XC-CACHE: Cross-Attending to Cached Context for Efficient LLM Inference

Anonymous ACL submission

Abstract

In-context learning (ICL) approaches typically leverage prompting to condition decoder-only language model generation on reference information. Just-in-time processing of a context is inefficient due to the quadratic cost of self-attention operations, and caching is desirable. However, caching transformer states can easily require almost as much space as the model parameters. When the right context isn't known in advance, caching ICL can be challenging. This work addresses these limitations by introducing models that, inspired by the encoder-decoder architecture, use cross-attention to condition generation on reference text without the prompt. More precisely, we leverage pre-trained decoder-only models and only train a small number of added layers. We use Question-Answering (QA) as a testbed to evaluate the ability of our models to perform conditional generation and observe that they outperform ICL, are comparable to fine-tuned prompted LLMs, and drastically reduce the space footprint relative to standard KV caching by two orders of magnitude.

1 Introduction

Large Language Models (LLMs) have propelled advances in language modeling and enabled automatic production of almost human-like text. Despite impressive progress, challenges persist in applying LLMs in practical settings such as the risk of *hallucinations* (or rather *confabulatations* (Bottou and Schölkopf, 2023; Millidge, 2023)) and of non-factual (Li et al., 2023a; Xu et al., 2024) or toxic content (Zou et al., 2023; Xhonneux et al., 2024) in their generated text. Moreover, without fine-tuning, it is surprisingly difficult to adapt these models to incorporate new information not included in their training data (Luo et al., 2023; Kalajdzievski, 2024). Indeed, while LLMs can answer queries about their training data somewhat accurately in a zero-shot fashion (Petroni et al., 2019), queries regarding information not available at training time often lead to inaccuracies (Maynez et al., 2020; Ji et al., 2023) as one would expect.

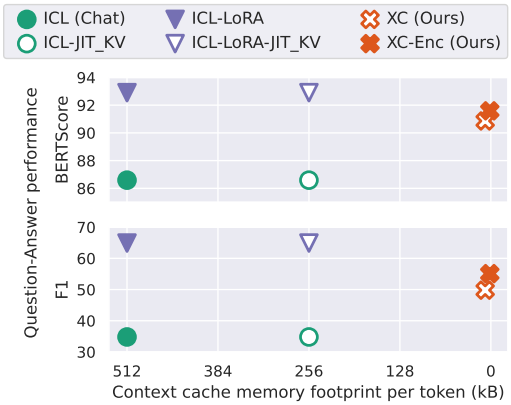


Figure 1: Average QA performance vs. caching memory footprint per context token. **The closer to the top-right corner, the better**. XC-LLAMA variants drastically reduce cache size at a small cost in accuracy.

This work focuses on grounding LLM generation on contextual information provided at inference time (Figure 2(a)). The most common approach for conditioning model generation is arguably *In-Context Learning* (ICL) (Radford et al., 2019; Brown et al., 2020): one prepends the relevant context to the prompt to generate an answer conditioned on the combined query and context. This technique is a core component of popular frameworks, such as Retrieval-Augmented Generation (RAG) (Lewis et al., 2020), with the specificity that the relevant context is not known *a priori*, but has to be retrieved from a corpus (Ram et al., 2023).

While somewhat effective and straightforward, ICL, as typically performed with decoder-only architectures, has flaws. On the one hand, ICL-based generation is known to present high variance with respect to the prompt template so that equivalent valid-looking prompt formats produce drastically different results (Chen et al., 2023). On the other

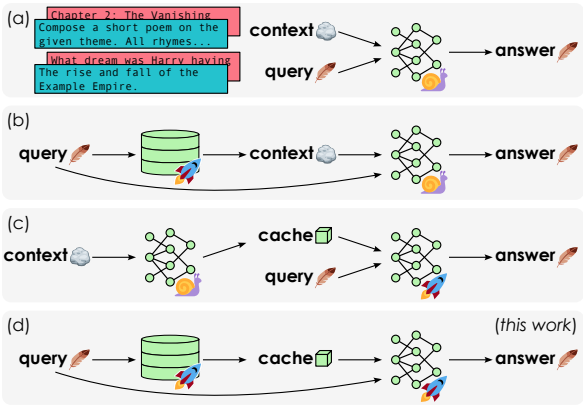


Figure 2: **Faster inference in context-conditional language modeling.** (a) A use case where the user's query must be interpreted within some context to generate an answer. This work focuses on cases where the query and the answer are both small (light), but the context is large (heavy). The resulting LLM time complexity is thus $\mathcal{O}\big(|\text{context}|^2\big)$ (slow). (b) In-context learning (ICL) and retrieval augmented generation (RAG) are two examples where the query is used to look up the context *from a finite corpus*. (c) In many cases, the context can be processed in advance to a cache enabling $\mathcal{O}\big(|\text{context}|(|\text{query}| + |\text{answer}|)\big)$ (fast) inference on a given query. (d) Finite context corpus may be processed offline, enabling fast execution at inference. Since cache size affects storage and communication costs, we search for models requiring smaller cache.

hand, ICL is costly in terms of time and space. Just-in-time processing of the context suffers from quadratic complexity on the length due to self-attention operations (Vaswani et al., 2017). The alternative is pre-processing and caching the context internal states (the so-called key-value or KV states) to speed up inference (Figure 2(c)). However, this *can require the same order of space as the model parameters themselves* (we give details in Section 2). Recent work has reduced the space requirements of KV caching by sub-sampling states (Xiao et al., 2023; Adnan et al., 2024), although at the cost of ignoring relevant content.

To overcome these limitations, we propose alternatives to ICL that perform conditional generation without injecting the relevant information in the prompt (Figure 2(a)), and seek to implement lightweight cache methods as illustrated in Figure 2(d). Our approach is reminiscent of the, now arguably legacy, encoder-decoder architectures, as it relies on cross-attention layers to condition generation on pre-computed context encodings. More precisely, we propose cross-context-cache (XC-CACHE), which stores only the outputs of the encoder hidden states and relies on cross-attention to ingest the cache at inference time. We instantiate XC-CACHE via two parameter-efficient approaches that leverage pre-trained decoder-only models and extend them with a separate encoder to process the context: one approach uses the frozen decoder as an encoder (called XC-LLAMA), and the other uses a small bi-directional encoder (called XC-LLAMAENC). Crucially, our encoder-decoder architectures are more amenable to caching the context states, requiring orders of magnitude less space than their ICL counterparts. When context caching is enabled, fine-tuned models result in higher accuracy performance, but demand a large memory footprint (and consequently, higher latency and cost). In contrast, our XC-CACHE approach substantially reduces cache memory requirements by nearly 98%; and as other efficiency-improving techniques like quantization (Frantar et al., 2023), this reduction comes at a minor cost in accuracy, as illustrated in Figure 1. Nevertheless, our method consistently outperforms ICL alternatives based on LLAMA 2 or GPT-3.5, as detailed in Section 5.

Overall, we advocate for a conceptual shift in architecture design for conditional generation, which should recenter on caching and make it integral to a model's operation rather than an afterthought. Our contributions are summarized as follows:

1. *Cacheability*: We provide evidence that encoder-decoder architectures are good candidates for conditional generation since our cache-friendly models *enhance model performance* compared to ICL, while *reducing cache memory footprint* by more than 98%.

2. *Parameter efficiency*: We show that *training a few cross-attention layers* (and optionally, a small encoder) suffices to convert decoders into encoder-decoder pairs. We contribute a mix of training tasks that enable context-conditional generation without costly ICL.

3. *Decoder-as-encoder*: We show that representations extracted from pre-trained causal decoders *can be used as-is* to replace an encoder.

2 Caching Representations

Let context, query and answer denote sequences of tokens from a shared vocabulary $\mathbb{V}$. We write $|\text{context}|$, $|\text{query}|$ and $|\text{answer}|$ the respective length of these sequences. Figure 2(a) illustrates an LLM

which, conditioned on context, produces an answer to a user-specified query.

Assumptions. In what follows, we make three main assumptions: (A1) the context is not unique and depends on the query; (A2) there exists a manageable amount of contexts to condition on; and (A3) the context is large, *i.e.*, $|\text{query}| + |\text{answer}| \ll |\text{context}|$. In doing so, we restrict ourselves to the regimes where processing contexts offline is both compelling (A1 and A3) and viable (A2).

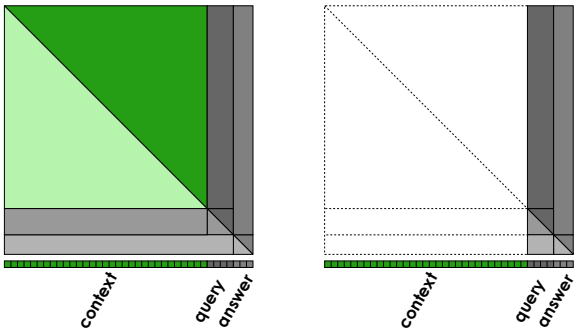
Many ICL use cases satisfy these assumptions. For example, when facing a number of different tasks, we may craft task-specific instructions, each detailing how to obtain the answer to the query (Figure 2(b)). The same applies to RAG-based question-answering when retrieved documents are encoded independently: the retriever selects from a corpus the most relevant documents (context) to answer the user question (query).

2.1 Inference Time Complexity

In general, the time complexity of LLM inference is dominated by its attention mechanism, which in the ICL setting is $\mathcal{O}\big((|\text{context}| + |\text{query}| + |\text{answer}|)^2\big)$. In the large context regime (A3), this simplifies to $\mathcal{O}\big(|\text{context}|^2\big)$: we can thus expect tangible inference speedups by improving how we handle the context. One way to achieve such speedups is to pre-process the contexts offline to some intermediate states (cache) and provide it to the model at inference (Figure 2(c)). This way, the quadratic cost of processing the context is paid once, allowing inference to simply look up a ready-made cache (Fig. 2(d)). Note that the incurred storage and communication overheads are linear in the size of the cache, which is linear in context length. In this setup, inference time complexity becomes $\mathcal{O}\big(|\text{context}|(|\text{query}| + |\text{answer}|)\big)$, *i.e.* linear in context length (Fig. 3), a significant speedup.

2.2 Practical costs of caching

Practical considerations might negatively affect a cached-enabled setup. **Loading** and **communication** overheads are both linear in cache size, motivating smaller cache. Any **extra operations** required at inference mitigates caching benefits. Caching methods may incur an (implicit or explicit) cost in the **quality** of the generated answer. See Appendix A for details.



(a) Without cache. (b) KV caching.

Figure 3: **Stylized representation of attention execution time** (area) for context-conditional language modeling. Dashed lines in (b) shows the savings when the context's (past) keys and values cache are provided. For causal models, the area below the diagonal represents execution time that could be saved by other means.

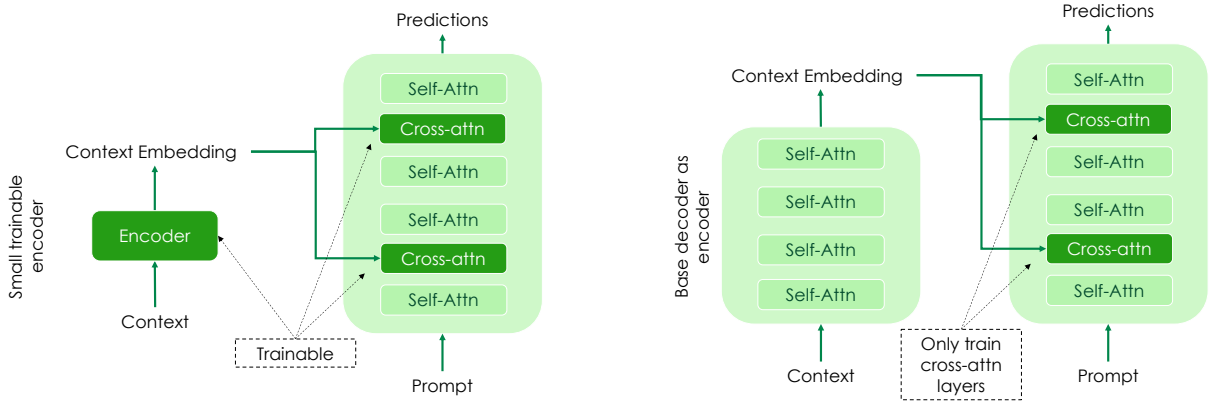
2.3 Approaches to Caching

KV Caching. The default approach is to store the (past) keys and value states generated while processing context, hereafter called KV caching. KV caching is commonly associated with a conversational setting, in which case the cache often remains on the GPU device between conversational rounds (Figure 2(d)). Other setups, such as serving multiple conversations, demand, however that we move cache from storage (and/or CPU RAM) to the GPU, incurring further latency overhead. As an example, for LLAMA 2-7B (Touvron et al., 2023) using 16 bits precision, we must move around a whopping 512 kB *per token*.[1] Smaller per-token cache sizes are thus desirable.

JIT-KV Caching. An alternative approach is to store the (past) hidden states of the model in the cache (in the case of LLAMA 2-7B, this would be half as big, *i.e.*, 256 kB per token). At inference time, once these hidden states are loaded on GPU, we can recover the full keys and values in $\mathcal{O}\big(|\text{context}|\big)$ well-parallelized operations (recall that a transformer's keys and values are obtained by a linear operation on the hidden states).We call this approach "just-in-time" or JIT-KV caching.

XC Caching (Ours). Both KV and JIT-KV caching perform the exact same computations as the original model without a cache. They both suffer from two types of costs while producing the same answer: the **size** of the cache and the **oper-**

[1] 32 layers, 32 heads per layer, 128 floating points per key *and* value (thus a factor 2), 2 bytes per 16 bit floating point



(a) Trainable small bi-directional encoder.

(b) Decoder as the (causal) encoder.

Figure 4: **XC-LLAMA's architectures.** A decoder-only model implements encoder-decoder architectures. Fine-tuning out in a parameter-efficient fashion via training only a small number of cross-attention layers.

ations to be performed at inference (*e.g.*, convert hidden states to keys and values). In this work, we propose XC caching (read *cross-context caching*), which considers the *quality loss* of the generated answer as a third such cost and present two models balancing these three costs. Both models employ cross-attention layers integrated into the decoder architecture, enabling the model to attend on context cached with a compact memory footprint at inference time. Models are described in Section 3.

3 XC: Cross Attending to Efficiently Cached Context

To reduce the memory footprint of caching, our approach draws inspiration from the encoder-decoder architecture, which until recently was the go-to design for conditional generation. Recently, decoder-only models have become more popular in part due to their data efficiency: the entire parameter budget is allocated to the decoder and not "wasted" on an encoder, with the additional advantage that all parameters are trained against all data tokens. Even though maintaining an external encoder may seem wasteful, we favor an encoder-decoder architecture: it better lends itself to pre-computing and caching context representations. Indeed, only the encoder output vectors need to be stored, as opposed to intermediate states across all of the decoder's self-attention layers (KV and JIT-KV cache).

In what follows, we refer to a model as composed of an encoder $\mathcal{E} : \mathbb{V}^{|\text{context}|} \mapsto \mathbb{R}^{d|\text{context}|}$, which takes in a context and outputs token-level representations of size d, and of a decoder $\mathcal{D} : \mathbb{V}^m \times \mathbb{R}^{d|\text{context}|} \mapsto \Delta^{|\mathbb{V}|}$, which takes as input the query and the context encodings and outputs an answer in the simplex of size $|\mathbb{V}|$. More precisely, the decoder $\mathcal{D}$ autoregressively outputs the parameters of a categorical distribution over the vocabulary.

For the sake of parameter efficiency and to leverage state-of-the-art pre-trained LLMs, we start from an existing decoder-only model and augment it with new cross-attention layers interleaved between existing self-attention layers, as illustrated in Figure 4. We consider two strategies to implement the encoder $\mathcal{E}$. The first one trains a *small encoder*, *i.e.*, $\mathcal{E}$ is orders of magnitude smaller than $\mathcal{D}$. The second one uses a *decoder as encoder*, *i.e.*, the frozen decoder-only model is used out-of-the-box as the encoder ($\mathcal{E} := \mathcal{D}$). More precisely, we use as encodings the representations extracted from the pre-trained $\mathcal{D}$ at its last layer before the language modeling head.

Choosing one approach over the other depends on practical considerations. If caching is possible and context representations can be computed offline, then using the decoder as an encoder is preferable for overall simplicity and parameter efficiency. Otherwise, a small encoder would make just-in-time processing of contexts significantly less costly. Both approaches inherit the advantages of trained decoder-only models while benefiting from using an encoder during inference. In particular, contextual information can be efficiently cached since only the output at $\mathcal{E}$'s top layer must be stored instead of the entire set of $\mathcal{D}$'s intermediate states.

Finally, to enable context conditioning, we train exclusively the newly added modules: the cross-attention layers in one setting and the small encoder in the other. The base decoder is kept frozen in both settings (even when it acts as an encoder). Con-

sequently, our training procedure does not affect the original parameters of $\mathcal{D}$, which can still be used as a general-purpose language model when the additional layers are removed.

4 Experimental Setup

We focus on the question-answering task and train on a combination of datasets where context, query, and answer triplets are available. QA is an ideal testbed for our approaches as it requires efficiently looking up external information and accounting for it at generation time.

4.1 Training Dataset

We build a training dataset by standardizing and pooling together the training partitions of five publicly available and diverse (open-domain, multi-hop, conversational) datasets: NATURAL QUESTIONS (NQ) (Kwiatkowski et al., 2019), HOTPOTQA (Yang et al., 2018), TOPIOCQA (Adlakha et al., 2022), MS MARCO (Bajaj et al., 2016), and SQUAD-V2 (Rajpurkar et al., 2018). Examples from the resulting training dataset contain a query (natural-language question), an answer (expected output), and one or more contexts (*e.g.*, knowledge base articles), where at least one of them contains the answer to the query. We refer to this context as *reference* context. We make use of a validation set for model selection and hyperparameter tuning. For datasets with a validation set but not a test partition, we use the validation set for testing and create a validation set by holding out 10% of randomly selected training samples. We apply a further filtering step in our training data and remove examples with contexts longer than 6,000 tokens, corresponding to less than 5% of the samples.

Evaluation is performed on the resulting test partitions of NATURAL QUESTIONS, HOTPOTQA, and TOPIOCQA. Further details and statistics of the datasets we considered can be found in Appendix B.

4.2 Auxiliary Tasks

In addition to training on the primary QA tasks, we optimize our models on context repetition tasks, as described below. The advantage of defining such auxiliary tasks is two-fold. On the one hand, they allow us to optimize the likelihood of all available tokens, even those used as input to the encoder. On the other hand, they help avoid sub-optimal solutions where the cross-attention layers behave as simple identity operators. Indeed, our models could learn to ignore the context but cannot do so when tasked to repeat the context.

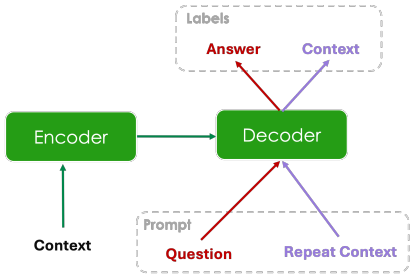


Figure 5: **Multitask training strategy.** Within an epoch, each example is presented to the model twice. In the first round, the model predicts the answer conditionally based on the context and the query. In the second appearance of an example within the epoch, we train the model to repeat or infill the context.

In practice, we use reserved tokens to instruct the model to repeat the context, either as-is or by infilling (Bavarian et al., 2022), *i.e.* returning it in the prefix-suffix-middle or the suffix-prefix-middle order, as done by Li et al. (2023b); Lozhkov et al. (2024). Such tasks introduce new variability, as the model learns to copy-paste the context and to find and replace its missing chunks, resulting in improved performance in the multi-epoch training setting. Note that we train our models on question-answering and context repetition on every training sample. We set one training epoch to correspond to two passes over the training dataset, where we perform the primary or the secondary tasks in each pass, as illustrated in Figure 5.

4.3 Implementation Details

We rely on the openly available pre-trained LLAMA 2 (Touvron et al., 2023) to define variations of XC-LLAMA. Specifically, all our empirical assessments use the 7 billion parameter version of LLAMA 2. For the variation of XC-LLAMA where a dedicated encoder is trained (referred to as XC-LLAMAENC from now on), we fine-tune as encoder a LONGFORMER (Beltagy et al., 2020), which is a BERT (Devlin et al., 2018) of approximately 125M parameters, pre-trained on relatively long documents using efficient self-attention. We note, however, that the longest publicly available LONGFORMER accepts inputs of at most 4,096 tokens, which is shorter than our maximum length of 6,000. We then increase the model's maximum input length by repeating the starting position embeddings. Upon fine-tuning, the model does handle

contexts longer than 4,096 tokens without noticeable problems.

As per adding cross-attention layers to XC-LLAMA, we introduce one cross-attention layer every few self-attention layers of the transformer. In particular, we found the 5-6 configuration (*i.e.*, inserting five cross-attention layers while skipping six self-attention ones) to work consistently well. We thus use this configuration throughout our evaluations. We remark that we strive to be parameter efficient and keep the parameter count of added modules below 10% of $\mathcal{D}$'s parameter count. Training is carried out with the ADAMW optimizer (Loshchilov and Hutter, 2017) with a batch size of 256 and for 40,000 steps (amounting to 4 training epochs). We use a linear learning rate scheduler with a warm-up phase and such that the learning rate reaches 0 at the last training step. A comprehensive list of hyperparameter values is shown in Table 8 in Appendix C.

4.4 Metrics and Evaluation

We compare our models against ICL methods for generating answers conditioned on context and query. We note that contexts in the cases we considered present a relatively low *signal-to-noise* ratio, as most of the tokens are related to the answer but are not at all relevant. In some more extreme situations, the posed question cannot be answered from the context, and models are expected (and trained or prompted to) indicate that it is not possible to answer based on the provided context.

We use the same metrics and evaluation setup described by Adlakha et al. (2023) – such as F1 SCORE, RECALL, METEOR, and ROUGEL – but keep F1 as our metric of focus. In addition, we evaluate BERTSCORE (Zhang et al., 2019) measured between predictions and ground-truth answers.

5 Results

5.1 Comparison with existing methods

We first compare our method to existing approaches for conditional generation. Our main baseline is ICL, *i.e.*, providing the context as part of the prompt. More specifically, we report baseline results for LLAMA 2-CHAT, which we found to perform better than the base pre-trained LLAMA 2. We further report results for GPT-3.5-TURBO, OpenAI's black-box model, presumably conditioned by prompting. For these ICL baselines, we selected the prompt templates based on generated answer quality on sample validation data (refer to Appendix C for details). Finally, we report the results of FUSION-IN-DECODER (FiD) (Izacard and Grave, 2021), a T5-based (Raffel et al., 2020) conditional generative model, which consistently proved to be state-of-the-art on QA tasks (Borgeaud et al., 2022; Wang et al., 2023a; Adlakha et al., 2023). Unlike the decoder-only backbone of our models, FiD is arguably no longer a general-purpose model, as all of its parameters are fine-tuned to perform QA. More importantly, as discussed in more depth in Section 5.2, pre-processing and caching context representations is not an option for FiD since it requires knowing the question at encoding time. Nevertheless, we introduce the FiD baseline to check where our models stand relative to established QA-specialized models.

Results are presented in Table 1. On the considered datasets, cross-attending to the contexts (XC-LLAMA or XC-LLAMAENC) considerably improves performance w.r.t prompting (LLAMA 2-CHAT). The gap varies depending on the dataset. We conjecture that this is due to the high variance induced by prompting, although it might exist an optimal prompt for each dataset to help close this gap. Thus, approaches that do not rely on the prompt offer the advantage of being more broadly applicable and, hence, more practical. We also note that even in the setting where the decoder is used as an encoder, cross-attending to contexts still yields better performance than ICL, no matter the base decoder we compare against. This suggests that the trained cross-attention layers compensate for the potential sub-optimality because the encoder representations are not explicitly trained for the task we evaluate. A broader set of results comprising more metrics and models can be found in Table 5.

5.2 Broader Analysis

The results in the previous section show that adding and fine-tuning dedicated parameters for context-conditioning improves performance relative to prompting. Based on this observation, in this section, we expand our evaluation to consider alternative approaches that leverage a small number of additional parameters that enable models to condition generation on reference contexts. In addition to prediction performance, we now also focus on *computational efficiency*; namely, we assess how amenable to pre-processing and caching representations of contexts different models are.

We thus fine-tune LORA adapters (Hu et al.,

Dataset	Model	F1	BERTSCORE
NQ	GPT-3.5 TURBO	57.80	90.87
	FID	59.05	91.75
	LLAMA 2-CHAT	41.26	87.43
	XC-LLAMA (Ours)	59.95	92.87
	XC-LLAMAENC (Ours)	**63.12**	**93.30**
HOTPOTQA	GPT-3.5 TURBO	39.37	87.83
	FID	45.60	90.56
	LLAMA 2-CHAT	29.63	86.02
	XC-LLAMA (Ours)	43.94	90.55
	XC-LLAMAENC (Ours)	**54.57**	**92.08**
TOPIOCQA	GPT-3.5 TURBO	40.18	87.52
	FID	31.22	85.95
	LLAMA 2-CHAT	33.45	86.33
	XC-LLAMA (Ours)	45.47	89.16
	XC-LLAMAENC (Ours)	**47.73**	**89.40**

Table 1: Question-Answer performance on three diverse information-seeking tasks. GPT-3.5 TURBO and LLAMA2-CHAT are given reference context through ICL (prompting), while our approach uses cross-attention layers to ingest reference embeddings.

Model	Cache size (kB/token)
LLAMA 2-ICL-KV	512
LLAMA 2-ICL-JIT-KV	256
LLAMA 2-ICL-LoRA	512
XC-LLAMA (Ours)	8
XC-LLAMAENC (Ours)	**1.5**

Table 2: Cache memory footprint per context token for different models. The encoder-decoder approach of XC-LLAMA allows for huge savings due to the need to store only the last hidden states instead of KV states throughout layers. Cache sizes here assume 16-bit precision.

2021) applied to the same LLAMA 2 decoder we use for XC-LLAMA. In other words, we fine-tune our ICL baseline from Section 5 to control for the effect of prompting in the performances we observe, which, as observed in Table 3, drastically improves QA accuracy relative to the ICL baselines reported in Table 1. However, despite enabling improvements in prediction performance, relying on LORA-style model adaptation still necessitates storing all KV states throughout every layer to cache contextual information, incurring significant costs. On the other hand, models with an encoder require caching the hidden states of only their last layer. This is more clearly depicted in Table 2, where we further consider a compromise scenario for ICL that caches hidden states at each layer rather than KV states. We refer to this variant of ICL as LLAMA 2-ICL-JIT-KV, since it performs just-in-time KV projection of cached hidden states, trading time for space.

Note that XC-LLAMA variations greatly reduce the caching footprint simply because they require storing only the last hidden states of the encoder. XC-LLAMAENC reduces space requirements even further, as the representation of LONGFORMER are lower dimensional relative to that of LLAMA 2.

Reducing cache size carries substantial practical implications, notably in reducing the memory footprint of a pre-processed knowledge base. For example, moving from an ICL approach to introducing an encoder could reduce cache size by orders of magnitude. This is particularly significant when storing pre-processed representations of vast datasets like the entire Wikipedia. Additionally, reducing the cache size leads to runtime savings by mitigating communication costs, as the volume of information transferred from disk to device is markedly reduced. Finally, scaling down the cache size frees up device memory at inference time, enabling longer generation or larger batch sizes for batched inference. See appendix A for details.

We provide a joint view of those two performance components – prediction and computational – in Figure 1, where we plot aggregate F1 and BERTSCORE across datasets versus the amount of cache per context token required by models. Note that models closer to the top-right corner are preferred since they are highly accurate at a low caching cost. While no optimal methods are identified in the Pareto sense, the Pareto optimal set consists of ICL models fine-tuned with LORA on one end, offering slightly higher aggregate BERTSCORE but requiring substantial caching space. On the other hand, models with an encoder make small sacrifices in prediction accuracy while significantly reducing the memory footprint. We also note that there is a gap between XC-LLAMA and XC-LLAMAENC, and the additional parameters introduced by XC-LLAMAENC yield a boost in accuracy and improve space efficiency.

Detailed QA results are reported in Table 3, which includes the GPT-3.5 TURBO ICL baseline and FiD, which is now fine-tuned on our training dataset. Again, we highlight that FiD, while included for performance reference, is not directly comparable to the other models and does not support caching in pre-processed contexts. Our reporting of FiD aims to provide insight into where our models stand relative to a specialized baseline. As previously mentioned, our models incur a slight reduction in prediction accuracy but achieve significant space savings, which proves advantageous in

Dataset	Model	F1	BERTSCORE
NQ	GPT-3.5 TURBO	57.80	90.87
	FID (fine-tuned)	**72.67**	**94.76**
	LLAMA 2 (LoRA)	67.38	93.27
	XC-LLAMA (Ours)	59.95	92.87
	XC-LLAMAENC (Ours)	63.12	93.30
HOTPOTQA	GPT-3.5 TURBO	39.37	87.83
	FID (fine-tuned)	53.54	89.64
	LLAMA 2 (LoRA)	**71.97**	**94.62**
	XC-LLAMA (Ours)	43.94	90.55
	XC-LLAMAENC (Ours)	54.57	92.08
TOPIOCQA	GPT-3.5 TURBO	40.18	87.52
	FID (fine-tuned)	41.52	86.54
	LLAMA 2 (LoRA)	**55.41**	**90.80**
	XC-LLAMA (Ours)	45.47	89.16
	XC-LLAMAENC (Ours)	47.73	89.40

Table 3: Question-Answer performance with ICL, carried out with a model specialized to a prompt template via LoRA. Note that the FiD model was pre-trained on NQ, hence its high performance on that dataset.

various practical scenarios. For a comprehensive overview, please refer to the full results in Table 5.

6 Related Work

Decoders as encoders. Repurposing pre-trained decoders is becoming a popular approach to leverage powerful language models for applications other than generative modeling. For example, GRIT (Muennighoff et al., 2024) converts a pre-trained causal decoder into a bi-directional encoder, yielding sentence-level embeddings while maintaining its ability to perform autoregressive generation of text. However, unlike the models we consider, this conversion requires fine-tuning all model parameters instead of additional ones. Parameter-efficient approaches to turn decoders into encoders were also proposed, such as in (Meng et al., 2024) and (BehnamGhader et al., 2024), where a pre-trained MISTRAL decoder (Jiang et al., 2023) is fine-tuned in a contrastive setting using LoRA adapters to yield sentence level representations for retrieval. Closer to our method is CODET5+ (Wang et al., 2023d), which also defines encoder-decoder architectures rather than turning decoders into sentence encoders. Similar to one of our variants (XC-LLAMA), it is assembled from two pre-trained decoders, one used as an encoder and the other as a decoder and linked with a few cross-attention operations. However, CODET5+ requires fine-tuning the entire and relatively large encoder. We show that a good pre-trained decoder has good enough representations, but one can improve upon it efficiently using just a very small trainable encoder.

Conditioning without prompts. One recent line of work has focused on controlling a model's generation by intervening in its parameters directly (Wang et al., 2023c; Zhang et al., 2023; Wang et al., 2023b), in particular, to either introduce or erase knowledge post-training. Such approaches would typically require accessing and rewriting the internal parameters of a pre-trained language model and are not amenable to frequently changing contexts such as in a Question-Answer setting.

Efficient inference. There exist several methods for improving inference speed and memory footprint. One approach is to lower the numerical precision or quantize the model weights and/or data, which has been shown to preserve the model accuracy with only 8 bits per weight (Dettmers et al., 2022), or even lower precision with 4, 3, or even 2 bits per weight (Frantar et al., 2023). Alternatively, the key-query cache can be compressed (Ainslie et al., 2023; Nawrot et al., 2024), although doing so requires fine-tuning. Finally, using Flash Attention (Dao et al., 2022) leads to significant savings for just-in-time processing of contexts. These methods orthogonally complement what we present in this paper and can be combined with XC-LLAMA.

7 Conclusion

We introduced XC-LLAMA as an approach to transform a pre-trained decoder-only language model into an encoder-decoder architecture that can condition generation on both the encoder inputs and the decoder query. This is achieved by integrating cross-attention layers interleaved in between existing self-attention layers of the pre-trained decoder. We describe two approaches for defining the encoder: using a copy of the decoder or introducing a trainable but small bi-directional encoder.

The proposed architecture allows for a reduction in caching space by a factor exceeding 300. When evaluated in the QA setting, we observe a higher prediction accuracy than standard ICL approaches achieved through both LLAMA 2 and GPT-3.5 TURBO. Additionally, we observe accuracy levels nearly on par with caching-intensive fine-tuned prompted models, providing a more caching-friendly alternative to prompted language models that proves highly practical. Identified limitations are discussed in Section 8.

8 Limitations

Most of the models discussed in this paper have impressive results in the QA setting. However, our experience working with these models reveals limitations primarily stemming from their reliance on an underlying language model. As such, these methods inherit potential flaws of the language model they build upon.

For instance, typical large language models are trained on vast amounts of text, likely including information related to questions in publicly available QA benchmarks. While such training on related data may offer a shortcut to models for correctly answering questions in the context that they have "memorized" during training, it can also introduce errors: the model may "remember" related but inaccurate information relative to a specific query. In simpler terms, prompting-based ICL approaches and potentially our models can bypass context and rely solely on memory to generate a continuation. This is undesirable as we aim to ensure that models accurately account for the reference context rather than pre-training's. Fine-tuning models address the abovementioned issue by encouraging reliance on the provided context, as we explicitly do with context repetition and filling auxiliary tasks. However, it also specializes in fine-tuning data to such an extent that it may hurt the generalization ability of datasets that deviate from what was observed during fine-tuning. We estimate to what extent models know the answer after pre-training in Table 6.

Unseen Datasets. To test the generalization ability of the models we evaluate to data non-identically distributed concerning fine-tuning, we curated a test dataset containing 16,290 reference documents, with roughly 5 question-answer pairs per document. The documents were written by human annotators, who wrote about imaginary scenarios, each featuring several subsections with titles and subtitles. Importantly, given that the reference documents in this dataset contain fictional or untrue information, we can reasonably assume that none of the models can rely on or be misled by their memory of the training data. Additionally, documents in this dataset and accompanying questions and answers exhibit unique styles that may differ from the fine-tuning data.

When evaluated in this more challenging out-of-distribution scenario, the accuracy of all models experiences a significant decline, as depicted in Figure 6, where we compare QA scores for all models

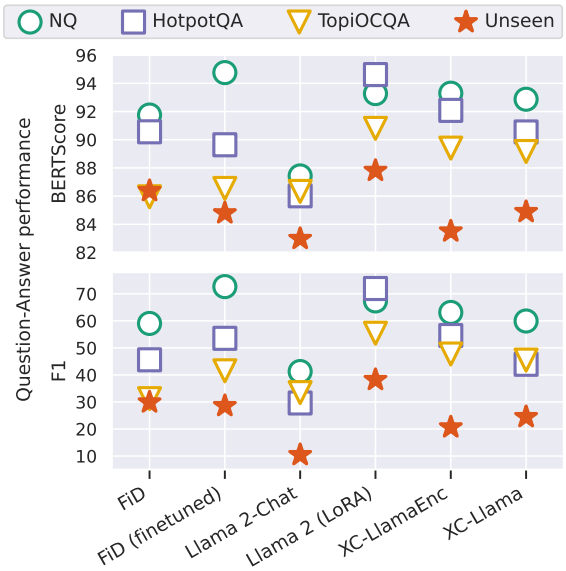


Figure 6: QA performance of various models on our three test datasets against our curated unseen dataset. All models struggle to generalize in this out-of-distribution scenario where the test dataset significantly differs from that used for training.

across all datasets, including the curated one labeled as *Unseen*. We hypothesize that, apart from the inability of models to rely on memory to answer queries, the main factor driving the accuracy decline is the divergence between the test dataset and the fine-tuning data. This discrepancy may arise from variations in writing style, document lengths, signal-to-noise ratio, and the presence of distracting content related to the question but not useful for its answer, among other potential variations. Identifying the exact sources of errors and enhancing the robustness of context-conditional models present promising avenues for future research.

Ethics Statement

We acknowledge the environmental impact of computational resources, including the carbon footprint associated with data storage and processing. By focusing on conditional generation through caching with a small memory footprint, we aim to minimize energy consumption and promote sustainability in computational practices.

While cache reduction can lead to speed improvements, we recognize the importance of additionally implementing safeguards to ensure responsible usage of computational models. This entails mitigating bias, fairness, and privacy issues, thus fostering ethical deployment of technology.

References

Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. 2023. Evaluating correctness and faithfulness of instruction-following models for question answering. *arXiv preprint arXiv:2307.16877*.

Vaibhav Adlakha, Shehzaad Dhuliawala, Kaheer Suleman, Harm de Vries, and Siva Reddy. 2022. Topiocqa: Open-domain conversational question answering with topic switching. *Transactions of the Association for Computational Linguistics*, 10:468–483.

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *arXiv preprint arXiv:2403.09054*.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Léon Bottou and Bernhardt Schölkopf. 2023. Borges and ai. *arXiv preprint arXiv:2310.01425*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yanda Chen, Chen Zhao, Zhou Yu, Kathleen McKeown, and He He. 2023. On the relation between sensitivity and accuracy in in-context learning. *Preprint*, arXiv:2209.07661.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Damjan Kalajdzievski. 2024. Scaling laws for forgetting when fine-tuning large language models. *arXiv preprint arXiv:2401.05605*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096.

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

Junyi Li, Xiaoxue Cheng, Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023a. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023b. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919.

Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2024. Sfr-embedding-mistral:enhance text retrieval with transfer learning. Salesforce AI Research Blog.

Beren Millidge. 2023. Llms confabulate not hallucinate. Accessed on April 3, 2024.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2024. Generative representational instruction tuning. *arXiv preprint arXiv:2402.09906*.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. 2024. Dynamic memory compression: Retrofitting LLMs for accelerated inference. *arXiv preprint arXiv:2403.09636*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. *choice*, 2640:660.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473.

Alexander Peysakhovich and Adam Lerer. 2023. Attention sorting combats recency bias in long context language models. *arXiv preprint arXiv:2310.01427*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *roceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Vol 2: Short Papers)*, pages 784—-789.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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

Boxin Wang, Wei Ping, Peng Xu, Lawrence McAfee, Zihan Liu, Mohammad Shoeybi, Yi Dong, Oleksii Kuchaiev, Bo Li, Chaowei Xiao, et al. 2023a. Shall we pretrain autoregressive language models with retrieval? a comprehensive study. *arXiv preprint arXiv:2304.06762*.

Weixuan Wang, Barry Haddow, and Alexandra Birch. 2023b. Retrieval-augmented multilingual knowledge editing. *Preprint*, arXiv:2312.13040.

Xiaoxing Wang, Xiangxiang Chu, Yuda Fan, Zhexi Zhang, Bo Zhang, Xiaokang Yang, and Junchi Yan. 2023c. Rome: Robustifying memory-efficient nas via topology disentanglement and gradient accumulation. *Preprint*, arXiv:2011.11233.

Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. 2023d. CodeT5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1069–1088, Singapore. Association for Computational Linguistics.

Sophie Xhonneux, David Dobre, Jian Tang, Gauthier Gidel, and Dhanya Sridhar. 2024. In-context learning can re-learn forbidden tasks. *arXiv preprint arXiv:2402.05723*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Chiyuan Zhang, Daphne Ippolito, Katherine Lee, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023. Counterfactual memorization in neural language models. *Preprint*, arXiv:2112.12938.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

A Practical costs of caching

Although a complexity linear in the context length is achievable while maintaining context quality, practical considerations might negatively affect a cached-enabled setup.

Loading and communication overheads. Even though long-term storage costs are relatively low, loading and communication latencies are directly proportional to the space footprint of the cached content. The two main bottlenecks here are loading cache from disk (if disk storage is used), and moving cache to the GPU's memory. If the total amount of information to cache fits in CPU RAM, there may be no need to store them on disk during inference, enabling better-than-linear returns on smaller caching sizes.

Extra operations. In the case of LLAMA 2, JIT-KV caching requires a cache that is half as big as the one of KV – 254 kB instead of 512 kB per token – and is thus clearly more advantageous on the loading and communication fronts. However, extra operations are required at inference time to convert the hidden states into the keys and values required by the model. Here these extra operations turn out to be cheap linear operations. Similarly, our XC caching approaches require very little storage (*i.e.*, 8 kB and 1.5 kB per token), which is a clear win on loading and communication fronts. However, extra layers are added to the original model in Sec. 3), which induces time costs that have to be accounted for in the overall speedup. Again, these extra operations turn out to be negligible compared to the benefit of caching.

Explicit quality cost. KV and JIT-KV caching both return *exactly* the same answer as the corresponding un-cached model: these caching strategies incur no (explicit) cost in answer quality compared to that un-cached model. In contrast, our XC caching strategy does not confer this guarantee: compromises between costs and benefits must be considered.

Implicit quality cost. All caching methods considered in this work share the same implicit "opportunity cost": processing context offline implies that the resulting cache is independent of the specifics of the query Depending on the use case, this might result in a loss of answer quality. For instance, in the QA setting, knowing the question when processing the context allows the model to focus on the information that is useful to generate an answer. Arguably, this results in better context representations as it reduces the risk of confusing the LLM with irrelevant information.

B Datasets

In this section we give some details about our training datasets and how we used them for training. Statistics about the datasets are in Table 4.

Dataset	# Train	# Valid	# Test
Natural Questions	83,118	2,240	5,067
HotpotQA	83,038	7,405	7,405
TopiOCQA	42,702	2,656	2,512
MS Marco	808,298	101,035	0
Squad V2	130,293	11,872	0
Totals	1,147,449	125,208	14,084

Table 4: Statistics for datasets used to train and test our models. For the first three datasets, we use the validation split of each to test our model and split the provided train set into *train* and *valid*.

Natural Questions (NQ). Aimed at evaluating open-domain question-answering tasks, Natural Questions (Kwiatkowski et al., 2019) presents a collection of real user questions from Google queries. Answers are written by human annotators and are based on Wikipedia articles (that may or may not contain the exact information needed for the question). For each sample in the dataset, the context we use as input is a paragraph containing the answer. We use the open version of NQ (Lee et al., 2019), which is the subset of the original dataset where the contexts have at most 100 tokens from English Wikipedia (indexed Dec 2018). This open variant of NQ does not provide the contexts relevant to each question. We thus use the questions in OPEN NQ to query the original NQ dataset and fetch the contexts associated with each question. Given that NQ provides multiple correct answers to a given query-context pair, we split each sample into multiple rows at train time, so that each question-context pair is paired with each one of the provided answers. For the test set, however, we evaluate a test sample once and compare model outputs to all the reference answers and consider the answer that maximizes the given metric.

Hotpot QA. The dataset contains open-domain questions that require at least two or more documents to be answered (Yang et al., 2018). Each sample in the dataset includes a list of contexts (paragraphs from English Wikipedia indexed Dec 2017), two of which contain useful information for answering the question, while the remainder are distractors. Thus, the answers in this dataset typically depend on more than one component of the context, so a model is expected to look up different parts of the context to compose an accurate answer.

TopiOCQA. A conversational dataset where samples represent information-seeking open-domain dialogue, the answer to each question in TopiOCQA (Adlakha et al., 2022) is based on 200-token passages from English Wikipedia (indexed Oct 2020). Human annotators generate these reference answers.

MS MARCO. Intended to evaluate reading comprehension and question answering, the questions from the MS MARCO dataset (Nguyen et al., 2016) are sampled from real user questions from Bing or Cortana. The reference passages are collected from the web (not necessarily Wikipedia) through the Bing search engine, and they may or may not be enough to answer the question. Answers are human-generated.

Squad V2. Squad v2 (Rajpurkar et al., 2018) consists of a small set of question/answer pairs generated by annotators and based on Wikipedia articles. The answers in Squad are spans from the original context or yes-no replies and hence are less abstractive.

We train our baselines and models with the train split of all five datasets mentioned above, concatenated and shuffled. We evaluate models on the validation split of the first three datasets – Natural Questions, Hotpot QA, and TopioCQA – as well as an unseen dataset we created.

C Complete set of results

Table 5 contains results for all methods, metrics, and for all datasets we accounted for in our empirical assessment.

D Inference results with no context

We evaluate our pretrained LLMs on the test split of our datasets without passing a context to base the answer on. We do this to have a better sense of how many of the answers they already know without ICL.

E Further details on baselines

GPT Details. We set temperature and presence and frequency penalty to 0, TOP-P to 1.0 and n to 1. Please refer to OpenAI documentation[2] to learn more about these parameters. To generate GPT answers, we prompt GPT with the following.

GPT Prompt

You are a helpful assistant who is able to generate brief and correct answers to questions, grounded on a given text.
You are now given a "question" and a "context" possibly containing the answer to the question.
Answer the question based only on the context given.
If the answer to the question is not in the context, then say UNANSWERABLE. Your answer must be concise and to the point.

Question: {question}
Context: {context}
Answer:

[2] https://www.openai.com/docs/

Dataset	Model	EM	Precision	Recall	F1	Rouge-L	METEOR	BERTSCORE
Natural Questions	GPT-3.5 TURBO	43.08	55.10	74.87	57.80	57.67	53.22	90.87
	FID	50.64	63.18	58.63	59.05	59.02	47.06	91.75
	FID (fine-tuned)	67.00	75.20	72.59	72.67	72.52	58.30	94.76
	LLAMA 2-CHAT	24.96	40.41	58.20	41.26	41.28	38.04	87.43
	LLAMA 2 (LoRA)	59.64	70.55	67.39	67.38	67.08	53.88	93.27
	XC-LLAMAENC (Ours)	56.77	65.93	62.89	63.12	62.99	49.99	93.30
	XC-LLAMA (Ours)	51.41	62.62	60.01	59.95	60.21	47.47	92.87
HotpotQA	GPT-3.5 TURBO	27.62	39.08	46.43	39.37	39.35	34.01	87.83
	FID	33.81	48.95	45.54	45.60	45.63	35.24	90.56
	FID (fine-tuned)	44.04	54.49	55.38	53.54	53.42	41.78	89.64
	LLAMA 2-CHAT	14.22	27.07	51.48	29.63	29.56	30.82	86.02
	LLAMA 2 (LoRA)	58.33	74.91	72.72	71.97	71.91	55.27	94.62
	XC-LLAMAENC (Ours)	43.29	56.88	54.90	54.57	54.63	41.13	92.08
	XC-LLAMA (Ours)	31.90	46.27	44.02	43.94	44.35	32.17	90.55
TopiOCQA	GPT-3.5 TURBO	17.18	44.29	45.77	40.18	39.23	35.50	87.52
	FID	16.43	53.56	27.17	31.22	31.02	20.56	85.95
	FID (fine-tuned)	24.94	49.80	41.69	41.52	41.31	32.31	86.54
	LLAMA 2-CHAT	13.76	42.56	36.87	33.45	32.88	26.95	86.33
	LLAMA 2 (LoRA)	25.89	61.29	57.43	55.41	54.92	47.19	90.80
	XC-LLAMAENC (Ours)	24.82	55.44	47.83	47.73	47.49	39.22	89.40
	XC-LLAMA (Ours)	19.17	52.49	46.64	45.47	44.94	38.37	89.16

Table 5: Question-Answer performance on three diverse information-seeking tasks. All models in this table that are trained/fine-tuned, did so on the same five datasets; the table reports the metrics of testing these models on different test splits.

Dataset	Model	EM	Precision	Recall	F1	Rouge-L	METEOR	BERTSCORE
Natural Questions	GPT-3.5 TURBO (No ICL)	4.42	15.33	51.58	20.45	19.88	27.53	84.01
	LLAMA 2-CHAT (No ICL)	8.62	15.66	26.34	16.06	16.30	14.44	83.39
HotpotQA	GPT-3.5 TURBO (No ICL)	6.01	16.52	49.51	20.83	20.41	26.05	84.12
	LLAMA 2-CHAT (No ICL)	5.27	10.19	15.39	10.42	10.58	9.50	81.29
TopiOCQA	GPT-3.5 Turbo (No ICL)	0.48	11.51	15.09	10.81	11.07	13.30	83.69
	LLAMA 2-CHAT (No ICL)	4.53	11.05	10.45	8.98	9.12	6.93	80.65

Table 6: Question-Answer performance when *no context* is given to the model (only a question).

LoRA Fine-tuning Details. We fine-tune the LLAMA 2 model using LoRA adaptation (Hu et al., 2021). To get a comparable number of trainable parameters as for XC-LLAMA, we allow LoRA to modify all three attention projection matrices (for the queries, keys, and values), and set r to 360, α to 360, and dropout ratio to 0.5. Note that all models, unless otherwise specified, are trained on the same training data that is a pooling of all five datasets discussed in Section 4.1. The prompt for this fine-tuned model follows.

LLAMA 2-7B (LoRA) Prompt

<|system|>
{context}
<|user|>
{question}
<|assistant|>

LLAMA 2-Chat Details. We initially fine-tuned both LLAMA 2 and LLAMA 2-CHAT (a version of LLAMA 2 fine-tuned for chat) to our dataset, but report results on LLAMA 2-CHAT as it was the model with superior performance. The prompt varies slightly between the version that includes the context (LLAMA 2-CHAT Prompt) and the version that excludes it (LLAMA 2-CHAT(No ICL) Prompt). Both prompt versions follow.

LLAMA 2-CHAT Prompt

[INST] <<SYS>>
Please answer the following question given the following passages. Please be brief. If you cannot answer the question, please reply with 'UNANSWERABLE'.

<</SYS>>

{context}
Question: {question}
[/INST]
Answer:

LLAMA 2-CHAT (No ICL) Prompt

[INST] <<SYS>>
Please answer the following question. Please be brief. If you cannot answer the question, please reply with 'UNANSWERABLE'.

<</SYS>>

Question: {question}
[/INST]
Answer:

F Sensitivity analysis: position of the reference context

We explore the sensitivity of the models to the positioning of the reference context (useful for answering the query) within the full the context given to the model. Past work has shown that standard architectures based self-attention and used for language modeling tend to focus on specific parts of the past content when generating a token, and suffer from primacy and recency bias (Liu et al., 2024; Peysakhovich and Lerer, 2023; Xiao et al., 2023). To explore the performance of our models against such position shifts, we turn our focus to the HOTPOTQA dataset, whose contexts are given by lists of (roughly) 10 sentences, out of which two are marked as *useful* to answering the posed question. We generate four variants of the dataset that differ by the positions of the useful contexts at the beginning of these context paragraphs, in the middle, at the end, and a variant in which the two useful contexts are far away from each other and placed at the first and last positions. We test both our models and LLAMA 2-CHAT (LoRA) on each of the datasets and see that all models are somewhat sensitive to the position of the useful contexts, but ranges are relatively small (*i.e.* less than 5%)as compared to more drastic scenarios such as what is reported in

the literature for ICL (cf. Figure 1 in (Liu et al., 2024) with ranges greater than 20%), suggesting that fine-tuned context-conditional language models are less biased to certain parts of the context.

Model	Positionis of the useful contexts				
	Shuffled	Beginning 0,1	Middle 4,5	End 8,9	Far apart 0,9
LLAMA 2-CHAT	29.63	**33.88**	30.45	32.80	32.05
LLAMA 2 (LoRA)	71.97	72.28	71.83	**74.35**	73.14
XC-LLAMA (Ours)	43.94	43.52	**44.30**	42.30	43.70
XC-LLAMAENC (Ours)	54.57	**57.65**	56.79	54.61	52.98

Table 7: Results on HotpotQA when we vary the position of the two useful contexts (out of a total of roughly 10 contexts per sample). We test beginning (positions 0, 1), middle (positions 4, 5), end (positions 8, 9) positions, and the case where the two contexts are far apart (positions 0, 9).

G Training hyperparameters

The main training hyperparameters used for XC-LLAMA are reported in Table 8.

Optimizer	ADAMW
Base learning rate	0.0002
Learning rate warmup steps	2,500
Maximum gradient norm	1
Batch size	256
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	0.000001
Weight decay	0.001
Cross-attention bias	False
Cross-attention dropout probability	0.2
Cross-attention final layer	True
Cross-attention hidden size	2,048
Number of Cross-attention layers	5
Cross-attention layer skips	6
Cross-attention number of attention heads	32
Cross-attention number of key value heads	32
Numerical precision	16-bit (BF16)
Number of training steps	40,000

Table 8: Training hyperparameters for XC-LLAMA.