Do Large Language Models Need a Content Delivery Network?

Yihua Cheng, Kuntai Du, Jiayi Yao, Junchen Jiang
The University of Chicago

Abstract

As the use of large language models (LLMs) expands rapidly, so does the range of knowledge needed to supplement various LLM queries. Thus, enabling modular and efficient injection of new knowledge in LLM inference is critical. We argue that compared to the more popular fine-tuning and in-context learning, using KV caches as the medium of knowledge could simultaneously improve the modularity of knowledge injection and the efficiency of LLM serving with low cost and fast response. To make it practical, we envision a *Knowledge Delivery Network* (KDN), a new component in LLM services that dynamically optimizes the storage, transfer, and composition of KV cache across LLM engines and other compute and storage resources. Just like content delivery networks (CDNs), such as Akamai, enabled the success of the Internet ecosystem through their efficient data delivery, KDNs will be critical to the success of LLM applications through their efficient knowledge delivery. An open-sourced KDN prototype: **https://github.com/LMCache/LMCache**.

1 Background and Motivation

Traditionally, machine learning models, such as computer vision [22, 30, 31, 38] and image generation [20, 32, 37], learn all the knowledge from the training data. However, with the rapid usage growth of large language models (LLMs), applications require the ability to inject external knowledge, unseen in training, into LLM inference. For instance, chatbots [1, 5, 16] and agent systems use the chatting histories as supplementary knowledge to generate personalized responses; enterprises use LLM agents to answer queries based on their internal databases, as new knowledge, using retrieval-augmented generation (RAG) [14, 25, 27, 28, 39]; and searching engines use LLM to read fresh and relevant web data from the internet for each user query [2, 6, 10].

Not only is providing more external knowledge among the most efficient ways to boost LLM quality [17, 28, 39, 41], but the requirements of knowledge injection also become more complex and diverse. For instance, in enterprise settings, data scientists and application operators often demand the flexibility that allows them to directly specify which documents (or which parts of a document) should or should *not* be used as the context to answer a given query [4, 9]. Moreover, as new data is being constantly collected [3, 8], the knowledge-injection method must be efficient enough to keep up with the rapidly evolving pool of knowledge.

The 2nd Workshop on Hot Topics in System Infrastructure (HotInfra'24), co-located with SOSP'24, November 3, 2024, Austin, TX, USA

Thus, the key question is *"how to design and implement a system to inject knowledge to LLMs?"*

Three knowledge-injection methods exist. *In-context* learning and *fine-tuning* are the two popular options employed by knowledge-augmented generation. Fine-tuning retrains an LLM on a new dataset using a set of labeled knowledge to update all or parts of the model's parameters. Unlike fine-tuning, in-context learning leverages the existing capabilities of the model to interpret and respond to new information, *without* altering the model's parameters. Yet, compared to using a fine-tuning model, in-context learning has much higher run-time computation overhead as it has to process a much longer input (*i.e.,* the prefill step), before the model can generate the first token.

An alternative, which we refer to as *KV-cache learning*, is to let the LLM pre-compute the *KV cache*[1] of the new-knowledge text so that when the same knowledge is needed to supplement another LLM query, the KV cache can be directly used by LLM. This way, LLMs can directly generate the response as fast as the fine-tuned model, without the extra computation overhead to prefill the text of the knowledge as in-context learning. However, KV-cache learning, with a straightforward implementation, will suffer from the large size of the KV caches.

Most research in the machine-learning communities has primarily focused on the generation quality of different knowledge-injection methods, in F1 score and accuracy [15, 29, 35]. Studies have shown that both in-context learning and fine-tuning can achieve high text-generation quality if they are configured appropriately by *machine-learning* engineers [11, 21, 44]. However, less is known about the tradeoffs of these methods presented to *system engineers* who implement and maintain the LLM infrastructure for knowledge-augmented LLM applications.

This paper sheds light on these methods from a ***system architecture's***[2] perspective (Figure 1). Specifically, we make two key arguments:

- *Existing knowledge-injection methods—in-context learning, fine-tuning, and KV-cache learning—mainly differ in the tradeoffs between their* **modularity** *(ease of adding new knowledge and flexibility to specify injected knowledge) and* **efficiency** *(in per-query cost and response delay).* (§2)
- *Compared to in-context learning and fine-tuning, KV-cache learning* **could** *improve both modularity and efficiency,*

[1]KV cache is the intermediate state when LLM prefill on a text [33, 34, 40], which represents the LLM's understanding of the knowledge.

[2]Architecture refers to the interface between the modules, and we leave the refinement of implementation to future work.

大型语言模型是否需要内容分发网络?

程怡华，杜坤泰，姚佳怡，姜俊辰 芝加哥大学

摘要

随着大型语言模型（LLMs）使用的快速扩展，补充各种LLM查询所需的知识范围也在扩大。因此，在LLM推理中实现模块化和高效的新知识注入至关重要。我们认为，与更流行的微调和上下文学习相比，使用KV缓存作为知识的媒介可以同时提高知识注入的模块化和LLM服务的效率，且成本低、响应快。为了使其具有实用性，我们设想了一个知识交付网络（KDN），这是LLM服务中的一个新组件，能够动态优化KV缓存的存储、传输和组合，跨越LLM引擎和其他计算及存储资源。就像内容交付网络（CDNs），例如Akamai，通过高效的数据交付促进了互联网生态系统的成功，KDN将在LLM应用的成功中发挥关键作用，通过其高效的知识交付。一个开源的KDN原型：https://github.com/LMCache/LMCache。

1 背景与动机

传统上，机器学习模型，如计算机视觉 [22, 30, 31, 38] 和图像生成 [20, 32, 37]，从训练数据中学习所有知识。然而，随着大型语言模型（LLMs）使用的快速增长，应用程序需要能够将训练中未见的外部知识注入到LLM推理中。例如，聊天机器人 [1, 5, 16] 和代理系统使用聊天历史作为补充知识来生成个性化响应；企业使用LLM代理根据其内部数据库回答查询，作为新知识，使用检索增强生成（RAG） [14, 25, 27, 28, 39]；搜索引擎使用LLM从互联网读取每个用户查询的新鲜和相关的网页数据 [2, 6, 10]。

不仅提供更多外部知识是提升LLM质量最有效的方法之一[17, 28, 39, 41]，而且知识注入的要求也变得更加复杂和多样化。例如，在企业环境中，数据科学家和应用操作员通常要求灵活性，以便他们可以直接指定哪些文档（或文档的哪些部分）应该或不应该用作回答特定查询的上下文[4, 9]。此外，随着新数据的不断收集[3, 8]，知识注入方法必须足够高效，以跟上快速发展的知识库。

因此，关键问题是“如何设计和实施一个系统以将知识注入大型语言模型（LLMs）？”

存在三种知识注入方法。上下文学习和微调是知识增强生成中使用的两种流行选项。微调使用一组标记知识在新数据集上重新训练LLM，以更新模型参数的全部或部分。与微调不同，上下文学习利用模型现有的能力来解释和响应新信息，而不改变模型的参数。然而，与使用微调模型相比，上下文学习的运行时计算开销要高得多，因为它必须处理更长的输入（即预填充步骤），才能生成第一个标记。

一种替代方案，我们称之为KV-cache学习，是让LLM预先计算新知识文本的KV缓存[1]，这样当需要相同知识来补充另一个LLM查询时，KV缓存可以被LLM直接使用。通过这种方式，LLM可以像微调模型一样快速生成响应，而无需像上下文学习那样额外计算填充知识文本的开销。然而，KV-cache学习由于其简单的实现，将面临KV缓存的巨大规模问题。

在机器学习社区，大多数研究主要集中在不同知识注入方法的生成质量上，关注F1分数和准确性 [15, 29, 35]。研究表明，如果机器学习工程师适当地配置，背景学习和微调都可以实现高质量的文本生成 [11, 21, 44]。然而，对于实施和维护知识增强型LLM应用程序的系统工程师来说，这些方法的权衡知之甚少。

本文从系统架构[2]的角度阐明了这些方法（图1）。具体而言，我们提出两个关键论点：

- 现有的知识注入方法——上下文学习、微调和KV缓存学习——主要在模块化（添加新知识的便利性和指定注入知识的灵活性）与效率（每个查询的成本和响应延迟）之间的权衡上有所不同。（§2）
- 与上下文学习和微调相比，KV-cache学习可以提高模块化和效率，

The 2nd Workshop on Hot Topics in System Infrastructure (HotInfra'24), co-located with SOSP'24, November 3, 2024, Austin, TX, USA

[1]KV cache is the intermediate state when LLM prefill on a text [33, 34, 40], which represents the LLM's understanding of the knowledge.

[2]Architecture refers to the interface between the modules, and we leave the refinement of implementation to future work.

	Modularity	Efficiency
In-context learning	✓	✗
Fine-tuning	✗	✓
KV-cache learning (*with Knowledge Delivery Networks*)	✓	✓

Figure 1. *Knowledge-injection methods make trade-offs between* **modularity** *and* **efficiency**. *With Knowledge Delivery Networks (KDNs), KV-cache learning can improve on both dimensions compared to in-context learning and fine-tuning.*

if a new system component, called **knowledge-delivery network (KDN)**, *optimizes the management of KV caches by harnessing several emerging research efforts.* (§3.2)

At a high level, a KDN serves as a backend of LLM-processed knowledge (*i.e.,* KV caches), which can be a part of an LLM-serving system or shared across multiple LLM-serving systems. Unlike the existing LLM-serving systems [23, 26, 46], which deeply couple KV caches with LLM engines (*i.e.,* binding KV caches with GPUs and managing KV caches inside inferencing), KDNs call for a clean ***separation*** between KV-cache management and LLM serving engines, for better modularity and efficiency.

We will outline the key components of a KDN, including a storage pool of KV caches that leverages KV-cache compression, a fast KV-cache streaming system to transfer KV caches between LLM serving engines, and a KV-cache blender module that dynamically puts together multiple pieces of knowledge stored in modularized KV caches. Using a prototype of KDN, we will show that some emerging research efforts have already provided preliminary techniques, which together can make highly efficient KDNs a reality.

2 LLM Knowledge-Injection From A System Perspective

Knowledge-augmented generation, particularly, fine-tuning and in-context learning, is well-studied in the AI literature.

Fine-tuning embeds a corpus of texts in the LLM's weights, so the fine-tuned model can directly respond to a user query with a low response delay. However, as the entire corpus of texts must be embedded in the model together, fine-tuning lacks the flexibility to add new knowledge and specify what knowledge should or should not be used.

In-context learning is the opposite of fine-tuning, as it allows the operators to specify which external knowledge should be used easily by putting the texts to the LLM's input. However, the compute overhead of prefilling will grow superlinearly with the input length, causing a long response delay when more external data is added to the input.

2.1 The Efficiency-Modularity Tradeoff

Both methods can achieve similar text-generation quality if used in the right way [11, 21, 44]. Instead of viewing these options only in terms of accuracy (*i.e.,* through the ML perspective), we compare the knowledge-injection methods along the following two *system-centric* metrics: **modularity** and **efficiency**.

Modularity: In the context of knowledge-augmented LLM, the modularity of a method includes two aspects. First, a modular approach should allow service providers to specify which knowledge to use and compose them easily. Second, the overhead (*e.g.,* time, cost) of injecting new knowledge into the LLM should be minimized.

In-context learning puts the new knowledge in the model's input, rather than the model itself. The separation of knowledge and model serves as the key to modularity—LLM service providers can specify which knowledge to use and easily compose different pieces of knowledge, which helps the LLM to avoid conflicting knowledge and improve the generation quality. In contrast, fine-tuning has poor modularity. Users cannot specify which knowledge in the fine-tuned model would be used to generate the answer. Moreover, fine-tuning needs to happen for every new knowledge and model, which may take hours to days to complete.

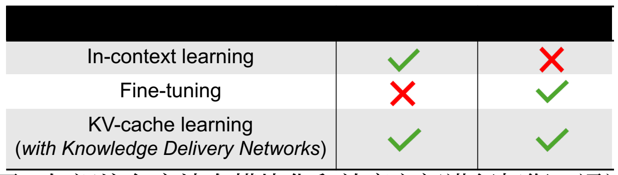
Efficiency: On the other hand, the efficiency of a knowledge-augmented LLM system is measured by per-query cost and response delay during LLM inference. Cost is the computation used for the LLM to handle a request, and response delay is defined as the time between the LLM receiving the request and the generation of the first token. Viewed through this lens, in-context learning is not ideal, because when using in-context learning, LLMs must spend a long time *prefilling* input text with the knowledge before generating the first token. In contrast, fine-tuning is better in terms of efficiency. Because the knowledge is embedded in the model's weights, the fine-tuned model can skip the long prefill.

In short, in-context learning is more modular but sacrifices efficiency, while fine-tuning, though achieving better efficiency, suffers from the overhead of incorporating and controlling the knowledge due to poor modularity.

2.2 KV-Cache Learning

Alternatively, *KV cache learning* stores the knowledge in LLM-generated KV cache, and injects knowledge by feeding the KV cache to the LLM, without modifying the model. A KV cache stores the knowledge in the form of the attention state generated by the model after it processes the text, so the KV cache, once generated, can be reused by the LLM to skip prefilling if the subsequent requests use the same context. When many queries reuse the same context, reusing its KV cache *could* reduce the per-query delay and compute usage, while still preserving the modularity as in-context learning. The idea of *KV-cache learning* has gained increasing attention in LLM services (*e.g.,* [7, 36, 46]).

Why storing knowledge in KV caches pays off? On the surface, the use of KV caches may seem merely a space-time tradeoff (trading KV cache storage space for shorter prefill), but the tradeoff is favorable for two reasons:



In-context learning	✓	✗
Fine-tuning	✗	✓
KV-cache learning *(with Knowledge Delivery Networks)*	✓	✓

图1. 知识注入方法在模块化和效率之间进行权衡。通过知识交付网络（KDN），KV-cache学习在这两个维度上相比于上下文学习和微调都有所改善。

如果一个新的系统组件，称为知识交付网络（KDN），优化了KV缓存的管理
通过利用几项新兴的研究工作。(§3.2)
在高层次上，KDN 作为 LLM 处理知识的后端（即 KV 缓存），可以是 LLM 服务系统的一部分或在多个 LLM 服务系统之间共享。与现有的 LLM 服务系统 [23, 26, 46] 不同，这些系统将 KV 缓存与 LLM 引擎深度耦合（即将 KV 缓存绑定到 GPU 并在推理过程中管理 KV 缓存），KDN 呼吁在 KV 缓存管理和 LLM 服务引擎之间进行清晰的分离，以实现更好的模块化和效率。

我们将概述 KDN 的关键组成部分，包括利用 KV 缓存压缩的 KV 缓存存储池、一个快速的 KV 缓存流系统，用于在 LLM 服务引擎之间传输 KV 缓存，以及一个 KV 缓存混合模块，动态地将存储在模块化 KV 缓存中的多个知识片段组合在一起。通过 KDN 的原型，我们将展示一些新兴的研究工作已经提供了初步的技术，这些技术可以共同使高效的 KDN 成为现实。

从系统的角度看 2 LLM 知识注入

知识增强生成，特别是微调和上下文学习，在人工智能文献中得到了充分研究。
微调将一组文本嵌入到LLM的权重中，因此微调后的模型可以直接以较低的响应延迟对用户查询作出回应。然而，由于整个文本语料库必须一起嵌入到模型中，微调缺乏添加新知识的灵活性，并且无法指定哪些知识应该或不应该被使用。
上下文学习与微调相反，因为它允许操作员通过将文本放入LLM的输入中轻松指定应使用哪些外部知识。然而，预填充的计算开销将随着输入长度的增加而超线性增长，当更多外部数据添加到输入中时，会导致响应延迟变长。

2.1 效率-模块化权衡

这两种方法如果以正确的方式使用，可以实现类似的文本生成质量 [11, 21, 44]。我们不仅仅从准确性（即通过机器学习的视角）来考虑这些选项，而是沿着知识注入方法进行比较。以下两个以系统为中心的指标：模块化和效率。

模块化：在知识增强的LLM背景下，方法的模块化包括两个方面。首先，模块化的方法应该允许服务提供商指定使用哪些知识并轻松组合它们。其次，将新知识注入LLM的开销（例如，时间、成本）应该最小化。

上下文学习将新知识放在模型的输入中，而不是模型本身。知识与模型的分离是模块化的关键——大型语言模型（LLM）服务提供商可以指定使用哪些知识，并轻松组合不同的知识片段，这有助于LLM避免冲突知识并提高生成质量。相比之下，微调的模块化较差。用户无法指定在微调模型中将使用哪些知识来生成答案。此外，微调需要针对每个新知识和模型进行，这可能需要数小时到数天才能完成。

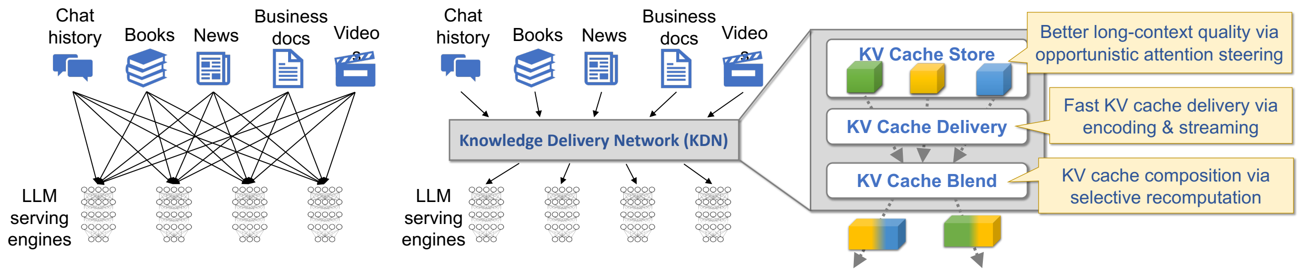
效率：另一方面，知识增强的 LLM 系统的效率通过每个查询的成本和 LLM 推理过程中的响应延迟来衡量。成本是 LLM 处理请求所需的计算量，而响应延迟被定义为 LLM 接收请求与生成第一个标记之间的时间。从这个角度来看，上下文学习并不理想，因为在使用上下文学习时，LLM 必须花费很长时间用知识预填输入文本，然后才能生成第一个标记。相比之下，微调在效率方面更好。因为知识嵌入在模型的权重中，微调后的模型可以跳过长时间的预填充。

简而言之，上下文学习更具模块化，但牺牲了效率，而微调虽然实现了更好的效率，却因模块性差而面临整合和控制知识的开销。

2.2 KV-缓存学习

或者，KV缓存学习将知识存储在LLM生成的KV缓存中，并通过将KV缓存输入到LLM中来注入知识，而不修改模型。KV缓存以模型在处理文本后生成的注意力状态的形式存储知识，因此一旦生成，KV缓存可以被LLM重用，以跳过预填充，如果后续请求使用相同的上下文。当许多查询重用相同的上下文时，重用其KV缓存可以减少每个查询的延迟和计算使用，同时仍然保持与上下文学习的模块化。KV缓存学习的理念在LLM服务中越来越受到关注（例如，[7, 36, 46]）。

为什么将知识存储在KV缓存中是值得的？表面上，使用KV缓存似乎仅仅是一个空间-时间权衡（用KV缓存存储空间换取更短的预填充时间），但这种权衡有两个原因是有利的：


Chat history
Books
News
Business docs
Video
LLM serving engines
Chat history
Books
News
Business docs
Video
Knowledge Delivery Network (KDN)
LLM serving engines
KV Cache Store
KV Cache Delivery
KV Cache Blend
Better long-context quality via opportunistic attention steering
Fast KV cache delivery via encoding & streaming
KV cache composition via selective recomputation

(a) Injecting knowledge via in-context learning (feeding text or raw data to LLMs)

(b) Injecting knowledge via Knowledge-Delivery Networks (feeding KV caches to LLMs)

(c) High-level structure of the proposed Knowledge Delivery Network (KDN)

Figure 2. *Architecture of a Knowledge Delivery Network (KDN).*

- *KV caches are reused a lot.* Many contexts, especially long contexts, are frequently reused by different queries and different users. This can be viewed as the LLM's version of the Pareto's rule: an LLM, like human, uses 20% of the knowledge for 80% of the time, which means knowledge is frequently reused. Just consider that if a user asks the LLM to read a book, it is unlikely that they will only ask one book-related question.
- *The size of KV caches increases slower than prefill delay.* As the context increases, the KV cache size grows *linearly* whereas the prefill delay grows *superlinearly*. And as the LLM gets bigger, more compute will happen at the feedforward layers which do not affect the KV cache size.

3 Knowledge Delivery Networks for Efficient Knowledge Injection

Limitations of Existing KV-Cache Systems: Despite the promise, existing KV-caching learning systems still have some technical limitations.

- *Limited storage for KV caches:* Currently, many serving engines only use the GPU/CPU memory locally accessible by an individual LLM instance to store KV caches. Such local-memory-only storage greatly *limits* the amount of KV caches that are stored and reused. For instance, a CPU memory of 64 GB can only store the KV cache of 160K tokens (two pdf reports) for a small-size LLM (Llama-34B). However, expanding the storage of KV caches to disk or remote servers would significantly constrain the bandwidth for loading the KV caches into GPU memory.
- *Prefix-only reusing:* To reuse the KV cache, most systems require that the text of the KV cache must be the *prefix* of the LLM input. Even though reusing the KV cache of the prefix is naturally supported by LLMs, this assumption of "sharing prefix only" severely limits its use cases. For instance, retrieval-augmented generation (RAG) concatenates *multiple* retrieved text chunks in the LLM input, so most reused texts will not be the prefix.
- *Degraded quality with long contexts:* Finally, as more texts are added to the input as LLM's context (*i.e.,* long context), the LLM's capability to capture important information might degrade, lowering the inference quality. Thus, when the KV caches are reused by more queries repeatedly, the degraded quality will also affect more queries.

3.1 Knowledge Delivery Architecture

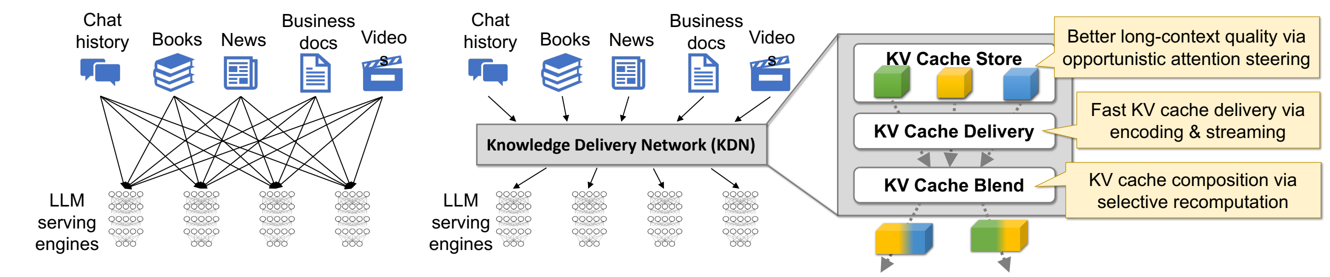
At first glance, these challenges facing prior KV-cache-based systems may look disparate. Yet, our insight is that they share a common need—***a separate KV-cache management system, which dynamically compresses, composes, and modifies KV caches to optimize the storage and delivery of KV caches and the LLM inference based on KV caches***. We refer to such a system as a ***Knowledge Delivery Network*** (**KDN**). As depicted in Figure 2, the envisioned architecture of a KDN consists of three main modules:

- The ***storage*** module stores the KV caches keyed by various texts and offline *modifies* the KV cache content such that the inference quality will be improved when the KV caches are fed to the LLM.
- The ***delivery*** module transmits the *compressed* KV caches from the storage device to the server running the LLM and decompresses them in GPU to be used by the LLM serving engine.
- The ***blending*** module dynamically *composes* multiple KV caches corresponding to different texts when these texts are put together in the context of an LLM query.

The existing LLM-serving systems [23, 26, 46] deeply couple KV caches with LLM engines. In contrast, the concept of KDN is to ***separate*** the management of KV caches and the LLM serving engines. Such separation will enable innovations on the storage, sharing, and use of KV caches, *without* needing to be deeply coupled with the fast-evolving LLM serving engines. By implementing these optimizations separately, a KDN serves as a critical new system module for LLM-serving ecosystems. It enables the decoupling of KV-cache management from LLM serving engines, which allows LLM serving engines to focus on fast query-driven inference, while the KDN can focus on the KV-cache-related optimizations independently.

3.2 Technical Roadmap

The architecture of KDN itself does not directly address the challenges associated with KV caches; it merely breaks a


Chat history
Books
News
Business docs
Videos
LLM serving engines
Chat history
Books
News
Business docs
Videos
Knowledge Delivery Network (KDN)
LLM serving engines
KV Cache Store
Better long-context quality via opportunistic attention steering
KV Cache Delivery
Fast KV cache delivery via encoding & streaming
KV Cache Blend
KV cache composition via selective recomputation

(a) Injecting knowledge via in-context learning (feeding text or raw data to LLMs)

(b) Injecting knowledge via Knowledge-Delivery Networks (feeding KV caches to LLMs)

(c) High-level structure of the proposed Knowledge Delivery Network (KDN)

图2. 知识交付网络（KDN）的架构。

- KV缓存被大量重用。许多上下文，尤其是长上下文，常常被不同的查询和不同的用户频繁重用。这可以看作是LLM版本的帕累托法则：LLM像人类一样，使用20%的知识来应对80%的时间，这意味着知识被频繁重用。想想看，如果一个用户让LLM阅读一本书，他们不太可能只问一个与书籍相关的问题。

- KV缓存的大小增长速度慢于预填充延迟。随着上下文的增加，KV缓存的大小线性增长，而预填充延迟则超线性增长。随着LLM的增大，前馈层的计算量会增加，但这不会影响KV缓存的大小。

3个高效知识注入的知识交付网络

现有KV缓存系统的局限性：尽管前景广阔，现有的KV缓存学习系统仍然存在一些技术限制。

- KV缓存的存储有限：目前，许多服务引擎仅使用单个LLM实例本地可访问的GPU/CPU内存来存储KV缓存。这种仅限本地内存的存储极大限制了存储和重用的KV缓存量。例如，64 GB的CPU内存只能存储160K标记（两个pdf报告）的小型LLM（Llama-34B）的KV缓存。然而，将KV缓存的存储扩展到磁盘或远程服务器将显著限制将KV缓存加载到GPU内存的带宽。

- 仅前缀重用：为了重用KV缓存，大多数系统要求KV缓存的文本必须是LLM输入的前缀。尽管LLM自然支持重用前缀的KV缓存，但这种“仅共享前缀”的假设严重限制了其使用案例。例如，检索增强生成（RAG）在LLM输入中连接多个检索到的文本块，因此大多数重用的文本将不是前缀。

- 降级质量与长上下文：最后，随着更多文本被添加到输入作为LLM的上下文（即长上下文），LLM捕捉重要信息的能力可能会降低推理质量。因此，当KV缓存被更多查询重复使用时，降低的质量也会影响更多查询。

3.1 知识交付架构

乍一看，这些面临先前基于KV缓存系统的挑战似乎各不相同。然而，我们的见解是，它们有一个共同的需求——一个独立的KV缓存管理系统，该系统动态地压缩、组合和修改KV缓存，以优化KV缓存的存储和交付以及基于KV缓存的LLM推理。我们将这样的系统称为知识交付网络（KDN）。如图2所示，设想中的KDN架构由三个主要模块组成：

- 存储模块存储由各种文本键控的KV缓存，并离线修改KV缓存内容，以便在将KV缓存输入到LLM时提高推理质量。

- 交付模块将压缩的 KV 缓存从存储设备传输到运行 LLM 的服务器，并在 GPU 中解压缩，以供 LLM 服务引擎使用。

- 混合模块在LLM查询的上下文中将这些文本放在一起时，动态组合多个对应于不同文本的KV缓存。

现有的LLM服务系统[23, 26, 46]将KV缓存与LLM引擎深度耦合。相比之下，KDN的概念是将KV缓存的管理与LLM服务引擎分开。这种分离将促进KV缓存的存储、共享和使用方面的创新，而无需与快速发展的LLM服务引擎深度耦合。通过单独实施这些优化，KDN作为LLM服务生态系统中的一个关键新系统模块。它使KV缓存管理与LLM服务引擎解耦，从而允许LLM服务引擎专注于快速查询驱动的推理，而KDN可以独立专注于与KV缓存相关的优化。

3.2 技术路线图

KDN 的架构本身并没有直接解决与 KV 缓存相关的挑战；它只是打破了一个

	Modularity	Efficiency	
	Time to inject new knowledge	Inference cost ($) per request	Response delay (s) per request
Fine-tuning	10 hours	0.0052	2.63
In-context learning	0	0.0149	10.91
KV-cache learning w/ KDN	0.25 hours	0.0059	2.97

Table 1. *Comparison between different knowledge-injection methods under a RAG setup. With KDN, KV-cache learning is* 40× *faster when incorporating new knowledge than fine-tuning, and achieves* 3.7× *faster and* 2.5× *cheaper during inference compared to in-context learning.*

potential solution into three modules. Fortunately, we observe that emerging research efforts could lead to a sufficient design for each module, thus making KDN practical.

KV-cache delivery: The recent KV cache compression techniques make it possible to cheaply store and quickly load KV caches outside GPU and CPU memory. For instance, CacheGen [33] compresses the KV cache by quantizing and then encoding it into binary strings. LLMLingua [24] introduces a smaller language model to identify and remove non-essential tokens in the knowledge's text, thus reducing the size of the corresponding KV cache. H2O [45] directly removes elements in the KV cache based on their importance calculated during the inference. By combining the above techniques, the memory footprint of the KV cache can be reduced by over 10×, drastically improving the loading speed and the storage cost of the KV cache.

KV-cache blending: Some recent works also improve the composability of the KV caches. CacheBlend [40], for instance, enables arbitrarily composing different KV caches by recomputing the cross-attention between KV caches, where the recomputation only needs 10% computation of prefilling the full text. PromptCache [19] lets users define a prompt template with different segments, which allows each segment's KV cache to be reused at different positions rather than prefix-only.

Offline KV-cache editing: By separating KV-cache management from the LLM-serving engines, KDNs open up new possibilities to improve inference quality. Recent works have shown that if the attention matrix of an LLM input is appropriately modified, the inference quality can significantly improve [13, 42]. In other words, when a KV cache is "deposited" to KDN, KDN not only stores it but also can actively influence the LLM inference quality by offline editing the KV cache and returning the edited KV cache when it is retrieved next time, all of which is done *without any change to the model itself or the input prompt.*

Interface with LLM serving engines: Currently, most LLM serving engines, such as vLLM [26], HuggingFace TGI [23], and SGLang [46], do not readily support the injection of externally provided KV caches as part of the LLM input. Instead, their internal implementation KV-cache management (*e.g.,* paged memory in vLLM) is deeply cobbled with the model inference implementation. One way to deploy KDN is to augment each LLM engine with a KDN as submodule of the LLM engine. However, as elaborated in §3.2, developing a performant KDN is a substantial undertaking, so it will cause much redundant engineering effort if each LLM engine maintains and evolves its KDNs. To avoid reinventing the wheel, the LLM serving engines could interface with a separate, shared KDN provider, via two APIs: *(i)* the LLM *stores* the KV cache with the associated text to KDN, and *(ii)* the LLM *retrieves* the KV cache from KDN using some text. Exposing these APIs is feasible, given most popular LLM engines are open-source. Still, several design questions remain. How to leverage heterogeneous links, such as NVLink, RDMA, or PCIe, to transfer KV caches? Can the APIs be shared with other LLM functions, like disaggregated prefilling?

Early promise: Based on these techniques, we implemented LMCache (https://github.com/LMCache/LMCache), a prototype of KDN. We compare the modularity and efficiency of KV-cache learning with KDN to fine-tuning and in-context learning under a RAG use case with the following setup:

- Total size of knowledge (in text): 2 million tokens.
- Each request: 8K tokens knowledge plus 2K tokens user chatting history[3].
- Language model: Llama 3.1 70B [18].
- Hardware: 2 Nvidia A40 GPUs.

Modularity is measured by the time[4] of injecting the knowledge base into LLM, and efficiency is measured by inference cost[5] and response delay per request. Table 1 shows that with the KDN, KV-cache learning can be 40× faster than fine-tuning when injecting new knowledge, and it also achieves 3.7× cheaper and 2.5× faster during inference time compared to in-context learning.

4 Conclusion

In short, this paper makes a case for (1) the separation between the management of knowledge, in the form of KV caches, and LLM serving engines, and (2) a Knowledge-Delivery Network (KDN) as a new LLM system component that harnesses recent research development to optimize the efficiency (in speed and cost) of KV-cache-based knowledge injection. We hope this paper can inspire more research to tackle the aforementioned problems.

[3]We assume the chatting history cannot be pre-learned by fine-tuning.
[4]The time for fine-tuning the model is estimated from Llama-Adapter [43].
[5]The inference cost is calculated based on the AWS cloud price [12].

	Modularity	Efficiency	
	Time to inject new knowledge	Inference cost ($) per request	Response delay (s) per request
Fine-tuning	10 hours	0.0052	2.63
In-context learning	0	0.0149	10.91
KV-cache learning w/ KDN	0.25 hours	0.0059	2.97

T表1. 在RAG设置下不同知识注入方法的比较。使用KDN时，KV-cache学习在融入新知识时比微调快40×，在推理过程中比上下文学习快3.7×且便宜2.5×。

潜在解决方案分为三个模块。幸运的是，我们观察到新兴的研究努力可能会为每个模块提供足够的设计，从而使KDN变得实用。

KV-cache 交付：最近的 KV 缓存压缩技术使得能够以低成本存储和快速加载 GPU 和 CPU 内存之外的 KV 缓存。例如，CacheGen [33] 通过量化然后将其编码为二进制字符串来压缩 KV 缓存。LLMLingua [24] 引入了一个更小的语言模型，以识别和删除知识文本中的非必要标记，从而减少相应 KV 缓存的大小。H2O [45] 根据推理过程中计算的重要性直接删除 KV 缓存中的元素。通过结合上述技术，KV 缓存的内存占用可以减少超过 10×，从而大幅提高 KV 缓存的加载速度和存储成本。

KV-cache 混合：一些最近的工作也改善了 KV 缓存的可组合性。例如，CacheBlend [40] 通过重新计算 KV 缓存之间的交叉注意力，使得可以任意组合不同的 KV 缓存，而重新计算只需要填充完整文本的 10% 计算量。PromptCache [19] 允许用户定义具有不同段落的提示模板，这使得每个段落的 KV 缓存可以在不同位置重用，而不仅仅是前缀。

离线KV缓存编辑：通过将KV缓存管理与LLM服务引擎分离，KDN开启了改善推理质量的新可能性。最近的研究表明，如果适当地修改LLM输入的注意力矩阵，推理质量可以显著提高[13, 42]。换句话说，当KV缓存被“存入”KDN时，KDN不仅存储它，还可以通过离线编辑KV缓存并在下次检索时返回编辑后的KV缓存，主动影响LLM的推理质量，所有这些都是在不改变模型本身或输入提示的情况下完成的。

与LLM服务引擎的接口：目前，大多数LLM服务引擎，如vLLM [26]、HuggingFace TGI [23]和SGLang [46]，并不支持将外部提供的KV缓存作为LLM输入的一部分。相反，它们的内部实现KV缓存管理（例如，vLLM中的分页内存）与模型推理实现紧密耦合。部署KDN的一种方法是将每个LLM引擎增强为KDN的子模块。LLM引擎。然而，正如§3.2中详细阐述的，开发一个高性能的KDN是一项巨大的工作，因此如果每个LLM引擎都维护和发展其KDN，将会造成大量冗余的工程工作。为了避免重复造轮子，LLM服务引擎可以通过两个API与一个单独的共享KDN提供者接口：（i）LLM将与KDN相关的文本存储在KV缓存中，以及（ii）LLM使用某些文本从KDN检索KV缓存。考虑到大多数流行的LLM引擎是开源的，暴露这些API是可行的。然而，仍然存在几个设计问题。如何利用异构链接，如NVLink、RDMA或PCIe，来传输KV缓存？这些API能否与其他LLM功能共享，例如分散的预填充？

早期承诺：基于这些技术，我们实现了LMCache (https://github.com/LMCache/LMCache)，这是KDN的一个原型。我们比较了KV-cache学习与KDN在RAG用例下的微调和上下文学习的模块性和效率，设置如下：

- 知识的总大小（以文本计）：200万标记。
- 每个请求：8K 令牌的知识加上 2K 令牌的用户聊天历史[3]。
- 语言模型：Llama 3.1 70B [18]。
- 硬件：2 个 Nvidia A40 GPU。

模块化通过将知识库注入LLM的时间[4]来衡量，效率则通过每个请求的推理成本[5]和响应延迟来衡量。表1显示，使用KDN时，KV-cache学习在注入新知识时可以比微调快40×，并且在推理时间上相比于上下文学习，它还实现了便宜3.7×和快2.5×。

4 结论

简而言之，本文主张 (1) 知识管理（以 KV 缓存的形式）与 LLM 服务引擎之间的分离，以及 (2) 知识交付网络（KDN）作为一个新的 LLM 系统组件，利用最近的研究进展来优化基于 KV 缓存的知识注入的效率（在速度和成本方面）。我们希望本文能够激发更多研究来解决上述问题。

[3]We assume the chatting history cannot be pre-learned by fine-tuning.
[4]The time for fine-tuning the model is estimated from Llama-Adapter [43].
[5]The inference cost is calculated based on the AWS cloud price [12].

References

[1] character.ai | personalized ai for every moment of your day. https://character.ai/. (Accessed on 09/07/2024).

[2] Enterprise search: an llm-enabled out-of-the-box search engine. https://io.google/2023/program/27cce05f-df4c-4ab2-9a59-5b466bdae0f9/. (Accessed on 09/07/2024).

[3] How many websites are there in the world? (2024) - siteefy. https://siteefy.com/how-many-websites-are-there/. (Accessed on 09/08/2024).

[4] Instruction to exclude certain information when generating answer - openai developer forum. https://community.openai.com/t/instruction-to-exclude-certain-information-when-generating-answer/470451. (Accessed on 09/08/2024).

[5] Introducing chatgpt | openai. https://openai.com/index/chatgpt/. (Accessed on 09/07/2024).

[6] Perplexity. https://www.perplexity.ai/. (Accessed on 09/07/2024).

[7] Perplexity. https://lmcache.github.io/2024-09-17-release/. (Accessed on 10/14/2024).

[8] Perplexity partners with elevenlabs to launch 'discover daily' podcast. https://www.perplexity.ai/hub/blog/perplexity-partners-with-elevenlabs-to-launch-discover-daily-podcast. (Accessed on 09/08/2024).

[9] Revolutionizing operational efficiency: Unifying analytics and observability for seamless decision-making - convivaimproving llm output by combining rag and fine-tuning. https://www.conviva.com/blog/revolutionizing-operational-efficiency-unifying-analytics-and-observability-for-seamless-decision-making/. (Accessed on 09/08/2024).

[10] Search - consensus: Ai search engine for research. https://consensus.app/search/. (Accessed on 09/07/2024).

[11] Simone Alghisi, Massimo Rizzoli, Gabriel Roccabruna, Seyed Mahed Mousavi, and Giuseppe Riccardi. Should We Fine-Tune or RAG? Evaluating Different Techniques to Adapt LLMs for Dialogue. *arXiv preprint arXiv:2406.06399*, 2024.

[12] Amazon Web Services. Ec2 on-demand instance pricing – amazon web services, 2024. Accessed on September 07, 2024.

[13] Anonymous. Model tells itself where to attend: Faithfulness meets automatic attention steering. In *Submitted to ACL Rolling Review - June 2024*, 2024. under review.

[14] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762, 2024.

[15] Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. Meta-learning via language model in-context tuning. *arXiv preprint arXiv:2110.07814*, 2021.

[16] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

[17] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[18] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[19] In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference, 2023.

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[21] Aman Gupta, Anup Shirgaonkar, Angels de Luis Balaguer, Bruno Silva, Daniel Holstein, Dawei Li, Jennifer Marsman, Leonardo O Nunes, Mahsa Rouzbahman, Morris Sharp, et al. RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture. *arXiv preprint arXiv:2401.08406*, 2024.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] HuggingFace. text-generation-inference, 2024.

[24] 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.

[25] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*, 2023.

[26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[28] Jiarui Li, Ye Yuan, and Zehua Zhang. Enhancing llm factual accuracy with rag to counter hallucinations: A case study on domain-specific queries in private knowledge-bases. *arXiv preprint arXiv:2403.10446*, 2024.

[29] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

[30] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[32] Yixin Liu, Kai Zhang, Yuan Li, Zhiling Yan, Chujie Gao, Ruoxi Chen, Zhengqing Yuan, Yue Huang, Hanchi Sun, Jianfeng Gao, et al. Sora: A review on background, technology, limitations, and opportunities of large vision models. *arXiv preprint arXiv:2402.17177*, 2024.

[33] Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, et al. Cachegen: Fast context loading for language model applications. *arXiv preprint arXiv:2310.07240*, 2023.

[34] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

[35] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*, 2023.

[36] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Kimi's kvcache-centric

参考文献

[1] character.ai | 为您一天中的每个时刻提供个性化的人工智能。https://character.ai/。（访问日期：2024年9月7日）。[2] 企业搜索：一个开箱即用的 LLM 驱动搜索引擎。https://io.google/2023/program/27cce05f-df4c-4ab2-9a59-5b466bdae0f9/。（访问日期：2024年9月7日）。[3] 世界上有多少个网站？（2024）- siteefy。https://siteefy.com/how-many-websites-are-there/。（访问日期：2024年9月8日）。[4] 生成答案时排除某些信息的指令 - openai 开发者论坛。https://community.openai.com/t/instruction-to-exclude-certain-information-when-generating-answer/470451。（访问日期：2024年9月8日）。[5] 介绍 chatgpt | openai。https://openai.com/index/chatgpt/。（访问日期：2024年9月7日）。[6] Perplexity。https://www.perplexity.ai/。（访问日期：2024年9月7日）。[7] Perplexity。https://lmcache.github.io/2024-09-17-release/。（访问日期：2024年10月14日）。[8] Perplexity 与 elevenlabs 合作推出“每日发现”播客。https://www.perplexity.ai/hub/blog/perplexity-partners-with-elevenlabs-to-launch-discover-daily-podcast。（访问日期：2024年9月8日）。[9] 革新运营效率：统一分析和可观察性以实现无缝决策 - conviva通过结合 RAG 和微调来提高 LLM 输出。https://www.conviva.com/blog/revolutionizing-operational-efficiency-unifying-analytics-and-observability-for-seamless-decision-making/。（访问日期：2024年9月8日）。[10] 搜索 - consensus：用于研究的人工智能搜索引擎。https://consensus.app/search/。（访问日期：2024年9月7日）。[11] Simone Alghisi, Massimo Rizzoli, Gabriel Roccabruna, Seyed Mahed Mousavi 和 Giuseppe Riccardi。我们应该微调还是 RAG？评估不同技术以适应 LLM 进行对话。arXiv 预印本 arXiv:2406.06399，2024。 [12] 亚马逊网络服务。按需 EC2 实例定价 - 亚马逊网络服务，2024。访问日期：2024年9月7日。 [13] 匿名。模型告诉自己关注哪里：忠实性与自动注意力引导相遇。在提交给 ACL Rolling Review - 2024年6月，2024。审稿中。 [14] Jiawei Chen, Hongyu Lin, Xianpei Han 和 Le Sun。基准测试检索增强生成中的大型语言模型。在人工智能 AAAI 会议论文集，卷 38，页 17754–17762，2024。 [15] Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis 和 He He。通过语言模型的上下文调优进行元学习。arXiv 预印本 arXiv:2110.07814，2021。 [16] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez 等。聊天机器人竞技场：一个通过人类偏好评估 LLM 的开放平台。arXiv 预印本 arXiv:2403.04132，2024。 [17] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu 和 Zhifang Sui。关于上下文学习的调查。arXiv 预印本 arXiv:2301.00234，2022。 [18] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan 等。Llama 3 模型群。arXiv 预印本 arXiv:2407.21783，2024。 [19] 在 Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal 和 Lin Zhong。提示缓存：低延迟推理的模块化注意力重用，2023。

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, 和 Yoshua Bengio. 生成对抗网络. ACM通讯, 63(11):139–144, 2020. [21] Aman Gupta, Anup Shirgaonkar, Angels de Luis Balaguer, Bruno Silva, Daniel Holstein, Dawei Li, Jennifer Marsman, Leonardo O Nunes, Mahsa Rouzbahman, Morris Sharp, 等. RAG与微调: 管道、权衡及农业案例研究. arXiv预印本 arXiv:2401.08406, 2024. [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, 和 Jian Sun. 深度残差学习用于图像识别. 在IEEE计算机视觉与模式识别会议论文集中, 页码770–778, 2016. [23] HuggingFace. 文本生成推理, 2024. [24] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, 和 Lili Qiu. Llmlingua: 压缩提示以加速大型语言模型的推理. arXiv预印本 arXiv:2310.05736, 2023. [25] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, 和 Graham Neubig. 主动检索增强生成. arXiv预印本 arXiv:2305.06983, 2023. [26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, 和 Ion Stoica. 大型语言模型服务的高效内存管理与分页注意力. 在第29届操作系统原理研讨会论文集中, 页码611–626, 2023. [27] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, 等. 知识密集型NLP任务的检索增强生成. 神经信息处理系统进展, 33:9459–9474, 2020. [28] Jiarui Li, Ye Yuan, 和 Zehua Zhang. 通过RAG增强LLM的事实准确性以对抗幻觉: 针对私有知识库中的领域特定查询的案例研究. arXiv预印本 arXiv:2403.10446, 2024. [29] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, 和 Colin A Raffel. 少量参数高效微调优于上下文学习且更便宜. 神经信息处理系统进展, 35:1950–1965, 2022. [30] Haotian Liu, Chunyuan Li, Qingyang Wu, 和 Yong Jae Lee. 视觉指令调优. 神经信息处理系统进展, 36, 2024. [31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, 和 Alexander C Berg. SSD: 单次多框检测器. 在计算机视觉–ECCV 2016: 第14届欧洲会议, 荷兰阿姆斯特丹, 2016年10月11–14日, 论文集, 第I部分14, 页码21–37. Springer, 2016. [32] Yixin Liu, Kai Zhang, Yuan Li, Zhiling Yan, Chujie Gao, Ruoxi Chen, Zhengqing Yuan, Yue Huang, Hanchi Sun, Jianfeng Gao, 等. Sora: 大型视觉模型的背景、技术、局限性和机会的综述. arXiv预印本 arXiv:2402.17177, 2024. [33] Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael Maire, Henry Hoffmann, Ari Holtzman, 等. Cachegen: 语言模型应用的快速上下文加载. arXiv预印本 arXiv:2310.07240, 2023. [34] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, 和 Anshumali Shrivastava. Scissorhands: 利用重要性假设的持久性进行LLM KV缓存压缩. 神经信息处理系统进展, 36, 2024. [35] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, 和 Yanai Elazar. 少量微调与上下文学习: 公平比较与评估. arXiv预印本 arXiv:2305.16938, 2023. [36] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, 和 Xinran Xu. Mooncake: Kimi 的KVCACHE中心

architecture for llm serving. *arXiv preprint arXiv:2407.00079*, 2024.

[37] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[38] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[39] Cunxiang Wang, Xiaoze Liu, Yuanhao Yue, Xiangru Tang, Tianhang Zhang, Cheng Jiayang, Yunzhi Yao, Wenyang Gao, Xuming Hu, Zehan Qi, et al. Survey on factuality in large language models: Knowledge, retrieval and domain-specificity. *arXiv preprint arXiv:2310.07521*, 2023.

[40] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

[41] Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. Inference scaling for long-context retrieval augmented generation. *arXiv preprint arXiv:2410.04343*, 2024.

[42] Qingru Zhang, Chandan Singh, Liyuan Liu, Xiaodong Liu, Bin Yu, Jianfeng Gao, and Tuo Zhao. Tell your model where to attend: Post-hoc attention steering for llms. *arXiv preprint arXiv:2311.02262*, 2023.

[43] Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention. *arXiv preprint arXiv:2303.16199*, 2023.

[44] Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. Raft: Adapting language model to domain specific rag. *arXiv preprint arXiv:2403.10131*, 2024.

[45] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023.

[46] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104*, 2023.

大型语言模型服务的架构。arXiv 预印本 arXiv:2407.00079，2024。[37] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser 和 Björn Ommer。使用潜在扩散模型进行高分辨率图像合成。在 IEEE/CVF 计算机视觉与模式识别会议论文集中，页码 10684–10695，2022。[38] Mingxing Tan, Ruoming Pang 和 Quoc V Le。Efficientdet：可扩展且高效的目标检测。在 IEEE/CVF 计算机视觉与模式识别会议论文集中，页码 10781–10790，2020。[39] Cunxiang Wang, Xiaoze Liu, Yuanhao Yue, Xiangru Tang, Tianhang Zhang, Cheng Jiayang, Yunzhi Yao, Wenyang Gao, Xuming Hu, Zehan Qi 等。关于大型语言模型中的事实性调查：知识、检索和领域特异性。arXiv 预印本 arXiv:2310.07521，2023。[40] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu 和 Junchen Jiang。Cacheblend：通过缓存知识融合实现快速的大型语言模型服务。arXiv 预印本 arXiv:2405.16444，2024。[41] Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang 和 Michael Bendersky。用于长上下文检索增强生成的推理扩展。arXiv 预印本 arXiv:2410.04343，2024。

[42] 张清如，辛丹丹，刘丽媛，刘晓东，余彬，高剑锋，赵拓。告诉你的模型在哪里关注：后期注意力引导用于大型语言模型。arXiv 预印本 arXiv:2311.02262，2023年。[43] 张任瑞，韩嘉铭，刘克里斯，高鹏，周傲君，胡向飞，严世林，卢攀，李洪生，乔宇。LLaMA-Adapter：使用零初始化注意力高效微调语言模型。arXiv 预印本 arXiv:2303.16199，2023年。[44] 张天俊，帕蒂尔·希希尔·G，贾南·贾因，沈晟，马泰·扎哈里亚，斯托伊卡·伊昂，戈恩萨雷斯·约瑟夫。Raft：将语言模型适应于特定领域的RAG。arXiv 预印本 arXiv:2403.10131，2024年。[45] 张振宇，盛颖，周天怡，陈天龙，郑连敏，蔡瑞思，宋赵，田元东，克里斯托弗·瑞，克拉克·巴雷特，王张扬，陈北迪。H2O：高效生成大型语言模型推理的重击者预言机。在ICML2023基础模型高效系统研讨会，2023年。[46] 郑连敏，尹良生，谢志强，黄杰，孙楚悦，余浩宇，曹诗怡，科齐拉基斯·克里斯托斯，斯托伊卡·伊昂，戈恩萨雷斯·约瑟夫等。使用sglang高效编程大型语言模型。arXiv 预印本 arXiv:2312.07104，2023年。