Compressing Many-Shots in In-Context Learning

Devvrit Khatri[*,δ,◇] **Pranamya Kulkarni**[◇] **Nilesh Gupta**[δ,◇] **Yerram Varun**[◇]
Liqian Peng[◇] **Jay Yagnik**[◇] **Praneeth Netrapalli**[◇] **Cho-Jui Hsieh**[†,◇]
Alec Go[◇] **Inderjit S Dhillon**[δ,◇] **Aditya Kusupati**[◇] **Prateek Jain**[*,◇]
[◇]Google [†]UCLA [δ]University of Texas at Austin

Abstract

Large Language Models (LLMs) have been shown to be able to learn different tasks without explicit finetuning when given many input-output examples / demonstrations through In-Context Learning (ICL). Increasing the number of examples, called "shots", improves downstream task performance but incurs higher memory and computational costs. In this work, we study an approach to improve the memory and computational efficiency of ICL inference by compressing the many-shot prompts. Given many shots comprising t tokens, our goal is to generate a m soft-token summary, where $m < t$. We first show that existing prompt compression methods are ineffective for many-shot compression, and simply using fewer shots as a baseline is surprisingly strong. To achieve effective compression, we find that: (a) a stronger compressor model with more trainable parameters is necessary, and (b) compressing many-shot representations at each transformer layer enables more fine-grained compression by providing each layer with its own compressed representation. Based on these insights, we propose **MemCom**, a layer-wise compression method. We systematically evaluate various compressor models and training approaches across different model sizes (2B and 7B), architectures (Gemma and Mistral), many-shot sequence lengths (3k-6k tokens), and compression ratios (3× to 8×). MemCom outperforms strong baselines across all compression ratios on multiple classification tasks with large label sets. Notably, while baseline performance degrades sharply at higher compression ratios, often by over 20-30%, MemCom maintains high accuracy with minimal degradation, typically dropping by less than 10%.

[*] Correspondence to: devvrit.03@gmail.com, prajain@google.com

1 Introduction

In-Context Learning (ICL) (Brown et al., 2020a) is a method by which language models learn a task by conditioning on examples of that task, provided as part of the input. Large Language Models (LLMs) have demonstrated remarkable abilities to generalize to new tasks simply by attending to a sequence of input-output pairs – commonly referred to as "shots" – followed by a test input. This approach has gained widespread popularity due to its flexibility and minimal setup. Unlike finetuning, ICL requires no updates to model parameters. As a result, the same model can be used for all downstream tasks without any finetuning overhead.

A common assumption has been that finetuning typically yields superior performance on downstream tasks compared to ICL. However, recent studies (Agarwal et al., 2024; Bertsch et al., 2025) demonstrate that this performance gap narrows significantly as the number of in-context examples increases. As the support for long context for LLMs increases, many-shot ICL poses as an attractive and practical alternative to finetuning – retaining the benefits of task flexibility and reduced overhead, while delivering improved accuracy through richer context.

Despite its promise, many-shot ICL introduces its own set of challenges, primarily in terms of memory and computation overhead during inference. To support many-shot learning, the key-value (KV) cache corresponding to all example tokens must be stored and maintained, leading to significant memory usage. Additionally, each test input must attend to these cached examples, which results in substantial computational cost—especially as the number of shots increases. This creates a bot-

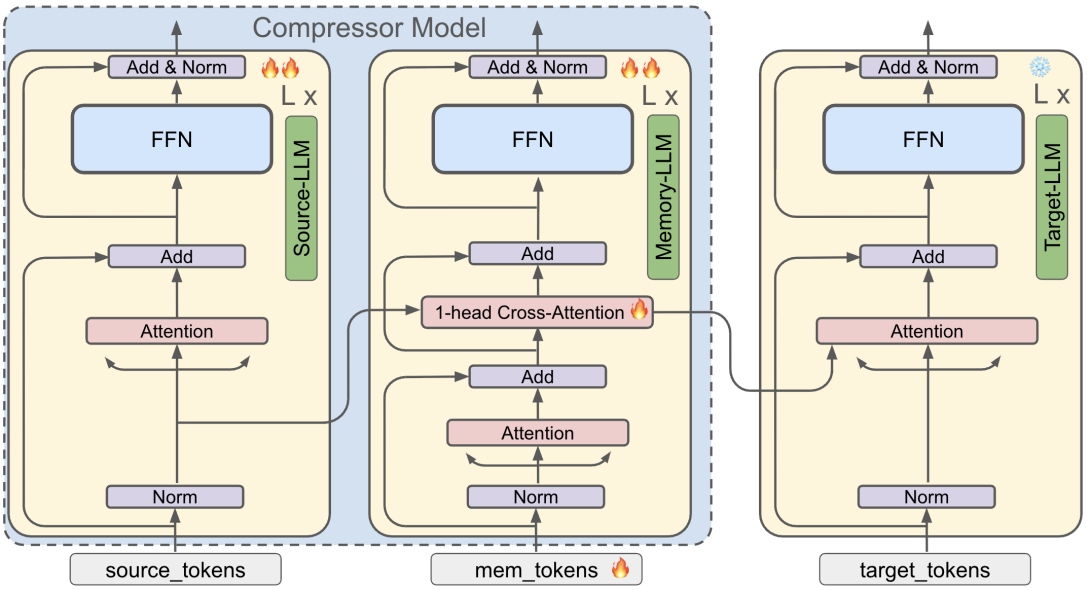


Figure 1: MemCom consists of two stacks of LLMs working as compressor. The Source-LLM processes the input tokens to be compressed. The Memory-LLM processed memory tokens, which cross-attends to the input representation in corresponding layer. Finally, during inference, Target-LLM (frozen ❄) processes test input and attends to the compressed representations at each layer. Training happens in two phases. In Phase-1, only the cross-attention module and the memory tokens are trained (represented by 🔥), while in Phase-2 the entire stack of Source-LLM and Memory-LLM is trained (🔥🔥), in addition to the memory tokens.

tleneck that limits the scalability of many-shot ICL in practical deployments.

In this work, we address the above challenge by introducing a LLM-based compression technique, MemCom, to form compressed KV cache. At inference time, the model attends only to this compressed cache, significantly reducing memory usage and attention computation. A practical use case for this approach arises in edge deployments of LLMs, where resource constraints demand efficient inference. In such scenarios, a hybrid Cloud-Edge architecture can be employed: the cloud performs the many-shot compression offline, and the resulting compressed representation is used by the edge device during inference. Hybrid strategies leveraging cloud-edge collaboration have been explored in various domains (Almeida et al., 2022; Chen et al., 2024; Laskaridis et al., 2020), including recent work that customizes edge LLMs using LoRA adapters hosted in the cloud (Bang et al., 2024). Similarly, our approach enables offline compression of in-context examples, making many-shot ICL practical and efficient for deployment in constrained environments.

MemCom consists of two stacks of LLMs working as compressor (Figure 1) producing a compressed KV cache of the many-shot input across layers. The Source-LLM (left) processes the original many-shot t input tokens. The Compressor-LLM (middle) takes a fixed set of m memory tokens and, at each layer i, performs cross-attention over the i-th layer representations of the t input tokens in the Source-LLM. The resulting compressed memory representations are passed to the Target-LLM (right), which is the deployed model responsible for inference. It processes the test inputs and, at each layer, attends to the compressed representations instead of the full input. Note that the target-LLM is frozen, and not finetuned. Since compression is performed offline, we opt for a powerful compressor to ensure high-quality, near-lossless compression. At inference time, the model only needs to attend to m tokens rather than the original t, substantially reducing memory and compute.

To the best of our knowledge, this is the first work to explore many-shot compression for in-context learning. While several prompt compression techniques have been proposed (Mu et al., 2023; Ge et al., 2024; Chevalier et al., 2023), they are primarily designed for short prompts (typically ¡1k tokens) and often operate by compressing only the final hidden

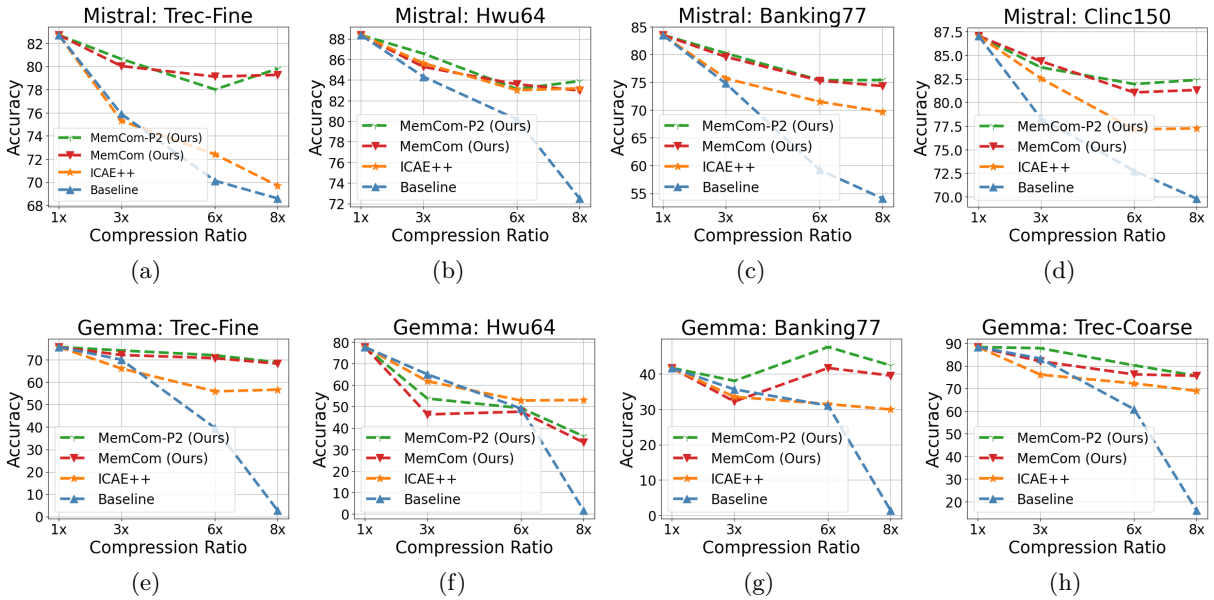


Figure 2: Comparison of compression performance across different compression ratios. As the compression ratio increases, the vanilla baseline ("Baseline") exhibits significant performance degradation. ICAE++ performs better than the vanilla baseline but remains less robust compared to MemCom, which maintains high performance even at high compression ratios. The top row shows results for Mistral-7B, where sequences of $6k$ tokens are compressed. The bottom row shows results for Gemma2-2B, with $3k$-token sequences being compressed. MemCom refers to our method's performance post Phase-1 training, which introduces comparatively lightweight compressor of 1-head cross attention in every layer. While MemCom-P2 refers to the performance post Phase-2 training, which has two full stacks of LLMs working as the compressor. Section 4 introduces the training methodology.

states. We tested the most relevant recent method, ICAE (Ge et al., 2024), to the demanding long-context many-shot setting, and found that it does not retain performance. Our analysis suggests two primary bottlenecks with such approaches for many-shot compression: first, the limited capacity of the compressor model, and second, the coarse nature of compressing using only final-layer compressed representations. Interestingly, a simple baseline of using fewer shots to fit the m-token budget proves surprisingly strong. In addition to increasing compressor capacity (i.e., using more trainable parameters), we identify that a more fine-grained compression strategy is crucial for improving compression quality. Thus, MemCom introduces a strong layer-wise compression, a key departure from prior methods, enabling each transformer layer to access its own tailored compressed representation of the many-shot context.

We perform evaluations on five classification tasks with large label sets (Table 1), and demonstrate that MemCom consistently outperforms the strongest baseline across compression ratios ranging from 3× to 8× Notably, as the compression ratio increases, the performance of the few-shot baseline degrades sharply, whereas MemCom maintains robust accuracy with only marginal decline (Figure 2).

2 Related Work

The impact of scaling many-shots in In-Context Learning. Brown et al. (2020b) demonstrated that scaling LLMs significantly enhances their few-shot learning capabilities, paving the way for models like GPT-3. Subsequent work (Agarwal et al., 2024; Bertsch et al., 2025; Li et al., 2023) explored the benefits of many-shot ICL, showing that increasing the number of demonstrations leads to substantial performance gains, often rivaling supervised fine-tuning. This underscores the potential of scaling demonstrations in ICL while emphasizing the growing need for efficient methods to manage the associated computational costs.

Shot-selection and KV cache compression. Previous works have explored demon-

stration selection as a strategy to improve ICL efficiency (Dong et al., 2024; Luo et al., 2024). However, these approaches still require storing the full set of many-shot examples, leaving the memory overhead issue unaddressed. Additionally, they operate entirely at inference time by dynamically retrieving relevant examples based on the test query. In contrast, our method is orthogonal—compression is performed offline, and there is no additional inference-time overhead. Other methods (Zhang et al., 2023; Xiao et al., 2024; Li et al., 2024) propose KV cache compression techniques that attend only to predefined token positions (e.g., start or end tokens). However, these strategies are incompatible with many-shot ICL, as different queries often rely on distinct and diverse demonstration sets.

Comparison with approaches to prompt compression for efficient inference. To address the computational challenges of long prompts, researchers have explored various prompt compression techniques. Chevalier et al. (2023) introduced AutoCompressors for recursively compressing long text into summary vectors. However, their training involves adapting the main model, while we work in the setting where target-model is frozen. Ge et al. (2024) proposed the In-context Autoencoder (ICAE), leveraging LLMs to encode contexts into memory slots. However, their work is based on a specific Prompt-with-Context (PwC) task they introduce in the paper, with contexts of length $< 1k$ tokens. Moreover, their aim is to provide a parameter efficient compressor, which we show in Section 5.1 is not effective for long-context many-shot compression. Jiang et al. (2023b) compress general prompts into concise natural language, whereas our aim is to provide compression for many-shot ICL task using memory-tokens, which don't necessarily translate into natural language. Mu et al. (2023) introduced "gisting," training LLMs to compress prompts into gist tokens. However, it requires modifying the main model, similar to Chevalier et al. (2023). Moreover, the evaluations are on very shot prompts of length < 50 tokens. Tan et al. (2024) proposed LLoCO, an offline context compression framework for long context, though trained particularly for task of document summarization. Moreover, their target-LLM requires LoRA updates, whereas we work in frozen-target LLM regime.

While existing prompt compression methods have shown promise, they have not studied the task of long-context many-shot compression. The complex interplay of numerous diverse examples within the many-shots require more fine-grained compression strategies with expressive compressors. Our proposed method, MemCom, aims to overcome these limitations by introducing a layer-wise compression approach coupled with a stronger compressor model. We operate in the frozen-target LLM regime, which aligns with practical deployment scenarios such as on-device inference at the edge. In such cases, modifying the model weights is often undesirable, as it may interfere with the LLM's broader capabilities. Additionally, access to the original training data is rarely available, making any finetuning prone to overfitting and distributional drift. The most relevant prior work under this constraint is ICAE (Ge et al., 2024), which we thoroughly compare with in Section 5.

3 Experimental Setup

We conduct our experiments using two different LLM architectures: Gemma2-2B (Team, 2024) and Mistral-7B-v0.3 (Jiang et al., 2023a), both using their base pre-trained checkpoints. To train our compressor models, we exclusively use a pretraining datasets, without any instruction tuning or task-specific finetuning. This stands in contrast to prior prompt compression works (Ge et al., 2024; Tan et al., 2024; Jiang et al., 2023b), which typically relies on additional supervision and task specific finetuning to guide the compression process. Our motivation for using pretraining data is threefold. First, in the context of ICL, labeled training data is often scarce or unavailable. Second, it is well-established that continued pretraining improves the downstream ICL capabilities of LLMs (Brown et al., 2020b). We view the compressor-assisted target model as an alternate architecture trained via next-token prediction: instead of attending to previous tokens explicitly, the model learns to predict next token by attending to the previous tokens' compressed representations. Thus, we hypothesize that training on large-scale pretraining data

Table 1: Datasets used in our work. We test each method on ICL tasks with varying large label sets and different domains. The average demonstration length is the average number of tokens using Mistral/Gemma tokenizer on each example shot.

Dataset	Domain	# Labels	Avg Demo Length	Example Output
Trec-Coarse	Questions	6	20.57/17.45	abbreviation, entity
Trec-Fine	Questions	47	20.38/18.11	manner, reason
HWU64	Conversational	64	20.89/18.89	music_query, social_post
Banking77	Financial	77	27.34/25.06	change_pin, request_refund
Clinc150	Multiple	151	20.01/18.05	cook_time, income

should also improve compression quality over time. Third, pretraining corpora are abundant, making it feasible to scale up training if resources permit—potentially improving compressor performance even more with further training.

The pretraining datasets used are a mix of FineWebEdu (Penedo et al., 2024) and SlimPajama (Soboleva et al., 2023). For downstream evaluations, we use a set of classification tasks with large label space. Table 1 and Appendix A.3 provide information of these tasks, including the number of classes and representative label examples. We do evaluations under two different sequence lengths: 3k tokens of many-shot input for Gemma2-2B and 6k tokens for Mistral-7B. To assess the effectiveness of compression, we experiment with three compression ratios: 3×, 6×, and 8×. Details of training hyperparameters such as learning rate, batch size, and weight decay are provided in Appendix A.2, and the training methodology for MemCom is described in Section 4.

4 MemCom Architecture and Training

MemCom consists of two LLM stacks forming a compressor, along with a set of learnable memory tokens. The first LLM, referred to as the Source-LLM, is initialized with copy of the target-LLM, and processes the many-shot input tokens to be compressed. The second LLM, the Memory-LLM, processes the memory tokens. It is initialized as a copy of the target-LLM as well, and in addition includes a randomly initialized 1-head cross-attention module at each layer. This module allows the memory tokens to attend to the corresponding layer representations from the Source-LLM. As a result, m memory tokens extract compressed information from t input tokens, producing a compact representation at each layer. The Target-LLM, which remains frozen during training and is used for inference, attends to these compressed memory tokens instead of the original many-shot input. This layer-wise transfer of compressed information enables the Target-LLM to leverage the many-shot context while avoiding the memory and compute overhead of attending to all t input tokens explicitly.

The compression mechanism, depicted in Figure 1 operates at each transformer layer. Specifically, for any given layer i, let $H^i_{source} \in \mathbb{R}^{t\times d}$ be the sequence of t source token input representations in the Source-LLM layer i. Similarly, let $H^i_{memory} \in \mathbb{R}^{m\times d}$ be the sequence of m memory token representations after the Memory-LLM's self-attention module in layer i. The core of the compression is a cross-attention module where the memory tokens query the source tokens: $O^i = CrossAttention(Q = H^i_{memory}, K = H^i_{source}, V = H^i_{source})$. The Target-LLM then uses these layer-specific compressed representations O_i as the key and value context for its attention modules, instead of attending to the original t tokens.

We train MemCom using a standard next-token prediction loss on the target tokens. As mentioned in Section 3, only pretraining data is used for training. During training, for Gemma2-2B we sample 4k-token sequences and randomly select a split point within the range $[2.7\text{k}, 3.4\text{k}]$. The tokens before this point are assigned as source tokens and the remaining tokens as target tokens. For Mistral-7B, we sample 8k-token sequences and apply the same strategy, selecting a random split point in the range $[5.7\text{k}, 6.3\text{k}]$ for source/target token split. Training is conducted in two phases.

Phase-1: Since the cross-attention modules and memory tokens are randomly initial-

ized—while the rest of the compressor is initialized from well-trained parameters, we begin by training only these randomly initialized components. This allows them to learn meaningful compression behavior without disrupting the pretrained parameters.

Phase-2: After initial convergence, we unfreeze the entire Source-LLM and Memory-LLM stacks and continue training both LLMs jointly, including the memory tokens and cross-attention layers.

Each phase is trained on approximately 80 billion tokens of pretraining data. The Target-LLM remains frozen throughout both phases. As we'll see in Section 5, most of the gains are achieved post Phase-1 training itself. While additional marginal gains can be achieved post Phase-2 training.

5 Experiments

5.1 Baseline Study and Motivating Insights

We begin our analysis by evaluating ICAE (Ge et al., 2024) as a baseline for many-shot compression. ICAE introduces a compressor LLM, initialized with the same parameters as the target LLM but augmented with LoRA adapters on the query and key projection matrices (Figure 3a). The input sequence is appended with m learnable memory tokens, and a forward pass is performed through the compressor. The output memory token representations are treated as the compressed context. During inference, these m compressed tokens are prepended to the target tokens in the frozen target LLM. Thereby, during inference the target tokens attend only to the compressed memory representations, rather than the full source sequence.

The ICAE training objective is similar to MemCom's, using next-token prediction on the target tokens. In addition, ICAE includes an auto-encoding loss, where the model attempts to reconstruct the original source tokens from the compressed memory representation. For a detailed description of ICAE's architecture and training procedure, we refer the reader to Ge et al. (2024). In our work, we train ICAE with just next-token prediction loss. Using auto-encoder loss in addition led to instability at higher learning rate, and non-optimal training at lower learning rate (Appendix A.2).

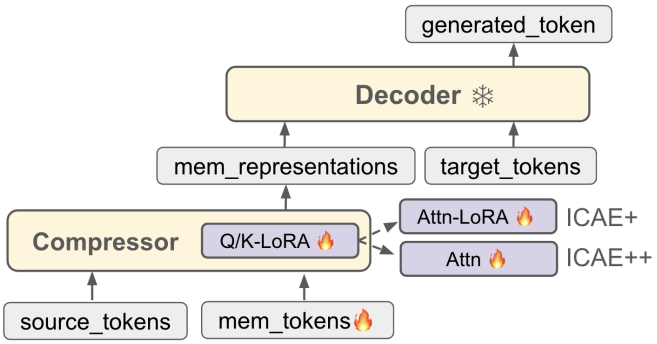
To understand the limitations of ICAE clearly, we study it at the larger scale of 7B Mistral model, longer sequence length of compressing $6k$ source tokens, and large compression ratio of $8\times$. Training follows the original setup in Ge et al. (2024) (except using only next token prediction loss), keeping LoRA rank as 32, and using similar training as our MemCom training procedure. Under the more challenging many-shot setting, we find that ICAE fails to perform meaningful compression, performing significantly lower than a simple baseline that fits fewer full shots within the same token budget. This suggests that the ICAE compressor, as originally proposed, is not powerful enough for compressing long many-shot sequences.

To verify the above hypothesis, we extend the LoRA parameters to value and post matrices as well (in addition to just query and key matrices as in original ICAE setup) in the attention module. We refer to this architecture as ICAE+. As seen in Figure 3b and Appendix C, the performance increases, confirming our hypothesis that long-context many-shot compression requires a stronger compressor. To test this even further, we make the entire attention module trainable in the compressor network of ICAE, rather than just adapting it via LoRA. We notice a significant performance boost as a result of this, with performance starting to reach close to the simple baseline. We call this modified ICAE architecture as ICAE++.

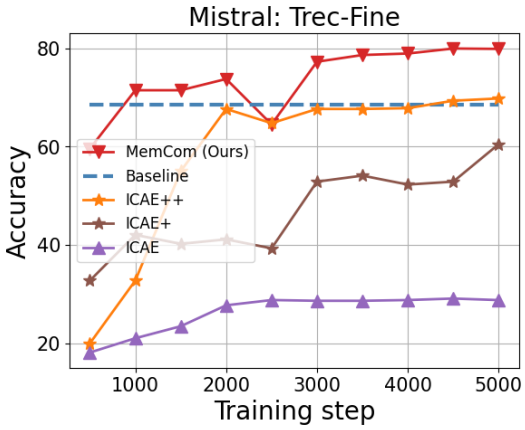
Next, we motivate our design choice of layer-wise compression, as opposed to using the final-layer output of a full forward pass as the compressed representation. In a vanilla ICL setting, the source tokens would be prepended to the target tokens, and the model would process the combined sequence through all layers. This means that, at each layer, the target tokens have access to rich and evolving representations of the source tokens.

In the compressed setting, however, the source tokens are removed and replaced with a compact representation. To preserve performance, the ideal scenario would be for the target tokens to receive representations that closely approximate the original source-token representations at each layer.

This motivates our approach in MemCom, where we perform layer-wise compression. Specifically, we introduce a single-head cross-



(a)



(b)

Figure 3: (a) Architecture of ICAE - The compressor is a copy of the decoder LLM, with LoRA parameters applied only to the Query and Key matrices. We extend this setup in ICAE+ by including LoRA parameters to value and post matrices as well. And we eventually extend to the entire attention module being trained, referred as ICAE++. (b) Performance improves as we move from *ICAE* $\rightarrow$ *ICAE+* $\rightarrow$ *ICAE++* $\rightarrow$ *MemCom*. The comparison is shown through performance on Trec-Fine across training steps, on Mistral-7B while compressing 6k source tokens by 8$\times$ into 768 memory tokens.

Table 2: MemCom performance on Mistral-7B across different compression ratios. The many-shots sequence length to compress is fixed at $6k$ tokens. The second column refers to the number of tokens m being attended to in each layer by the target tokens. For baseline, this means using many-shots comprising m tokens. For ICAE++ and MemCom, it refers to m memory tokens. $m = 2k,\ 1k,\ 768$ tokens represent $3\times,\ 6\times,$ and $8\times$ compression ratio respectively.

Task $\rightarrow$	$m \downarrow$	TREC-Coarse	TREC-Fine	HWU64	Banking77	Clinc-150
Baseline	$6k$	86.58	82.71	88.36	83.53	87.05
Baseline	$2k$	83.53	75.89	84.32	74.81	78.34
ICAE++		80.56	75.31	85.53	75.71	82.55
MemCom		83.33	80.03	85.27	79.61	**84.37**
MemCom-P2		**85.01**	**80.64**	**86.56**	**80.27**	83.72
Baseline	$1k$	82.20	70.12	80.16	59.22	72.75
ICAE++		78.55	72.40	83.01	71.51	77.15
MemCom		84.50	**79.12**	**83.58**	75.32	81.06
MemCom-P2		**86.20**	77.01	83.12	**75.39**	**81.95**
Baseline	786	82.88	68.60	72.50	54.09	69.81
ICAE++		78.01	69.70	83.18	69.71	77.25
MemCom		**83.31**	79.27	82.97	74.35	81.32
MemCom-P2		83.30	**79.80**	**83.89**	**75.42**	**82.38**

attention module in each layer of the compressor. This module allows the memory tokens to attend to the Source-LLM's representations at the corresponding layer and produce compressed outputs tailored for that layer in the Target-LLM. This fine-grained, layer-wise transfer of information enables the model to more faithfully approximate the behavior of the original many-shot attention. Note that inference-time latency of our layer-wise compression and ICAE's full forward pass compression remains the same - as the target tokens attend to m tokens in each layer in both the algorithms. Figure 3b demonstrates that transitioning from the ICAE++ architecture to our MemCom approach with layer-wise cross-attention, trained under the same Phase-1-only setting, leads to a clear performance improvement. This result supports our central hypothesis that fine-grained, layer-wise compression is substantially more effective for many-shot ICL. To further validate this hypothesis, we present additional experiments in Section 5.2.

5.2 Evaluations

For evaluation, we compare our method against two baselines: (1) the vanilla baseline of fitting fewer full shots within the compressed token budget, and (2) ICAE++, our stronger variant of ICAE. As a reference point, we also report the performance when using all the source tokens without any compression, which serves as a natural upper bound for the task.

We present results for Mistral-7B in Table 2 and Gemma2-2B in Table 3 . For the Clinc-150 dataset, we found that it was not possible to fit at least one example per class within the 3k-token source budget used for Gemma2-2B. Since the full source sequence itself would be incomplete, evaluating compression methods under this setting would not be meaningful. Therefore, we report Clinc-150 results only for Mistral-7B, where the 6k-token source budget is sufficient to include at least one example from all classes.

In both Table 2 and Table 3, the second column indicates the number of tokens m that the target tokens attend to at each layer. For the baseline, this corresponds to fitting as many full-shot examples as possible within m tokens. For ICAE++ and MemCom, it represents the number of memory tokens. MemCom-P2 refers to MemCom after Phase-2 training. We observe the following resuts:

First, we observe that moving from compression through a full-model forward pass, as in ICAE++, to layer-wise compression, as in MemCom after Phase-1 training, consistently leads to improved performance. Specifically, MemCom outperforms ICAE++ across all benchmarks on Mistral-7B when compressing 6k source tokens. On Gemma2-2B, the same trend holds across all tasks except HWU64. Investigating this reverse trend on HWU64 remains an open direction for future work.

Second, we observe that MemCom Phase-2 training generally leads to improved performance over Phase-1 training. This effect is more pronounced at lower compression ratios on Gemma2-2B. However, as we move to larger source sequence lengths (6k tokens) and stronger models like Mistral-7B, we find that Phase-1 training already provides the majority of the gains, with Phase-2 offering marginal additional improvements. This is encouraging, as it suggests that a lightweight compressor—comprising a simple 1-layer cross-attention module per layer—is sufficient to capture most of the benefits of many-shot compression at scale.

Third, we find that while the performance of the baseline and ICAE++ degrades sharply with increasing compression ratios, MemCom maintains high accuracy with minimal performance drop (Figure 2). This effect is particularly pronounced with stronger models like Mistral-7B and longer source sequences of 6k tokens. Notably, this robustness at larger sequence lengths highlights the practical utility of MemCom for real-world long-context ICL applications.

Ablation: Cross-attention module design choice. In Appendix D, we study the effect of varying the structure of the cross-attention module used in MemCom to mult-head attention (MHA) or multi-query attention (MQA). Our conclusion is using 1-head cross-attention provides the best performance overall, and we therefore adopt this configuration for all other experiments in the paper.

Table 3: MemCom performance on Gemma2-2B across different compression ratios. The many-shots sequence length to compress is fixed at $3k$ tokens. $m = 1k$, 512, 384 tokens represent $3\times$, $6\times$, and $8\times$ compression ratio respectively. MemCom-P2 refers to MemCom after Phase-2 training.

Task →	$m \downarrow$	TREC-Coarse	TREC-Fine	HWU64	Banking77
Baseline	$3k$	88.31	75.77	77.72	41.70
Baseline	$1k$	83.24	70.13	**65.02**	35.63
ICAE++		76.11	66.16	61.8	33.52
MemCom		82.01	72.12	46.34	30.20
MemCom-P2		**87.78**	**74.24**	53.75	**38.05**
Baseline	512	60.83	39.35	49.06	31.04
ICAE++		72.22	55.95	**52.81**	31.49
MemCom		76.22	70.80	47.61	41.69
MemCom-P2		**80.21**	**72.09**	49.28	**47.63**
Baseline	384	16.11	2.74	1.64	1.36
ICAE++		68.93	56.71	**53.05**	29.94
MemCom		**75.60**	68.34	35.31	39.55
MemCom-P2		**75.60**	**68.91**	36.17	**42.43**

6 Limitations and Future Work

We primarily evaluate MemCom on classification tasks with large output spaces (Table 1). While we conducted preliminary experiments on several generation tasks—including summarization (XSum (Narayan et al., 2018)), question answering (TriviaQA (Joshi et al., 2017)), sequential reasoning (Sequential Parity (Agarwal et al., 2024)), and arithmetic reasoning (GSM8K (Cobbe et al., 2021))—we observed limited differences between the baselines. Specifically, the performance of a baseline using fewer shots (to fit within the compressed token budget) was already comparable to a baseline using all shots without compression. In such cases, where the uncompressed baseline itself shows minimal sensitivity to the number of shots, it is unsurprising that compression does not yield substantial gains. We believe that extending our approach to even longer sequence lengths, beyond those explored here, could make many-shot compression more impactful for generation tasks as well. This is since the performance gap between using fewer and more shots is likely to widen with larger contexts.

Another limitation is MemCom's substantial training cost (80-160B tokens), necessary for building powerful compressors. This cost will escalate with longer sequences due to increased token processing and quadratic attention complexity. Thus, an important direction for future work is to explore strategies for making compressor training more efficient.

Third, many-shot compression may be unsuitable for tasks that require precise access to individual input tokens. For example, in low-resource language to English translation tasks (Agarwal et al., 2024), ICL operates by mapping specific tokens from the source language to their English counterparts. As the model needs to preserve fine-grained token information rather than relying on a compressed abstraction, in our preliminary experiments we noticed degraded performance on such tasks.

7 Conclusion

In this work, we studied the problem of compressing the information contained in many-shot prompts for In-Context Learning (ICL). We showed that existing prompt compression methods, when applied directly, fail to perform effectively in the many-shot setting. To address this, we proposed MemCom, a novel approach that employs a more powerful compressor and performs layer-wise compression to produce fine-grained representations. In contrast to prior methods that use the final embedding of a compressor model as a static summary, MemCom transfers compressed information at each layer, enabling more faithful reconstruction of the original context.

Through extensive experiments on classification tasks with large label spaces, we demonstrated that MemCom consistently outperforms strong baselines. Notably, at high 8x

compression ratios where baseline accuracy can plummet by over 70%, MemCom maintains robust performance with accuracy drops typically below 10%. This provides a promising path toward efficient utilization of many-shot ICL, without incurring the substantial memory and compute costs of the vanilla setting.

8 Acknowledgment

We would like to thank Peyman Milanfar, Arun Suggala, and Sanjiv Kumar for their valuable feedback on an earlier version of this paper.

References

Rishabh Agarwal, Avi Singh, Lei M. Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, John D. Co-Reyes, Eric Chu, Feryal Behbahani, Aleksandra Faust, and Hugo Larochelle. 2024. Many-shot in-context learning. *Preprint*, arXiv:2404.11018.

Mario Almeida, Stefanos Laskaridis, Stylianos I. Venieris, Ilias Leontiadis, and Nicholas D. Lane. 2022. Dyno: Dynamic onloading of deep neural networks from cloud to device. *ACM Transactions on Embedded Computing Systems*, 21(6):1–24.

Jihwan Bang, Juntae Lee, Kyuhong Shim, Seunghan Yang, and Simyung Chang. 2024. Crayon: Customized on-device llm via instant adapter blending and edge-server hybrid inference. *Preprint*, arXiv:2406.07007.

Amanda Bertsch, Maor Ivgi, Emily Xiao, Uri Alon, Jonathan Berant, Matthew R. Gormley, and Graham Neubig. 2025. In-context learning with long-context models: An in-depth exploration. *Preprint*, arXiv:2405.00200.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020a. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020b. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. 2020. Efficient intent detection with dual sentence encoders. *Preprint*, arXiv:2003.04807.

Yuxuan Chen, Rongpeng Li, Zhifeng Zhao, Chenghui Peng, Jianjun Wu, Ekram Hossain, and Honggang Zhang. 2024. Netgpt: A native-ai network architecture beyond provisioning personalized generative services. *Preprint*, arXiv:2307.06148.

Arthur Chevalier, Thomas Scialom, Antoine Dray, Laurent Romary, and Benoît Sagot. 2023. Adapting language models to compress contexts. *arXiv preprint arXiv:2310.14033*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

Ziyang Ge, Bowen Liu, Hao Zhang, Aston Chen, Zhiyuan Liu, and Maosong Sun. 2024. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2401.16109*.

Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Chin-Yew Lin, and Deepak Ravichandran. 2001. Toward semantics-based answer pinpointing. In *Proceedings of the First International Conference on Human Language Technology Research*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *Preprint*, arXiv:2310.06825.

Shaohan Jiang, Yu Liu, Ge Wang, Haifeng Li, Yi Wang, Xin Zhang, Nan Wang, Lin Zhang, Aston Chen, Zhiyuan Liu, and 1 others. 2023b. Llmlingua: Compressing prompts for accelerated inference of large language models. In *International Conference on Machine Learning*, pages 15143–15167. PMLR.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *Preprint*, arXiv:1705.03551.

Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. 2019. An evaluation dataset for intent classification and out-of-scope prediction. 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 1311–1316, Hong Kong, China. Association for Computational Linguistics.

Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, page 1–15. ACM.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Preprint*, arXiv:2404.14469.

Yutao Li, Ziyang Yao, Zhiqing Zhang, Tianyi Zhao, Chen Zhang, Aston Chen, Zhiyuan Liu, and Maosong Sun. 2023. In-context learning with many demonstration examples. *arXiv preprint arXiv:2310.18389*.

Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. 2019. Benchmarking natural language understanding services for building conversational agents. *Preprint*, arXiv:1903.05566.

Man Luo, Xin Xu, Yue Liu, Panupong Pasupat, and Mehran Kazemi. 2024. In-context learning with retrieved demonstrations for language models: A survey. *Preprint*, arXiv:2401.11624.

Yuchen Mu, Zhiqing Zhang, Aston Chen, Zhiyuan Liu, and Maosong Sun. 2023. Learning to compress prompts with gist tokens. *arXiv preprint arXiv:2310.15082*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *Preprint*, arXiv:1808.08745.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *Preprint*, arXiv:2406.17557.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Zihan Tan, Zhiqing Zhang, Aston Chen, Zhiyuan Liu, and Maosong Sun. 2024. Lloco: Learning long contexts offline. *arXiv preprint arXiv:2402.04696*.

Gemma Team. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Amin Vahdat and Mark Lohmeyer. 2023. Announcing cloud tpu v5e ga for cost-efficient ai model training and inference. Google Cloud Blog.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Zhenyu (Allen) Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *ArXiv*, abs/2306.14048.

A Experimental Setup

A.1 Training

For training MemCom and ICAE++, we use a mixture of FineWebEdu (Penedo et al., 2024) and SlimPajama (Soboleva et al., 2023) datasets. For Mistral training, we filter all sequences that are smaller than 7k tokens, and trim them to 8k sequence lengths (using padding if needed). During training, we randomly split each sequence between $[5.7k, 6.3k]$ tokens as source tokens, and the remaining as target tokens. We use m memory tokens, where $m \in \{2048, 1024, 768\}$ for Mistral-7B.

Similarly, for Gemma-2 2B we filter out all sequences less than 3.7k, and trim them to 4k sequence lengths. Source tokens are split in range $[2.7k, 3.4k]$ and the remaining are considered as target tokens. Number of memory tokens $m \in \{1024, 512, 384\}$

We used 512 TPUv5 accelerator (Vahdat and Lohmeyer, 2023) for all of our training runs. Phase-1 of MemCom training and ICAE++ took around 12/18 hours for Gemma/Mistral model, whereas Phase-1 of MemCom took 24/36 hours.

A.2 Hyperparameters

Batch size for Mistral is kept at 1024, and for Gemma at 2048. Weight decay is kept at 0 for all experiments. Learning rate (LR) is tuned in the range $\{2e-4,\ 1e-4,\ 5e-5,\ 2e-5,\ 1e-5,\ 5e-6,\ 1e-6,\ 8e-7\}$. We use the largest learning rate possible that does not lead to instability. For all the MemCom Phase-1 training, this happens to be $2e-4$. For MemCom Phase-2 training, this was $2e-6$, except $8\times$ compression ratio on Mistral where this was $8e-7$. For ICAE++, as mentioned in Section 5.1, training with Autoencoder loss led to unstable training at higher learning rate, and was relatively stable at $5e-6$. We show the next token prediction train loss plots across training steps in Figure 4a for ICAE++ when AE loss was enabled as well during training. Without AE loss, and only using next token prediction loss, ICAE++ training was stable with LR $= 2e-4$. A warmup of 1.5k steps was used for ICAE++, 0.5k for MemCom Phase-1, and 1.5k for MemCom Phase-2.

A.3 Downstream Tasks

We use 5 downstream classification tasks with large label space. These are: TREC-Coarse and TREC-fine (Li and Roth, 2002; Hovy et al., 2001), HWU64 (Liu et al., 2019), Banking77 (Casanueva et al., 2020), Clinc150 (Larson et al., 2019). More details are provided in Table 1.

For each dataset, the construction of few-shot input prompts, constrained to t tokens, involved a round-robin sampling strategy ensuring class balance. We iteratively selected one random shot per class. This process was repeated until the token budget was nearly filled. In the concluding iteration, to precisely meet the t-token limit, if a selected shot exceeded the budget, it was dropped and the loop to add shots was ended. Thus, we few-shot prompt construction followed a class-balanced sampling procedure.

We first created a unified candidate pool by merging its original train and test splits. From this pool, 20 distinct instances per class were randomly selected to serve as our evaluation queries. For each selected query, a many-shot demonstration prompt of up to t tokens was constructed using the method detailed above.

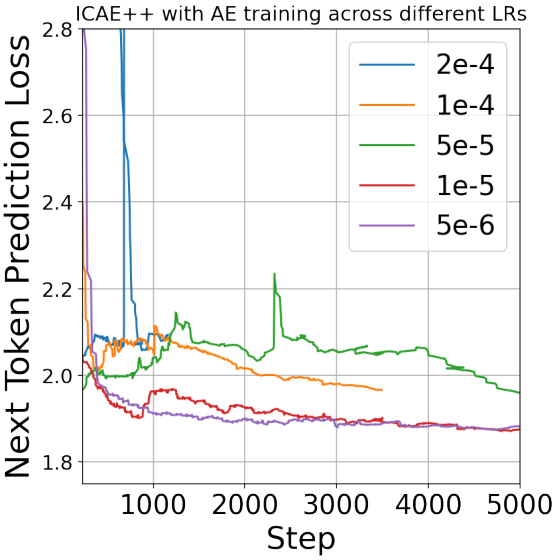
B Dataset and Model Licenses

All the datasets and models used are publicly available and were utilized in accordance with their respective licenses, all of which permit research use. Specifically, Banking77, TREC, and HWU64 is available under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. The Clinc150 dataset is available under Creative Commons Attribution 3.0 International (CC BY 3.0) license. FineWebEdu is under Open Data Commons Attribution License (ODC-By) v1.0 license and under CommonCrawl's Terms of Use, and SlimPajama is under Apache 2.0 license.

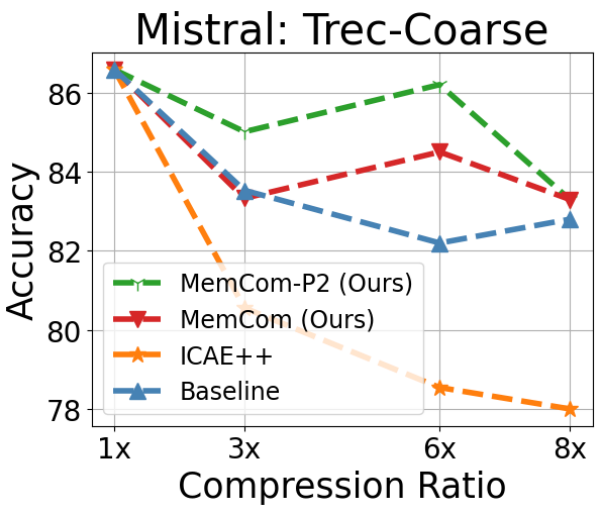
Mistral has Apache license 2.0 and Gemma has Gemma Terms of Use as its license, both permitting use of the models for research purposes.

C Baseline Study and Motivating Insights

As mentioned in Section 5.1, we study ICAE and gradually change its architecture to en-



(a) ICAE++ training curve under different learning rates, when using both next token prediction loss and autoencoder loss during training. We see that the largest learning rate having stable training is $5e-6$.



(b) Comparison of compression performance across different compression ratios on Mistral model for Trec-Coarse benchmark. We show the remaining plots in Figure 2

Figure 4

able more powerful compressor. Due to resource constraints, we do our study on just the most relevant setting - using the larger model Mistral-7B, longest source length sequence of $6k$ tokens, and highest compression ratio of $8\times$. As we see in Figure 3b, moving from *ICAE* $\rightarrow$ *ICAE+* $\rightarrow$ *ICAE++* $\rightarrow$ *MemCom* results in improved performance. We also show this improvement across other benchmarks in Table 4.

Moreover, as mentioned in Section A.2, ICAE++ with Autoencoder loss performs worse compared to without Autoencoder loss. We show this performance difference in Table 5.

D Ablation

Cross-attention module design choice. We study the effect of varying the structure of the cross-attention module used in MemCom. Specifically, we compare using multi-head attention (MHA), multi-query attention (MQA), and our default 1-head attention, with all variants trained from scratch. Since Mistral employs MQA in its self-attention layers, another plausible design choice is to initialize the MQA cross-attention with the same parameters as the model's MQA self-attention. However, we find that this variant also doesn't outperforms 1-head cross-attention overall.

Based on the results shown in Table 6, we conclude that using 1-head cross-attention, initialized from scratch, provides the best performance overall. We therefore adopt this configuration for all other experiments in the main paper.

Task →	TREC-Coarse	TREC-Fine	HWU64	Banking77	Clinc-150
Baseline-6k	86.58	82.71	88.36	83.53	87.05
Baseline-768	82.88	68.60	72.50	54.09	69.81
ICAE	51.67	28.81	32.34	20.97	34.46
ICAE+	71.11	56.10	54.43	34.03	44.74
ICAE++	78.01	69.70	83.18	69.71	77.25
MemCom	83.31	79.27	82.97	74.35	81.32

Table 4: Baseline-6k refers to the baseline using all 6k source tokens worth of shots. Baseline-768 is the 8× compression comparable baseline, which uses 8× less tokens worth of shots. Performance increases as we move from *ICAE* → *ICAE+* → *ICAE++* → *MemCom*. This is because of having a strong model (*ICAE* → *ICAE+* → *ICAE++*) and more fine grained information transfer through layer-wise compression (*ICAE++* → *MemCom*).

Task →	TREC-Coarse	TREC-Fine	HWU64	Banking77	Clinc-150
Baseline-6k	86.58	82.71	88.36	83.53	87.05
Baseline-768	82.88	68.60	72.50	54.09	69.81
ICAE++ with AE	58.33	32.62	17.73	22.60	27.75
ICAE++	78.01	69.70	83.18	69.71	77.25

Table 5: Baseline-6k refers to the baseline using all 6k source tokens worth of shots. Baseline-768 is the 8× compression comparable baseline, which uses 8× less tokens worth of shots. ICAE++ with AE refers to the ICAE++ architecture trained with Autoencoder loss in addition to next token prediction loss. We observe that ICAE++ performs better without Autoencoder loss.

Table 6: MemCom comparison on Mistral-7B with different cross-attention modules, on 8x compression ratio. Each experiments is trained on Phase-1 training only. The Baseline performance in the second row is using all 6k source tokens. MQA* refers to MQA initialized with same parameters as the model's self-attention module.

Task →	TREC-Coarse	TREC-Fine	HWU64	Banking77	Clinc-150
Baseline	86.58	82.71	88.36	83.53	87.05
1-head attention	**83.31**	**79.27**	82.97	74.35	81.32
MHA	75.11	69.36	**85.23**	74.82	83.41
MQA	72.23	62.04	83.64	**75.36**	83.22
MQA*	77.78	69.82	71.41	62.99	**83.38**