSW, FIT achieves lower (better) hamming distance on significantly more samples than other baselines.

attributed to platform-specific nuances: the Nitro Enclave runs a Linux OS image, while the SGX enclave inside a Linux VM additionally runs Gramine's library OS. We next discuss our key observations on how FIT compares to prior state-of-the-art attacks.

Leveraging both access dependencies and an understanding of many-to-one relationships between objects and pages greatly improves attack efficacy. FIT achieves lower hamming distances for all applications than the compared baselines by a significant margin. Its best performance is in the LLM use case, with a median hamming distance of 0 and 91% accuracy in predicting the entire sequence of tokens used. This is because the object-level accesses correspond to tokens from a limited medical vocabulary (754 tokens), resulting in such sparse accesses across the embedding table that 95.9% of observed page accesses in the training dataset correspond to only one embedding entry access. The size of an embedding table entry in the quantized BLOOM-560M model is also 1088 bytes, which limits the number of possible embedding entries in the rest of the pages to only 3. FIT is much more likely to predict the correct embedding entry given a page than in the DLRM use case, where an entry of any embedding table is 64 bytes. With DLRM, FIT achieves a median normalized hamming distance of 0.27, around 19 correct predictions out of 26. FIT's efficacy on HNSW falls in the middle, with a median normalized hamming distance of 0.05.

In contrast, the Naive Bayes baseline predictions are based solely on the frequency distribution of accessed objects, conditioned on the observed page. Since it considers each access in isolation, the simple Naive Bayes approach cannot leverage relationships between accesses in a sequence, achieving median normalized hamming distances of 0.77 for DLRM and 0.76 for HNSW use cases. However, this baseline achieves a low Hamming distance of 0.07 on LLM, due to the high percentage of one-to-one object-to-page mappings. In fact, the total percentage of embedding table entry accesses it predicts correctly overall is 93%, which closely matches the percentage of page accesses corresponding to a single embedding table entry. We attribute the remaining gap in performance

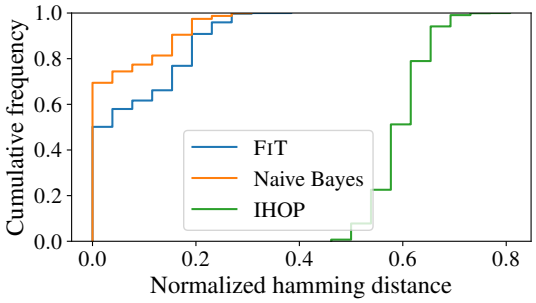

Figure 8: **Normalized hamming distance distribution given 1-to-1 page to embedding table entry mappings for DLRM.** IHOP accuracy improves significantly compared to the realistic 1-to-many case, but its inability to scale to large embedding tables prevents it from outperforming FIT and the Naive Bayes baseline.

between Naive Bayes and FIT to our approach's ability to leverage the natural language dependencies between tokens in each sequence.

As discussed earlier, IHOP suffers from having to glean fine-grained mappings from coarse-grained accesses. However, it can still map some pages to objects, achieving median normalized hamming distances of 0.85 for DLRM, 0.94 for LLM, and 0.92 for HNSW. Surprisingly, the unique object-to-page mappings in the LLM case do not improve IHOP's performance, possibly due to the setup or heuristics used to solve its optimization problem. We also note that IHOP has weaker assumptions on adversarial information, as it requires only *statistical* training information about page access patterns rather than ground-truth information about the underlying embedding table entries accessed for each sequence of page accesses. This ground-truth knowledge can feasibly be collected in the current use case; we intend to adapt our approach to weaker assumptions in future research (§7).

Without many-to-one object-to-page mappings, attack efficacy improves. To confirm that many-to-one mappings are the primary limiter on IHOP's efficacy, we evaluate the attacks on an idealized setup of DLRM such that each embedding table entry resides on a unique page. We note that such a scenario is unlikely in practice, as an OS adversary does not

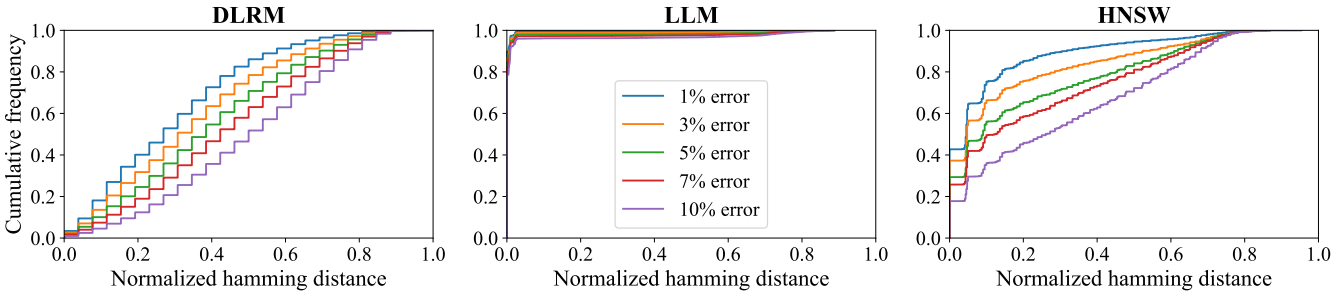

Figure 9: **Normalized hamming distance distribution for various error rates in observed page traces.** For all evaluated use cases, FIT's accuracy decreases with increasing error rate in page observations, but not significantly.

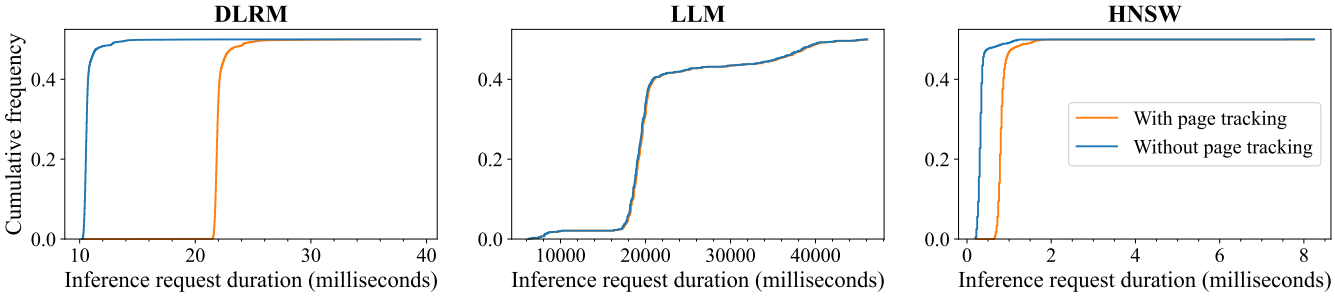

Figure 10: **Cumulative distribution of request duration.** The mean latency overhead added to each request by page tracking ranges from 1.01× for LLM to 2.53× for HNSW.

control how an application places its data across allocated memory, only which memory region(s) the application is allocated for a particular data structure. As shown in Figure 8, IHOP correctly predicts all entries in an access sequence for a significant portion of the testing set, barring the excluded embedding tables — the hamming distances being lower-bounded by 12. However, FIT also improves over the one-to-one object-to-page mapping, with the added advantage of handling larger tables. Naive Bayes performs the best because it stores the one-to-one object-to-page mapping seen in training and consults it during testing; if this scenario were possible, the adversary could indeed rely on this mapping without needing more sophisticated methods.

Without access dependencies, attack efficacy is upper-bounded. To understand the importance of access dependency information to our attack, we consider a scenario where accesses are independent. FIT and IHOP would reduce to the level of Naive Bayes, as the only available information would be from the frequency of page- and object-level accesses. They would perform no better than existing frequency analysis attacks that assume independent accesses [8, 11, 33].

5.5 Practical Considerations for the Attack

We now discuss the practical aspects of FIT, specifically the stages of recording page accesses and training the generative model. We focus on the Nitro Enclave executions of our evaluated use cases, but observe similar trends for SGX.

Sensitivity to measurement error. Our page trace collection process observes measurement errors for up to 1% of page accesses for DLRM in a Nitro Enclave and 1.4% in an SGX Enclave, and $< 1\%$ for LLM and HNSW on both platforms. When analyzing the trace of page observations, the adversary can encounter two classes of errors. The first class includes false positives: accesses to monitored pages that are not to the monitored data structure and stem from accesses to other application data within the same page range, e.g., the embedding table's underlying tensor metadata and methods, which may be stored on a different page from the embedding table entry being accessed. The second class includes false negatives, e.g., embedding table pages that are expected to be accessed but have their accessed bit unset, are more sporadic and may be due to background processes clearing the accessed bit [23]. When it is unclear which singular page was accessed for some object-level access, the adversary heuristically replaces the access with the first page of the monitored data structure. As we see next, the attacker model can tolerate higher error rates.

Figure 9 shows how the normalized hamming distances achieved by FIT are impacted by the error rate in the page tracking phase, which we increase by replacing a random sample of pages in the access sequences of the testing set. FIT's performance degrades gracefully as the error rate increases to 10%, at worst resulting in a median hamming distance of 0.5 for DLRM (half of the access sequence is correctly predicted). Meanwhile, LLM is minimally affected by injected errors, likely because the attacker can leverage the repetition of identical token sequences across all prompts due to their use of a

medical diagnostic template. We acknowledge that there are more sophisticated approaches to page tracking that provide information in a higher amount or fidelity, as introduced in prior work (§2.2). Our more straightforward approach captures a lower bound on the adversary's performance, as FIT is adaptable to noise and can only do better with higher-quality observations.

Latency overheads. We first consider the *online* latency overhead that the adversary's tracking and recording of page accesses adds to the duration of handling one client request on the cloud server. The adversary is motivated to minimize this latency overhead, making it less likely for clients to suspect an attack. As shown in Figure 10, the latency added by our page tracking implementation varies between use cases as a function of the number of PTEs that need to be monitored. In DLRM, 11 ms on average are added to the total inference time for each request, resulting in an overhead of 2.06×. In LLM, the embedding table spans a much wider address range, requiring on average 140 ms to access all its PTEs; however, this added latency is negligible compared to the overall inference time of ∼ 20 *seconds* per request, which is due to the LLM's slower execution on CPU. In contrast, HNSW has a mean request duration of 0.8 ms. While page tracking adds only 0.5 ms to the request duration, this translates to a latency overhead of 2.53×. Moreover, we note that all of our measurements exclude network latencies for processing a request, which would further reduce the relative impact of tracking page accesses.

Attack overheads also include the *offline* time needed to train the attack model and run inference on the recorded accesses. FIT's model for DLRM was trained on 1M sequences for 5 epochs, requiring about 43 GPU hours, and the IHOP attack takes approximately 25 hours given truncated access sequences. For LLM, FIT requires 6 GPU hours to train on 500K sequences for 30 epochs, while IHOP only takes 5 minutes to complete due to the small number of embedding table entries accessed over all inference requests. Finally, for HNSW, FIT takes less than one GPU hour to train for 40 epochs on 22.5K sequences, while IHOP takes 6 hours.

6 Related Work

We discussed prior page access pattern attacks in §2; we now discuss related works in other directions.

ML-assisted inference attacks. Machine learning (ML) is a powerful and increasingly popular tool for inferring secrets from information leakage, especially from noisy side channels. Prior side-channel attack papers have successfully leveraged ML at various levels of the cloud stack, including the operating system [81], network [7, 29, 57, 60], and low-level hardware [30, 37]. The closest to our work are attribute inference attacks that leverage ML models to infer private information about individuals from publicly available data [22, 62].

Exploiting correlations in data. Privacy analyses of data with statistical dependencies are emerging in the community. As discussed in §5.2, IHOP [53] introduces a statistical attack that leverages correlated queries to identify plaintext accesses to encrypted storage; MAPLE [36] uses a similar approach. Other works have found that attackers can exploit non-i.i.d. training data to enhance the accuracy of membership inference attacks [31] and steal secret model parameters in federated learning [75].

Embedding table attacks and defenses. Prior works have identified embedding table access patterns as an attack vector for learning private user information. A study on DLRM demonstrated that given the exact embedding table indices accessed during an inference request, an attacker could extract the identity of the user behind the request and their sensitive attributes, e.g., age and gender [27]. More recent works have proposed leakage mitigation techniques based on ORAM and private information retrieval protocols [40, 55].

Learning access patterns for prefetching. A language modeling approach has previously been applied to learning memory access patterns for cache and DRAM prefetching [13, 28, 79]. Prior works have trained LSTMs, a type of recurrent neural network, to learn sequences of accesses as a sentence and predict future accesses as the next "word" in the sentence. Recent approaches use more sophisticated encoder-decoder models to predict future *sequences* of memory accesses [47, 80].

Our task differs from the prefetching use case in that, rather than predicting the subsequent memory accesses, we need to *translate* a sequence of fine-grained accesses in the plaintext world from coarse-grained accesses in the encrypted world. Unlike a hardware-based component that must make online decisions about what to prefetch, our assumed adversary resides in software and can perform offline analysis to predict private attributes based on observed access patterns.

7 Discussion & Future Work

Our attack represents the first step toward understanding the privacy risks associated with correlated page access patterns over sensitive data. In this section, we discuss possible directions for future research.

Countermeasures. As state-of-the-art oblivious data access mechanisms [25] have been shown to be vulnerable to correlation-based access pattern attacks [53], it is crucial to develop performant defenses. This is further exacerbated as the other extreme — ORAM-based solutions — hide access correlations and are secure against FIT but incur poly-logarithmic bandwidth and latency overheads — much too high for most practical use cases [25]. We believe it is possible to develop

a practical countermeasure that hides application-specific access correlations, providing practical security without incurring ORAM-level overheads; exploring this solution space is a promising avenue for future work.

Adaptations of attack to other settings. While our study focuses on cloud-based CCE deployments with a privileged adversary, many similar real-world deployment settings are also vulnerable to correlation-based attacks, such as Searchable Symmetric Encryption (SSE). However, FIT's reliance on detailed ground-truth knowledge — e.g., knowledge of the internals of application-level object placement across OS-allocated memory — limits its application to such use cases. Adapting our approach to settings with weaker adversarial assumptions and limited auxiliary information will offer more insight into the implications of leaking correlational access patterns in such settings.

8 Conclusion

In this work, we have explored the design of a new page access pattern attack against cloud applications deployed within server-side CCEs. Our attack, FIT, draws insights from language modeling to leverage access correlations across pages and reconstruct sensitive objects in cloud applications, even when many objects may reside on the same page. Our evaluations show that FIT, on average, can accurately reconstruct $71.7-99.9\%$ of the object-level access sequences, given only coarse-grained page-level accesses. Our attack presents a realistic portrayal of vulnerabilities resulting from page access pattern leakages, motivating the design of new countermeasures that strike a practical balance between security and performance.

Acknowledgements

We thank the anonymous reviewers and the shepherd for their informative and valuable feedback. This work used the Delta system at the National Center for Supercomputing Applications through allocation CIS240310 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF awards #2138259, #2138286, #2138307, #2137603, and #2138296. This work was also supported in part by the NSF's awards #2047220, #2054957, #2112562, #2147946, and a NetApp Faculty Fellowship.

Ethics Considerations

Our ethical analysis considers the principles of "Beneficence" and "Respect for Persons". From a consequentialist perspective, making our novel attack approach known allows the security community to develop countermeasures and ultimately improve the privacy guarantees of cloud systems. Accordingly, our next step is to explore realistic threat scenarios for the proposed attack and practical mitigations. Although public disclosure of this attack may enable an adversary to carry it out, we argue that users and researchers will also be enabled to defend against it or opt for secure platforms where this attack cannot be mounted; thus, the benefits of this decision outweigh the potential harms.

We identify all users of cloud services as stakeholders. From a deontological perspective, we must respect their right to know the privacy risks associated with the services they use, making it the morally correct decision to share our knowledge of the attack. We have taken measures to respect people's privacy by deriving all experimental data from publicly available and anonymized datasets. The data collection portion of our attack was carried out on open-source applications and confined to AWS and Azure virtual machines under our control, causing no harm to other users of AWS and Azure.

Open Science

Adhering to the open science policy of USENIX Security, we have made our research artifacts available on Zenodo (https://doi.org/10.5281/zenodo.15602651) and GitHub (https://github.com/yale-nova/found-in-translation). We provide (i) source code for our access pattern attack, (ii) relevant datasets used in our evaluations, and (iii) documentation on setting up and running our experimental pipeline.

References

[1] AMD. AMD secure encrypted virtualization (SEV). https://www.amd.com/en/developer/sev.html, 2023.

[2] Ghous Amjad, Seny Kamara, and Tarik Moataz. Forward and backward private searchable encryption with sgx. In *Proceedings of the 12th European Workshop on Systems Security*, EuroSec '19, New York, NY, USA, 2019. ACM.

[3] ARM. Security technology building a secure system using TrustZone technology. *ARM Limited White Paper*, 2009.

[4] ARM. Introducing Arm confidential compute architecture. https://developer.arm.com/documentation/den0125/0100/Arm-CCA-Software-Architecture, 2021.

[5] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Fabio

Scotti, and Alessandro Piva. Privacy-preserving fingercode authentication. In *ACM Workshop on Multimedia and Security*, MM&Sec '10, page 231–240, New York, NY, USA, 2010. ACM.

[6] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in NLI: Ways (not) to go beyond simple heuristics, 2021.

[7] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, July 2019.

[8] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[9] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space, 2016.

[10] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page Table-Based attacks on enclaved execution. In *USENIX Security 17*, pages 1041–1056, Vancouver, BC, August 2017.

[11] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679, 2015.

[12] Anrin Chakraborti and Radu Sion. Concuroram: High-throughput stateless parallel multi-client ORAM. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[13] Chandranil Chakraborttii and Heiner Litz. Learning i/o access patterns to improve prefetching in ssds. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV*, page 427–443, Berlin, Heidelberg, 2020. Springer-Verlag.

[14] T-H Hubert Chan and Elaine Shi. Circuit opram: Unifying statistically and computationally secure orams and oprams. In *Theory of Cryptography Conference*, pages 72–107. Springer, 2017.

[15] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M. Sadegh Riazi. SANNS: Scaling up secure approximate k-Nearest neighbors search. In *USENIX Security 20*, pages 2111–2128, August 2020.

[16] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019.

[18] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library, 2024.

[19] Joshua J. Engelsma, Anil K. Jain, and Vishnu Naresh Boddeti. Hers: Homomorphically encrypted representation search. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(3):349–360, 2022.

[20] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In Ian Goldberg and Mikhail J. Atallah, editors, *Privacy Enhancing Technologies*, pages 235–253, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[21] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[22] Neil Zhenqiang Gong and Bin Liu. You are who you know and how you behave: Attribute inference attacks via users' social friends and behaviors. In *USENIX Security 16*, pages 979–995, Austin, TX, August 2016.

[23] Mel Gorman. *Understanding the Linux Virtual Memory Manager*. Prentice Hall PTR, 2004.

[24] Gramine library OS with Intel SGX support. https://github.com/gramineproject/gramine.

[25] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *USENIX Security 20*, pages 2451–2468, 2020.

[26] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 982–995, 2020.

[27] Hanieh Hashemi, Wenjie Xiong, Liu Ke, Kiwan Maeng, Murali Annavaram, G. Edward Suh, and Hsien-Hsin S. Lee. Data leakage via access patterns of sparse features in deep learning-based recommendation systems, 2022.

[28] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1919–1928. PMLR, 10–15 Jul 2018.

[29] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security 16*, pages 1187–1203, Austin, TX, August 2016.

[30] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.

[31] Thomas Humphries, Simon Oya, Lindsey Tulloch, Matthew Rafuse, Ian Goldberg, Urs Hengartner, and Florian Kerschbaum. Investigating membership inference attacks under data dependencies. In *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*, pages 473–488, 2023.

[32] Intel. Intel trust domain extensions (Intel TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html, 2023.

[33] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Ndss*, volume 20, page 12, 2012.

[34] Rishabh Jain, Scott Cheng, Vishwas Kalagi, Vrushabh Sanghavi, Samvit Kaul, Meena Arunachalam, Kiwan Maeng, Adwait Jog, Anand Sivasubramaniam, Mahmut Taylan Kandemir, and Chita R. Das. Optimizing cpu performance for recommendation systems at-scale. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. ACM.

[35] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[36] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Jamie DeMaria, Andrew Park, and Amos Treiber. MAPLE: MArkov process leakage attacks on encrypted search. Cryptology ePrint Archive, Paper 2023/810, 2023.

[37] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[38] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy*, pages 336–353, 2020.

[39] Criteo AI Lab. Kaggle display advertising dataset, 2018.

[40] Maximilian Lam, Jeff Johnson, Wenjie Xiong, Kiwan Maeng, Udit Gupta, Yang Li, Liangzhen Lai, Ilias Leontiadis, Minsoo Rhu, Hsien-Hsin S. Lee, Vijay Janapa Reddi, Gu-Yeon Wei, David Brooks, and Edward Suh. Gpu-based private information retrieval for on-device machine learning inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '24, page 197–214, New York, NY, USA, 2024. ACM.

[41] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using sgx. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2019.

[42] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[43] Alexander Marvi, Brad Slaybaugh, Ron Craft, and Rufus Brown. VMware ESXi zero-day used by Chinese espionage actor to perform privileged guest operations on compromised hypervisors. https://cloud.google.com/blog/topics/threat-intelligence/vmware-esxi-zero-day-bypass, 2023.

[44] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP '13*, 2013.

[45] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A

cryptographic inference service for neural networks. In *USENIX Security 20*, pages 2505–2522, August 2020.

[46] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy*, pages 279–296, May 2018.

[47] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, NetAI'18, page 48–53, New York, NY, USA, 2018. ACM.

[48] Krishna Giri Narra, Zhifeng Lin, Yongqin Wang, Keshav Balasubramanian, and Murali Annavaram. Origami inference: Private inference using hardware enclaves. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 78–84, 2021.

[49] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems, 2019.

[50] Olga Ohrimenko, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma. Observing and preventing leakage in mapreduce. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1570–1581, 2015.

[51] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious {Multi-Party} machine learning on trusted processors. In *USENIX Security 16*, pages 619–636, 2016.

[52] OpenAI. GPT-4 technical report, 2024.

[53] Simon Oya and Florian Kerschbaum. IHOP: Improved statistical query recovery against searchable symmetric encryption through quadratic optimization. In *USENIX Security 22*, pages 2407–2424, Boston, MA, August 2022.

[54] Learn Linux Project. Linux paging and swapping. https://www.learnlinux.org.za/courses/build/internals/ch05s03.

[55] Rachit Rajat, Yongqin Wang, and Murali Annavaram. Laoram: A look ahead oram architecture for training large embedding tables. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. ACM.

[56] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T. Lee, Hsien-Hsin S. Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 26–39, 2021.

[57] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security 17*, pages 1357–1374, Vancouver, BC, August 2017.

[58] Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. Efficient llm inference on cpus, 2023.

[59] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, page 317–328, New York, NY, USA, 2016. ACM.

[60] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1928–1943, New York, NY, USA, 2018. ACM.

[61] Mahdi Soleimani, Grace Jia, and Anurag Khandelwal. Weave: Efficient and expressive oblivious analytics at scale. In *USENIX OSDI*, 2025.

[62] Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models. *arXiv preprint arXiv:2310.07298*, 2023.

[63] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[65] Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. Ddxplus: A new dataset for automatic medical diagnosis, 2022.

[66] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations*, 2019.

[67] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models, 2019.

[68] Jo Van Bulck, Frank Piessens, and Raoul Strackx. SGX-Step: A practical attack framework for precise enclave execution control. In *2nd Workshop on System Software for Trusted Execution (SysTEX)*, pages 4:1–4:6. ACM, October 2017.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[70] Leonid Velikovich, Ian Williams, Justin Scheiner, Petar Aleksic, Pedro Moreno, and Michael Riley. Semantic lattice processing in contextual automatic speech recognition for google assistant. In *Interspeech 2018*, pages 2222–2226, 2018.

[71] Midhul Vuppalapati, Kushal Babel, Anurag Khandelwal, and Rachit Agarwal. SHORTSTACK: Distributed, fault-tolerant, oblivious data access. In *USENIX OSDI*, pages 719–734, Carlsbad, CA, July 2022.

[72] Sheng Wang, Zihao Zhao, Xi Ouyang, Qian Wang, and Dinggang Shen. Chatcad: Interactive computer-aided diagnosis on medical image using large language models. *arXiv preprint arXiv:2302.07257*, 2023.

[73] BigScience Workshop. Bloom: A 176b-parameter open-access multilingual language model, 2023.

[74] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344, 2019.

[75] Zuobin Xiong, Zhipeng Cai, Daniel Takabi, and Wei Li. Privacy threat and defense for federated learning with non-i.i.d. data in aiot. *IEEE Transactions on Industrial Informatics*, 18(2):1310–1321, 2022.

[76] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[77] Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. Variational neural machine translation. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 521–530, Austin, Texas, November 2016. Association for Computational Linguistics.

[78] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, Boston, MA, March 2017. USENIX Association.

[79] Pengmiao Zhang, Ajitesh Srivastava, Benjamin Brooks, Rajgopal Kannan, and Viktor K. Prasanna. Raop: Recurrent neural network augmented offset prefetcher. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '20, page 352–362, New York, NY, USA, 2021. ACM.

[80] Pengmiao Zhang, Ajitesh Srivastava, Anant V. Nori, Rajgopal Kannan, and Viktor K. Prasanna. Transformap: Transformer for memory access prediction, 2022.

[81] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and XiaoFeng Wang. OS-level side channels without procfs: Exploring cross-app information leakage on iOS. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[82] Zihao Zhao, Sheng Wang, Jinchen Gu, Yitao Zhu, Lanzhuju Mei, Zixu Zhuang, Zhiming Cui, Qian Wang, and Dinggang Shen. Chatcad+: Towards a universal and reliable interactive cad using llms. *IEEE Transactions on Medical Imaging*, 2024.

[83] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298, Boston, MA, March 2017. USENIX Association.

[84] Jinhao Zhu, Liana Patel, Matei Zaharia, and Raluca Ada Popa. Compass: Encrypted semantic search with high accuracy. Cryptology ePrint Archive, Paper 2024/1255, 2024.