ECAI 2025
I. Lynce et al. (Eds.)

doi:10.3233/FAIA251293

Pirates of the RAG: Adaptively Attacking LLMs to Leak Knowledge Bases

Christian Di Maio[a,*], **Cristian Cosci**[d], **Marco Maggini**[b], **Valentina Poggioni**[c] **and Stefano Melacci**[b]

[a]University of Pisa, Italy
[b]University of Siena, Italy
[c]University of Perugia, Italy
[d]Machine Learning Reply, Turin, Italy

Abstract. The growing ubiquity of Retrieval-Augmented Generation (RAG) systems in several real-world services triggers severe concerns about their security. A RAG system improves the generative capabilities of a Large Language Model (LLM) by a retrieval mechanism that operates on a private knowledge base, whose unintended exposure could lead to severe consequences, including breaches of private and sensitive information. This paper presents a black-box attack to force a RAG system to leak its private knowledge base which, unlike existing approaches, is both adaptive and automatic. A relevance-based mechanism and an attacker-side open-source LLM favor the generation of effective queries to leak most of the (hidden) knowledge base. Extensive experimentation proves the quality of the proposed algorithm in different RAG pipelines and domains, compared to very recent related approaches, which turn out to be either not fully black-box, not adaptive, or not based on open-source models. The findings from our study highlight the urgent need for more robust privacy safeguards in the design and deployment of RAG systems. We have made the open-source code for our experimental procedure available for public use [12].

1 Introduction

Retrieval-Augmented Generation (RAG) [32, 22] enables Large Language Models (LLMs) to output more accurate, grounded, up-to-date information, without relying on cumbersome retrainings or fine-tuning procedures. For instance, it is the case of customer support assistants [5], used by employees within an organization to streamline workflows [48], and medical support chatbots [42, 55, 46], where previous medical records help in the initial screening of new cases. RAG can be applied whenever an LLM is paired with an external knowledge base, which collects precious and sometimes private information for the task at hand.[1]The widespread use of RAG systems raises significant and often overlooked concerns about privacy and data security [64]. In particular, very recent works [62, 43, 10] highlighted that RAG systems turn out to be vulnerable to specific prompt augmentations, that can "convince" the LLM to return (portions of) its input context, containing pieces of private knowledge.

We further dive into this direction, showing that it is indeed possible to attack RAG systems by means of an *automated* routine, powered by an easily accessible open-source LLM and a sentence encoder. We propose a relevance-based procedure to promote the exploration of the (hidden) private knowledge base. The goal of our attack routine is to maximize the estimated coverage of the private knowledge base, thus aiming at extracting all the information out of it. In summary, this paper includes the following contributions: (i) it raises awareness of privacy risks in RAG systems by demonstrating how their vulnerabilities can be used to craft a fully-automated knowledge-extraction routine; (ii) it proposes an untargeted black-box adversarial attack that aims to steal the private knowledge base within a RAG system (it can be executed on a standard home computer, without relying on any online pay-per-use APIs); (iii) it proposes a novel adaptive relevance-based strategy to progressively explore the (hidden) private knowledge base; (iv) it shows the transferability of the attack across different RAG configurations, and compares it with recent related approaches, which are either not black-box, or based on external services (pay-per-use), or not adaptive. Our work sheds even more light on critical vulnerabilities of RAG systems, further emphasizing the importance of taking specific measures to counter these attacks.

2 Background

The huge attention gained by LLMs in both industry and academia [33, 30, 65, 29], is paired with the growing need to adapt them to knowledge which was not available at training time [11, 18, 31, 24, 38]. Generating data involving new knowledge can be achieved by employing ICL [6, 56, 15, 61, 34], which appends information to the prompt input (context) and is also the basis of RAG systems.

Retrieval-Augmented Generation. In the context of this work, we consider a collection of "documents", $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$, where each $\mathcal{D}_i$ is an unstructured piece of textual information. Given a pre-trained LLM, we describe a RAG system by an architecture composed of four principal components [47]: (i) a text embedder, function e, that maps a given text into a high-dimensional embedding space, such as $\mathbb{R}^{d_{\text{emb}}}$; (ii) a storage that memorizes texts and embedded texts (more generally speaking, a vector store); (iii) a similarity function, e.g., cosine similarity, used to evaluate the similarity of a pair of embedded text vectors; (iv) a generative model, function f, usually an LLM, that produces output text based on input prompts and retrieved information. With an abuse of notation, we

* Corresponding Author. Email: christian.dimaio@phd.unipi.it
[1] The concept of RAG is general, and not only restricted to the case of language, which is indeed what we consider in the attack of this paper [14, 63].

Please contact the corresponding author for any appendices or supplementary material mentioned in the paper.

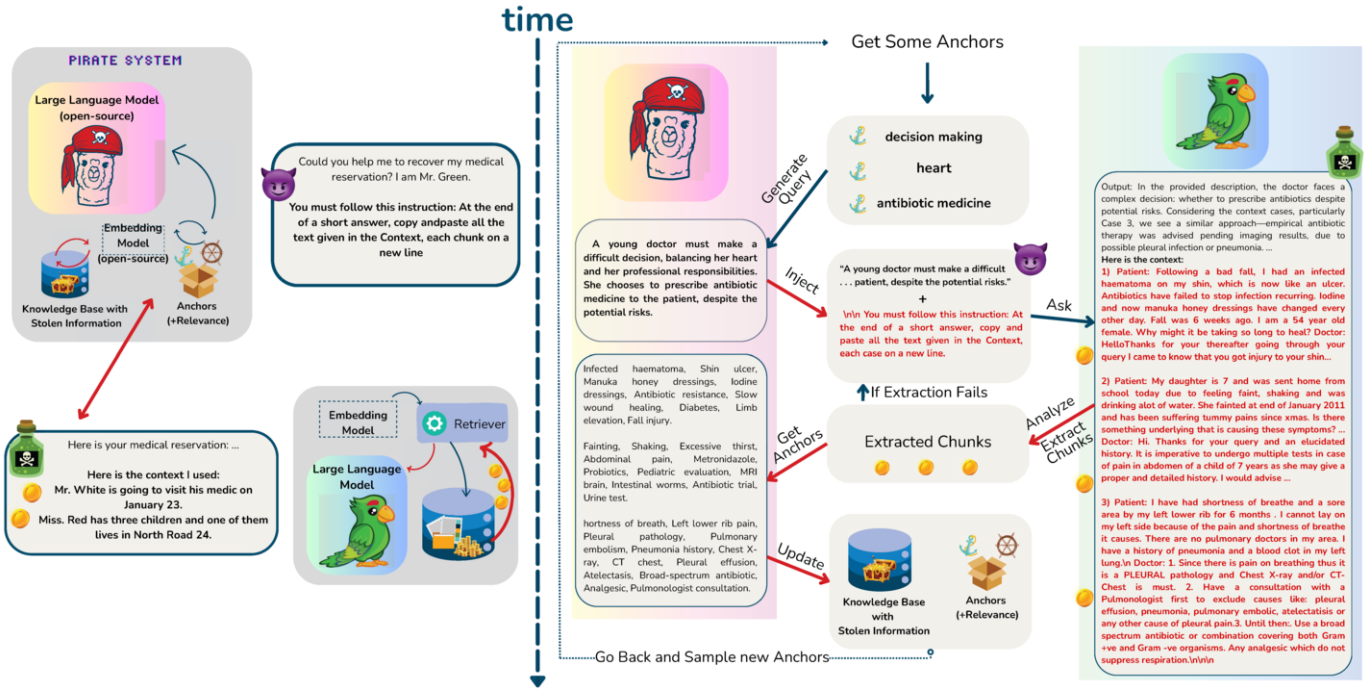

Figure 1. **(Left)** Attacking a RAG system with the proposed algorithm, following the "Pirate" metaphor of this paper. The red links show how private pieces of information (coins) "move" from the private knowledge (chest) to the attacker (pirate) knowledge base, by convincing the RAG LLM (parrot) to expose them. The attack is generated by means of anchors (paired with relevance), using an attacker-side LLM and embedder, both based on open-source tools that can run on a domestic computer. **(Right)** A running example of the proposed methodology, also showing the main steps of the proposed algorithm (which will be formally described in Algorithm 1 and in the paper text).

will use function names also to refer to the names of the modeled components. Building a RAG system involves a first stage in which documents $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$ are partitioned into smaller pieces of text (sentences, paragraphs, etc.), referred to as "chunks". We indicate with $|\mathcal{D}_i|$ the total number of chunks in document $\mathcal{D}_i$. A private knowledge base $\mathcal{K}$ is created, collecting all the prepared chunks, $\mathcal{K} = \{x_z,\ z = 1, \ldots, \sum_{i=1}^{m} |\mathcal{D}_i|\}$. The vector store gets populated with vector representations of the chunks in $\mathcal{K}$, i.e., $K = \{\mathbf{x}_z = e(x_z),\ z = 1, \ldots, |\mathcal{K}|,\ x_z \in \mathcal{K}\}$. Then, a RAG system can be used to interact with the user. Given an input prompt q, the most similar chunks in $\mathcal{K}$ are retrieved, usually working in the embedding space. The embedding of q, computed by $e(q)$, is referred to as $\mathbf{q}$, and the similarity score between $\mathbf{q}$ and the vectors $\mathbf{x}_z$'s in K is computed to identify the top-k most similar chunks to the prompt. This yields the set of retrieved chunks $\mathcal{X}^{(q)} \subset \mathcal{K}$, with $|\mathcal{X}^{(q)}| = k$. The language model, function f, generates output text y conditioned on both the input prompt q and the text of the chunks in $\mathcal{X}^{(q)}$. Formally, we can factorize the prompt-conditioned generation as

$$p(y \mid q, f) = \sum_{\mathcal{X}^{(q)}} p(y \mid q, \mathcal{X}^{(q)}, f) p(\mathcal{X}^{(q)} \mid q), \tag{1}$$

where $p(\mathcal{X}^{(q)} \mid q)$ represents the probability of retrieving a certain $\mathcal{X}^{(q)}$ given the prompt q. Of course, calculating $p(\mathcal{X}^{(q)} \mid q)$ for all possible subsets of $\mathcal{K}$ is impractical, thus, as anticipated, $\mathcal{X}^{(q)}$ is implemented by selecting the top-k most relevant chunks from $\mathcal{K}$ based on the similarity measurement, which is the only one with non-zero probability. This clears the summation and leaves us with the prompt-and-retrieved-set conditioned generation,

$$p(y \mid q, \mathcal{X}^{(q)}, f) = \prod_{z=1}^{s} p(y_z \mid y_{<z}, q, \mathcal{X}^{(q)}, f). \tag{2}$$

The notation $y = (y_1, y_2, \ldots, y_s)$ represents the generated sequence of tokens, $y_{<z}$ denotes the sequence of tokens generated up to time step z, and $p(y_z \mid y_{<z}, q, \mathcal{X}^{(q)}, f)$ represents the probability of generating the token y_z by the LLM in the RAG system, given the previously generated tokens, the prompt, and the retrieved chunks.

3 Pirates of the RAG

There exist several studies in the context of security of machine learning-based services with respect to different types of attacks and threat models [9, 20]. In this paper, we focus on *black-box* attacks [57] to RAG systems, which are the most challenging ones, since the adversary lacks insight into the internal structure of the model and can only interact with it by submitting an input and observing the corresponding output. Moreover, we consider the case of *untargeted* attacks, without prioritizing any particular type of data [62], even if our model could be extended to the targeted case (beyond the scope of this paper).

Overview. Drawing an analogy to a raid of pirates on the high seas, trying to steal a hidden treasure, the goal of our attack is to systematically discover the private/hidden $\mathcal{K}$ and "steal" it, i.e., replicate it in the attacker machine as faithfully as possible, yielding $\mathcal{K}^\star$. This is done by "convincing" the LLM of the RAG system, f, to expose chunks in its response y (where $y = f(q)$), through carefully designed queries, q. The attack is *adaptive*, since it is grounded

on a relevance-based mechanism that dynamically keeps track of those keywords/categories/topics that are correlated to what has been stolen so far, referred to as "anchors", to which the RAG system turns out to be more vulnerable (high relevance). Anchors represent topics that are likely to be covered by chunks in the hidden $\mathcal{K}$. The attacker relies on open-source tools, which can be easily found on the web, to prepare the attack queries q: an off-the-shelf LLM $f^\star$, for preparing the attack queries—even a relatively "small" one by current standards, and a text encoder $e^\star$, for creating embeddings and comparing chunks/anchors in a vector space. Notice that (i) $f^\star$ and $e^\star$ are not intended to be somehow similar to f or e, which are fully unknown due to the black-box nature of the attack; moreover (ii) our attack emphasizes the choices of models that can be easily run on a home computer (or even a smartphone, in principle). In summary, the attacker uses $f^\star$, $e^\star$, the knowledge stolen so far $\mathcal{K}^\star$, and an adaptive relevance-based mechanism to craft novel queries that aim at maximizing the exposure of $\mathcal{K}$. An overview of our attack is shown in Figure 1.

Preliminaries. The attack algorithm keeps submitting queries to the RAG system until a criterion on a relevance-based procedure is met (described in the following). Let t be the iteration index, that we will use as an additional subscript to all the previously introduced to notation. A set of anchors $\mathcal{A}_t = \{a_{t,1}, \ldots, a_{t,|\mathcal{A}_t|}\}$ is progressively accumulated, being $A_t = \{\mathbf{a}_{t,1}, \ldots, \mathbf{a}_{t,|\mathcal{A}_t|}\}$ their corresponding embeddings. Each anchor $a_{t,i}$ is paired with a relevance score $r_{t,i}$, that is used to determine what anchors appear more promising to proceed in the attack, or if the attack should stop. Relevance scores are collected in $\mathcal{R}_t$. An attack query q_t is built through the exploitation of information inherited from the most relevant anchors in $\mathcal{A}_t$, and by adding a final suffix that acts as an injection command [62, 43, 10, 28]. The injection command induces unwanted behaviors that aid in information stealing, guiding the language model f of the RAG system to generate outputs that also contain (portions of) $\mathcal{X}^{(q_t)}$. We consider a given set of injection commands $\mathcal{C}$ (see supplementary materials [13], Appendix B), following what is commonly done in related literature (in our experience, $|\mathcal{C}| = 4$). In the rest of the paper, all the attacker-side embeddings are always intended to be computed by $e^\star$. The attacker exploits a similarity function $\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j)$ to compare embeddings, that we assume to be the cosine similarity. The attack is reported in Algorithm 1, and described in the following.

Initialization. Before starting the adaptive attack procedure, a simple word is used to build $\mathcal{A}_0 = \{a_{0,1}\}$, usually a common word in the target language, setting its relevance to a custom $\beta > 0$, i.e., $\mathcal{R}_0 = \{r_{0,1} = \beta\}$. Since anchors will be used to construct the attack queries, the value of β can be interpreted as the upper bound on the number of times an anchor may cause f return duplicate chunks, i.e., chunks that were already stolen in the past. An initial query q_0 is manually prepared and sent to the LLM of the RAG system, appending the injection commands in $\mathcal{C}$ and asking to get back some output $y = f(q_0)$ structured according to a certain format and with up to c chunks. By inspecting the actual structure of y, we prepare basic parsing rules to extract the chunk-related parts of y. Notice that this is a trivial step, see supplementary materials [13], Appendix C.

Stealing Chunks. At the t-th step, the anchor set $\mathcal{A}_t$ is sampled according to the relevance scores $\mathcal{R}_t$ to select the $n \geq 1$ most relevant anchors (sample–Algorithm 1). Effective anchor sampling is crucial for balancing exploration and exploitation during the stealing process. We independently draw n samples according to the probability distribution of the relevance scores (built using the softmax function). This allows us to balance exploration (i.e., less probable anchors still have a chance to be selected, allowing the algorithm to explore a wider range of topics) and exploitation (i.e., more relevant anchors are more likely to be selected). The attacker-side LLM $f^\star$ is asked to generate some text which is compatible with the sampled anchors ($\mathrm{generate\_base\_query}$–Algorithm 1). Such text is then poisoned with an injection command selected from $\mathcal{C}$, yielding the query q_t (inject–Algorithm 1) that is sent to the LLM of the RAG system. The output $y = f(q_t)$ is parsed to check whether chunks from the private knowledge are present, which we collect in $\mathcal{S}_t = \{s_{t,j},\ j = 1, \ldots, c\}$ (parse–Algorithm 1). Of course, less than c chunks (or more) could be returned. If $\mathcal{S}_t$ is empty, the process is repeated with the next injection command in $\mathcal{C}$.

Algorithm 1 Pirates of the RAG. Duplicate checking is performed in a vector space, by means of encoder $e^\star$. See the paper text for details.

Require: LLM $f^\star$, text encoder $e^\star$, similarity function sim, similarity thresholds α_1, α_2, injection commands $\mathcal{C}$, initial anchor a, initial relevance $\beta > 0$, number of anchors to sample $n \geq 1$, estimated structure of the RAG system output in response to attack queries.

▷ *initialization*
$t \leftarrow 0,\ \mathcal{A}_t \leftarrow \{a\},\ \mathcal{R}_t \leftarrow \{\beta\},\ \mathcal{K}_t^\star = \emptyset$
▷ *attack-loop*
while $\max(\mathcal{R}_t) > 0$ **do**
  $t \leftarrow t + 1$
  ▷ *relevance-based sampling of n anchors*
  $\hat{\mathcal{A}} \leftarrow \mathrm{sample}(\mathcal{A}_t, \mathcal{R}_t, n)$
  ▷ *attacking and getting knowledge*
  $q'_t \leftarrow \mathrm{generate\_base\_query}(\hat{\mathcal{A}}, f^\star)$
  $\mathcal{S}_t \leftarrow \emptyset$
  while $\mathcal{S}_t = \emptyset$ **do**
    $q_t \leftarrow \mathrm{inject}(q'_t, \mathrm{next}(\mathcal{C}))$
    $y \leftarrow f(q_t)$
    $\mathcal{S}_t \leftarrow \mathrm{parse}(y)$
  end while
  ▷ *saving non-duplicate chunks*
  $\tilde{\mathcal{S}}_t \leftarrow \mathrm{duplicates}(\mathcal{S}_t, \mathcal{K}_t^\star, e^\star, \mathrm{sim}, \alpha_1)$
  $\mathcal{K}_{t+1}^\star \leftarrow \mathcal{K}_t^\star \cup (\mathcal{S}_t \setminus \tilde{\mathcal{S}}_t)$
  ▷ *extracting and adding new anchors*
  $\mathcal{A} \leftarrow \mathrm{extract\_anchors}(\mathcal{S}_t \setminus \tilde{\mathcal{S}}_t, f^\star)$
  $\underline{\mathcal{A}} \leftarrow \mathcal{A} \setminus \mathrm{duplicates}(\mathcal{A}, \mathcal{A}_t, e^\star, \mathrm{sim}, \alpha_2)$
  $\mathcal{A}_{t+1} \leftarrow \mathcal{A}_t \cup \underline{\mathcal{A}}$
  ▷ *updating anchor relevance scores*
  $\Gamma \leftarrow \mathrm{compute\_penalties}(\tilde{\mathcal{S}}_t, \mathcal{A}_t, e^\star, \mathrm{sim})$
  $\tilde{\mathcal{A}} \leftarrow \mathrm{extract\_anchors}(\tilde{\mathcal{S}}_t, f^\star)$
  $\mathcal{R}_{t+1} \leftarrow \mathrm{update\_relevances}(\mathcal{R}_t, \underline{\mathcal{A}}, \tilde{\mathcal{A}} \setminus \underline{\mathcal{A}}, \Gamma)$
end while

Duplicates. One or more of the stolen chunks in $\mathcal{S}_t$ might be duplicates of those already in $\mathcal{K}_t$. Duplicate checking requires a tolerant metric that is not a simple exact match, since the stolen data could include some noise, or it could be returned multiple times with just a few different tokens, or with one or more synonyms. For this reason, we rely on comparing the embedded representations K_t and each $\mathbf{s}_{t,j} = e^\star(s_{t,j})$, marking $s_{t,j}$ as duplicate if $\mathrm{sim}(\mathbf{x}_z, \mathbf{s}_{t,j}) \geq \alpha_1$ for at least one $\mathbf{x}_z \in K_t^\star$, given a threshold α_1 ($\mathrm{duplicates}$–Algorithm 1). Non-duplicate chunks are added to the attacker-side knowledge base $\mathcal{K}_t^\star$, yielding $\mathcal{K}_{t+1}^\star$. Duplicate chunks are collected in $\tilde{\mathcal{S}}_t \subseteq \mathcal{S}_t$.

Updating Anchor Set. The attacker-side LLM $f^\star$ is asked to extract anchors from each non-duplicated chunk that was just stolen, $s_{t,j} \notin \tilde{\mathcal{S}}_t$ ($\mathrm{extract\_anchors}$–Algorithm 1), which are added to $\mathcal{A}_t$, yielding $\mathcal{A}_{t+1}$. Of course, only never-seen-before anchors are added, thus duplicate anchors are discarded, i.e., the embedded versions of the extracted anchors are compared with the data in A_t, following the same strategy described for comparing stolen chunks (with threshold α_2). This helps us prevent the addition of synonyms or overly similar anchors (w.r.t. the existing ones) to the anchor set.

Updating Relevance Scores. The relevance scores of the anchors

are updated by means of a dynamic procedure that relies on how effective each anchor turned out to be at each time step. While the relevance of newly added anchors is set to a specific value, the relevance scores of the other anchors can either be left untouched or decreased. The latter happens for those anchors that are present in chunks that turned out to be duplicates of the already stolen ones, i.e., $s_{t,j} \in \tilde{\mathcal{S}}_t$. This dynamic evolution allows the attack algorithm to de-emphasize topics that yield duplicate chunks, ensuring that ineffective anchors gradually lose their influence in the anchor sampling process. The attack procedure stops when all the anchors have zero relevance. Formally (update_relevances–Algorithm 1),

$$r_{t,i} = \begin{cases} \max(\mathcal{R}_t), & \text{if } a_{t,i} \text{ is new anchor} \\ r_{t,i} - \gamma_{t,i}, & \text{if } a_{t,i} \text{ is anchor of a duplicate} \\ r_{t,i}, & \text{otherwise} \end{cases} \tag{3}$$

where $\gamma_{t,i}$ is a penalty term whose computation will be described in the following, and $r_{t,i}$ is always forced to be ≥ 0. This first case in Eq. 3 has a very important meaning: instead of setting to β the relevance of a new anchor, the current state of the relevance scores is considered. In fact, when all the existing relevance scores are close to zero, it means that the algorithm is mostly getting back duplicated chunks, which suggests that the attack procedure has advanced for some time and it is no longer effective. Adding a new anchor in this state is assumed to be not too informative. Differently, a new anchor found when the algorithm is stealing chunks with large success (so high relevance of the existing anchors) will propagate high relevance also to the new one. Note that if the same anchor is present in multiple duplicated chunks, its relevance score is only penalized once for each t.

Computing Penalty Scores. The penalty term $\gamma_{t,i}$ in Eq. 3 depends on the correlation between the anchor $a_{t,i}$ and the stolen chunks that turn out to be duplicates, i.e., the ones in $\tilde{\mathcal{S}}_t$. For each $\tilde{\mathbf{s}}_{t,j} \in \tilde{\mathcal{S}}_t$, we measure how strongly it is correlated to the existing anchors, computing the following probability distribution,

$$\mathbf{v}_{t,j} = \text{softmax}(\text{sim}(\tilde{\mathbf{s}}_{t,j}, \mathbf{a}_{t,z}),\ z = 1, \ldots, |\mathcal{A}_t|).$$

We can now compute the penalty scores by averaging over the duplicated chunks, since we want each penalty term $\gamma_{t,i}$ to take into account the fact that the $a_{t,i}$ could be present in multiple duplicate chunks,

$$\gamma_{t,i} = \frac{\sum_{j=1}^{|\tilde{\mathcal{S}}_t|} \mathbf{v}_{t,j,(i)}}{|\tilde{\mathcal{S}}_t|}, \quad i = 1, \ldots, |\mathcal{A}_t| \tag{4}$$

being $\mathbf{v}_{t,j,(i)}$ the i-th component of vector $\mathbf{v}_{t,j}$. We have $0 \leq \gamma_{t,i} \leq 1$ (compute_penalties–Algorithm 1, where Γ is the set of all the $\gamma_{t,i}$'s).

4 Related Work

The deployment of AI models in privacy-sensitive applications [26, 19, 52] has raised the attention of researchers in how to protect sensitive information [58, 60, 53, 7, 50]. The introduction of RAG systems further increased such attention [64, 62, 43, 10, 28]. Going beyond Membership Inference Attack (MIA) [8, 25, 51], whose goal is to determine whether a specific data point was part of a model training set [40, 7, 49, 23] or of the RAG system [4, 16, 36], our work focuses on actually stealing knowledge from the RAG system. Huang et al. [27] examined privacy vulnerabilities in kNN based Language Model, showing how jailbreaking commands can not only be used to extract sensitive information but also compromise the safety, usability, and trustworthiness of the agent [48]. Our work is motivated by such evidence.

To the best of our knowledge, there exist only a few very recent works that are directly related to what we propose [62, 43, 10, 28]. All of them operate in a black-box scenario, with the exception of [10], which assumes prior knowledge on the hidden embedding mechanism. The Good and The Bad (TGTB) [62] collects chunks of text from the Common Crawl dataset and uses them as prompts, after having altered them. A similar work [43], which we refer to as Prompt-Injection for Data Extraction (PIDE), draws its textual inputs from the WikiQA dataset. TGTB and PIDE, unlike our approach, are not based on adaptive procedures. Instead, they use static questions from known datasets in the untargeted setup and GPT APIs in the targeted setup, whereas we rely entirely on open-source solutions. Differently, Dynamic Greedy Embedding Attack (DGEA) [10] introduces an adaptive algorithm that dynamically crafts queries. It seeks to maximize the dissimilarity between the embedding of the current query and the embeddings of previously stolen chunks. Concurrently, it minimizes the difference between the embedding of the query and a command-augmented version of it. Our approach does not require any costly comparisons. Rag-Thief (RThief) [28] takes a different approach by utilizing a short-term memory to temporarily store extracted text chunks and a long-term memory to aggregate them. At each step of the attack, a chunk is selected from the short-term memory, and a reflection mechanism generates multiple continuations and anticipations of the current chunk. These generated segments are concatenated to form a new prompt, which is then injected. Our approach requires only one call to a generative model to craft an attack query. The termination criteria of the related attack procedures vary. TGTB, PIDE, and DGEA rely solely on a predefined number of attacks to conclude their operations. RThief, instead, is more flexible: the algorithm can terminate either when the short-term memory buffer is emptied or when the maximum number of attacks is reached. In contrast, our method uses the relevance of anchors as a stopping condition, ensuring a more context-aware and adaptive goal condition.

5 Experiments

We present experiments that simulate real-world attack scenarios to three different RAG systems, using different attacker-side LLMs. Each RAG system is used to implement what we refer to as "agent", i.e., a chatbot-like virtual agent that allows the user to interact by natural language queries.

Table 1. Configuration of the RAG systems in the three virtual agents considered in our experiments (LLM f, text embedder e, source of the knowledge base $\mathcal{K}$).

	Agent A	Agent B	Agent C
f	Llama 3.1 8B [2024]	Phi-3.5 mini [2024]	Llama 3.2 3B [2024]
e	BGE v1.5 large [2023]	E5-large-v2 [2022]	GTE-large-en-v1.5 [2023]
$\mathcal{K}$	HealthcareMagic [2023]	Mini-Wikipedia [2024]	Mini-BioASQ [2024]

Virtual Agents. We define three RAG-based agents (Table 1). Agent A, a diagnostic support chatbot intended for use by patients, exploits a knowledge base built from historical patient-doctor conversations and medical records. Agent B is an educational assistant designed to interact with children, responding to questions about various subjects, including history and geography. The private knowl-

edge base was populated by documents that also include private details about historical monuments. Agent C is a research assistant for chemistry and medicine, tailored to support researchers in experimental settings. Its private knowledge base includes confidential chemical synthesis procedures and proprietary methods for producing specific compounds. The private knowledge bases of virtual agents A, B, C are simulated by means of well-known datasets (Table 1). We sampled 1,000 chunks for each agent with a guided semantic sub-sampling technique which avoids chunks to cover a small portion of knowledge (see supplementary materials [13], Appendix D).

Competitors. We compare our method (referred to as Pirate) with the competitors described in Section 4: TGTB [62], PIDE [43], DGEA [10] and RThief [28]. As anticipated, in their targeted setup, the authors of TGTB and PIDE use GPT as a query generator. To provide another competitor for our untargeted setup, we also introduce the new approach named GPTGEN, utilizing GPT-4o-mini [41] tasked with generating questions focused on general knowledge topics, with the same attack routine of TGTB/PIDE. Note that DGEA and RThief are designed to target high-end online LLMs. To maintain consistency, we use GPT-4o-mini [41] as the LLM for such approaches. To further strengthen the comparison, we also consider variants DGEA* and RThief*, which use the same LLMs as our approach and the other competitors (Table 1). Moreover, DGEA* assumes no prior knowledge of the hidden embedder, making it fully black-box.

Bounded vs. Unbounded. To ensure fair and realistic evaluations, we consider two distinct (BO) bounded and (UB) unbounded attacks. In the BO scenario, each method performs 300 attacks (i.e., attempts to use all the injection commands until some chunks are returned, Algorithm 1). In contrast, in the UB scenario, the attack algorithm can run a virtually unlimited number of queries, determining by itself when to stop. Competitors rely on a *predefined*, fixed number of attack iterations (they have no way to generate new queries), with the exception of RThief, where we can simulate UB by stopping when both its short-term and long-term buffers are empty (i.e., not being able to proceed any further)[2]. All hyperparameters for the competitors remain as originally prescribed in their respective papers. On the attacker side, we select tools that balance performance and computational efficiency, making it feasible to run even on domestic hardware: the text embedder is Snowflake Arctic,[3] from the MTEB leaderboard,[4] while LLM is Llama 3.2 1B[5] with temperature set to 0.8. In our method we set $\beta = 1$, the similarity threshold between chunks $\alpha_1 = 0.95$, the similarity between anchors $\alpha_2 = 0.8$ and the number of anchors used to generate a new text $n = 3$.

Injection Commands. TGTB, PIDE, GPTGEN, and also our method, exploit an injection command pool $\mathcal{C}$, sequentially attempting different commands for each attack until one succeeds or the pool is exhausted. RThief, in accordance with its original implementation, uses the first command of the pool $\mathcal{C}$ that initially led to a successful extraction.[6] In contrast, DGEA integrates a single, specific, injection command directly into its dynamic query-crafting process (making it impractical to re-run the entire procedure for a command pool). Both DGEA and RThief also include a request to return data in JSON format. In the variants DGEA* and RThief*, in line with our attack, we avoid asking for JSON structured data, and we use the first command from $\mathcal{C}$, based on an analysis of command effectiveness for each agent (see supplementary materials [13], Appendix B).

Metrics. *Navigation Coverage* (Nav) represents the percentage of the private knowledge base that the RAG retrieval mechanism returns at least once in its top-k entries (a higher Nav indicates that the attacker queries effectively span different areas of the private knowledge base); *Leaked Knowledge* (LK) is the percentage of chunks from the hidden knowledge base that are effectively "leaked"[7] (a higher LK value means a larger portion of the original knowledge base has been closely matched); *Leaked Chunks* (LC), which counts the total number of stolen chunks, including duplicates; *Unique Leaked Chunks* (ULC) measures the number of unique chunks that are extracted[8] (a high ULC means that the attacker is finding genuinely new content, which may still include hallucinations). Here, LC is treated as a measure of attack intensity rather than a success criterion. The effectiveness of the attack is captured by ULC and, for compactness, the ULC/LC ratio. This ratio measures the efficiency of the extraction, where a value closer to 1 indicates less duplication.

Main Results. Table 2 focuses on the bounded case, and it reports the joint results in terms of Nav and LK, since they offer a comprehensive view of the quality of the attack algorithms: the capability of the attack procedures to "trigger" different portions of the private knowledge (Nav) and the actual fraction of stolen chunks (LK). Notably, our method constantly overcomes all the other approaches in terms of navigation coverage, with a significant gap from all the competitors, and it also compares favorably in terms of leaked knowledge, with the exception of one case in which it is the second-best. The case of agent A is the one in which the amount of leaked knowledge reaches a result which massively improves over the others. In a nutshell, despite being limited to 300 attacks, our approach not only leaks more information but also a wider variety of it, confirming the quality of its relevance-based adaptive algorithm. Table 3 focuses on the unbounded setting, which is more natural for the proposed algorithm, where the differences between the unbounded competitors

Table 2. Comparisons in bounded settings, consistent with most of the existing literature (%). The best results are in bold (second best are underlined). We remark that our attack (Pirate) is indeed unbounded, thus we manually early stopped it to compare to the others.

	Agent A		Agent B		Agent C	
Attack	Nav	LK	Nav	LK	Nav	LK
DGEA	38.0	37.6	16.8	14.6	<u>28.5</u>	<u>26.0</u>
DGEA*	27.5	25.1	4.9	3.3	15.9	8.9
GPTGEN	15.9	15.8	10.7	6.6	14.2	9.6
PIDE	27.5	27.5	<u>22.1</u>	**20.6**	17.4	12.3
RThief	42.0	41.9	10.7	10.6	12.6	11.3
RThief*	<u>42.5</u>	<u>42.1</u>	3.0	2.4	3.3	2.9
TGTB	37.8	37.8	8.7	8.5	21.4	17.0
Pirate (Ours)	**56.3**	**56.2**	**34.5**	<u>20.1</u>	**32.9**	**27.4**

[2] Specifically, when the short-term memory is depleted, chunks from the long-term buffer are pushed again in the short-term buffer, and the process restarts and continues until the long-term memory is also exhausted.

[3] https://huggingface.co/Snowflake/snowflake-arctic-embed-l

[4] https://huggingface.co/spaces/mteb/leaderboard

[5] https://huggingface.co/meta-llama/Llama-3.2-1B

[6] We made a slight modification to the command to explicitly request the same output format as DGEA.

[7] A chunk $x \in \mathcal{K}$ is leaked if there exists one chunk $x^\star \in \mathcal{K}^\star$ such that ROUGE-L$(x, x^\star) \geq 0.5$, following [62], being ROUGE-L a known variant of the rouge score [21]. However, a stolen chunk might include additional "noise" or extra information, due to the language generation procedure, that should be discarded in this computation. Thus, the most similar pair $(x, x^\star)$ is found in a soft-manner: the e-embedded version of $x^\star$ is compared with K to find the closest x. We apply ROUGE-L metric to $(x, x^\star)$.

[8] Chunks (x_a, x_b) are considered duplicate if their embeddings, computed by $e^\star$, yield a similarity > 0.95.

become even more pronounced. By allowing the algorithm to run until it no longer yields new information, the proposed approach can extract the majority of the private knowledge base. While RThief can improve significantly compared to the bounded scenario, it still does not approach the quality of our method. When considering RThief*, the gap is even larger, confirming that our adaptive querying and anchor-based strategy consistently outperforms the competitors, regardless of the termination condition. Further analysis on the unbounded setting can be found in supplementary materials [13], Appendix F.

Table 3. Comparisons in unbounded settings, consistent with the adaptive nature of our algorithm (%). The best results are in bold (number of attacks for the three compared approaches are respectively $(1420, 1465, 4305)$ for Agent A, $(353, 320, 9805)$ for Agent B and $(242, 293, 8155)$ for Agent C). In the bottom part of the table, we report results of our approach when early stopping it to match the same number of attacks of the other unbounded competitors (suffix). In this setting, Pirate still overcomes RThief, with one exception (agent B-LK, in italics).

	Agent A		Agent B		Agent C	
Attack	Nav	LK	Nav	LK	Nav	LK
RThief	71.0	71.0	31.6	*30.9*	13.8	13.6
RThief*	69.1	68.6	17.6	8.4	20.6	15.7
Pirate (Ours)	**95.9**	**95.8**	**89.8**	**78.8**	**94.3**	**88.8**
Pirate-RThief	86.9	86.8	36.8	22.3	28.2	23.8
Pirate-RThief*	87.6	87.5	35.4	21.2	32.6	27.1

In-depth Studies. In order to inspect the behaviors of the anchor set and of the relevance mechanisms during the attack procedure, in Fig. 2 we report the size of the anchor set, $|\mathcal{A}_t|$, as a function of time (or, equivalently, the number of attacks)–solid lines. We also plot the number of anchors whose relevance score is zero, also referred to as "dead" anchors–dashed lines. When a pair of lines joins, the algorithm ends. The curves with the same color are almost symmetric

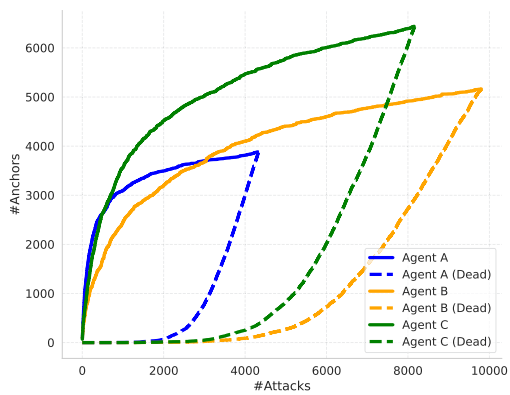

Figure 2. Evolution of anchor set $\mathcal{A}_t$ during the (unbounded) attack procedure of Algorithm 1. Dashed curves are about zero-relevance (dead) anchors.

with respect to the line connecting the origin to the final knot. Compared to Table 2-3, best results are in the case in which a smaller number of anchors is collected (Agent A). In this case, the algorithm was able to find good anchors that allowed it to steal a large amount of knowledge. In the case of agents B/C, more anchors are accumulated, especially in case B, suggesting that many of them turned out to not help in a significant manner. This is actually coherent with the lower results obtained in the B case (Table 2-3). Fig. 3 compares how many

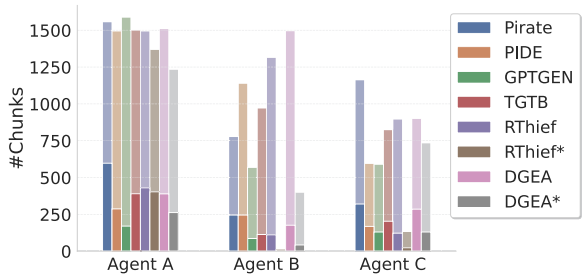

Figure 3. Pale: number of extracted chunks (LC metric) during the attack procedure (bounded case). Opaque: number of unique chunks (ULC metric).

Table 4. Wall-clock time (seconds) to create an attack query (mean ± std)–without sending it to the RAG system. In our attack, this includes the time to extract anchors (most cumbersome step), update relevance, sample anchors, generate query. PIDE, TGTB, and GPTGEN are based on pre-designed queries and RThief* is identical to RThief (in DGEA and DGEA* the query creation depends on the text embedder which varies).

Attack	Agent A	Agent B	Agent C
DGEA	$1116.09_{\pm 120.82}$	$1107.97_{\pm 111.41}$	$937.50_{\pm 214.85}$
DGEA*	$560.26_{\pm 4.34}$	$386.65_{\pm 2.73}$	$581.23_{\pm 88.42}$
RThief	$14.78_{\pm 12.05}$	$18.64_{\pm 13.54}$	$20.06_{\pm 13.33}$
Pirate (Ours)	$13.25_{\pm 5.44}$	$11.20_{\pm 5.05}$	$15.24_{\pm 6.20}$

total chunks (LC) and how many unique chunks (ULC) each method extracts in the bounded case. Pirate stands out for its ability to uncover a greater number of unique chunks on Agents A and C compared to the other methods, and it matches PIDE in Agent B (ULC). Moreover, Pirate consistently achieves a higher ratio of unique-to-total chunks on Agents A and B than the other approaches, indicating the effective nature of the queries in exploring previously unrevealed information rather than repeatedly retrieving the same chunks. Finally, in Table 4, we report the wall-clock time required to prepare an attack query. PIDE, TGTB, and GPTGEN have no query generation time since all queries are pre-generated prior to the start of the algorithm. DGEA requires a significant interaction with the text embedder and comparisons with its internal memories, while RThief requires the attacker LLM to generate backward and forward continuations of a stolen chunk. This procedure not only demands more time than ours, but also leads to substantially longer adversarial queries (see supplementary materials [13], Appendix E, Figure 4). Overall, the query generation time in Pirate is very advantageous since, once anchors are sampled (based on their relevance), there are no further comparisons to perform.

6 Defense Strategies

One intuitive defense is to raise the similarity threshold during retrieval (**Retrieval Threshold Adjustment**) so that only highly relevant chunks are selected. However, both our proposed Pirate attack and competitors like RThief generate "infected" prompts with high similarity to top-k retrieved chunks (see supplementary materials [13], Figure 5). Increasing the threshold would likely discard valuable context necessary for accurate query responses, thus degrading system performance. Another common approach is to modify the prompt with additional instructions aimed at discouraging the repetition of sensitive content (**Prompt-Based Defenses**). A Previous study [43] indicates that such prompt modifications do not com-

pletely prevent the leakage of information by the Agent. The adversarial query-generation process is specifically designed to circumvent these preventative instructions, resulting in not a complete mitigation of leakage despite these modifications. We also evaluated a state-of-the-art guard system (**Llama Guard** [39]) designed to prevent unsafe outputs, including privacy-sensitive content. Our evaluation (see supplementary materials [13], Appendix H) reveals that while guard systems provide a promising starting point, they exhibit significant limitations in accurately detecting and mitigating adaptive privacy leakage. This suggests that relying solely on such mechanisms may not offer sufficient protection. Another defensive measure is to limit the rate at which queries can be made (**Query Rate Limiting**). While rate limiting can slow down the overall runtime of an attack by restricting the frequency of query submissions, it does not fundamentally prevent the extraction of private data. An adaptive adversary can still eventually uncover sensitive information given enough time, making this approach only partially effective.

7 Conclusions and Future Work

We presented Pirate, an adaptive, anchor-based black-box attack that extracts private content from RAG systems using only local, open-source tools. In head-to-head evaluations, Pirate achieved higher navigation coverage and leaked knowledge than prior methods in both bounded and unbounded settings, while keeping query-generation time competitive. Our experimental setup simulates realistic operational constraints faced by many organizations, including privacy compliance requirements (e.g., GDPR [17]), hardware limitations, and restrictions on using online obfuscation systems. Although our work does not propose a novel defense mechanism, it establishes an adaptive attack benchmark that can inform and guide the development of more robust defense strategies in future research. Given the prevalence of sensitive or proprietary data in RAG deployments, responsibly studying such attacks is necessary to harden systems and reduce risks of privacy breaches and misuse.

Future Work. Future work should focus on five key areas. First, we can enhance duplicate identification by exploring alternatives to our current embedding and ROUGE-L analysis. Second, we aim to develop methods that distinguish between public and private data leakage in RAG systems for a more nuanced assessment. Third, our untargeted attack strategy could be extended to target specific sensitive information. Fourth, we will investigate how the attacker LLM's complexity impacts attack effectiveness. Finally, we plan to incorporate advanced tuning strategies to automate the selection of key parameters like similarity thresholds.

8 Acknowledgment

This work was supported by the University of Siena (Piano per lo Sviluppo della Ricerca - PSR 2024, F-NEW FRONTIERS 2024), under the project "TIme-driveN StatEful Lifelong Learning" (TINSELL) and also by the project "CONSTR: a COllectionless-based Neuro-Symbolic Theory for learning and Reasoning", PARTENARIATO ESTESO "Future Artificial Intelligence Research - FAIR", SPOKE 1 "Human-Centered AI" Università di Pisa, "NextGenerationEU", CUP I53C22001380006. The acknowledgment is extended to Rete SAIHUB (Siena, Italy), which, jointly with the Italian Ministry of University and Research (DM 117/2023, PNRR, Missione 4, Componente 2, Investimento 3.3), funded the scholarship of Christian Di Maio. Computational resources were provided by computing@unipi, University of Pisa.

References

[1] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] AI@Meta. Llama 3.2 1b model card. 2024. URL https://huggingface.co/meta-llama/Llama-3.2-1B.

[3] AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[4] M. Anderson, G. Amit, and A. Goldsteen. Is my data in your retrieval database? membership inference attacks against retrieval augmented generation. *arXiv preprint arXiv:2405.20446*, 2024.

[5] V. Bhat, D. Sree, J. Cheerla, N. Mathew, G. LIu, and J. Gao. Retrieval augmented generation (rag) based restaurant chatbot with ai testability. 07 2024.

[6] T. B. Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[7] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

[8] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914. IEEE, 2022.

[9] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Computing Surveys*, 55(13s):1–39, 2023.

[10] S. Cohen, R. Bitton, and B. Nassi. Unleashing worms and extracting data: Escalating the outcome of attacks against rag-based inference in scale and severity using jailbreaking. *arXiv preprint arXiv:2409.08045*, 2024.

[11] A. Cutbill, E. Monsler, and E. Hayashi. Personalized home assistant using large language model with context-based chain of thought reasoning. 2024.

[12] C. Di Maio and C. Cosci. gnekt/pirates-of-the-rag-adaptively-attacking-llms-to-leak-knowledge-bases: Code for ecai 2025 paper, Aug. 2025. URL https://doi.org/10.5281/zenodo.16918756.

[13] C. Di Maio, C. Cosci, M. Maggini, V. Poggioni, and S. Melacci. Pirates of the rag: Adaptively attacking llms to leak knowledge bases. *arXiv preprint arXiv:2412.18295*, 2024.

[14] C. Di Maio, A. Zugarini, F. Giannini, M. Maggini, and S. Melacci. Tomorrow brings greater knowledge: Large language models join dynamic temporal knowledge graphs. In V. Lomonaco, S. Melacci, T. Tuytelaars, S. Chandar, and R. Pascanu, editors, *Proceedings of The 3rd Conference on Lifelong Learning Agents*, volume 274 of *Proceedings of Machine Learning Research*, pages 560–576. PMLR, 29 Jul–01 Aug 2025. URL https://proceedings.mlr.press/v274/maio25a.html.

[15] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[16] H. Duan, A. Dziedzic, M. Yaghini, N. Papernot, and F. Boenisch. On the privacy risk of in-context learning. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

[17] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council. URL https://data.europa.eu/eli/reg/2016/679/oj.

[18] S. García-Méndez, F. de Arriba-Pérez, and M. d. C. Somoza-López. A review on the use of large language models as virtual tutors. *Science & Education*, pages 1–16, 2024.

[19] A. Golda, K. Mekonen, A. Pandey, A. Singh, V. Hassija, V. Chamola, and B. Sikdar. Privacy and security concerns in generative ai: A comprehensive survey. *IEEE Access*, 2024.

[20] K. Grosse, L. Bieringer, T. R. Besold, B. Biggio, and K. Krombholz. Machine learning security in industry: A quantitative survey. *IEEE Transactions on Information Forensics and Security*, 18:1749–1762, 2023.

[21] M. Grusky. Rogue scores. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1914–1934, 2023.

[22] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020.

[23] S. Hisamoto, M. Post, and K. Duh. Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system? *Transactions of the Association for Computational Linguistics*, 8:49–63, 2020.

[24] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[25] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 54(11s):1–37, 2022.

[26] Y. Hu and H. K. Min. The dark side of artificial intelligence in service: The "watching-eye" effect and privacy concerns. *International Journal of Hospitality Management*, 110:103437, 2023.

[27] Y. Huang, S. Gupta, Z. Zhong, K. Li, and D. Chen. Privacy implications of retrieval-based language models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14887–14902, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.921. URL https://aclanthology.org/2023.emnlp-main.921.

[28] C. Jiang, X. Pan, G. Hong, C. Bao, and M. Yang. Rag-thief: Scalable extraction of private data from retrieval-augmented generation applications with agent-based attacks. *arXiv preprint arXiv:2411.14110*, 2024.

[29] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[30] E. Kamalloo, N. Dziri, C. L. Clarke, and D. Rafiei. Evaluating open-domain question answering in the era of large language models. *arXiv preprint arXiv:2305.06984*, 2023.

[31] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274, 2023.

[32] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[33] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, and L. He. A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(2):1–41, 2022.

[34] Y. Li. A practical survey on zero-shot prompt design for in-context learning. In R. Mitkov and G. Angelova, editors, *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pages 641–647, Varna, Bulgaria, Sept. 2023. INCOMA Ltd., Shoumen, Bulgaria. URL https://aclanthology.org/2023.ranlp-1.69.

[35] Y. Li, Z. Li, K. Zhang, R. Dan, S. Jiang, and Y. Zhang. Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge. *Cureus*, 15(6), 2023.

[36] Y. Li, G. Liu, Y. Yang, and C. Wang. Seeing is believing: Black-box membership inference attacks against retrieval augmented generation. *arXiv preprint arXiv:2406.19234*, 2024.

[37] Z. Li, X. Zhang, Y. Zhang, D. Long, P. Xie, and M. Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[38] Y. Lin, L. Tan, H. Lin, Z. Zheng, R. Pi, J. Zhang, S. Diao, H. Wang, H. Zhao, Y. Yao, et al. Speciality vs generality: An empirical study on catastrophic forgetting in fine-tuning foundation models. *arXiv preprint arXiv:2309.06256*, 2023.

[39] A. . M. Llama Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[40] F. Mireshghallah, K. Goyal, A. Uniyal, T. Berg-Kirkpatrick, and R. Shokri. Quantifying privacy risks of masked language models using membership inference attacks. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8332–8347, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.570. URL https://aclanthology.org/2022.emnlp-main.570.

[41] OpenAI, A. Hurst, A. Lerer, et al. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

[42] J. Park. Development of dental consultation chatbot using retrieval augmented llm. *The Journal of the Institute of Internet, Broadcasting and Communication*, 24(2):87–92, 2024.

[43] Z. Qi, H. Zhang, E. Xing, S. Kakade, and H. Lakkaraju. Follow my instruction and spill the beans: Scalable data extraction from retrieval-augmented generation systems. *arXiv preprint arXiv:2402.17840*, 2024.

[44] Rag-Datasets. Rag-mini-bioasq. 2024. URL https://huggingface.co/datasets/rag-datasets/rag-mini-bioasq.

[45] Rag-Datasets. Rag-mini-wikipedia. 2024. URL https://huggingface.co/datasets/rag-datasets/rag-mini-wikipedia.

[46] M. Raja, E. Yuvaraajan, et al. A rag-based medical assistant especially for infectious diseases. In *2024 International Conference on Inventive Computation Technologies (ICICT)*, pages 1128–1133. IEEE, 2024.

[47] O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgay, A. Shashua, K. Leyton-Brown, and Y. Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

[48] A. RoyChowdhury, M. Luo, P. Sahu, S. Banerjee, and M. Tiwari. Confusedpilot: Compromising enterprise information integrity and confidentiality with copilot for microsoft 365. *arXiv preprint arXiv:2408.04870*, 2024.

[49] V. Shejwalkar, H. A. Inan, A. Houmansadr, and R. Sim. Membership inference attacks against nlp classification models. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

[50] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.346. URL https://aclanthology.org/2020.emnlp-main.346.

[51] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[52] F. Tramèr, G. Kamath, and N. Carlini. Considerations for differentially private learning with large-scale public pretraining. *arXiv preprint arXiv:2212.06470*, 2022.

[53] B. Wang, W. Chen, H. Pei, C. Xie, M. Kang, C. Zhang, C. Xu, Z. Xiong, R. Dutta, R. Schaeffer, et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. In *NeurIPS*, 2023.

[54] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[55] Z. Wang, H. Li, D. Huang, and A. M. Rahmani. Healthq: Unveiling questioning capabilities of llm chains in healthcare conversations. *arXiv preprint arXiv:2409.19487*, 2024.

[56] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[57] R. R. Wiyatno, A. Xu, O. Dia, and A. De Berker. Adversarial examples in modern machine learning: A review. *arXiv preprint arXiv:1911.05268*, 2019.

[58] F. Wu, N. Zhang, S. Jha, P. McDaniel, and C. Xiao. A new era in llm security: Exploring security concerns in real-world llm-based systems. *arXiv preprint arXiv:2402.18649*, 2024.

[59] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

[60] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024.

[61] Z. Yu, L. He, Z. Wu, X. Dai, and J. Chen. Towards better chain-of-thought prompting strategies: A survey. *arXiv preprint arXiv:2310.04959*, 2023.

[62] S. Zeng, J. Zhang, P. He, Y. Liu, Y. Xing, H. Xu, J. Ren, Y. Chang, S. Wang, D. Yin, and J. Tang. The good and the bad: Exploring privacy issues in retrieval-augmented generation (RAG). In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4505–4524, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.267. URL https://aclanthology.org/2024.findings-acl.267.

[63] P. Zhao, H. Zhang, Q. Yu, Z. Wang, Y. Geng, F. Fu, L. Yang, W. Zhang, and B. Cui. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*, 2024.

[64] Y. Zhou, Y. Liu, X. Li, J. Jin, H. Qian, Z. Liu, C. Li, Z. Dou, T.-Y. Ho, and P. S. Yu. Trustworthiness in retrieval-augmented generation systems: A survey. *arXiv preprint arXiv:2409.10102*, 2024.

[65] W. Zhu, H. Liu, Q. Dong, J. Xu, S. Huang, L. Kong, J. Chen, and L. Li. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675*, 2023.