# Learned Structure in Cartridges: Keys as Shareable Routers in Self-Studied Representations

**Maurizio A. Diaz**
mau@mit.edu

## Abstract

A bottleneck for long-context LLM inference is the linearly growing KV cache. Recent work has proposed CARTRIDGES, an approach which leverages offline compute to train a much smaller KV cache than is typically required for a full document (up to 40x less memory usage at inference time). In this paper, we present the first mechanistic exploration of the learned CARTRIDGE key-value cache structure. In particular, we propose that (1) CARTRIDGE keys act as stable, shareable retrieval routers for the compressed corpora and (2) most of the learned compression occurs within the CARTRIDGE value vectors. We present empirical evidence of our routing theory across tasks, model families, and model sizes; for example, we can ablate the learned CARTRIDGE key vectors between tasks with little performance loss. Finally, we propose a slight improvement in initialization called Sampled Chunk Initialization (SCI). We suggest that SCI can lead to faster CARTRIDGE convergence than previously demonstrated in the literature. Our findings lay the groundwork for broader empirical study of CARTRIDGE training optimization which may be crucial for further scaling.

## 1 Introduction

As context windows grow for large language models (LLMs) [26], users expect to process larger corpora. Common use-cases are large code bases [20], financial filings and market data [11], legislature [8], or personal files [1]. The most common solution is in-context learning (ICL) wherein we provide the full context to the downstream model; however, ICL-based inference relies on a key-value (KV) cache that grows linearly with context window utilization [4]. Because the cache grows linearly, we need to consider (1) the memory consumption of storing context for ICL and (2) the resulting reduction in throughput due to increased pressure on-device bandwidth and compute [6]. This leaves users with an undesirable tradeoff between convenience and inference cost.

Recent work tackling long-context inference have primarily focused on server-level optimizations. Server-level approaches most notably include prompt (or prefix) caching, prefix-sharing (cascade and bifurcated attention, Hydragen [13][28][2]), and distributed approaches like ring attention.[18] However, these methods do not directly address compute and bandwidth pressures. In response, a growing roster of cache slimming protocols has emerged and we can divide them into two classes: token reducers and key-value compressors. Token reducers rely on summarization, chunking, or filtering to reduce corpora directly via heuristics or natural language [12][3]. Key-value compressors directly leverage sparsity or query oracles to project KV caches into lower rank spaces [7][24].

While these approaches are promising, they typically come with a quality tradeoff. A more recent key-value compression method proposes a two-stage pipeline [5]:

1. **Self-Study:** Exhaustively chunk the corpus and prompt a model to quiz itself on the content. These synthetic rollouts can be massively parallel and help build a diverse training dataset.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Workshop on Mechanistic Interpretability.

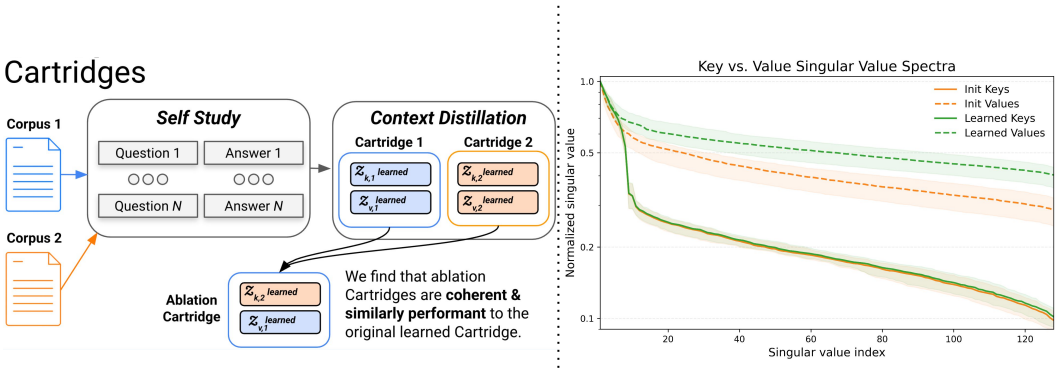


Figure 1: **(Left)** CARTRIDGES learn to compress long-context documents by first generating synthetic conversations about the corpus and then training a small KV cache using the synthetic traces. This process is called SELF-STUDY. Leveraging a context distillation objective, we back-propagate SELF-STUDY traces into trainable KV caches while keeping the rest of the model frozen. **(Right)** Here we plot the layer-wise mean singular value spectra of a LLAMA 3.1 8B CARTRIDGE's KV vectors before and after training [19]. The resulting key vectors are stable while the learned value vectors increase in singular value, representing a more efficient use of representation space due to compression.

2. **Context Distillation:** Once we have our conversational traces, we initialize the model with a fixed-size key-value cache of length $p \ll d_{\mathcal{C}}$ where $\mathcal{C}$ is our Corpus. Then, we can train on a context distillation objective to align our CARTRIDGE-initialized model's next token distribution with the SELF-STUDY synthetic traces [9][23].

The synthetic trace generation process is SELF-STUDY and the final context-distilled cache is called a CARTRIDGE. CARTRIDGES exhibit interesting properties, including composability, high recall accuracy on benchmarks, and a tensor structure that is ideal for high throughput prefix-sharing inference engines *e.g.* Tokasaurus [14].

In this paper, we contribute a preliminary investigation into the mechanisms of a trained CARTRIDGE. In particular, we study the structure of the learned CARTRIDGE key and value vectors across model families, scale, and corpora. We can summarize our results as follows:

- We observe that keys barely change during CARTRIDGE training whereas values change significantly. We use singular value analysis to summarize this trend and argue that value vectors learn to use their representation dimensionality more effectively to maximize compression. This result likely explains why random initialization failed to converge in the original CARTRIDGES paper.
- We ran an ablation experiment where we train two CARTRIDGES with shared initializers on different tasks and then swap the learned key vectors but not the values. We note that responses remain coherent and the resulting performance loss is mild, which aligns with our first observation.
- We investigate a simple but effective initialization scheme for our CARTRIDGE. Rather than using the first $p$ tokens of our corpus to initialize the CARTRIDGE (the main method used in the original paper), we randomly sample chunks throughout the full corpus to maximize structural diversity. This approach leads to statistically significant improvements for training convergence.

## 2 Preliminaries

In this section, we position CARTRIDGES as a new parameter efficient fine-tuning (PeFT) technique that borrows directly from prefix-tuning. We briefly summarize prior work on PeFT mechanistic interpretability, highlighting the lack of related work to prefix-tuning based PeFT methods. Next, we provide more details on CARTRIDGES with a focus on the relevant experimental parameters

and notation for our downstream experiments. Finally, we briefly cover the relevant datasets and benchmarks to reduce friction when framing our analyses in Sections 3 and 4.

### 2.1 Related Work

There is a significant body of work exploring task-specific LLM specialization. Common PeFT techniques are low-rank adapters (LoRA) and prefix-tuning, with the former being the most popular method [10][17]. CARTRIDGES primarily borrow from prefix-tuning: both methods prepend a learned representation to the input sequence and rely on the model to treat the trained prefix as real tokens. However, despite the widespread popularity of PeFT, there is comparatively little research focused on interpreting how LoRAs or prefix-tuning change models at a mechanistic level.

**Interpreting LoRA** Recent interpretability work focuses on LoRA-driven mechanistic changes when tuning for narrowly scoped tasks. For example, Lee *et al.* studied sparse neural activations during LoRA for nuclear safety applications, while Nijasure et al. found that middle layers (5-15) and specific MLP projections drive ranking performance [16][21]. These works focus on neuron-level changes and layer contributions, whereas our analysis examines the geometric structure and transferability of learned representations.

**Interpreting Prefix-Tuning** Prefix-tuning, which is much less popular than LoRA in practice, has even less relevant interpretability work. However, Petrov *et al.* provide a theoretical analysis of when prefix-tuning works well [22]. They theoretically argue that prefix-tuning "cannot change the relative attention pattern over the content" and we find that this theory aligns strongly with our experimental findings. More concretely, they argue that key vectors have fixed directional biases established during pre-training; our ablation experiments (Figure 3) support this claim directly on real downstream tasks.

### 2.2 Relevant CARTRIDGES Notation

We choose to preserve notation from the original CARTRIDGES paper to make cross-referencing easier for the reader. Taking that into consideration, we provide a brief overview of relevant definitions.

**LLMs** For an LLM $\mathcal{F}$ we define the output distribution $p_{\text{model}}(\cdot) = \mathcal{F}(\cdot|\mathbf{x})$ which is a categorical distribution over the model's vocabulary $\mathcal{V}$. Given a corpus $\mathcal{C}$ and queries $q$, we can define in-context learning (ICL) decoding of our queries as $\mathcal{F}(\cdot|\mathcal{C} \oplus q)$.

**CARTRIDGE** A CARTRIDGE is a KV cache of size $p \ll n_{\mathcal{C}}$ which aims to augment an LLM $\mathcal{F}$ such that the LLM behaves as if a Corpus $\mathcal{C}$ of length $n_{\mathcal{C}}$ were fully within context. Throughout the paper, we call this CARTRIDGE $Z \in \mathbb{R}^{L \times p \times d_{\text{head}} \times 2}$ and define the augmented $\mathcal{F}$ as $\mathcal{F}_Z$. Given $l \in [1, L]$, where $l$ indexes our model layers, we can summarize the difference between traditional ICL and Cartridges as:

$$\text{ICL KV Cache:} \quad \underbrace{(\mathbf{k}[1], \mathbf{v}[1]), \ldots, (\mathbf{k}[n_{\mathcal{C}}], \mathbf{v}[n_{\mathcal{C}}])}_{\text{KV pairs for } \mathcal{C}}, \underbrace{(\mathbf{k}[n_{\mathcal{C}}+1], \mathbf{v}[n_{\mathcal{C}}+1]), \ldots}_{\text{KV pairs for } q}$$

$$\text{CARTRIDGE KV Cache:} \quad \underbrace{(\mathbf{z}^{(l)}_{\text{k},1}, \mathbf{z}^{(l)}_{\text{v},1}), \ldots, (\mathbf{z}^{(l)}_{\text{k},p}, \mathbf{z}^{(l)}_{\text{v},p})}_{\text{Trainable pairs in } Z}, \underbrace{(\mathbf{k}[1], \mathbf{v}[1]), \ldots}_{\text{KV pairs for } q}$$

During CARTRIDGE training, only the parameters composing our CARTRIDGE $Z$ of sequence length $p$ are trainable while the rest of $\mathcal{F}$ remains frozen.

### 2.3 Datasets and Benchmarks

For consistency, we re-use or reproduce the core datasets from the original CARTRIDGES paper.

**LONGHEALTH** is a question answering benchmark containing 20 fictional patients' detailed medical records. Each record is composed of multiple cases and each case is approximately 5,500 words long. For the LONGHEALTH corpus $\mathcal{C}_{\text{LongHealth}}$, each prompt $q$ is a multiple choice question with 5 options. In accordance with the original paper, we use 10 full patient records concatenated to form a 100k+ token context corpus with 200 questions to measure accuracy. This formulation is different from the

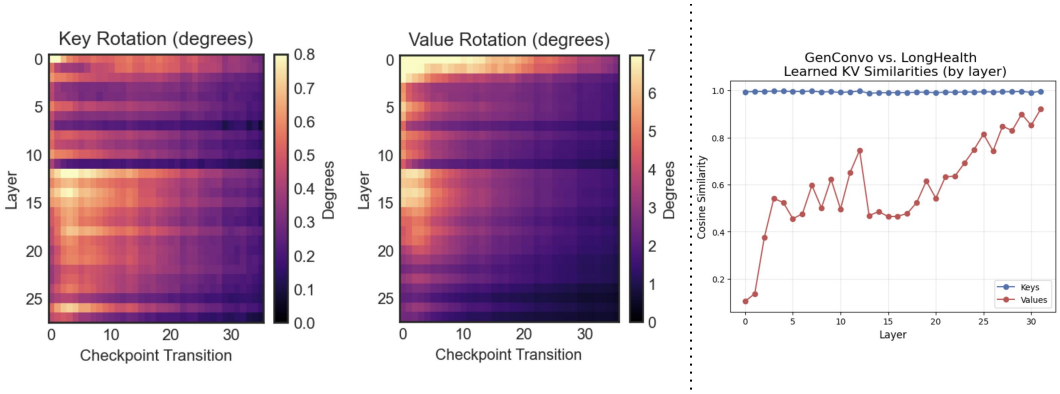


Figure 2: **(Left)** We reproduced a LONGHEALTH CARTRIDGE from the original paper. To do so, we trained a LLAMA 3.2 3B CARTRIDGE with length $p = 2048$ for 3072 optimizer updates (batch_size=64, sequence_length=1024). We checkpointed the CARTRIDGE every 96 optimizer steps and, for all layers $l \in [1, L]$, plotted the key and value vector rotation between each checkpoint. Notably, value rotations are often a full order of magnitude larger than key rotations and they continue late into the training process. **(Right)** We trained two CARTRIDGES on separate tasks: GENCONVO and LONGHEALTH from the original paper. Afterwards, we plotted the layer-wise cosine similarity between the two fully trained CARTRIDGES and note that their learned key vectors are highly similar. We explore this further in Figure 3 where we show that we can swap these key vectors with minimal downstream performance loss. On the other hand, the learned value vectors differ the most within the layers that experience the most vector rotations throughout learning.

original LONGHEALTH paper's evaluation setting which evaluates each patient independently. We prefer the CARTRIDGES formulation because it showcases longer-context reasoning and requires that the model delineate between different patient cases.[1]

**GENCONVO** is a synthetic conversational dataset based on FINANCEBENCH's 2022 AMD 10-K corpus. Our GENCONVO conversations are based on four different Q&A prompts that encourage structural, multi-hop, mathematical, and factual retrieval reasoning (Appendix C.2). We reproduce GENCONVO with Claude Sonnet 4.0 instead of 3.7 due to changes in rate limits between these models. To maximize throughput and balance API rate limit loads, we use inference-time scaling library VERDICT for GENCONVO synthesis [15]. Throughout the paper, we rely on GENCONVO perplexity to measure a CARTRIDGE's ability to compress financial reasoning. While the original GENCONVO produces 16 answers per question, we opted to generate 50 manually validated answers per question for our evaluation set.

**ARXIV** is the tutorial SELF-STUDY dataset from the CARTRIDGES GitHub. The ARXIV corpus is the source `.tex` file for the original CARTRIDGES paper (40k tokens and 64k SELF-STUDY rollouts). We only use ARXIV to train QWEN3's ABLATIONCARTRIDGE model used in Figure 3. We do not run evaluations using ARXIV.

## 3 What representation does a CARTRIDGE learn?

In this section, we'll explore the evolution of the CARTRIDGE key and value vectors throughout training. First, we'll analyze the geometric restructuring of the CARTRIDGE KV cache from initialization to convergence (Figure 2). Next, we'll formalize our singular value analysis and argue that it provides a summary of training dynamics. Finally, we'll run an ablation experiment to show that CARTRIDGE key vectors can be swapped between learned tasks with minimal performance loss (Figure 3).

[1] In LONGHEALTH, inter-patient differences can be subtle. For example, `patient_01` might receive an ibuprofen prescription that's only 50mg different from `patient_02`. By mixing the corpus, we require the model to perform multi-hop reasoning beyond memorizing just one ibuprofen dosage.

### 3.1 Observed Training Dynamics

Let's briefly refresh the SELF-STUDY context distillation objective. First, we build a teacher distribution $p_{\text{teacher}}(\cdot) = \mathcal{F}(\cdot|\mathcal{C} \oplus q)$ over the synthetic conversations derived from the corpus we want to compress. In our case, we borrow the same conversation "seed prompts" from the original paper (Appendix C.1). We then train our student $p_{\text{student}}(\cdot) = \mathcal{F}_Z(\cdot|q)$ to minimize

$$\mathcal{L} = \mathbb{E}_{(\mathcal{C},q)\sim\mathcal{D}_{\text{synth}}} \left[\text{KL}(p_{\text{teacher}} \| p_{\text{student}})\right]$$

where $\mathcal{D}_{\text{synth}}$ is the synthetic conversation dataset from SELF-STUDY, and we optimize only the trainable parameters $Z = \{(\mathbf{z}^{(l)}_{\text{k},i}, \mathbf{z}^{(l)}_{\text{v},i})\}_{l=1}^{L}$ across all layers. For convenience, we say a CARTRIDGE $Z$'s size is $p$.[2]

During this distillation, we note that our trainable key vectors rarely change whereas our value vectors change often; in particular, we measure these changes via the cosine similarity between CARTRIDGE checkpoints (Figure 2) *e.g.* $\cos(Z^{(t)}, Z^{(t+1)})$. Our hypothesis is that the key vectors rotate minimally enough that given two CARTRIDGES initialized via the same method, we could swap their key vectors arbitrarily. If this is true, it implies that CARTRIDGE key vectors act as stable routers to compressed value payloads. The magnitude difference in rotations reflects this division of labor: while values require continuous refinement for compression, keys stabilize once effective routing is established.

### 3.2 Singular Value Analysis

**Algorithm 1** Singular Value Analysis for Trained CARTRIDGE

**Initialize:** $\mathcal{S} \leftarrow \{\}$ ▷ Spectral collections
**for** vector $\in$ {KEYS, VALUES} **do**
  $\mathcal{S}[\text{vector}] \leftarrow \emptyset$
  **for** $l = 1$ **to** $L$ **do** ▷ Each layer
    $T^{(l)} \leftarrow \text{EXTRACT}(C, l, \text{vector})$ ▷ $\mathbb{R}^{h \times p \times d_{\text{head}}}$
    $X^{(l)} \leftarrow \text{RESHAPE}(T^{(l)}, [h \cdot p, d_{\text{head}}])$ ▷ Flatten heads
    $\sigma^{(l)} \leftarrow \text{SVD\_VALS}(X^{(l)})$ ▷ Singular values
    $\tilde{\sigma}^{(l)} \leftarrow \sigma^{(l)} / \sigma^{(l)}_1$ ▷ Normalize by largest
    $\mathcal{S}[\text{vector}] \leftarrow \mathcal{S}[\text{vector}] \cup \{\tilde{\sigma}^{(l)}[:k]$ ▷ Fetch the top $k$ values
  **end for**
**end for**
**return** {MEDIAN($\mathcal{S}$[KEYS]), MEDIAN($\mathcal{S}$[VALUES])}

While informative, we found the cosine similarity heatmaps too busy to establish a scalable, cross-model trend. As a response, we designed an analysis which runs a layer-wise singular value decomposition and then normalizes those values (Algorithm 1). In addition to reporting the median trend line, we also provide inter-quartile range bands (IQR) at the 25th and 75th percentile. Here's what we observed:

**Trends Across Scale** We found the high-level trend of stable routing keys and compressed value payloads consistent across model sizes (Appendix B). It seems that the model's pretrained prior is strong enough that it's more effective to focus on value re-alignment and compression than finding a more effective key structure (Figure 3).

**Differences Between Model Families** In Figure 3, we leverage our singular value analysis to explore why QWEN3 responds differently to our ablation than LLAMA (-7% vs -4-5% performance drop). We observe that QWEN3 key vectors are more heterogeneous across layers than LLAMA (via the IQR bands). The wider IQR bands directly correspond to QWEN3's larger performance drop, suggesting layer-wise key specialization reduces transferability.

[2]The most intuitive analog for $p$ is to think of a CARTRIDGE as a KV cache composed of $p$ tokens. Therefore, the context distillation objective is to compress the SELF-STUDY corpora into that KV cache such that $p$ is much smaller than the original corpus.

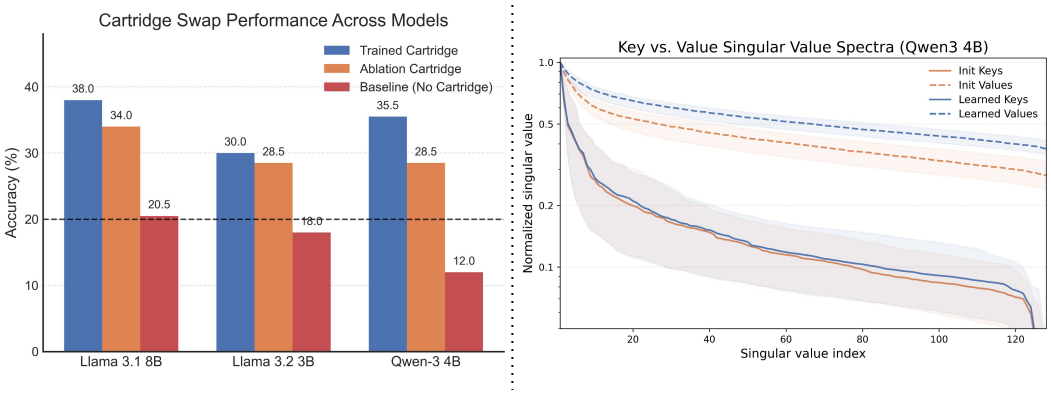


Figure 3: **(Left)** We present three LONGHEALTH evaluation settings: a baseline with no CARTRIDGE (red), a model with a LONGHEALTH-trained CARTRIDGE (blue), and a model where we swap its LONGHEALTH-trained CARTRIDGE key vectors with keys from a different task (orange). We call the latter an **ABLATIONCARTRIDGE.** For the LLAMA models, we swap in key vectors trained on GENCONVO and for QWEN3 we use key vectors trained on ARXIV data [27]. While the vector swap leads to a slight performance loss, the ABLATIONCARTRIDGE still outperforms both a random choice baseline and the model's baseline performance. **(Right)** We reran our KV cache singular value analysis on QWEN3. First, we noticed that QWEN3 exhibits the same training-time value vector singular increase of the LLAMA family. Second, QWEN3's key vector singular values are higher variance than LLAMA which might explain the larger performance loss during ablation.

Furthermore, based on how little CARTRIDGE key vectors drift throughout training we recognize that initialization might play a larger role in SELF-STUDY-based distillation than we thought. We explore this more in (Figure 4) where we show statistically significant faster convergence by randomly sampling the corpus to initialize our CARTRIDGE.

### 3.3 Key Vector Ablations

Informed by our CARTRIDGE KV cache singular value analysis, we decided to run an ablation experiment on the LONGHEALTH benchmark used in the original paper (Figure 3). For two trained CARTRIDGES $Z_A = \{(\mathbf{z}^A_{\text{k},i}, \mathbf{z}^A_{\text{v},i})\}_{i=1}^p$ and $Z_B = \{(\mathbf{z}^B_{\text{k},i}, \mathbf{z}^B_{\text{v},i})\}_{i=1}^p$ from different tasks $\{A, B\}$, we construct the ABLATIONCARTRIDGE:

$$Z_{\text{AB}} = \{(\mathbf{z}^B_{\text{k},i}, \mathbf{z}^A_{\text{v},i})\}_{i=1}^p$$

In comparison to the CARTRIDGE-enabled LLMs $\mathcal{F}(Z_A)$ and $\mathcal{F}(Z_{\text{AB}})$, we also define a baseline model $\mathcal{F}(Z_\emptyset)$ which has no CARTRIDGE. We call $\mathbf{z}^A_{\text{k},i}$ and $\mathbf{z}^B_{\text{k},i}$ **transferable** if $\text{ACC}(\mathcal{F}(Z_{\text{AB}})) > \text{ACC}(\mathcal{F}(Z_\emptyset))$ and the answer overlap between $\mathcal{F}(Z_A)$ and $\mathcal{F}(Z_{AB})$ is statistically significant (see: Table 1).

For our ablation experiments, we train all models for 512 optimizer steps with a batch size of 128 and a packed sequence length of 1024. For all experiments, we train $Z_A$ on LONGHEALTH and $Z_B$ on

Table 1: We ran a hypergeometric statistical test ($N = 200$) on the Q&A correctness overlap between trained and ablated CARTRIDGES. We confirm that the overlap is statistically significant compared to random chance in 5-question multiple choice Q&A, where $n_{\text{train}}$ and $n_{\text{ablated}}$ represent correct answers out of 200 questions and $n_{\text{overlap}}$ counts questions answered correctly by both. Notably, despite experiencing the largest performance loss due to ablation, QWEN3-4B has the most overlap of the three models we tested.

| Model | $n_{\text{train}}$ | $n_{\text{ablated}}$ | $n_{\text{overlap}}$ | $p$-value |
|---|---|---|---|---|
| Llama 3.1 8B | 76 | 68 | 34 | 0.0156* |
| Llama 3.2 3B | 60 | 57 | 28 | 0.0003* |
| Qwen-3 4B | 71 | 57 | 40 | <0.0001* |

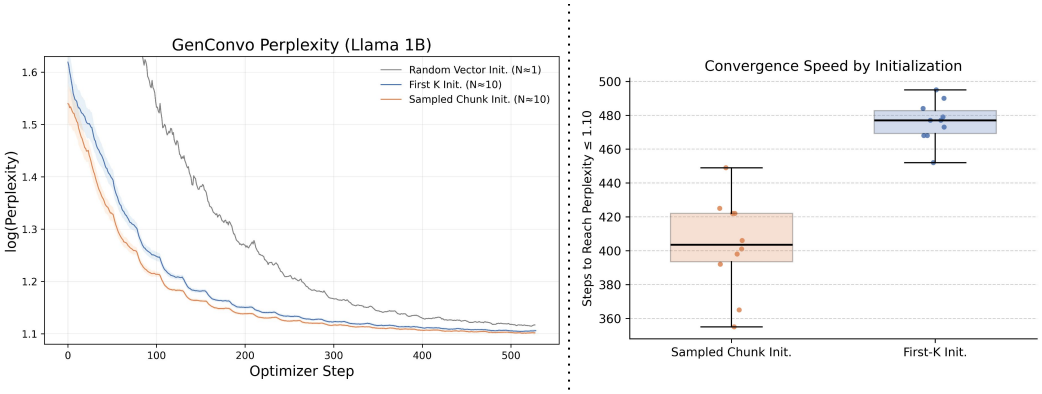


Figure 4: **(Left)** We plot the perplexity for 10 LLAMA-1B GENCONVO training runs with both Sampled Chunk Initialization (SCI) and the original paper's First $k$ Tokens Initialization. Additionally, we include a random vector initialization run for comparison. For all our SCI experiments, we chose chunksize=64 because it was the midpoint in our $n$-gram diversity vs. context length analysis (Appendix A). **(Right)** A different view on the perplexity graph, we can visualize convergence speed over our runs as a box plot. Setting a target threshold of perplexity $= 1.10$ to define convergence, we can run a paired $t$-test to confirm that SCI converges at a statistically faster rate ($p < 0.05$) than the original paper's **First-$k$ Token Initialization** scheme.

either GENCONVO or ARXIV. The CARTRIDGE sequence length $p$ is held consistent at 2048 and the initializing document is the default initializer from the CARTRIDGES repository (**gradients.txt**, the Wikipedia article for gradients). Surprisingly, we find that keys from **First $k$ Token Initialization** are transferable for LLAMA 3.2 3B, LLAMA 3.1 8B, and QWEN3.

## 4 The importance of good initialization

In this section, we propose and briefly benchmark an improved initialization scheme for CARTRIDGES. Inspired by our observation of training dynamics and KV cache geometry, we suggest that CARTRIDGE initialization benefits from the structural diversity of random sampling.

### 4.1 Known Initialization Schemes

The original CARTRIDGES paper explored a few approaches for CARTRIDGE initialization:

- **Random Vector Initialization**: random vector initialization (RVI) for prefix tuning is known to be unstable from prior literature.[17] In the original CARTRIDGES paper, the authors note that random vector initialization fails to converge and performs poorly compared to other methods. In hindsight, it's clear why: instead of focusing on value compression, a randomly initialized prefix is burdened with a joint optimization problem of both routing *and* compression.
- **First-$k$ Token Initialization**: using the first $k$ tokens from the target corpus $C$ is a natural first-pass approach and is the initialization scheme used by the original paper. By sampling from real text, we are leveraging structures the model is already comfortable working with.
- **Summary-Based Initialization**: alternatively, we could use a lighter-weight summarization model to initialize a CARTRIDGE. While attractive, we de-prioritized this approach to avoid explicitly stacking models which can introduce further uncertainty when benchmarking performance.

### 4.2 Sampled Chunk Initialization

We present Sampled Chunk Initialization (SCI) (Algorithm 2), a principled alternative to the original paper's **First-$k$ Token Initialization**. In Figure 4 we show that SCI converges faster than both **First-$k$ Token Initialization** and random initialization. In (Appendix B.4), we present further singular value analyses of the three initialization methods in (Figure 4). We note that our trained CARTRIDGES using

**Algorithm 2** Sampled Chunk Initialization (SCI)

**Require:** Corpus $\mathcal{C}$ with $n_{\text{total}}$ tokens
**Require:** Target cache size $p$, chunk size $c$
$\mathbf{x} \leftarrow \text{TOKENIZE}(\mathcal{C})$ ▷ Full corpus tokenization
$n_{\text{chunks}} \leftarrow \lfloor p/c \rfloor$
$\mathbf{s} \sim \text{Uniform}(\{0, 1, \ldots, n_{\text{total}} - c\}, n_{\text{chunks}})$
$\mathbf{x}_{\text{init}} \leftarrow []$
**for** $i = 1$ **to** $n_{\text{chunks}}$ **do**
    $\mathbf{chunk}_i \leftarrow \mathbf{x}[s_i : s_i + c]$
    $\mathbf{x}_{\text{init}} \leftarrow \mathbf{x}_{\text{init}} \oplus \mathbf{chunk}_i$
**end for**
$\mathbf{x}_{\text{init}} \leftarrow \mathbf{x}_{\text{init}}[: p]$ ▷ Truncate to target size
**return** $\text{FORWARDPASS}(\mathcal{F}, \mathbf{x}_{\text{init}})$

RVI hardly deviate from their spectra at initialization-time. Notably, the initialization spectra stays close to 1, aligning with random matrix theory which observes that randomly sampled orthogonal vectors maintain singular values close to one [25]. This differs significantly from the spectra from token-based initialization methods which decay in a consistent, structured manner.

To confirm that SCI is better than **First-$k$ Token Initialization**, we run a simple paired-$t$ statistical test. Let $\mu$ represent the mean steps to reach perplexity threshold 1.10. Then if we can reject the null hypothesis $\{H_0 : \mu_{\text{SCI}} = \mu_{\text{first-k}}\}$ against the alternative $\{H_1 : \mu_{\text{SCI}} < \mu_{\text{first-k}}\}$ we can claim statistically significant better convergence. In Figure 4 we reject $H_0$ with $p < 0.05$, confirming faster convergence.

## 5 Limitations and Future Work

### 5.1 Limitations

**Scale** We ran all experiments on single A10G, A100, and H100 spot instances. Future work should explore larger models and train for longer periods of time. Except for our pure 1:1 reproduction of the original paper in Figure 2, the rest of our models are approximately 4-6x optimizer steps under-trained compared to the best performing models from the original paper. In the appendix, we show that our singular value analysis holds for the 1:1 reproduced model (Appendix B.3).

**Baselines** When testing initialization schemes, we had to decide between method diversity and statistical significance. Future work could explore more initialization techniques, including more robust baselines *e.g.* summary-based initializations. Additionally, it would be insightful to test convergence on multiple tasks instead of just GENCONVO.

### 5.2 Future Work

**Training and Serving** Given that keys change little throughout training, it might be worth training CARTRIDGES with fully frozen key vectors. Beyond being an easy win for training efficiency, there might be desirable properties of frozen keys at inference-time. Suppose we only need trainable values for corpora compression: can we imagine a serving engine that exploits this to hot-swap CARTRIDGE value vectors at inference time while maintaining fixed key vectors? This may be a promising direction given the rise of prefix-sharing optimized serving engines like Tokasaurus [14].

Further research may also explore if there exists key vector initialization structures that are particularly good for certain tasks; or, even better, key vector initializations that are universally well-performing.

**Implications for Prefix-Tuning** While we focused on CARTRIDGES, mechanistic interpretability work for prefix-tuning remains sparse. Our findings provide a strong experimental foundation to support Petrov *et al.*'s theoretical arguments regarding the limitations of prefix-tuning. Further mechanistic work could explore the overlap between CARTRIDGE training dynamics and prefix-tuning; for example, do prefix-tuned key vectors also exhibit stability? Or, can they learn significantly novel attention patterns?

Beyond supporting existing theory, our work raises questions about which tasks are appropriate for CARTRIDGES. Future work could explore counter-examples to the patterns we observed during our experiments. Does there exist some task where keys must significantly reroute to perform well and does this delay convergence? Counter-examples could help formalize a boundary between compression-friendly CARTRIDGE tasks and tasks better suited for prefix-tuning or LoRA.

## 6 Conclusion

In this paper, we present the first mechanistic study of CARTRIDGES, a prefix-tuning based method for compressing long-context corpora prior to inference. Our analyses showed that CARTRIDGE key vectors act as a stable routing mechanism throughout training whereas the value vectors absorb most of the representational load required for compression. Based on this observation, we ran a set of key vector ablation experiments that confirmed that CARTRIDGE key vectors are shareable across tasks. Building on our mechanistic study of key vector structure, we introduce a simple sampled-chunk initialization scheme (SCI) and showed that it accelerates convergence.

Collectively, these findings provide experimental validation of Petrov *et al.*'s theoretical constraints on prefix-tuning, suggesting that there is a fundamental division of labor between keys and value vectors during prefix-tuning. Rather than being limitations, we suggest that these constraints could inform further training- and inference-time optimizations for CARTRIDGES and other prefix-tuning methods.

By presenting an empirical study of CARTRIDGES' learned structure, we hope to invite further mechanistic research and discussion for understanding PeFT methods more broadly.

## Acknowledgments and Disclosure of Funding

We would like to thank Prime Intellect for providing the compute for this project, Ileana Rugina for feedback, and Sabri Eyuboglu for open-sourcing the code that made this project possible.

## References

[1] Simran Arora and Christopher Ré. Can foundation models help us achieve perfect secrecy? *arXiv preprint arXiv:2205.13722*, 2022.

[2] Ben Athiwaratkun, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Haifeng Qian, Hantian Ding, Qing Sun, Jun Wang, Jiacheng Guo, Liangfu Chen, Parminder Bhatia, Ramesh Nallapati, Sudipta Sengupta, and Bing Xiang. Bifurcated attention: Accelerating massively parallel decoding with shared prefixes in llms, 2024.

[3] Yu-Neng Chuang, Tianwei Xing, Chia-Yuan Chang, Zirui Liu, Xun Chen, and Xia Hu. Learning to compress prompt in natural language formats. *arXiv preprint arXiv:2402.18700*, 2024.

[4] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[5] Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025.

[6] Yao Fu. Challenges in deploying long-context transformers: A theoretical peak performance analysis, 2024.

[7] Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023.

[8] Neel Guha, Julian Nyarko, Daniel Ho, Christopher Ré, Adam Chilton, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel Rockmore, Diego Zambrano, 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.

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

[10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[11] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

[12] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

[13] Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

[14] Jordan Juravsky, Ayush Chakravarthy, Ryan Ehrlich, Sabri Eyuboglu, Bradley Brown, Joseph Shetaye, Christopher Ré, and Azalia Mirhoseini. Tokasaurus: An llm inference engine for high-throughput workloads, June 2025.

[15] Nimit Kalra and Leonard Tang. Verdict: A library for scaling judge-time compute. *arXiv preprint arXiv:2502.18018*, 2025.

[16] Yoon Pyo Lee. Mechanistic interpretability of lora-adapted language models for nuclear reactor safety applications, 2025.

[17] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *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)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics.

[18] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023.

[19] Meta. The llama 3 herd of models, 2024.

[20] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[21] Atharva Nijasure, Tanya Chowdhury, and James Allan. How relevance emerges: Interpreting lora fine-tuning in reranking llms, 2025.

[22] Aleksandar Petrov, Philip Torr, and Adel Bibi. When do prompting and prefix-tuning work? a theory of capabilities and limitations. In *The Twelfth International Conference on Learning Representations*, 2024.

[23] Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

[24] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[25] Terence Tao. *Topics in Random Matrix Theory*, volume 132 of *Graduate Studies in Mathematics*. American Mathematical Society, 2012.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[27] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[28] Zihao Ye, Ruihang Lai, Bo-Ru Lu, Chien-Yu Lin, Size Zheng, Lequn Chen, Tianqi Chen, and Luis Ceze. Cascade inference: Memory bandwidth efficient shared prefix batch decoding, February 2024.

# A N-Gram Diversity of Initialization Methods

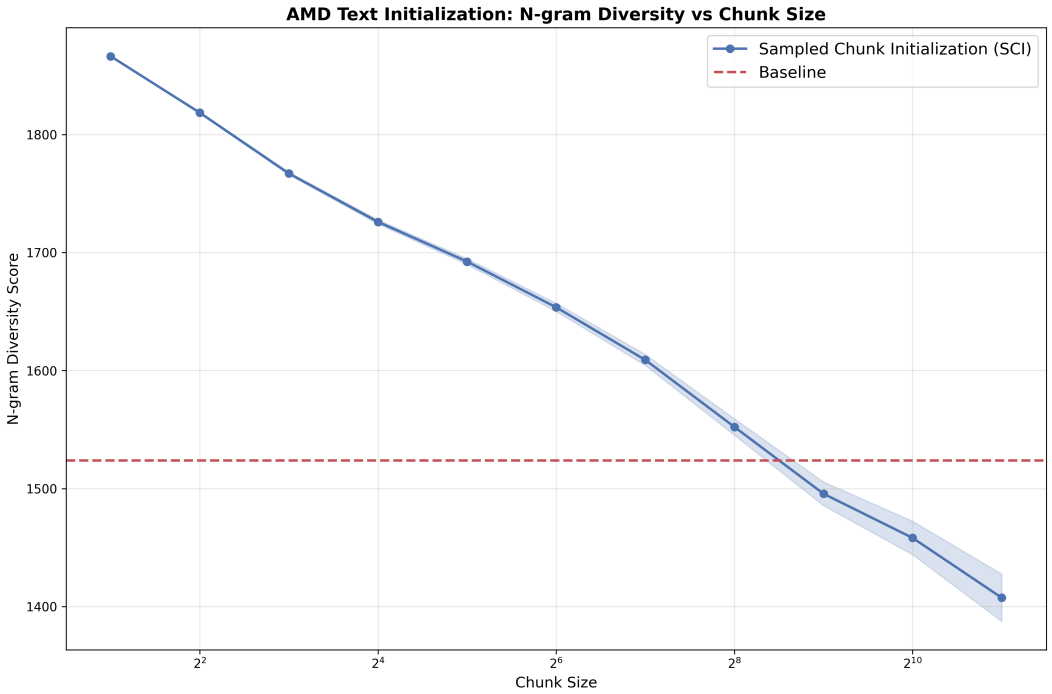


Figure 5: Here we plot the N-gram diversity of Sampled Chunk Initialization (SCI) vs. the **First $k$ Token Initialization** baseline. We note that $2^6$ is approximately the midway point when trading diversity for chunk length, so we chose 64 as the chunksize when running our experiments in Figure 4.

# B Additional Singular Value Analyses

Here we present additional supporting singular value analysis prompts to support our observations across scale, model families, and training regimes.

## B.1 LLAMA Across Scales

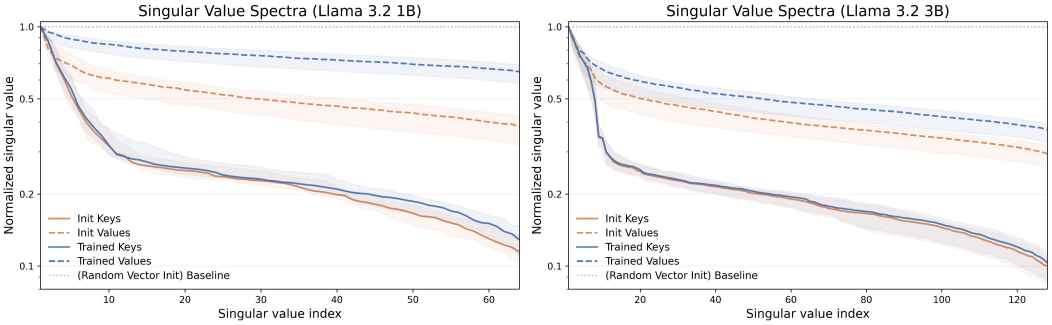


Figure 6: Here we have the singular value spectra for both LLAMA 1B and LLAMA 3B. We observe the same trends from the main paper: keys remain stable and values generally shift up in singular value. Both these models were trained on LONGHEALTH with the paper's **First $k$ Token Initialization** scheme. However, they are relatively undertrained at only 512 optimizer updates with a batch size of 64 and packed sequence length of 1024. In Appendix B.3, we note that training further reduces the IQR band variance for LLAMA 3.2 3B.

### B.2 QWEN3 Across Scales

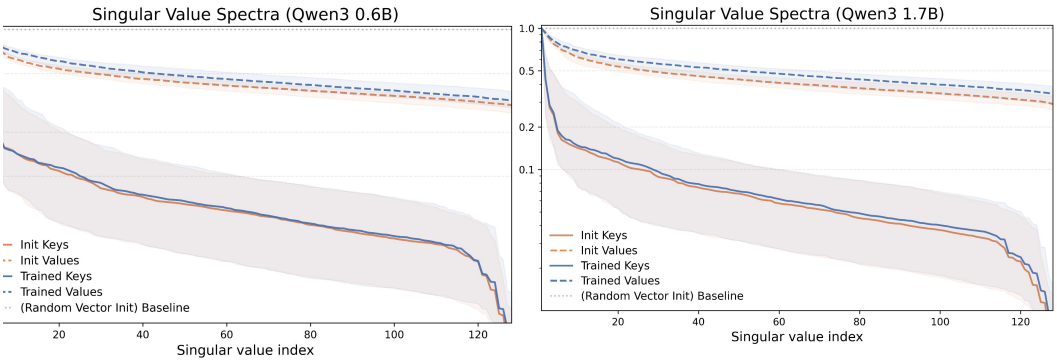


Figure 7: We present the singular value spectra for QWEN3 0.6B and QWEN3 1.7B on LONGHEALTH. Notably, we see the higher IQR badnd variance from Figure 3 repeated for the other QWEN3 models. Considering the results from LLAMA, it's possible that further training could reduce this variance given that these models are under-trained at only around 320 optimizer steps. Despite this undertraining, we still see a general upward singular value trend for the CARTRIDGE value vectors; however, we cannot claim statistical significance for the smaller QWEN3 models without more data.

### B.3 LLAMA Strict Replication

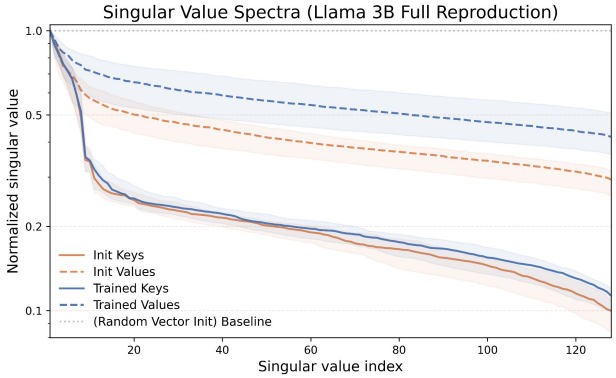


Figure 8: This is the singular value spectra of a LLAMA 3B model trained for 3072 steps on LONGHEALTH. We see the trend holds at scale; notably, given more optimizer steps the value vectors rose in singular value more than the under-trained LLAMA 3B from Appendix B.1.

### B.4 Across Different Initializations

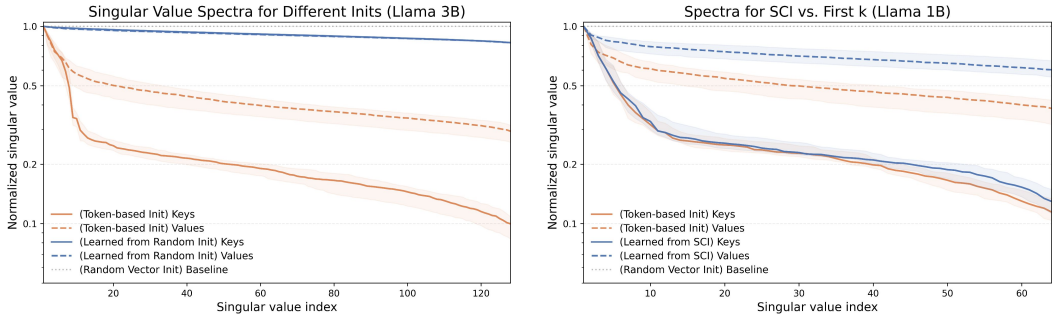


Figure 9: We compare two initialization methods against **First $k$ Tokens Initialization (Left)** Here the singular value spectra of random vector initialization (RVI) is remarkably different from our token-based approaches. Notably, both RVI keys and vectors stay close to the random orthogonal vector baseline. This suggests that keys fail to learn an effective routing structure for downstream tasks. **(Right)** Here is the singular value spectra for SCI. It looks more like what we expect to see from a successfully trained CARTRIDGE: stable keys and shifted values due to compression.

# C Prompts

## C.1 SELF-STUDY Prompts

We use the same SELF-STUDY prompts from the Cartridges GitHub.

```
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.'"
        ),
        (
            "Can you structure the information in {{subsection}} of {{document}} "
            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}'"
    )


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


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


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


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)


def generic_seed_prompt(**kwargs):
    return (
        f"Please generate a single chat message to begin a conversation about the information in the corpus. Ask a question about the corpus or make a request."
    )
```

### C.2 GENCONVO Prompts

We use modifications of the original GENCONVO prompts from the CARTRIDGES paper. These prompts yield similar Q&A pairs but encourage better instruction following from the generating model.

**Factual Prompt Template**

Generate a factual recall question about a specific entity, date, or name from the document.

**Format:** "Who/What/When [specific question]?"

**Answer:** Must be an exact entity name, date, or proper noun from the document (2–4 words max).

The answer should be unambiguous and directly stated in the document.

**Reasoning Prompt Template**

Generate a mathematical reasoning question requiring calculation over document values.

**Format:** "What is the [percentage/ratio/difference] of [specific calculation]?"

**Answer:** Must be a precise number with units (e.g., "12.5%", "$2.3M", "1.8x").

Question should require combining 2+ values from different parts of the document.

**Counting Prompt Template**

Generate a counting question about document structure or content frequency.

**Format:** "How many [items] are [condition]?"

**Answer:** Must be a single integer (e.g., "7", "23").

Focus on countable elements like sections, tables, mentions of specific terms, or occurrences.

**Synthesis Prompt Template**

Generate a multiple choice question testing document comprehension.

**Format:** Question with 5 options (A/B/C/D/E).

**Answer:** Single letter (A, B, C, D, or E). E always means "There is not enough information to answer the question".

Question should require understanding main themes, risks, or business model elements.