CARTRIDGES: Lightweight and general-purpose long context representations via self-study

Anonymous Authors[1]

Abstract

Large language models are often used to answer queries grounded in large text corpora (*e.g.* codebases, legal documents, or chat histories) by placing the entire corpus in the context window and leveraging in-context learning (ICL). Although current models support contexts of 100K–1M tokens, this setup is costly to serve because the memory consumption of the KV cache scales with input length. We explore an alternative: training a compressed KV cache offline on each corpus. At inference time, we load this trained KV-cache, which we call a CARTRIDGE, and decode a response. Critically, the cost of training a CARTRIDGE can be amortized across all the queries referencing the same corpus. However, we find that the naive approach of training the CARTRIDGE with next-token prediction on the corpus is not competitive with ICL. Instead, we propose SELF-STUDY, a training recipe in which we generate synthetic conversations about the corpus and train the CARTRIDGE with a context-distillation objective. We find that CARTRIDGES trained with SELF-STUDY replicate the functionality of ICL, while being significantly cheaper to serve. On challenging long-context benchmarks, CARTRIDGES trained with SELF-STUDY match ICL performance while using $38.6\times$ less memory and enabling $26.4\times$ higher throughput. SELF-STUDY also extends the model's effective context length (*e.g.* from 128k to 484k tokens on MTOB) and surprisingly, leads to CARTRIDGES that can be composed at inference time without retraining.

1. Introduction

A common large language model (LLM) usage pattern involves placing a large corpus into the context window and querying the model about it. For instance, a user or organization may use LLMs to understand codebases (Nam et al., 2024), financial documents (Islam et al., 2023), legal texts (Guha et al., 2023; Zheng et al., 2025), textbooks (Ouellette et al., 2025), or personal files (Arora & Ré, 2022). LLMs excel here due to in-context learning (ICL), enabling accurate responses to diverse queries (e.g., factual Q&A, summarization, code generation) (Dong et al., 2022).

Despite its flexibility, this usage paradigm is memory-intensive. ICL requires maintaining a KV cache that grows linearly with the input length. For example, LLaMA 70B needs 84 GB of memory (at 16-bit precision) to answer a single question over a 128k-token context (Dubey et al., 2024). This severely limits user throughput: on a single H100 GPU, LLaMA 8B's peak thoughput (tokens/s) drops by $77\times$ when increasing the context from 1k to 120k tokens (Figure 2).

Prior work has thus explored reducing KV cache memory usage. For instance, prompt compression methods reduce the number of tokens stored in the cache using summarization, retrieval, or self-information filtering (Lewis et al., 2020; Jiang et al., 2023b; Li, 2023; Chuang et al., 2024), while KV cache compression techniques directly compress the stored key-value pairs (Ge et al., 2023a; Zhang et al., 2023; Tang et al., 2024; Oren et al., 2024). Unfortunately, these methods raise sharp memory-quality tradeoffs: in experiments on challenging long-context tasks, we find that performance rapidly degrades when applying these methods with compression ratios greater than $2\times$ (see Figure 3).

Motivated by the observation that the cost of preparing a corpus-specific representation can be amortized across many queries, we explore a complementary approach based on offline training. Given a specific corpus (e.g., a financial filing) we freeze the LLM and train a compressed KV cache by backpropagating loss into the key and value vectors, resembling prefix tuning (Li & Liang, 2021; Lester et al., 2021). We call the trained representation for the corpus a "CARTRIDGE." At inference time, we load the trained CARTRIDGE for the corpus, append the user's question, and decode. As users often ask multiple questions about the same corpus, each CARTRIDGE can be trained once and reused. And since practitioners are often interested in common corpora (e.g., SEC filings, or Wikipedia), a CARTRIDGE can be distributed online akin to traditional models. This approach also integrates cleanly with existing inference servers, which are already designed to manage per-user KV caches (Kwon et al., 2023; Zheng et al., 2024).

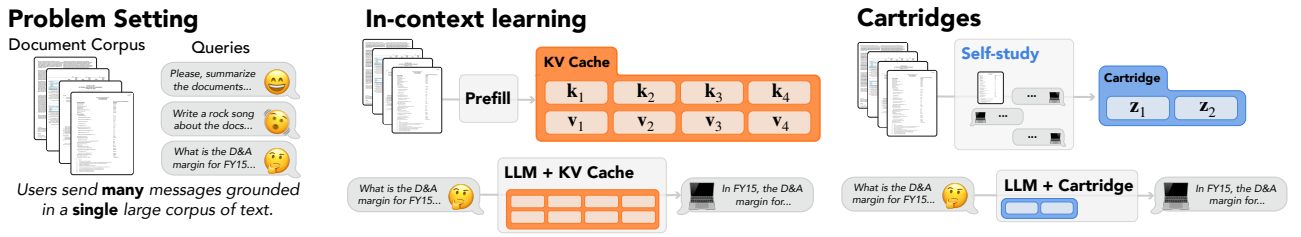

Figure 1. **Producing CARTRIDGES via self-study**. For a given document corpus, we train a CARTRIDGE by distilling the corpus into a parameterized KV cache representation through a process we call SELF-STUDY. At inference time, this CARTRIDGE may be loaded into an LLM, which can then be used to answer diverse queries about the corpus, simulating in-context analysis of the corpus while requiring substantially less memory.

Compressed KV caches reduce memory usage, but achieving ICL-equivalent functionality requires CARTRIDGES to satisfy three non-trivial conditions. **First**, CARTRIDGES should replicate the generality of ICL, and provide accurate answers across diverse questions (Dong et al., 2022). The training procedure to enable this functionality is unclear; we experimentally find that naive training on corpora allow for model memorization but do not support ICL-equivalent general reasoning. **Second**, a CARTRIDGE should replicate ICL's structural awareness—its ability to reason over document structure, and understand how distant parts of a corpus relate or depend on each other. This is critical for corpora like codebases or legal agreements, where information in one section often affects another (Hron, 2025). Naïve compression methods may overlook or discard such long-range and hierarchical dependencies, thus reducing performance. **Third**, CARTRIDGES should be composable. Just as users may flexibly combine different documents in a LLM context window (via concatenation), they should also be capable of combining different CARTRIDGES, depending on which documents are of interest. This kind of knowledge composition in LLMs is recognized as a challenging problem (Huang et al., 2023).

To address these challenges, we propose SELF-STUDY: a simple, automated recipe for training general, information-preserving, and composable CARTRIDGES for any text corpus. SELF-STUDY consists of two steps: (1) generating synthetic training data by prompting the model to quiz itself in diverse formats—e.g., Q&A, summarization, hierarchical structuring—encouraging broad coverage, generality, and structural understanding; and (2) distilling the corpus into a compressed cache using a context-distillation objective (Bhargava et al., 2024; Snell et al., 2022) that aligns the model's next-token predictions with and without access to the original text. To preserve global dependencies, SELF-STUDY augments synthetic examples with model-generated table-of-contents, positional context, and retrieval-augmented views over distant corpus chunks. This enables the model to learn long-range relationships during training. Remarkably, SELF-STUDY also yields CARTRIDGES that are composable without joint optimization: multiple CARTRIDGE can be concatenated and queried together, emulating ICL's ability to flexibly answer queries over multiple documents concatenated in context.

We evaluate SELF-STUDY on a broad suite of challenging and popular benchmarks that pair a single long context (100k-484k tokens) with a diverse set of queries (Islam et al., 2023; Adams et al., 2024; Tanzer et al., 2023). We make three findings. **First**, CARTRIDGES extends the quality-memory frontier—averaged across the benchmarks, CARTRIDGES produced with SELF-STUDY match ICL quality while consuming $38.6\times$ less memory, enabling a $26.4\times$ increase in peak throughput (tokens per second). These memory reductions and speedups represent an order of magnitude improvement over state-of-the-art cache compression baselines (*e.g.* DuoAttention (Xiao et al., 2024b)). **Second**, CARTRIDGES enables context length extrapolation. On the MTOB benchmark (Tanzer et al., 2023), where models must translate from Kalamang, a low-resource language, into English, we use SELF-STUDY with LLAMA-8B to construct a small CARTRIDGE from a 484k token textbook. This CARTRIDGE outperforms ICL over the first $130,000$ tokens of the textbook by 11.0 chrF points and matches the performance ICL over a curated and subset of the textbook in context. **Third**, we show that CARTRIDGES can be concatenated and used in a single decoding pass, similar to ICL, and can be used to extrapolate beyond the context length of ICL in Figure 5.

2. Related work

See Appendix B for a more detailed discussion of prior work.

Parameter Efficient Fine-Tuning Prior work has explored a range of strategies, prompt distillation (Kujanpää et al., 2024; Snell et al., 2024), self-instruction (Nayak et al., 2024), and domain-specific training (Colombo et al., 2024) for adapting langague models. Variants of this approach train $\mathcal{C}$ into smaller modules ("adapters") which can be added to the model, which have parameter efficiency benefits (Su et al., 2025; Hu et al., 2022; Li & Liang, 2021;

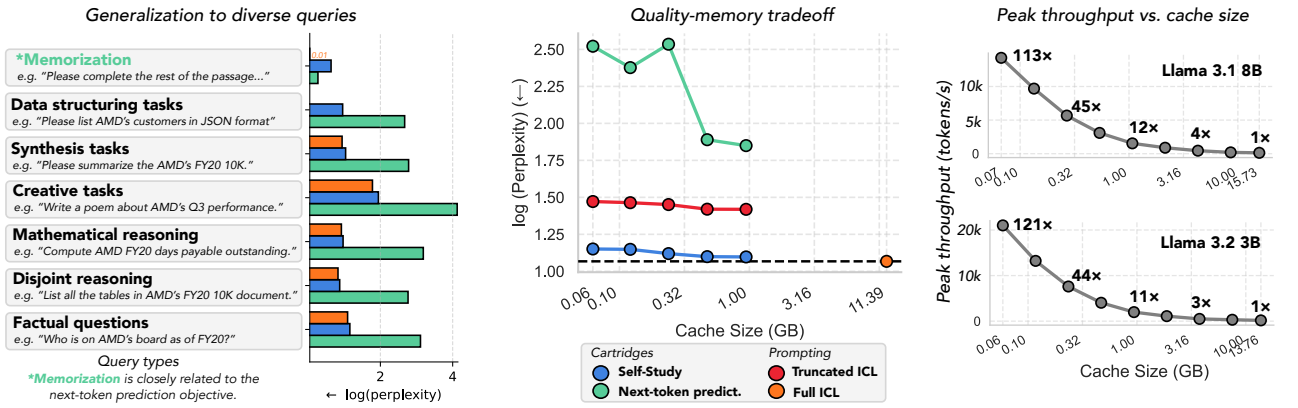

Figure 2. **CARTRIDGES trained with SELF-STUDY balance the generality and memory consumption tradeoff.** We compare four methods on the GENCONVO dataset: CARTRIDGES trained with next-token prediction over $\mathcal{C}$, CARTRIDGES trained with SELF-STUDY, full ICL, and truncated ICL, a prompt compression method in which we truncate the $\mathcal{C}$ to the first k tokens. (**Left**) We evaluate on different slices from the GENCONVO dataset. CARTRIDGES trained with next-token prediction performs well on memorization queries, which resemble it's training distribution, but cannot generalize to other queries like the other methods. (**Center**) The x-axis measures the size of the KV cache in GB for the different methods. The y-axis shows log-perplexity on the GENCONVO dataset averaged over the query types. (**Right**) Peak throughput (tokens/s) measured for different cache sizes for LLAMA-3B and LLAMA-8B with SGLang (Zheng et al., 2024) on an 1xH100 (See Appendix A).

Lester et al., 2021; Meng et al., 2024). A number of works have explored the idea of composing multiple different parameter-efficient adapters through various aggregation operations (Zhao et al., 2024b; Huang et al., 2023; Xiao et al., 2024a; Zhao et al., 2024a; Yadav et al., 2024; Wu et al., 2024; Gou et al., 2023; Li et al., 2024a).

Most similar to our work are recent knowledge injection methods, which aim to internalize the corpus $\mathcal{C}$ into model weights, allowing models to answer queries from parameter knowledge as opposed to ICL (Kujanpää et al., 2024) (Mao et al., 2025) (Su et al., 2025). Recent work like LIFT (Mao et al., 2025) uses synthetic data to train a per-document adapter for long-context documents, but does not study the throughput improvements stemming from the lack of a large KV cache. Our approach improves quality-memory (and thus quality-throughput) tradeoff frontier, matching ICL performance and supporting composability while keeping the memory footprint small.

Architectures Research has also examined more memory efficient alternatives to traditional attention (Vaswani et al., 2017), which leverage sparsity (Beltagy et al., 2020; Child et al., 2019; Zaheer et al., 2020; Team et al., 2024) or alter the structure of attention (Shazeer, 2019; Ainslie et al., 2023), among other strategies (Liu et al., 2024a; Zhang et al., 2025; Arora et al., 2024). Certain variants (i.e., grouped-query attention) appear in popular models like Llama-3, which we study in our experiments.

Prompt and KV-cache compression As the size of the KV cache drives the model memory footprint, research has examined different strategies for reducing the cache size. One set of approaches focus on making the prompt smaller—explicit methods alter the prompt text through summarization and filtering (Jiang et al., 2023b; Li, 2023; Chuang et al., 2024; Zhang et al., 2024a; Pan et al., 2024), while implicit methods compress prompt representations into a set of "soft" tokens (Chevalier et al., 2023; Yen, 2024; Ge et al., 2023b; Mu et al., 2023; Qin et al., 2023; Lester et al., 2021). Another set of approaches exploits observations about the mathematical structure of the KV cache (Yu et al., 2024; Chang et al., 2024), often finding that because a small number of keys dominate the attention scores of subsequent queries, non-impactful key-value pairs (or tokens) can be dropped (Ge et al., 2023a; Zhang et al., 2023; Tang et al., 2024; Oren et al., 2024; Li et al., 2024b) or merged (Wang et al., 2024; Zhang et al., 2024c; Wan et al., 2024).

3. The CARTRIDGE paradigm

We first formalize our problem and establish preliminaries (Section 3.1). We then describe the CARTRIDGE paradigm (Section 3.3). Finally, we describe the training procedure and provide motivation for SELF-STUDY, our method for training CARTRIDGES, which we introduce in Section 4.

3.1. Problem setup

We assume a setting in which users issue a stream of diverse queries about a common corpus of text. We denote the corpus as $\mathcal{C}$ and the query set as $Q = \{q_1, q_2, \ldots, q_m\}$. Illustrative examples of $\mathcal{C}$ include legal filings, financial documents, code repositories, chat histories, and medical

records.

Example: Financial Analysis

$\mathcal{C}$ may correspond to the 2025 Form 10-K filing (U.S. Securities and Exchange Commission, 2011) for AMD, which is almost 100k tokens. The queries an analyst might ask an LLM to answer with respect to this form are diverse, including: (1) recalling factual information, (2) performing mathematical reasoning over values, or (3) even generating creative responses (e.g., a poem) grounded in the 10-K's information.

Let $R = \{r_1, r_2, \ldots, r_m\}$ denote the responses the LLM produces for the queries. Users have two objectives. First, they wish to maximize the quality of responses R under some utility function (or alternatively, minimize the loss under some cost function). Second, they wish to minimize the LLM's memory footprint while it is answering questions with respect to the document. This is because larger memory footprints decrease throughput and necessitate more expensive hardware (Figure 2, Right).

3.2. Preliminaries: Language models and KV caches

Let $\mathcal{F}$ denote some language model and $\mathcal{F}(\cdot|\mathbf{x})$ correspond to distribution of the language model conditioned on some text $\mathbf{x} \in \mathcal{V}^n$.

Recall an LLM $\mathcal{F}$ accepts as input a sequence of N tokens $\mathbf{x} \in \mathcal{V}^N$ drawn from a discrete vocabulary $\mathcal{V} \subset \mathbb{Z}$ of tokens, each represented by a unique integer. $\mathcal{F}$ outputs a probability distribution over $\mathcal{V}$ for each token $x_i \in \mathcal{V}$ given the preceding tokens $\mathcal{F}(x[i]|\mathbf{x}[:\ i-1])$. Each token x_i in $\mathbf{x}$ is embedded into a d-dimensional space, yielding a matrix $\mathbf{u} \in \mathbb{R}^{N\times d}$. The matrix $\mathbf{u}$ is passed through a stack of L model layers, which each mix the matrix along the N and d dimensions, with layer ℓ outputting $\mathbf{y}^l \in \mathbb{R}^{N\times d}$. The final $\mathbf{y}^L$ is mapped to the logits over $\mathcal{V}$.

Most modern language models use the Transformer architecture based on self-attention (Vaswani et al., 2017). Given an input $\mathbf{u} \in \mathbb{R}^{N\times d}$ for sequence length N and embedding dimension d, it computes the output $y^l \in \mathbb{R}^{N\times d}$ via the softmax over projections $\mathbf{q}, \mathbf{k}, \mathbf{v} = \mathbf{u}\mathbf{W}_q, \mathbf{u}\mathbf{W}_k, \mathbf{u}\mathbf{W}_v$:

$$\mathbf{y}_i = \sum_{j=1}^{i} \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_j/\sqrt{d})\mathbf{v}_j}{\sum_{m=1}^{i}\exp(\mathbf{q}_i^\top \mathbf{k}_m/\sqrt{d})} \tag{1}$$

where weight matrices $\boldsymbol{W}_q$, $\boldsymbol{W}_k$ and $\boldsymbol{W}_v$ for each layer are learned during training. Every new output $\mathbf{y}_n$ is conditioned on prior $\{\mathbf{k}_i, \mathbf{v}_i\}_{i=1}^{n-1}$, which grows in n. This is why a long corpus $\mathcal{C}$ produces large memory footprints, as the size of the KV cache scales linearly in the length of $\mathcal{C}$.

3.3. Formalizing CARTRIDGES

Our goal is to learn a CARTRIDGE for a given corpus $\mathcal{C}$. A CARTRIDGE is a small set of parameters $Z \in \mathbb{R}^*$ (*i.e.* an adapter (Li & Liang, 2021; Hu et al., 2022)) that augments an LLM $\mathcal{F}$ and cause it to behave as if it had $\mathcal{C}$ in its context window. Formally, let $\mathcal{F}_Z(\cdot|q)$ denote the distribution of $\mathcal{F}$ augmented with Z given a query q. For all $q \in Q$, we want to ensure that samples $r_Z \sim \mathcal{F}_Z(\cdot|q)$ are as good or better than the ICL sample $r_q \sim \mathcal{F}(\cdot|\mathcal{C} \oplus q)$, according to some query-specific scoring function. In order for $\mathcal{F}_Z(\cdot|q)$ to match or exceed the behavior of $\mathcal{F}(\cdot|\mathcal{C} \oplus q)$, three important criteria should be met.

- **Displays generality**: Because Q might span a diverse range of question types (e.g., mathematical reasoning, factual recall comprehension, summarization, and more), it is essential that $\mathcal{F}_Z$ can generalize across different $q \in Q$. **This is non-trivial because Q is unknown when Z is being learned.** If $\mathcal{F}_Z$ does not generalize, then practitioners may need to learn different Z for different distributions of queries, which increases the cost of the CARTRIDGE. Ideally, Z should only need to be learned once, yet work for multiple types of queries.
- **Captures long range dependencies**: Z should also capture long range dependencies contained within $\mathcal{C}$. In many settings, correctly answering different $q \in Q$ requires reasoning about the order of information presented in $\mathcal{C}$. It is not clear how to capture these dependencies in Z.
- **Capable of composition**: Ideally, the representation of Z and mechanism by which $\mathcal{F}$ utilizes it could allow for composition, without any particular joint training of CARTRIDGES. Given Z_1 and Z_2 corresponding to $\mathcal{C}_1$ and $\mathcal{C}_2$, ideally $\mathcal{F}_{[Z_1,Z_2]}(q)$ is similar to $\mathcal{F}(\cdot|\mathcal{C}_1 \oplus \mathcal{C}_2 \oplus q])$

3.4. Parameterizing CARTRIDGES

We parameterize Z using a simplified version of prefix-tuning (Li & Liang, 2021). Specifically, we allocate a KV cache composed of *trainable* key and value vectors $\mathbf{z}_i^{\text{k}}, \mathbf{z}_i^{\text{v}} \in \mathbb{R}^d$. The size of the full $Z \in \mathbb{R}^{L\times p\times d\times 2}$, is controlled by the hyperparameter p. The memory footprint of Z is equivalent to a KV cache for a prompt with p tokens.

In ICL, the KV cache for $\mathcal{F}_\mathcal{C}(q)$ (where $\mathcal{C}$ is of length $n_\mathcal{C}$ and Q is of length n_Q) would contain $n_\mathcal{C} + n_Q$ key-value pairs, with the first $n_\mathcal{C}$ corresponding to $\mathcal{C}$ and the last n_Q corresponding to Q:

ICL KV Cache

$$\underbrace{(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_{n_\mathcal{C}}, \mathbf{v}_{n_\mathcal{C}})}_{\text{KV pairs for } \mathcal{C}}, \underbrace{(\mathbf{k}_{n_\mathcal{C}+1}, \mathbf{v}_{n_\mathcal{C}+1}) \ldots}_{\text{KV pairs for } q}$$

CARTRIDGE KV Cache

$$\underbrace{(\mathbf{z}^{\text{k}}_1, \mathbf{z}^{\text{v}}_1), \ldots, (\mathbf{z}^{\text{k}}_p, \mathbf{z}^{\text{v}}_p)}_{\text{Trainable KV pairs in } Z}, \underbrace{(\mathbf{k}_{n_{\mathcal{C}}+1}, \mathbf{v}_{n_{\mathcal{C}}+1}) \ldots}_{\text{KV pairs for } q}$$

To train a CARTRIDGE, we substitute the key-value pairs corresponding to $\mathcal{C}$ with Z, and directly optimize them by back-propagating the loss into the key and value vectors. **Critically, we freeze all parameters of the model, only training the key and value vectors in Z.** We discuss the choice of loss in Section 4.2.

Initialization Prior work finds that optimizing a randomly initialized cache Z_{KV} is unstable and leads to degraded performance (Li & Liang, 2021). Instead, these works initialize the trainable cache with a smaller dimensionality d and then re-project it to the original dimension with an MLP. In contrast, we find that proper initialization of Z allows us to directly optimize the full cache without reparametrization. Specifically, we initialize Z to the KV cache corresponding to the first p tokens of the corpus $\mathcal{C}$. Alternatively, we could use a summary of the corpus or filter tokens using off-the-shelf prompt compression strategies (Xiao et al., 2024b). In Appendix A, we show that our initializations lead to stable training and faster convergence than the random initialization used in prior work.

Why this parameterization? We note that the parameter-efficient fine-tuning literature provides numerous other ways to augment an LLM with a set of additional parameters (Li & Liang, 2021; Hu et al., 2022; Lester et al., 2021). In Appendix A.1, we perform a preliminary comparison of CARTRIDGES with each parameterization.

3.5. Serving CARTRIDGES

A CARTRIDGE can be served efficiently with minimal changes to existing LLM inference servers (Zheng et al., 2024; Kwon et al., 2023). Because a CARTRIDGE is a KV cache, it can be loaded directly into the KV cache slots using existing mechanisms for handling cached prefixes. LLM inference servers are heavily optimized for managing distinct KV-caches for multiple users (Ye et al., 2025), meaning cartrdiges can be served at high throughputs using existing inference servers. Decoding tokens with a CARTRIDGE is identical to serving a request with a prefix of length p. See Figure 2 for the relationship between prefix length and throughput. This contrasts with other methods like LoRA, which require custom infrastructure to serve efficiently to multiple users (Chen et al., 2024).

4. SELF-STUDY: A self-supervised method for compressing long contexts

In this section, we describe SELF-STUDY, a simple approach for training a CARTRIDGE Z on any corpus of text. The design of SELF-STUDY is motivated by experiments showing how CARTRIDGES trained with a simpler recipe fail to generalize to diverse user queries.

Motivating observations The naive method for constructing a CARTRIDGE would be to fine-tune the parameters of Z with the next token prediction objective on the corpus text directly. We show results experimenting with this approach in Figure 2, where we evaluate on a dataset derived from FinanceBench (Islam et al., 2023), which we refer to as GENCONVO (see appendix D for details). GENCONVO contains multiple types of questions (*e.g.* synthesis, reasoning). We find that the naïve next-token prediction approach can memorize with near perfect perplexity (Figure 2 left), while consuming $107\times$ less memory than ICL (Figure 2 center). However, generalization to other slices is poor, as shown in Figure 2. We seek a training objective that allows the responses from a model that uses the CARTRIDGE to generalize to a diverse set of user queries, resembling ICL.

Motivated by these observations, we describe our synthetic data generation in Section 4.1. As we show in Figure 2, fine-tuning a CARTRIDGE on this synthetic data allows models to use the CARTRIDGE generate responses to many types of queries that match the quality of queries generated with ICL. See Figure 1 for a visualization of the CARTRIDGE approach.

4.1. Self-supervised synthetic data to avoid overfitting

Towards training general CARTRIDGES, we propose using LLM generated synthetic data to generate our training dataset $\mathcal{D}_{\text{train}}$.

Overall synthetic data pipeline Our overall pipeline puts information from the corpus $\mathcal{C}$ in context and prompts the model to have a conversation with itself about the corpus to generate the synthetic query-response pairs as shown in Algorithm 1. We represent the concatenation of two vectors with $x \oplus y$

The conversation is generated by iteratively sampling generations from two LLM participants A and B (which are the same model). We maintain two different conversation histories: A's starts with a *user* message containing a seed prompt s (*e.g. "Please start a conversation by asking a question about the document above."*) followed by alternating *assistant* and *user* messages from A and B, respectively. B's conversation history does not include the seed prompt and contains the same messages as A's but with the roles of A and B swapped. Both have the subcorpus $\tilde{\mathbf{c}}$ in the system prompt. To build a training dataset, we sample m_{train} independent conversations and concatenate the messages from A and B into a single sequence of tokens:

$$\mathcal{D}_{\text{train}} = \{\mathbf{x}^{(j)} = \mathbf{a}_1^{(j)} \oplus \mathbf{b}_1^{(j)} \oplus \mathbf{a}_2^{(j)} \oplus \mathbf{b}_2^{(j)} \oplus \cdots \oplus \mathbf{a}_k^{(j)} \oplus \mathbf{b}_k^{(j)}\}_{j=1}^{m_{\text{train}}} \tag{2}$$

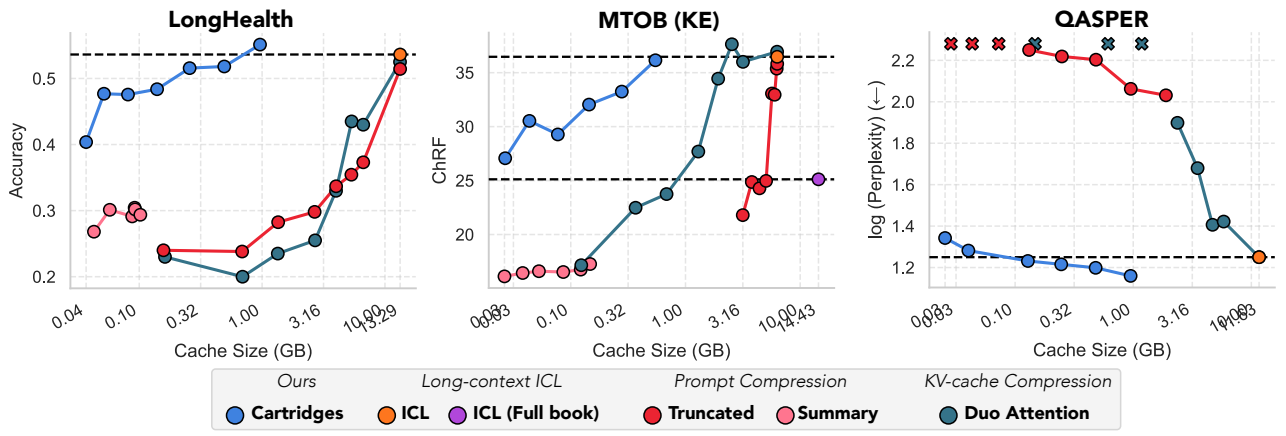

Figure 3. CARTRIDGES **matches ICL quality with lower memory costs.** We measure LLAMA-3B response quality (y-axis) against KV cache memory (x-axis) for different methods, at different KV cache sizes. The dashed line marks the quality of standard ICL.

Algorithm 1 SELF-STUDY: Data Generation

Input: $\mathcal{C}$: `Corpus`, $\mathcal{F}$: `Model`
Output: $\{\mathbf{a}_1, \mathbf{b}_1, \ldots, \mathbf{a}_k, \mathbf{b}_k\}$: `Convo`

$\tilde{\mathbf{c}} \leftarrow$ `chunk`$(\mathcal{C})$ ▷ **(1)** Get a **subcorpus** of $\mathcal{C}$ that fits in the context window
$\mathbf{s} \leftarrow$ `get_seed_prompt`$(\tilde{\mathbf{c}})$ ▷ **(2)** Get a prompt to **seed** the first message from A
for $i = 1$ to k **do** ▷ **(3)** Sample a **conversation** with k back and forths
  $\mathbf{a}_i \sim \mathcal{F}(\cdot \mid \tilde{\mathbf{c}} \oplus \mathbf{s} \oplus \mathbf{a}_1 \oplus \cdots \oplus \mathbf{b}_{i-1})$ ▷ **(3.1)** Sample A's message with $\tilde{\mathbf{c}}$ and $\mathbf{s}$ in context
  $\mathbf{b}_i \sim \mathcal{F}(\cdot \mid \tilde{\mathbf{c}} \oplus \mathbf{a}_1 \oplus \cdots \oplus \mathbf{b}_{i-1} \oplus \mathbf{a}_i)$ ▷ **(3.2)** Sample B's message with $\tilde{\mathbf{c}}$ in context
end for
return $\{\mathbf{a}_1, \mathbf{b}_1, \ldots, \mathbf{a}_k, \mathbf{b}_k\}$

where each $\mathbf{x}^{(j)}$ is a concatentation of the messages. Note that all of the datasets on which we evaluate in the main paper involve a single-turn. So, we set $k = 1$, generating a synthetic conversation with one user message and one assistant message. In Appendix A, we evaluate the multi-turn capabilities of CARTRIDGES.

Note that the `chunk` and `get_seed_prompt` functions expose two different ways to control the data distribution of the synthetic data. We find that these two design decisions are critical for training high quality CARTRIDGES with SELF-STUDY.

Chunking We use short subcorpora $\tilde{c}$ (between 512 and 4096) tokens to let the LLM focus on different parts of the corpus when generating data. This is motivated by observations in prior work (Liu et al., 2024c; Narayan et al., 2025). Furthermore, chunking also allows us to train CARTRIDGES on corpora longer than the model's context window.

Seed prompts Instead of using just one seed prompt, we curate a list of five different seed prompt types: *structuring*, *summarization*, *question*, *use cases*, and *creative*. The full list of seed prompts used in our experiments is provided in Appendix C. Critically, in all our experiments the seed prompts are **generic**: they do not mention anything related to the specifics of the corpora we evaluated (*e.g.* no mention of translation for MTOB or medical terms for LongHealth). We use the same set of seed prompts in all of our main results. In Section 5.4, we ablate the use of diverse seed prompts and find that it improves performance over a single generic seed prompt by up to 4.8 accuracy points (43.6 → 48.4 on LONGHEALTH).

4.2. SELF-STUDY context-distillation objective

Given a fine-tuning dataset $\mathcal{D}_{\text{train}}$, we adapt standard techniques from the model distillation literature (Kim & Rush, 2016).We let $\mathcal{F}(\cdot|\mathbf{x})$ denote the next token distribution given some input text $\mathbf{x}$. Our *teacher* is the model with the subcorpus, $\tilde{\mathbf{c}}$, in context $\mathcal{F}(\cdot|\tilde{\mathbf{c}})$ and our *student* is the same model adapted with a trainable cache $\mathcal{F}_Z(\cdot)$. We use a classic distillation objective (Hinton et al., 2015) that minimizes the KL-divergence between the teacher and student next-token distributions over a sequence of tokens $\mathbf{x} \in \mathcal{D}_{\text{train}}$:

$$\arg\min_{Z} \sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} \sum_{i=1}^{|\mathbf{x}|} D_{\text{KL}}\Big(\mathcal{F}(\cdot|\tilde{\mathbf{c}} \oplus \mathbf{x}[:i]) \;\;||\;\; \mathcal{F}_Z(\cdot|\mathbf{x}[:i])\Big) \tag{3}$$

In Appendix A, we provide an ablation that using a context-distillation objective improves accuracy for the (*e.g.* 3.7 accuracy points on LONGHEALTH).

5. Results

We describe experiments evaluating the effectiveness of CARTRIDGES trained with SELF-STUDY in various long-

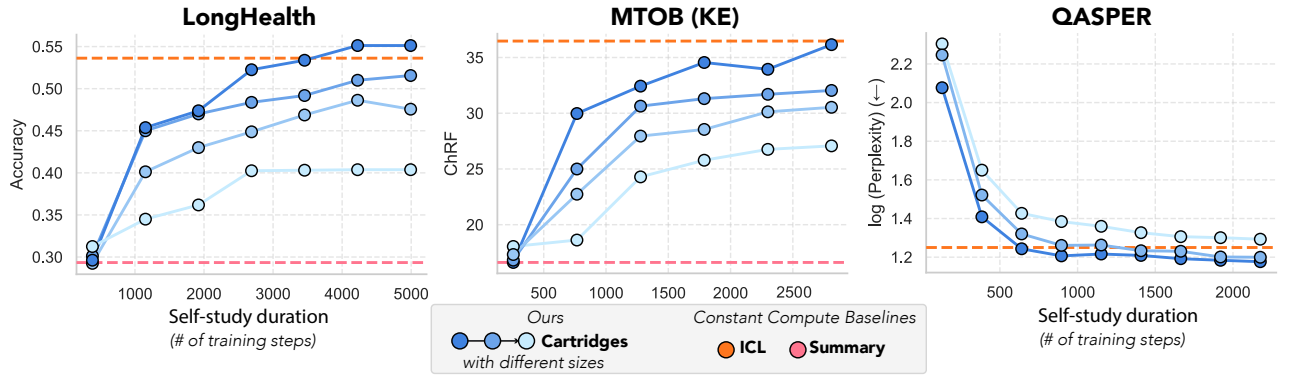

Figure 4. **Scaling SELF-STUDY compute.** These plots show how quality improves as we scale the training compute with SELF-STUDY. In all plots, the x-axis shows the total number of global training steps with batch size 64 and maximum sequence length 1024. No synthetically generated data is reused (*i.e.* training proceeds for one epoch). Curves are provided for CARTRIDGES of varying sizes ($p \in \{128, 512, 2048, 8192\}$). (**Left**) The y-axis shows accuracy on LONGHEALTH (Adams et al., 2024) with LLAMA-8B. (**Middle**) The y-axis shows the chrF on MTOB (Tanzer et al., 2023) with LLAMA-3B. (**Right**) The y-axis shows log-perplexity (lower is better) on QASPER (Dasigi et al., 2021) with LLAMA-3B.

context scenarios. Our results support the following claims. **First**, CARTRIDGES trained with SELF-STUDY can match or outperform ICL while maintaining generality and reducing serving costs (Section 5.1). **Second**, SELF-STUDY is effective on corpora longer than the context window of the LLM (Section 5.2). **Third**, when we concatenate two different CARTRIDGES without any joint training, the model can respond to queries requiring information from both CARTRIDGES (Section 5.3). Finally, we include ablations to assess the relative benefits of different aspects of SELF-STUDY and CARTRIDGES (Section 5.4).

Datasets We study datasets consisting of diverse (q, r) pairs about a single long document. Across datasets, $\mathcal{C}$ ranges between 100k and 484k tokens. Our datasets are drawn from popular long-context benchmarks, with some used as-released and others modified to meet this structure. These include: LONGHEALTH (Adams et al., 2024), MTOB (Tanzer et al., 2023), and QASPER (Dasigi et al., 2021). We evaluate LLM response quality using accuracy for LONGHEALTH, log perplexity for QASPER, and character n-gram f-score (chrF) for MTOB (Tanzer et al., 2023; Popović, 2015). Because each dataset effectively consists of a "single" document, we train a single CARTRIDGE per dataset at a given hyperparameter setting. Appendix D provides further details.

5.1. Pushing the quality/cost tradeoff frontier

We assess how CARTRIDGES produced with SELF-STUDY fare in quality and cost against baselines for LONGHEALTH and QASPER on LLAMA-3B. For both datasets, $\mathcal{C}$ fits within the model context window (128k tokens). We compare to traditional ICL, two prompt compression baselines (prompt truncation and prompt summarization using GPT-4o (OpenAI, 2024)), and a state-of-the-art KV cache compression baseline (Duo Attention (Jiang et al., 2023a; Xiao et al., 2024b)). We evaluate memory use in terms of KV cache size: the size of the KV cache for the ICL model and prompt compression methods, the size of the CARTRIDGE, and the size of the compressed KV cache for KV cache compression methods like DuoAttention.

Figure 3 presents our main results. On both LONGHEALTH and QASPER, we find cache sizes at which CARTRIDGES outperforms ICL. Compared against ICL, CARTRIDGES offers substantial memory savings at comparable performance: up to $10\times$ for LONGHEALTH, and up to $100\times$ for QASPER. In contrast, compression baseline methods see performance degradations at compression factors as low as $2\times$. Crucially, the small memory footprint of CARTRIDGES allows for much higher peak throughput (tokens/s). As Figure 2 (right) shows, cache sizes which match performance of ICL allow for almost $26\times$ higher throughput.

We also observe that CARTRIDGE performance scales as we increase the amount of compute used in self-study: the longer an CARTRIDGE is trained, the greater task performance. Figure 4 plots the performance for differentially sized CARTRIDGES as a function of the number of training steps. Across all sizes, we observe a steady positive correlation between performance and compute.

5.2. Extending the effective context window

We evaluate whether SELF-STUDY allows us to accurately process corpora that exceed the context window length. To study this, we consider the MTOB dataset, and LLAMA-8B, which has a context window of 128k tokens. MTOB provides two different long documents: a full 484k token latex textbook and a shorter 60k token version, which was

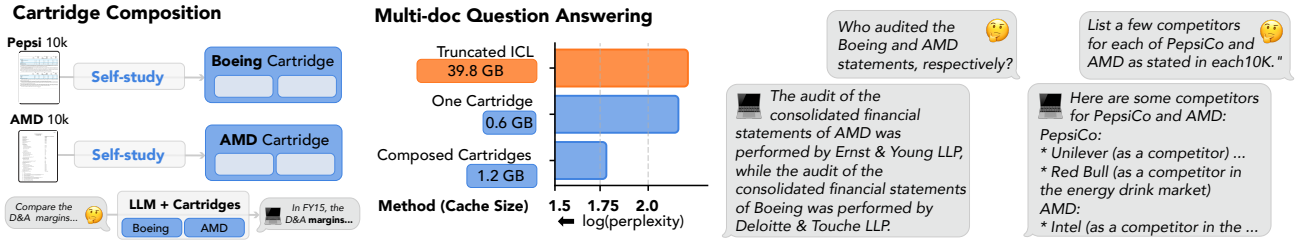

Figure 5. **CARTRIDGE Composition.** (**Left**) Illustration of CARTRIDGE composition, where two independently trained CARTRIDGES (one for a PepsiCo 10k and one for an AMD 10k) are concatenated without any additional training. (**Middle**) We evaluate composition on a dataset of multi-document questions requiring information in two different $approx$100k token documents with LLAMA-3B (see Appendix D). The x-axis shows log-perplexity (lower is better) on gold-standard answers. We compare CARTRIDGE composition with an (a) ICL baseline where we truncate the document to fit in the 128k token context length and (b) an CARTRIDGE baseline where we only include the CARTRIDGE for one of the documents. (**Right**) Examples of responses to multi-document questions using composed cartridges.

manually-curated by the dataset authors to exclude content not relevant to the translation task. Even though the 484k textbook is 356k tokens *longer* than LLAMA-8B's context window length, we can produce an CARTRIDGE for the full textbook thanks to the chunking strategy of SELF-STUDY.

Figure 3 (middle plot) shows the performance of CARTRIDGES of various sizes trained with SELF-STUDY. We show that As a point of comparison, we provide the results for KV cache baseline methods on the smaller 60k token textbook, and also include ICL on a truncated version of the long textbook. Like above, we observe that CARTRIDGE can match the performance of ICL while requiring substantially less memory. CARTRIDGES also outperform competitive baselines at every KV cache size, by up to 11.0 chrF points.

5.3. Composing CARTRIDGES

We evaluate if independently trained CARTRIDGES can be *composed* in order to serve queries about multiple different (see Figure 5, Left). We train CARTRIDGES across sizes $\{512, 1024, 2048, 4096\}$ and long 10k documents from AMD, Pepsi, AMEX, and Boeing (Islam et al., 2023). For each pair of CARTRIDGES pairwise (6 pairs per cache size), we evaluate using a dataset of *multi-document questions*, i.e., requiring information from both 10-ks. Surprisingly, we find composition not only leads to coherent LLM generations *off-the-shelf without any re-training* (Figure 5, Right), but also substantially outperforms the use of a single CARTRIDGE (*i.e.* for only AMD) or ICL (which struggles due to context length limits) (Figure 5, Center) on the multi-document questions.

5.4. Ablating SELF-STUDY design choices

We perform ablations to study different aspects of SELF-STUDY and CARTRIDGE parameterization. We provide full results in Appendix A and highlight key findings here.

First, we ablate the SELF-STUDY design choices discussed in Section 4. We show that using a context distillation objective (defined in Section 4.2) on the synthetic conversation data improves accuracy by 3.7 points (47.6 $\rightarrow$ 51.3 on LONGHEALTH) over a next-token prediction objective on the same synthetic data. See Appendix A for details on these results. Furthermore, we demonstrate how training with SELF-STUDY, as opposed to a simpler next-token-prediction objective on the raw corpus $\mathcal{C}$, improves the generality of the CARTRIDGES in Figure 2. We also confirm that using a small set of diverse seed prompts (defined in Section 4.1) improves over a single generic seed prompt by up to 4.8 accuracy points (43.6 $\rightarrow$ 48.4 on LONGHEALTH).

Next, we evaluate the parameterization and initialization choices discussed in Section 3. We show that initializing the CARTRIDGE to a KV cache of the first p tokens of the corpus improves training stability and convergence. We also compare a prefix/ parameterization, like the one we use, with LoRA. We observe differences in the generalization on tasks unrelated to the document like MMLU (Hendrycks et al., 2020). Appendix A gives a detailed analysis of the parameterizations.

6. Discussion and conclusion

We propose CARTRIDGES as an alternative to ICL for settings where many different user messages reference the same corpus of text. We demonstrate across diverse set of language model workloads that, when trained via SELF-STUDY, they match ICL's response quality while substantially reducing memory consumption (38.6$\times$ memory reduction across our evaluations) and increasing peak throughput (26.4$\times$ higher tokens per second). CARTRIDGES are simple to train, composable, and compatible with existing LLM serving infrastructure.

However, compared with ICL, SELF-STUDY is not without limitations. Using SELF-STUDY to produce a KV-cache is much more costly than simply running standard ICL

pre-fill. With our unoptimized implementation, training an ICL-quality CARTRIDGE takes $\sim$ 30 minutes on a single $8\times$H100 node (for Llama 8B). So our work does not provide a drop-in replacement for ICL, but rather demonstrates one way to tradeoff increased compute for reduced memory when constructing a KV-cache. This tradeoff is extremely advantageous in many settings: users often issue many queries over the same corpus and SELF-STUDY can be trained offline on idle or underutilized compute (*e.g.* at night when user load is low (Jaiswal et al., 2025; Goel et al., 2025)). Furthermore, there is ample room for optimizations (*e.g.* improved shared-prefix attention kernels (Dao et al., 2022; Ye et al., 2025)) that would make SELF-STUDY training procedure faster and more data-efficient.

Looking forward, we envision CARTRIDGES enabling a broad class of context-aware AI applications that are intractablewith ICL today, from medical assistants that know a patient's full medical history to LLM-powered IDEs that understand entire codebases.

References

Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., Harrison, M., Hewett, R. J., Javaheripi, M., Kauffmann, P., et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Adams, L., Busch, F., Han, T., Excoffier, J.-B., Ortala, M., Löser, A., Aerts, H. J., Kather, J. N., Truhn, D., and Bressem, K. Longhealth: A question answering benchmark with long clinical documents. *arXiv preprint arXiv:2401.14490*, 2024.

Ainslie, J., Lee-Thorp, J., De Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku. 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Arora, S. and Ré, C. Can foundation models help us achieve perfect secrecy? *arXiv preprint arXiv:2205.13722*, 2022.

Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zinsley, D., Zou, J., Rudra, A., and Ré, C. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024.

Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bhargava, A., Witkowski, C., Detkov, A., and Thomson, M. Prompt baking. *arXiv preprint arXiv:2409.13697*, 2024.

Chang, C.-C., Lin, W.-C., Lin, C.-Y., Chen, C.-Y., Hu, Y.-F., Wang, P.-S., Huang, N.-C., Ceze, L., Abdelfattah, M. S., and Wu, K.-C. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

Chari, V., Qin, G., and Van Durme, B. Kv-distill: Nearly lossless learnable context compression for llms. *arXiv preprint arXiv:2503.10337*, 2025.

Chen, L., Ye, Z., Wu, Y., Zhuo, D., Ceze, L., and Krishnamurthy, A. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024.

Chevalier, A., Wettig, A., Ajith, A., and Chen, D. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*, 2023.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Chuang, Y.-N., Xing, T., Chang, C.-Y., Liu, Z., Chen, X., and Hu, X. Learning to compress prompt in natural language formats. *arXiv preprint arXiv:2402.18700*, 2024.

Colombo, P., Pires, T. P., Boudiaf, M., Culver, D., Melo, R., Corro, C., Martins, A. F., Esposito, F., Raposo, V. L., Morgado, S., et al. Saullm-7b: A pioneering large language model for law. *arXiv preprint arXiv:2403.03883*, 2024.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

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

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Gandhi, S., Gala, R., Viswanathan, V., Wu, T., and Neubig, G. Better synthetic data by retrieving and transforming existing datasets. *arXiv preprint arXiv:2404.14361*, 2024.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023a.

Ge, T., Hu, J., Wang, L., Wang, X., Chen, S.-Q., and Wei, F. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023b.

Goel, K., Mohan, J., Kwatra, N., Anupindi, R. S., and Ramjee, R. Niyama: Breaking the silos of llm inference serving. *arXiv preprint arXiv:2503.22562*, 2025.

Gou, Y., Liu, Z., Chen, K., Hong, L., Xu, H., Li, A., Yeung, D.-Y., Kwok, J. T., and Zhang, Y. Mixture of cluster-conditional lora experts for vision-language instruction tuning. *arXiv preprint arXiv:2312.12379*, 2023.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Guha, N., Nyarko, J., Ho, D., Ré, C., Chilton, A., Chohlas-Wood, A., Peters, A., Waldon, B., Rockmore, D., Zambrano, D., et al. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in Neural Information Processing Systems*, 36:44123–44279, 2023.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hron, J. Legal ai benchmarking: Evaluating long context performance for llms. https://www.thomsonreuters.com/en-us/posts/innovation/legal-ai-benchmarking-evaluating-long-context-performance-for-llms/, 2025.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Huang, C., Liu, Q., Lin, B. Y., Pang, T., Du, C., and Lin, M. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*, 2023.

Islam, P., Kannappan, A., Kiela, D., Qian, R., Scherrer, N., and Vidgen, B. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

Jaiswal, S., Jain, K., Simmhan, Y., Parayil, A., Mallick, A., Wang, R., Amant, R. S., Bansal, C., Rühle, V., Kulkarni, A., et al. Serving models, fast and slow: optimizing heterogeneous llm inferencing workloads at scale. *arXiv preprint arXiv:2502.14617*, 2025.

Jayalath, D., Wendt, J. B., Monath, N., Tata, S., and Gunel, B. Long-range tasks using short-context llms: Incremental reasoning with structured memories. *arXiv preprint arXiv:2412.18914*, 2024.

Jiang, F. Identifying and mitigating vulnerabilities in llm-integrated applications. Master's thesis, University of Washington, 2024.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. LLMLingua: Compressing prompts for accelerated inference of large language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023b.

Kim, Y. and Rush, A. M. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Kujanpää, K., Valpola, H., and Ilin, A. Knowledge injection via prompt distillation. *arXiv preprint arXiv:2412.14964*, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel,

T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Li, D., Ma, Y., Wang, N., Ye, Z., Cheng, Z., Tang, Y., Zhang, Y., Duan, L., Zuo, J., Yang, C., et al. Mixlora: Enhancing large language models fine-tuning with lora-based mixture of experts. *arXiv preprint arXiv:2404.15159*, 2024a.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Li, Y. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering. *arXiv preprint arXiv:2304.12102*, 2023.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024b.

Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Liu, A., Liu, J., Pan, Z., He, Y., Haffari, G., and Zhuang, B. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37, 2024b.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024c.

Mao, Y., Xu, Y., Li, J., Meng, F., Yang, H., Zheng, Z., Wang, X., and Zhang, M. Lift: Improving long context understanding of large language models through long input fine-tuning. *arXiv preprint arXiv:2502.14644*, 2025.

Meng, F., Wang, Z., and Zhang, M. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

Mu, J., Li, X., and Goodman, N. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., and Myers, B. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

Narayan, A., Biderman, D., Eyuboglu, S., May, A., Linderman, S., Zou, J., and Re, C. Minions: Cost-efficient collaboration between on-device and cloud language models. *arXiv preprint arXiv:2502.15964*, 2025.

Nayak, N. V., Nan, Y., Trost, A., and Bach, S. H. Learning to generate instruction tuning datasets for zero-shot task adaptation. *arXiv preprint arXiv:2402.18334*, 2024.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Oren, M., Hassid, M., Yarden, N., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

Ouellette, L. L., Motomura, A., Reinecke, J., and Masur, J. S. Can ai hold office hours? *Available at SSRN 5166938*, 2025.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Pan, Z., Wu, Q., Jiang, H., Xia, M., Luo, X., Zhang, J., Lin, Q., Rühle, V., Yang, Y., Lin, C.-Y., et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*, 2024.

Popović, M. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pp. 392–395, 2015.

Qin, G., Rosset, C., Chau, E. C., Rao, N., and Van Durme, B. Dodo: Dynamic contextual compression for decoder-only lms. *arXiv preprint arXiv:2310.02409*, 2023.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Riaz, H., Bhabesh, S., Arannil, V., Ballesteros, M., and Horwood, G. Metasynth: Meta-prompting-driven agentic scaffolds for diverse synthetic data generation. *arXiv preprint arXiv:2504.12563*, 2025.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

Russak, M., Jamil, U., Bryant, C., Kamble, K., Magnuson, A., Russak, M., and AlShikh, W. Writing in the margins: Better inference pattern for long context retrieval. *arXiv preprint arXiv:2408.14906*, 2024.

Saxena, U., Saha, G., Choudhary, S., and Roy, K. Eigen attention: Attention in low-rank space for kv cache compression. *arXiv preprint arXiv:2408.05646*, 2024.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Snell, C., Klein, D., and Zhong, R. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Su, W., Tang, Y., Ai, Q., Yan, J., Wang, C., Wang, H., Ye, Z., Zhou, Y., and Liu, Y. Parametric retrieval augmented generation. *arXiv preprint arXiv:2501.15915*, 2025.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Tanzer, G., Suzgun, M., Visser, E., Jurafsky, D., and Melas-Kyriazi, L. A benchmark for learning to translate a new language from one grammar book. *arXiv preprint arXiv:2309.16575*, 2023.

Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

U.S. Securities and Exchange Commission. How to read a 10-k, 2011. URL https://www.sec.gov/answers/reada10k.htm. Accessed: 2025-05-14.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wan, Z., Wu, X., Zhang, Y., Xin, Y., Tao, C., Zhu, Z., Wang, X., Luo, S., Xiong, J., and Zhang, M. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *arXiv preprint arXiv:2406.13035*, 2024.

Wang, Z., Jin, B., Yu, Z., and Zhang, M. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024.

Wu, X., Huang, S., and Wei, F. Mixture of lora experts. *arXiv preprint arXiv:2404.13628*, 2024.

Xiao, C., Zhang, Z., Song, C., Jiang, D., Yao, F., Han, X., Wang, X., Wang, S., Huang, Y., Lin, G., et al. Configurable foundation models: Building llms from a modular perspective. *arXiv preprint arXiv:2409.02877*, 2024a.

Xiao, G., Tang, J., Zuo, J., Guo, J., Yang, S., Tang, H., Fu, Y., and Han, S. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024b.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks, 2024c. URL https://arxiv.org/abs/2309.17453.

Yadav, P., Raffel, C., Muqeeth, M., Caccia, L., Liu, H., Chen, T., Bansal, M., Choshen, L., and Sordoni, A. A survey on model moerging: Recycling and routing among specialized experts for collaborative learning. *arXiv preprint arXiv:2408.07057*, 2024.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024b.

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., and Ceze, L. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025. URL https://arxiv.org/abs/2501.01005.

Yen, H. Long-context language modeling with parallel context encoding. Master's thesis, Princeton University, 2024.

Yu, H., Yang, Z., Li, S., Li, Y., and Wu, J. Effectively compress kv heads for llm. *arXiv preprint arXiv:2406.07056*, 2024.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zhang, Q., Zhang, H., Pang, L., Zheng, H., and Zheng, Z. Adacomp: Extractive context compression with adaptive predictor for retrieval-augmented large language models. *arXiv preprint arXiv:2409.01579*, 2024a.

Zhang, R., Wang, K., Liu, L., Wang, S., Cheng, H., Zhang, C., and Shen, Y. Lorc: Low-rank compression for llms kv cache with a progressive compression strategy. *arXiv preprint arXiv:2410.03111*, 2024b.

Zhang, Y., Du, Y., Luo, G., Zhong, Y., Zhang, Z., Liu, S., and Ji, R. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning*, 2024c.

Zhang, Y., Liu, Y., Yuan, H., Qin, Z., Yuan, Y., Gu, Q., and Yao, A. C.-C. Tensor product attention is all you need. *arXiv preprint arXiv:2501.06425*, 2025.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Zhao, Z., Gan, L., Wang, G., Zhou, W., Yang, H., Kuang, K., and Wu, F. Loraretriever: Input-aware lora retrieval and composition for mixed tasks in the wild. *arXiv preprint arXiv:2402.09997*, 2024a.

Zhao, Z., Shen, T., Zhu, D., Li, Z., Su, J., Wang, X., Kuang, K., and Wu, F. Merging loras like playing lego: Pushing the modularity of lora to extremes through rank-wise clustering. *arXiv preprint arXiv:2409.16167*, 2024b.

Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Zheng, L., Guha, N., Arifov, J., Zhang, S., Skreta, M., Manning, C. D., Henderson, P., and Ho, D. E. A reasoning-focused legal retrieval benchmark. In *Proceedings of the 2025 Symposium on Computer Science and Law*, pp. 169–193, 2025.

Zhou, Y., Song, S., Liu, B., Xi, Z., Jin, S., Fan, X., Zhang, Z., Li, W., and Huang, X. Elitekv: Scalable kv cache compression via rope frequency selection and joint low-rank projection. *arXiv preprint arXiv:2503.01586*, 2025.

A. Extended Results

In this section, we ablate the main design choices of CARTRIDGES and SELF-STUDY.

A.1. CARTRIDGE design choices: parameterization and initialization

In our experiments, we parameterize the CARTRIDGE with a simplified version of prefix-tuning and initialize with a truncated KV-cache (see Section 3.4). In this section, we describe ablation experiments motivating these design choices. First, we compare two different CARTRIDGE parameterizations (Figure 6): simplified prefix-tuning (Li & Liang, 2021) and low-rank adaptation (LoRA) (Hu et al., 2022). Then, we demonstrate the importance of proper CARTRIDGE initialization (Figure 7).

Parameterization We evaluate CARTRIDGES trained on corpora from LONGHEALTH or QASPER on both *in-domain* (*i.e.* questions from LONGHEALTH or QASPER) and *out-of-domain* (*i.e.* questions from an unrelated benchmark, MMLU (Hendrycks et al., 2020)) queries.

We find that the prefix-tuning parameterization is more effective than a memory-matched LoRA parameterization on both in-domain and out-of-domain queries. This is illustrated in Figure 6 (Left), where we see that prefix-tuning occupies the top-right corner of the plot (high accuracy on both MMLU and the target dataset).

Notably, we find that as we increase the CARTRIDGE size with LoRA tuning, performance on out-of-domain queries (MMLU) drops significantly. At 1.06 GB (LoRA rank 1632), MMLU accuracy drops from 60.0% to 45.3%. This drop in performance is highly correlated with the size of the CARTRIDGE, suggesting that LoRA is not well-suited to large Cartridges, which we show in Figure 3 are important for recovering ICL performance. In contrast, with prefix-tuning the accuracy only drops to 54.3% at 1.06 GB. This degradation is mostly invariant to the size of the CARTRIDGE (54.7% at 0.15 GB), demonstrating that out-of-domain performance is robust across CARTRIDGE sizes.

On in-domain queries, prefix-tuning also outperforms LoRA, but the gap is smaller. Across all CARTRIDGE sizes, the best LONGHEALTH accuracy prefix-tuning achieves is 55.6% at 0.96 GB, while the best LoRA accuracy is 47.25% at 0.26 GB. Interestingly, LoRA accuracy at the largest CARTRIDGE sizes is lower; 41.3% at 0.96. It is possible that this is due to the out-of-domain degradation of LoRA we discussed above. Since queries in LONGHEALTH test set are quite different from the synthetic queries generated by SELF-STUDY (*e.g.* they are multiple choice and require some complicated reasoning traces), out-of-domain robustness may be also important for "in-domain" performance.

It isn't clear why prefix-tuning is so much more robust than LoRA to out-of-domain performance degradation. It is surprising given the similarity between a KV-cache and an MLP – both are linear transformations separated by a non-linearity. It is possible that this is due to the difference in the activation function (SiLU vs. Softmax). We leave a more detailed investigation into the root cause of this difference for future work.

Initialization The standard way of initializing a k token CARTRIDGE in our main paper is using the KV cache from the first k tokens of the source document. In Figure 7, we ablate different initialization source. We try two additional initalizations: *random vectors* and *random tokens*.

For *random vectors*, we simply initialize the parameters of the CARTRIDGE from a component-wise standard normal distribution. For *random tokens*, we initialize the CARTRIDGE as the KV cache of the first k tokens of arbitrary text (specifically, the Wikipedia page for gradient). The important difference between the these two strategies is that for *random tokens* the initial CARTRIDGE is "valid" KV cache produced by the model, while for *random vectors* it is not.

Freezing the attention sink A small yet important detail of training a CARTRIDGE is that we do not let the first token's key and value vectors to be trainable. As studied in (Xiao et al., 2024c), the first key vector, which corresponds to the beginning of sequence token and is thus the same for *every sequence*, acts as an "attention sink". We observed that when training a CARTRIDGE, allowing those key and value vectors to be trainable led to training instability (see Figure 8). For example, on some runs the MMLU accuracy would dip to below 30%.

A.2. SELF-STUDY design choices: data-generation and objective

In SELF-STUDY training, we use a multi-turn, seeded data-generation process and a context-distillation training objective (see Section 4). In this section, we ablate these design choices, comparing against the performance of SELF-STUDY with simpler data-generation and objectives.

Data Generation In Section 4.1, we describe how we use five different seed prompt types when generating data with Algorithm 1. These prompt types, *structuring*, *summarization*, *question*, *use cases*, and *creative*, are described in more detail in Appendix C.1.

In this section, we compare the performance of SELF-STUDY with these five prompt types against SELF-STUDY with a single prompt: *"Please generate a single chat message to begin a conversation about the information in the*

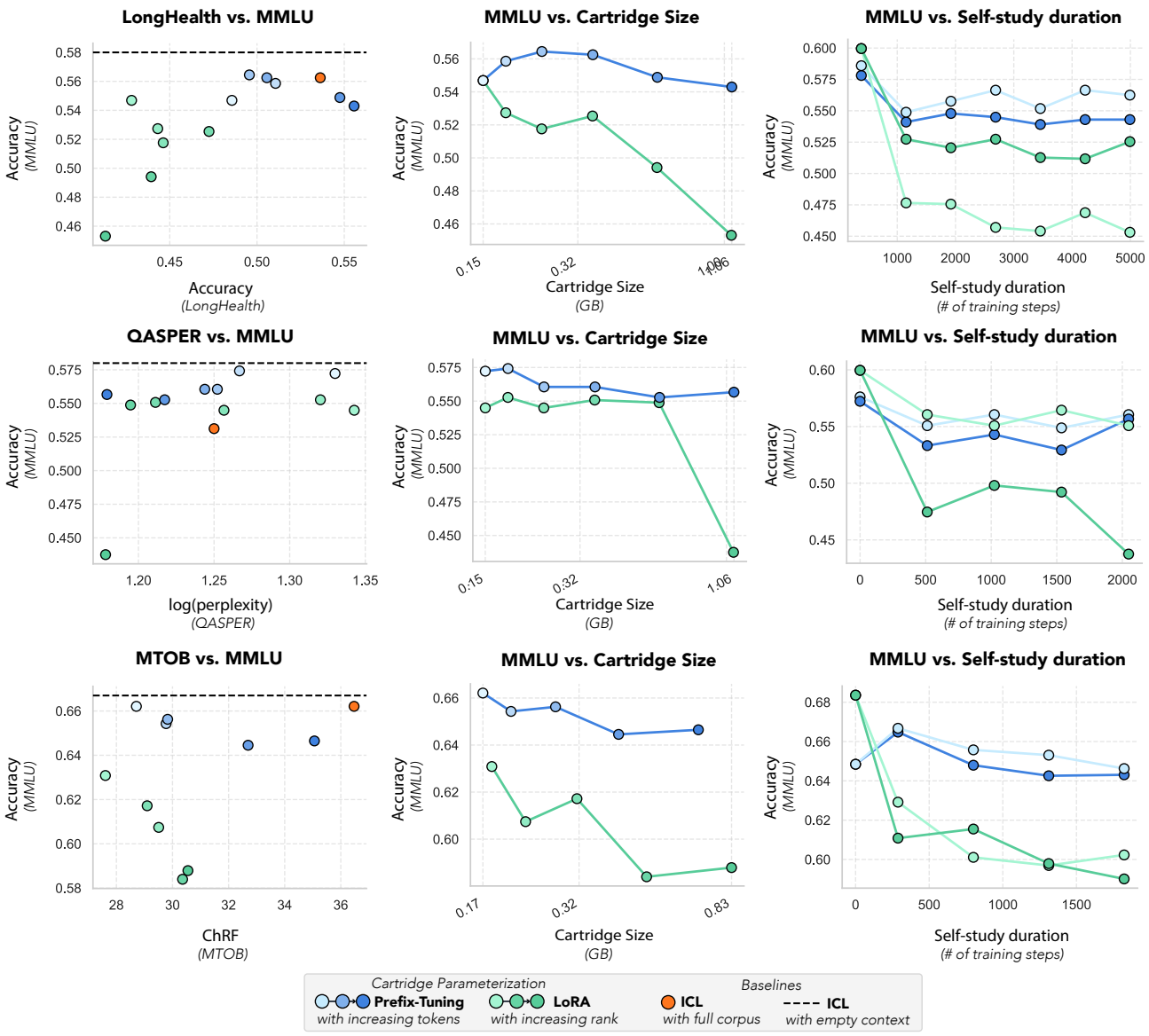

Figure 6. **Comparing CARTRIDGE parameterizations.** We train CARTRIDGES using SELF-STUDY on the corpora from LONGHEALTH (Top), QASPER (Middle), and MTOB (Bottom) using two different parameterizations: simplified prefix-tuning (as described in Section 3.4) and low-rank adaptation (LoRA) (Hu et al., 2022). We experiment with different CARTRIDGE sizes and choose LoRA rank and prefix-tuning cache size to align on memory consumption. We evaluate the performance of the CARTRIDGES on questions from the target dataset (LONGHEALTH or QASPER) using the same protocol as in Figure 3 and also on questions from MMLU (Hendrycks et al., 2020) that are unrelated to the corpora. (**Left**) The x-axis shows accuracy on MMLU and the y-axis shows accuracy on the target dataset. Each point represents a different CARTRIDGE size. (**Center Left**) The x-axis shows CARTRIDGE size in GB, and the y-axis shows accuracy on MMLU. (**Center Right**) The x-axis shows self-study duration in training steps, and the y-axis shows accuracy on MMLU. The shade of the points represents the size of the CARTRIDGE. (**Right**) The x-axis shows self-study duration in training steps, and the y-axis shows accuracy on the target dataset.

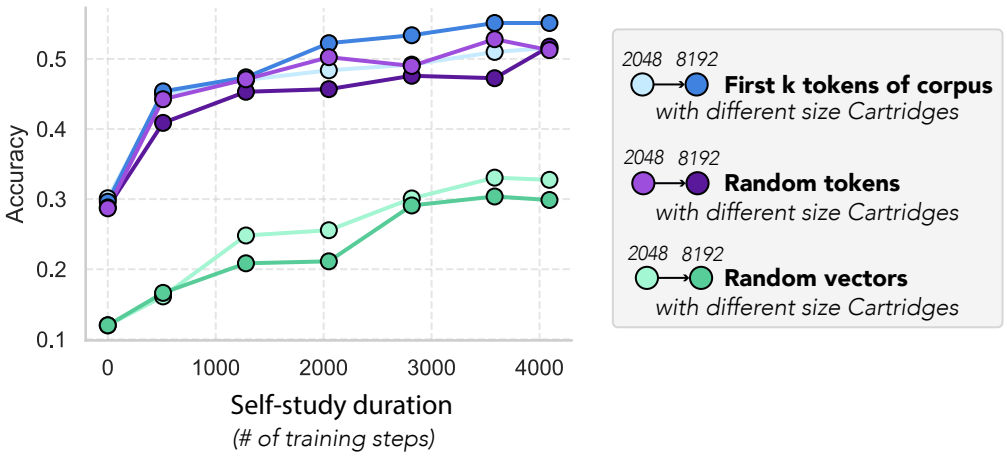

Figure 7. **Ablating CARTRIDGE initalization**. We train a CARTRIDGES using SELF-STUDY on the corpora from LONGHEALTH with 3 different initialization strategies. The x axis is the number of training steps and the y axis is the accuracy on longhealth. The blue lines are the results when initializing the CARTRIDGE using the KV cache from the first k tokens of the document. The purple lines are initializing the CARTRIDGE from the KV cache of unrelated text. The green lines is initializing the CARTRIDGE with random vectors. Initializing from the first k tokens leads to slightly stronger results than initializing from the KV cache of random text. This difference may be more prominent on other corpora where the first k tokens are more relevant to solving the downstream task.

corpus. Ask a question about the corpus or make a request."

Across three datasets, we find that using the five different prompt types during SELF-STUDY leads to higher quality CARTRIDGES (see Figure 10). On MTOB with CARTRIDGES of size 1024 tokens, we see a 7.9 point ChRF improvement (24.1 $\rightarrow$ 32.0). On LONGHEALTH, the improvement is 5.5 accuracy points (45.8 $\rightarrow$ 51.3).

Interestingly, on QASPER, we see no benefit from using the five different prompt types. It is possible this is because the queries in the QASPER dataset are mostly factual questions that do not require complex reasoning like LONGHEALTH and MTOB do.

Training Objective In Section 4, we describe the context-distillation objective we use (Snell et al., 2022; Kim & Rush, 2016; Bhargava et al., 2024). This approach requires that we collect top output probabilities from the in-context model's output distribution during data generation. A simpler alternative would be to just use a next-token prediction objective with cross-entropy.

In our comparison we find that this simpler objective underperforms the context-distillation objective (see Figure 10). Most notably, on MTOB with 2048 token CARTRIDGES, context-distillation outperforms next-token prediction by 8.3 ChRF points (24.9 $\rightarrow$ 33.2). On LongHealth, the gap is 3.7 accuracy points (47.6 $\rightarrow$ 51.3).

As shown in Figure 10, quality seems to be consistently improving with more SELF-STUDY compute. It is possible, therefore, that by spending more during SELF-STUDY with the next-token prediction objective, we could close the gap. However, for a fixed amount of SELF-STUDY compute, context-distillation is considerably more effective.

These results demonstrate how context-distillation plays an important role in efficiently recovering ICL performance with SELF-STUDY.

A.3. Throughput measurement details

We provide details for the throughput measurements in Figure 2. We use the state-of-the-art SGLang inference system, with default parameters (Zheng et al., 2024). We measure timings using an H100 GPU.

We first determine the largest batch size b that fits in GPU memory, given a cache of size k tokens. We then randomly initialize b CARTRIDGES of size k and pre-load the CARTRIDGES into GPU memory. We finally measure the time taken to decode 128 tokens per sequence. The CARTRIDGES and decoded tokens are appended to a KV-cache during generation. We report the average of 5 iterations after using 3 warm-up iterations.

B. Extended Related Work

In this section, we provide a more in-depth discussion of the place our work occupies in the broader literature. The structure below mirrors the structure of our paper: first we discuss work related to the parameterization and initialization of CARTRIDGES (Appendix B.1), then we cover work that inspired the design of SELF-STUDY (Appendix B.2),

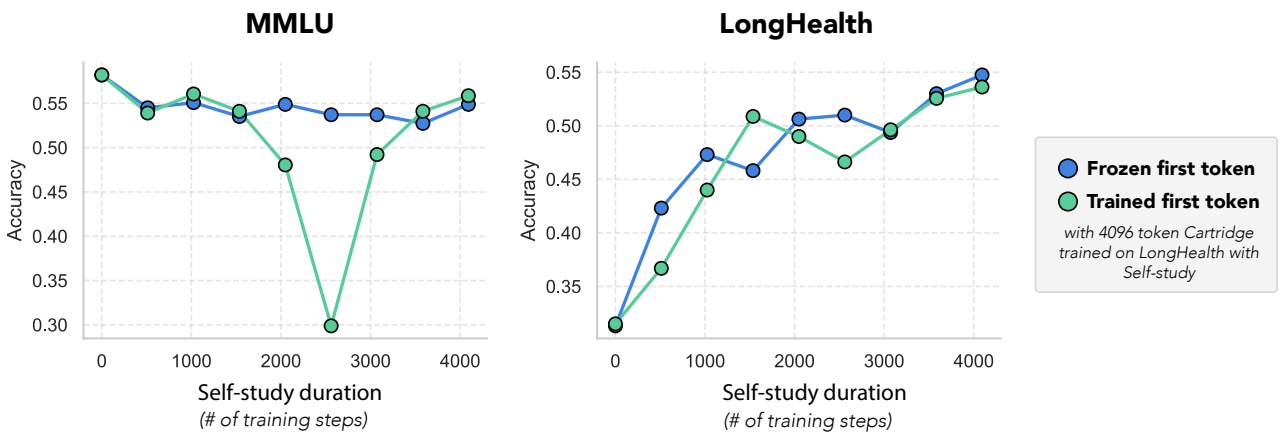

Figure 8. **Freezing the attention sink**. In both plots, the y-axis is accuracy the x-axis is training step. The left plot is MMLU accuracy and the right plot is LONGHEALTH accuracy. The green line is an exemplifies the phenomenon we observed where if the key and value vectors for the first token were trainable, we would often see training instability. In the plot on the left, the MMLU score dips to below 30% before recovering.

and finally we describe other approaches aimed at reducing the size of the KV-cache, many of which we compare against in our experiments(Appendix B.3).

B.1. Prior work related to the parameterization of CARTRIDGES

Below we discuss prior work from the parameter-efficient fine-tuning literature that inform the way we parameterize CARTRIDGES in our work.

B.1.1. PARAMETER-EFFICIENT FINE-TUNING (PEFT)

In order to adapt large language models (LLMs) to particular domains or tasks in a more compute and memory-efficient manner, several parameter-efficient fine-tuning (PEFT) methods have been developed. Some of the most widely used PEFT methods include Low-Rank Adaptation (LoRA) (Hu et al., 2022), prefix-tuning (Li & Liang, 2021), and prompt-tuning (Lester et al., 2021).

Leveraging prior observations that fine-tuned language models exhibit an intrinsic low rank structure, Hu *et al.* propose LoRA, which freezes model parameters and injects trainable rank decomposition matrices between each transformer layer. LoRA exhibits on-par or better fine-tuning quality while reducing the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times (Hu et al., 2022).

Li *et al.* and Lester *et al.* both take a different approach to lightweight fine-tuning, proposing tunable "prefixes" and "soft prompts" respectively to prepend to queries in order to steer the model to desired outputs. Li *et al.* proposes prefix-tuning, which learns a continuous representation for the activation of the prefix at each transformer layer. These learned activations are then prepended to activations obtained by passing the input prompt through the frozen transformer. In contrast, Lester *et al.* proposes prompt-tuning, which optimizes at the discrete token level and prepends a series of learnable tokens to the input prompt. Both methods show strong performance while greatly reducing the number of learnable parameters and improving compute and memory efficiency for language model adaptation.

Principal Singular values and Singular vectors Adaptation (PiSSA) (Meng et al., 2024) is another more recent PEFT method that attempts to ameliorate the slow convergence problems of LoRA. PiSSA initializes the LoRA rank decomposition matrices with the principal components of the original matrix, and exhibits faster convergence and enhanced performance compared to LoRA on several tasks, including GSM8K and MATH.

Several of these methods, especially LoRA, have been adapted specifically for distilling knowledge provided in context into the parameters of a language model. Some of those methods are described in the sections below, and this work is an extension of prefix-tuning for long-context tasks.

B.1.2. PARAMETER-EFFICIENT ADAPTER COMPOSITION AND MERGING

A number of works have explored the idea of composing multiple different parameter-efficient adapters (*e.g.* LoRAs) by summing them together, concatenating them, or using a dynamic mixture of experts (Zhao et al., 2024b; Huang et al., 2023; Xiao et al., 2024a; Zhao et al., 2024a; Yadav et al., 2024; Wu et al., 2024; Gou et al., 2023; Li et al., 2024a). For example, Huang *et al.* propose LoraHub, a framework for

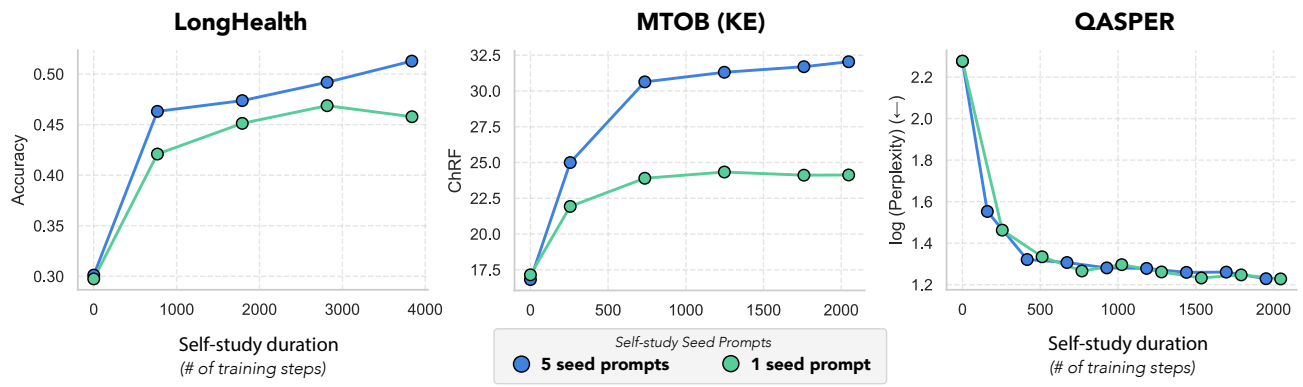

Figure 9. **Diverse seed prompts improve quality.** We generate synthetic data according to Algorithm 1 and ablate the choice of seed prompts sampled on Line 2. We consider two approaches: using a single, broad seed prompt (Green) or randomly sampling one of five different types of seed prompts (Blue). We train CARTRIDGES using self-study with these two strategies on LONGHEALTH, MTOB and QASPER corpora. In all plots, the x axis is the number of training steps, and the y axis is either accuracy (for LONGHEALTH and MTOB) or perplexity on ground truth answer (for QASPER). We use an CARTRIDGE size of 1024 tokens.

dynamically weighting and composing multiple language model adapters (Huang et al., 2023). Given a set of LoRA modules for different upstream tasks and new unseen task with in-context examples, LoraHub dynamically weights the LoRAs and composes a new LoRA module for the task. Similarly, Zhao *et al.* propose a method for dynamically *retrieving* the most relevant language model LoRAs for a given task (Zhao et al., 2024a).

B.1.3. PARAMETRIC KNOWLEDGE INJECTION

Several recent works have explored methods for integrating external knowledge directly into model parameters, known as parametric knowledge injection (Kujanpää et al., 2024; Mao et al., 2025; Su et al., 2025). To the best of our knowledge, these studies are the closest in scope to ours. Like ours, these works address the problem of parametric knowledge injection: how to store large text corpora within parameters of a language model. Some use simple synthetic data generation pipelines or context-distillation objectives. Unlike our work, these studies do not highlight the memory reduction and throughput advantages of parametric knowledge injection techniques. We highlight other differences below.

One parametric knowledge injection method, recently proposed by Kujanpaa *et al.*, is prompt distillation, in which a teacher model with access to privileged knowledge generates question-answer pairs. These pairs are then used to train a LoRA adapter for a student model (identical to the teacher model, but without access to privileged information) using a distillation objective (i.e. mimicking the teacher's full token distribution) (Kujanpää et al., 2024). This closely resembles our context-distillation objective, which we also found works better than next-token prediction. However, unlike our work, Kujanpaa *et al.* only train LoRA adapters of a single size (rank 1024) and don't assess memory reductions with respect to full in-context learning. Indeed, they do not evaluate against long-context ICL baselines at all, focusing instead on a comparison with RAG. Furthermore, they evaluate on a relatively simple long-context setting – a concatenation of SQUAD passages (Rajpurkar et al., 2016) – which does not exhibit long range dependencies or require reasoning the way MTOB and LONGHEALTH do.

Similarly, Mao *et al.* propose Long Input Fine-tuning (LIFT), which fine-tunes a language model using a typical next-token prediction objective on overlapping segments of the corpus, as well as instruction tuning on question answer pairs generated from the corpus. Unlike our work, Mao *et al.* find that synthetic Q/A pairs "offer minimal benefit and can even degrade performance due to overfitting" (Mao et al., 2025). The difference in our findings is perhaps due to the fact that they only generate *ten* synthetic examples, whereas we generate *tens of thousands*. Furthermore, they use a weaker ICL baseline (Llama 3 8B) that only has 8k tokens of context. Any contexts longer than 8k tokens are truncated before being fed to the ICL baseline.

Finally, Su *et al.* proposes Parametric Retrieval Augmented Generation (Parametric RAG), in which each document has a corresponding LoRA adapter, trained on an augmented dataset consisting of the document, rewritten versions of the document, and question-answer pairs generated from the document. At inference time, a retriever is used to determine relevants documents, and the corresponding LoRA adapters are merged (Su et al., 2025). This method demonstrates significant gains over RAG on a variety of tasks, including WikiMultihopQA.

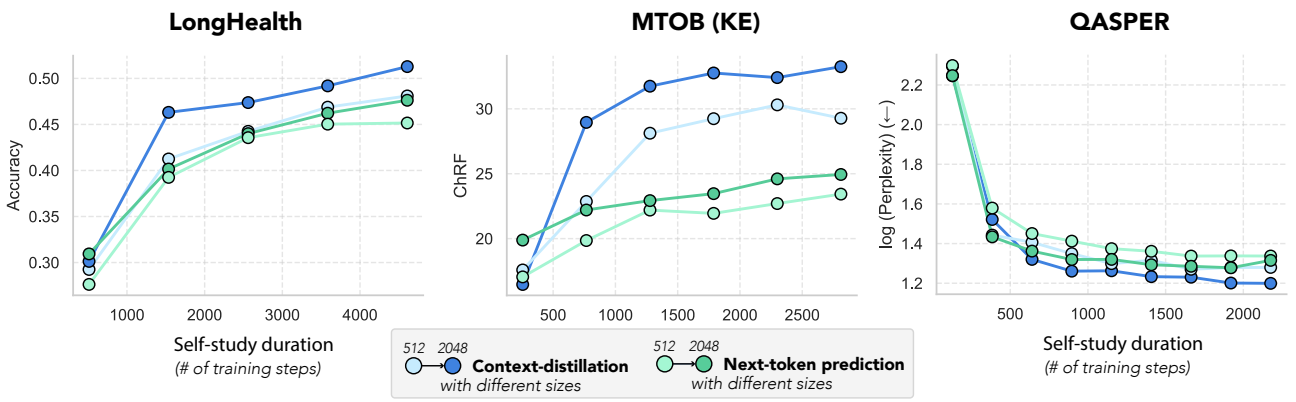

Figure 10. **Context-distillation objective improves training efficiency**. We train CARTRIDGES using SELF-STUDY on the corpora from LONGHEALTH (Left), MTOB (Center) and QASPER (Right) using two loss functions: a next token prediction loss (green) and a distillation loss (blue). We evaluate the performance of the CARTRIDGES on questions from the target dataset (LONGHEALTH, MTOB or QASPER) using the same protocol as in Figure 4. In all plots, the x axis is the number of training steps, and the y axis is either accuracy (for LONGHEALTH and MTOB) or perplexity on ground truth answer (for QASPER). The shade of the points represents the size of the CARTRIDGE. Using a distillation loss achieves higher accuracy (or lower perplexity for QASPER) across datasets and CARTRIDGE sizes.

B.2. Prior work related to SELF-STUDY

B.2.1. SELF DISTILLATION AND CONTEXT DISTILLATION

Self-distillation is another method used to internalize the performance gains provided by information in context (e.g. scratchpads, informative instructions) into the model parameters. In "Learning by Distilling Context", the authors distill a model with instructions and scratchpads in context into parameters by conditioning the model on "[instructions] + [task-input]" to predict "[scratch-pad] + [final answer]"; then fine-tuning the same model to predict its own "[final answer]" conditioned on the "[task-input]", without seeing the "[instructions]" or using the "[scratch-pad]" (Snell et al., 2024).

B.2.2. SYNTHETIC DATA GENERATION

Due to the ubiquitous need for high quality data for fine-tuning (e.g. for use with the methods described above), a large body of work has focused on generating high quality synthetic data (Nayak et al., 2024) (Abdin et al., 2024) (Gandhi et al., 2024) (Riaz et al., 2025). For example, Bonito is a model that is fine-tuned to generate synthetic data (Nayak et al., 2024), and MetaSynth is a method proposed by Riaz *et al.* that uses a language model to orchestrate several expert LLMs for domain-specific synthetic data generation (Riaz et al., 2025). The training process for Phi-4, a 14 billion parameter language model, also incorporates significant amounts of synthetically generated data (Abdin et al., 2024). Incorporating synthetic data, in conjunction with new post-training techniques, allows Phi-4 to surpass its teacher model on STEM QA tasks, as well as perform well for its size on reasoning benchmarks. These works demonstrate the potential for synthetic data generation methods to augment the capabilities of language models.

B.3. Reducing the size of the KV cache

In this section, we discuss existing approaches for reducing the size of the KV cache.

First, in Appendix B.3.3, we describe works that propose architectural changes to the multi-head attention operation, which reduce the memory footprint of the KV cache. Next, in Appendix B.3.1, we discuss *prompt compression* methods, which reduce the size of the KV cache by converting a long sequence of input embeddings into a shorter one. They can be split into hard-token methods, which output discrete tokens from the vocabulary, and soft-token methods, which output new token embeddings not from the vocabulary. Finally, in Appendix B.3.2, we describe *KV cache compression* methods. These methods directly modify the key and value matrices in the KV cache. Compared with prompt compression methods, these are more expressive because they can produce a KV cache that no sequence of input embeddings could have produced.

The methodology proposed in our work relies on cache-tuning, which could be viewed as a form of KV cache compression.

B.3.1. PROMPT COMPRESSION

Hard-token prompt compression Some works aim to reduce the size of KV cache by converting a longer text into a shorter text (Jiang et al., 2023b; Li, 2023; Chuang et al., 2024; Zhang et al., 2024a; Pan et al., 2024). These methods

are typically referred to as *hard-token* prompt compression methods because the resulting KV cache comes from discrete tokens from the vocabulary. Compared with soft-token prompt methods, these methods work well with black-box API models.

These methods can be broadly classified into two categories: filtering and summarization based methods. Filtering methods cut text from the original prompt using heuristics such as self-information. For example, LLMLingua and Selective-Context use a smaller LLM to filter a long prompt (*e.g.* dropping redundant tokens) before passing it to the main model (Jiang et al., 2023b; Li, 2023). Summarization methods paraphrase a long prompt into a smaller number of tokens (Chuang et al., 2024).

Soft-token prompt compression with adapted LLMs In one line of work, researchers train a model (typically an adapted LLM) to compress a long prompt into a smaller number of soft tokens (Chevalier et al., 2023; Yen, 2024; Ge et al., 2023b; Mu et al., 2023; Qin et al., 2023).

For example, *Autocompressors* and *In-context Autoencoders* (ICAE) are LLMs that are fine-tuned to output embeddings which can be used in soft-token prompts (Chevalier et al., 2023; Ge et al., 2023b). Autocompressors are trained with full-parameter fine-tuning and leverage a recursive strategy to generate the soft prompts, whereas ICAEs are trained with LoRA and use a single forward pass to generate the soft prompts. A number of other works also propose using an auxiliary model to produce soft-tokens from a long prompt (Ge et al., 2023b; Qin et al., 2023). *Gisting* is another method that differs from those above in that it uses the same LLM to compress the prompt into soft tokens as it uses to generate the response (Mu et al., 2023).

Soft-token prompt compression via gradient-descent Soft tokens can also be produced by optimizing input token embeddings with gradient descent. This idea, called *prompt tuning*, was first proposed for the purpose of conditioning a frozen langauge model to perform specific tasks (Lester et al., 2021). As such, it is an important part of the parameter-efficient fine-tuning literature and is discussed in more detail in Appendix B.1.1. Since then, Li *et al.* has extended prefix tuning techniques to long-context settings, proposing a new method called prefix propagation, which conditions prefixes on previous hidden states to achieve superior performance on long-document tasks compared to prefix tuning (Li et al., 2024a).

B.3.2. KV CACHE COMPRESSION

Hard-token KV cache compression Motivated by the observation that, in some settings, a small number of keys dominate the attention scores of subsequent queries, several works have proposed *KV cache eviction policies* wherein keys and values are dynamically dropped during generation (Ge et al., 2023a; Zhang et al., 2023; Tang et al., 2024; Oren et al., 2024). For example, H20 drops keys and values from *generated tokens* based on a running sum of historical attention scores (Zhang et al., 2023). Similarly, SnapKV drops keys and values from *prompt tokens* based on a window of queries from the end of the prompt (Li et al., 2024b).

A major limitation of eviction methods is that once a key is evicted, it cannot be recovered. Instead of evicting keys permanently, another line of work focuses on selectively loading keys from KV cache to SMs. While these works do not reduce memory consumption of the KV cache, they can speed up inference by making better use of GPU memory bandwidth (Ribar et al., 2023; Tang et al., 2024). For example, the Quest method estimates critical tokens at each decoding step and selectively loads them to SMs (Tang et al., 2024).

Compared with the hard-token *prompt compression* methods, KV-cache compression methods allow fine-grained control at the level of an attention head. This means that a token can be dropped from one attention head but not another.

Soft-token KV cache compression with merging In another line of work, instead of evicting tokens from the KV cache, researchers propose merging similar tokens (Wang et al., 2024; Zhang et al., 2024c; Wan et al., 2024; Liu et al., 2024b). For example, Cache Merge (CaM) takes keys marked for eviction and merges them instead, using a weighting scheme based on attention weights (Zhang et al., 2024c). Wang *et al.* builds on this work by clustering key states into "merge sets" based on cosine similarity, and merging states within a "merge set" with a Gaussian kernel weighting scheme, which upweights states more similar to a pivotal state chosen as the token with the largest total attention score (Wang et al., 2024). Wan *et al.* expands on both these works with Dynamic Discriminative Operations (D2O), which performs optimizations at both the layer and token levels. D2O adjusts the KV cache budget for each layer based on its attention density and uses an exponential moving average mechanism to dynamically determine when a previously discarded token is similar enough to retained tokens to be merged back in (Wan et al., 2024). All of these works demonstrate promising results, offering similar or better performance on several tasks compared to a full cache with a 50% or more reduction in cache size. However, there is still room for further improvement, as these methods still fail to match full cache performance in several tasks, and even a 50% reduction in cache size may still be prohibitively expensive for very large models or very long contexts. Additionally, these works do not evaluate the effectiveness of these methods in long-context settings.

Soft-token KV cache compression with low-rank projection A number of works leverage the observation that the KV cache exhibits low-rank structure to develop compression methods (Yu et al., 2024; Chang et al., 2024; Zhang et al., 2024b; Zhou et al., 2025; Saxena et al., 2024). Similar to compression methods based on merging, compression methods based on low-rank adaptation achieve performances similar to or exceeding full caches on several tasks at 50% compression, while experiencing performance degradation upon further compression.

Soft-token KV cache compression with adapted LLMs Above we discussed how some works adapt an LLM to output a shorter sequence of soft tokens given a long context. Similarly, one could adapt an LLM to output a smaller KV cache given a long context. While less explored than the analagous prompt compression approach, there is at least one published method that falls into this category. In *KV-distill*, the authors add LoRA adapters to an LLM's query projections and train them to to produce queries which aggregate information from prior tokens (Chari et al., 2025). The adapter is applied selectively to some tokens and only these tokens are kept in the KV cache. The idea is that these selected tokens can act as sinks to collect information from prior tokens. The adapter is trained with a distillation objective between a compressed and uncompressed KV cache. However, unlike our work, KV-distill does not use any training at test time.

Soft-token KV cache compression with gradient-descent The idea of treating the keys and value matrices in a KV cache as weights and training them with gradient descent was first discussed in the prefix-tuning paper (Li & Liang, 2021). In this work, the method was not applied to long-contexts, but rather as a parameter-efficient fine-tuning method that can be applied to training datasets with input-output pairs, so we discuss it in more detail in B.1.1. Since then, we are not aware of works that have applied this technique to handle long-contexts.

B.3.3. Architectural changes

A number of works have proposed architectural changes to the original multi-head attention (MHA) operation (Vaswani et al., 2017) that reduce the memory footprint of the KV cache. Because they fundamentally alter the architecture, these methods are not immediately compatible with pre-trained models using the standard MHA operation.

The earliest works in this direction developed fixed sparsity patterns in the attention map (Beltagy et al., 2020; Child et al., 2019; Zaheer et al., 2020). For example, many works use a sliding window sparsity pattern wherein each token attends to a fixed window of tokens around it. These approaches reduce the size of the KV cache because they require only keeping around a fixed number of tokens in the KV cache. More recently, some large language models have adopted sliding window sparsity in a subset of layers/heads (Team et al., 2024).

While the methods above reduce the size of the cache by introducing sparsity at the token-level, another class of methods changes the structure of the attention heads. Multi-query attention (MQA), the earliest of such modifications, uses multiple query heads but only a single key and value head (Shazeer, 2019). While MQA dramatically reduces the size of the KV cache, it can lead to a significant drop in the expressive power of the model. Grouped-query attention (GQA) is a middle ground between MQA and MHA that allows a group of query heads to attend to a single key and value head (Ainslie et al., 2023). Many frontier models use GQA, including the Llama 3 architecture, which we use in our experiments (Dubey et al., 2024; Jiang, 2024; Yang et al., 2024a). More recently, a number of other architectural modifications have been proposed including including Multi-head Latent Attention (Liu et al., 2024a) and Tensor Product Attention (Zhang et al., 2025).

In another line of work, researchers observe that without the softmax operation in the attention mechanism (*i.e.* linearizing the attention operator), the KV cache can be faithfully represented by the fixed size matrix $K^\top V$ (Arora et al., 2024). This allows us to represent the KV cache with a single matrix whose size is independent of the context length.

Indeed, a large body of work has focused on developing architectures with fixed-size memory consumption (*i.e.* models that do away with the KV cache). Notable examples include state-space models (Gu & Dao, 2023), RNNs (Beck et al., 2024), and other linear attention variants (Arora et al., 2024; Yang et al., 2024b).

Prior work shows that there are tradeoffs between the memory consumption of an architecture and the ability of a model to perform recall-intensive tasks, when controlling for compute (*i.e.* FLOPs) (Arora et al., 2024). In this context, our work shows that by increasing compute (*i.e.* FLOPs), we can reduce the memory consumption of a model without sacrificing performance. In **??**, we provide a prelinary theoretical analysis relating SELF-STUDY with recurrent architectures. However, future work should explore the relationship between CARTRIDGES and recurrent models in more depth.

B.3.4. Orchestration for long-context

In this section, we describe strategies for managing long-contexts by orchestrating calls to LLMs. For instance, the approach by (Russak et al., 2024) involves summarizing chunks of the context and then combining the summaries. Similarly, PRISM (Jayalath et al., 2024) treats the context

as a sequence of chunks, capturing key information in a structured data format. MemGPT (Packer et al., 2023) introduces a virtual memory paging system, drawing inspiration from operating systems. As context length reaches the limit of available memory, the system strategically determines which information to retain.

C. Extended method description

There are three key details of SELF-STUDY: the seed prompts, the chunking strategy, and the loss. In this section we discuss the seed prompts and chunking strategy, and in Appendix A.2 we further ablate the loss function we use.

C.1. SELF-STUDY seed prompts

As discussed in Algorithm 1, we seed the synthetic conversation generation with a prompt that elicits conversations about different aspects of the document. For each conversation, we randomly sample one of the following functions and create a seed prompt by calling it:

Structuring Seed Prompt Generator

```python
def structuring_seed_prompt(**kwargs):
    DATA_FORMATS = [
        "JSON",
        "YAML",
        "TOML",
        "INI",
        "XML",
        "plain text",
    ]

    data_format = random.choice(DATA_FORMATS)

    EXAMPLES = [
        (
            "Can you structure the information in {{subsection}} of {{document}} related to {{something specific}} "
            f"in the following format: {data_format}? "
            "Be sure to include precise information like any dates, times, names, and numerical values.'"
        ...

    ]

    example = random.choice(EXAMPLES)

    return (
        f"Please generate a single chat message instructing an LLM to structure the information in {data_format}. "
        "Output only the chat message itself and absolutely nothing else. "
        "Make sure it is clear what section and document you are asking about. "
        f"The message can follow the following template, filling in details from the corpus: \n\n'{example}'"
    )
```

Summarization Seed Prompt Generator

```python
def summarization_seed_prompt(**kwargs):
    prompts = [
        (
            "Please generate a single chat message instructing an LLM to summarize part of the corpus. "
            "Make sure the instruction is very explicit about the section of the corpus that you want to summarize. "
            "Include details (ids, names, titles, dates, etc.) that make it clear what you are asking about. "
        ),
        (
            "Please generate a single chat message instructing an LLM to summarize a section. "
            "Make sure the instruction is explicit about the section that should be summarized and the document it is from."
        ),
    ]
    prompt = random.choice(prompts)
    return prompt
```

Question Seed Prompt Generator

```python
def question_seed_prompt(**kwargs):
    prompts = [
        (
            "Generate a question for an LLM that will test its knowledge of the information in the corpus above. "
            "In your question be sure to include details (ids, names, titles, dates, etc.) that make it clear what you are asking about. "
            "Output only a single question. Do NOT include any other text or explanation other than the question."
        ),
        (
            "Generate a message for an LLM that will test its knowledge of the information in the corpus above."
            "Be sure to include details (ids, names, titles, dates, etc.) in the question so that it can be answered without access to the corpus (i.e. closed-book setting). "
            "Output only a single question. Do NOT include any other text or explanation other than the question."
        ),
        (
            "You are helping to quiz a user about the information in the corpus. "
            "Please generate a question about the subsection of the corpus above. "
            "Be sure to include details (ids, names, titles, dates, etc.) in the question to make it clear what you are asking about. "
            "Answer only with the question, do not include any other text."
        ),
    ]
    prompt = random.choice(prompts)
    return prompt
```

Use Case Seed Prompt Generator

```python
def use_case_seed_prompt(**kwargs):
    prompt = (
        "You are working to train a language model on the information in the following corpus. "
        "Your primary goal is to think about practical, real-world tasks or applications that someone could achieve using the knowledge contained within this corpus. "
        "Consider how a user might want to apply this information, not just recall it. "
        "After considering potential use cases, your task will be to generate a sample question that reflects one of these downstream applications. "
        "This question/instruction/task should be something a user, who has access to this corpus, might ask when trying to accomplish their specific goal. "
        "Output only a single question. Do NOT include any other text or explanation other than the question."
    )
    return prompt


```

Creative Seed Prompt Generator

```python
def creative_seed_prompt(**kwargs):
    prompt = [
        (
            "You are having a creative conversation inspired by the information in the corpus. "
            "Please generate a question for your conversation partner to start off the discussion. "
            "Answer only with the question, do not include any other text."
        ),
    ]
    return random.choice(prompt)
```

C.2. SELF-STUDY chunking

For the SELF-STUDY data generation process, we extract uniformly random token-level chunks from the input corpus $\mathcal{C}$. A corresponding textual description is generally prepended to each chunk $\tilde{c}$ to contextualize it when generating the seed prompt. This approach helps the model focus on different parts of the corpus and generate diverse synthetic examples. The specific chunking parameters and descriptions are tailored to each dataset:

- **LONGHEALTH:** Chunks are sampled with a minimum size of 512 tokens and a maximum size of 4096 tokens. The accompanying description is: *'Below is a section of a patient's medical record. It is part of a larger corpus of medical records for $N_{patients}$ different patients.'*
- **AMD/FinanceBench:** Fixed-size chunks of 8192 tokens are utilized. No specific descriptive text is prepended to these chunks.
- **MTOB:** Chunks are sampled with a minimum size of 512 tokens and a maximum size of 4096 tokens. The description used is: *'The following is an excerpt from a grammar book about the Kalamang language.'*
- **QASPER:** Following our general methodology, chunks are sampled with a minimum size of 512 tokens and a maximum size of 4096 tokens. A generic description is used to contextualize the chunk as an excerpt from a research paper, in line with the nature of the Qasper dataset.

D. Datasets

D.1. GENCONVO

To evaluate the ability of our approach to handle diverse queries over long documents, we generated the GENCONVO dataset. We created GENCONVO using the AMD 2022 10-K filing, a document from the FinanceBench corpus (Islam et al., 2023). The primary purpose of GENCONVO is to simulate a wide range of tasks a user might ask a model to perform given a long document, thereby testing the model's comprehension, reasoning, and ability to extract varied types of information. The generation process relies on Claude Sonnet 3.7 (Anthropic, 2024) and is structured as follows:

1. **Document Input:** The entire source document (e.g., the AMD 2022 10-K, which is less than 200,000 tokens and fits within the model's context window) is provided to Claude Sonnet 3.7.
2. **Question Generation:** A series of distinct prompt templates (detailed below), designed to elicit different reasoning traces (e.g., factual recall, synthesis, multi-hop reasoning), are used to generate questions. For the given document and each prompt template, we ask the model to generate 16 unique questions. This involves providing the model with the full document content alongside the specific question-generation prompt.
3. **Answer Generation:** Subsequently, for each generated question, Claude Sonnet 3.7 is prompted again with the original full document and the generated question to produce an answer. This process ensures that the answers are grounded in the provided document.

We hope GENCONVO provides a challenging benchmark that moves beyond simple fact retrieval, assessing a model's capacity for deeper understanding and more complex information processing over long contexts. The following prompt templates were utilized for the question generation phase:

Factual Prompt Template

```
Please generate a question to test
someone's ability to remember factual
details from the document. The answer
should be a few tokens long and be a
factual detail from the statement, such
as a number, entity, date, title, or
name.
This question should not be common
knowledge: instead, it should be
something that is only answerable via
information in the document.
```

Knowledge Prompt Template

```
Please generate a question that requires
combining information mentioned both
inside and outside the document.
This question should require using a
fact from the document and also a fact
that you are confident about, but is not
mentioned in the document. For instance:
- What are the founding dates of the
companies that got acquired this year?
This is a good question because the names
of the acquired companies are mentioned
in the document and the founding dates
are not mentioned. - What is the name
of the CEO's spouse? This is a good
question because the name of the CEO
is mentioned in the document and the
spouse's name is not mentioned.
The answer should be a fact that is a
few tokens long such as a number, entity,
date, title, or name.
```

Disjoint Prompt Template

```
Please generate a multi-hop question
that tests someone's ability to use
factual information mentioned in at least
two very different sub-sections of the
document.
This question shouldn't be a standard
question about this kind of document.
Instead, it should ask about two
particularly disconnected ideas, like
comparing information about the amount of
owned space for the company headquarters
with the amount of dollars of estimated
liability or comparing the revenue number
with the number of employees.
This question should also test one's
ability to do retrieval: do not give
away part of the answer in the question.
Ensure that for one to get the correct
answer to the question, they need to
understand the document.
The answer should be a short: for
example, a number, entity, date, title,
or name.
```

Synthesize Prompt Template

```
Please generate a question that requires
synthesizing and aggregating information
in the document.
For instance, you could ask someone to
summarize a page of the document, list
all the key competitors mentioned in
the document, or summarize the company's
business model.
```

Structure Prompt Template

```
Please generate a question that requires
understanding the structure of the
document.
This question should be more about the
structure of the document, rather than
the precise statement details. For
instance, you could ask someone to
list the titles of all the sections
in the document, describe the document
structure, report the total number of
pages, ask which section amongst two
sections comes first, or report the
section with the largest number of
tables.
```

Creative Prompt Template

```
Please generate a question about the
document to test someone's ability to
comprehend the content of the document.
This question specifically should be
focused on their ability to generalize
the information about the document to a
strange question of sorts.
This question shouldn't be a standard
question about this kind of document, it
should ask to do something abnormal and
creative, like writing a poem about a
financial document.
```

Counting Prompt Template

```
Please generate a question that requires
counting how frequently different events
occur in the document.
This question should be about statistical
properties of the document, rather than
the statement details. For instance, you
could ask someone to count the number of
times the word "million" is mentioned or
count the length of the shortest section
title.
The answer should be a number.
```

Reasoning Prompt Template

```
Please generate a question that requires
mathematical reasoning over the values in
the document.
This question should require going beyond
the facts directly mentioned in the
statement, such as asking to compute the
```

```
percentage increase in revenue between
two years, find the largest expense
category, or calculate difference in
profit between two years.
The answer should be a number.
```

D.2. LONGHEALTH

LONGHEALTH is a benchmark for evaluating large language models ability to analyze and interpret long clinical texts (Adams et al., 2024). The benchmark consists of 20 fictional clinical case reports (each containing between 5,090 and 6,754 word) and 400 multiple-choice questions based on them.

In our experiments, the context $\mathcal{C}$ consists of the reports for a *panel* of n patients. We evaluate in two settings:

1. LONGHEALTH HALF with $n = 10$ patients, with a full panel of approximately 100k tokens, which fits in the context length of the LLAMA 3 models.
2. LONGHEALTH FULL with $n = 20$ patients, with a full panel of approximately 200k tokens.

The questions are categorized into information extraction, negation, and sorting.

A **sorting** question is included below:

```
Please answer the question below about
the following patient: ID patient_03,
Name: Mr. John Williams, Birthday:
1956-08-08 00:00:00, Diagnosis: Multiple
Myeloma
<question>
Mr. Williams received multiple
radiologic examinations. In which order
did she receive them?
</question>
<options>
CT Whole Body > MR Spine Scan > CT Spine
Scan > PSMA-PET-CT Scan > CT Chest > CT
Whole Body > Whole Body CT scan
Whole Body CT scan > CT Spine Scan > CT
Whole Body > MR Spine Scan > CT Chest >
PSMA-PET-CT Scan > CT Whole Body.
CT Whole Body > CT Whole Body > CT Chest
> CT Chest > PSMA-PET-CT Scan > MR Spine
Scan > CT Spine Scan > Whole Body CT scan
> Chest X-ray
CT Chest > CT Spine Scan > CT Whole Body
> Whole Body CT scan > PSMA-PET-CT Scan >
MR Spine Scan > CT Whole Body
Whole Body CT scan > CT Spine Scan > CT
Whole Body > MR Spine Scan > CT Chest >
CT Whole Body > PSMA-PET-CT Scan
</options>
You should first think step by step.
Then give your final answer exactly as
it appears in the options. Your output
should be in the following format:
<thinking> {{YOUR_THOUGHT_PROCESS}}
</thinking>

<answer>
{YOUR_ANSWER}
</answer>
```

An example of a **negation** question is included below:

```
Please answer the question below
about the following patient: ID
patient_01, Name: Anna Sample,
Birthday: 1970-01-01 00:00:00,
Diagnosis: DLBCL
<question>
Which of these examinations were
never performed in Mrs. Sample?
</question>
<options>
Bone marrow aspiration
CSF aspiration
MRI of the head
Pulmonary function testing Cardiac
stress testing
</options>
You should first think step by step.
Then give your final answer exactly
as it appears in the options. Your
output should be in the following
format:
<thinking> {{YOUR_THOUGHT_PROCESS}}
</thinking>

<answer>
{YOUR_ANSWER}
</answer>
```

D.3. MTOB

The Machine Translation from One Book (MTOB) benchmark tests a large language model's ability to learn to translate between English and Kalamang, a low-resource language with virtually no web presence (Tanzer et al., 2023). The core task is to perform translation (Kalamang to English, and English to Kalamang) by primarily relying on a single comprehensive grammar book and a small set of accompanying linguistic resources.

The source documents provided by the MTOB benchmark are:

- **A grammar of Kalamang**: A comprehensive grammar textbook, with the original source provided in LaTeX format. This book details the phonology, morphology, and syntax of Kalamang.

- **Bilingual Word List (W)**: A list of Kalamang words with their part-of-speech tags and English descriptions.
- **Parallel Kalamang-English Corpus (S)**: A collection of 375 paired Kalamang-English sentences.

The MTOB authors preprocessed the grammar textbook from its original LaTeX source into several plaintext splits for their baseline experiments. These include:

- G^m **(Medium-length chunk)**: A plaintext segment of approximately 50,000 tokens consisting of an overview chapter, a morpheme table from the grammar book, and the complete bilingual word list (W).
- G^l **(Long-length chunk)**: A larger plaintext segment of approximately 100,000 tokens, containing chapters from the grammar book that the MTOB authors deemed most important for the translation task.
- **Full Plaintext Textbook (G)**: The entire grammar book converted to plaintext.

The combination of the long-length chunk (G^l), the parallel sentences (S), and the word list (W) exceeds the context window of Llama 3 models. We use the medium-length chunk G^m and the parallel sentence list S as input for our ICL baseline.

D.4. QASPER

QASPER is a benchmark for evaluating the ability of large language models to answer questions about scientific papers (Dasigi et al., 2021). To create a challenging multi-query long-context setting resembling the setup described in Section 3.1, we concatenate 16 papers all related to *QA NLP models* to form out corpus $\mathcal{C}$. In total, there are 78 questions about these 16 papers in the dataset, which we use as the queries Q.

Because the dataset only includes short answers and ground-truth spans containing evidence for each answer, we rewrite the answers in a longer, more conversational format using GPT-4.1 and use these as the targets when evaluating.