Efficient Many-Shot In-Context Learning with Task-Level Compression and Instance-Level Retrieval

Masayuki Kawarada **Ukyo Honda** **Syo Takase**
CyberAgent, Japan
{kawarada_masayuki,honda_ukyo,takase_sho}@cyberagent.co.jp

Abstract

Many-shot in-context learning (ICL) lets large language models exploit hundreds or thousands of demonstrations, but its cost grows with the prompt. We argue that many-shot demonstrations carry two kinds of information: *task-level* information, such as task structure, which emerges only across many examples, and *instance-level* information, such as exact input-output associations and rare exceptions, which resides in individual demonstrations. Existing efficiency methods commit all information to a single placement: demonstration retrieval keeps a few demonstrations as raw text, while context compression places the full pool into a compact memory. We study where each kind of information should be placed. On ManyICLBench, each one-sided method falls short on tasks that depend on the information it discards, while combining the two retains much of the full many-shot performance at a much smaller inference budget. Synthetic tasks further show that compressed memory and retrieved examples can complement each other but sometimes interfere.

1 Introduction

Large language models (LLMs) adapt to new tasks through In-Context Learning (ICL; Brown et al., 2020) by conditioning on demonstrations in the prompt. With longer context windows, many-shot ICL with hundreds or thousands of demonstrations provides richer information about the target task (Agarwal et al., 2024), but its cost grows with the prompt: longer demonstration prefixes increase prefill computation and enlarge the key-value (KV) cache (Bertsch et al., 2025).

We argue that many-shot demonstrations carry two kinds of information. *Task-level* information, such as task structure and label semantics, emerges only across many examples, while *instance-level* information, such as exact input-output associations and rare exceptions, resides in individual demonstrations. Existing efficiency methods preserve only one side (Figure 1): demonstration retrieval (Liu et al., 2022) keeps a few query-relevant demonstrations as raw text but loses global structure, while context compression places the full pool into a latent representation (Nawrot et al., 2024; Chevalier et al., 2023) or a textual summary (Honda et al., 2025) but weakens rare local evidence.

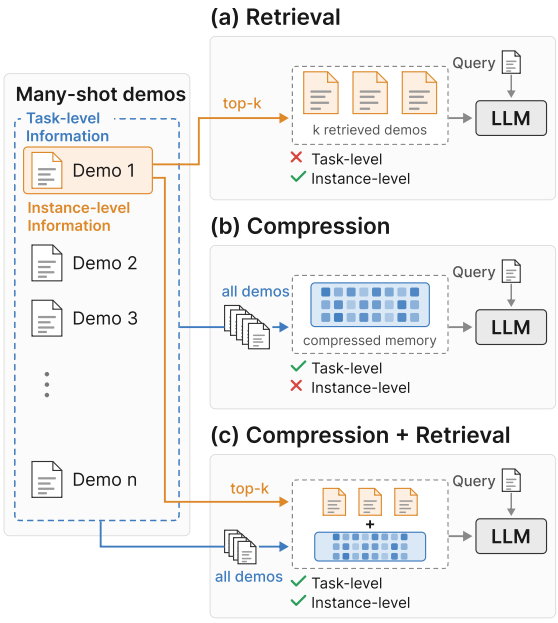


Figure 1: Three ways to place many-shot demonstrations under a limited budget. (a) Retrieval keeps a few demonstrations verbatim, preserving instance-level information (orange) but only part of the task-level information (blue); (b) compression covers all demonstrations but loses instance-level details first; (c) combining them aims to keep both.

In this work, we study *where* each kind of information should be placed when the full pool cannot be kept as raw text. We hypothesize that task-level information, being more robust to compression, belongs in compressed memory, whereas instance-level information must remain in raw text. On ManyICLBench (Zou et al., 2025), we compare

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 3547–3555
November 6-10, 2026 ©2026 Association for Computational Linguistics

compression only, retrieval only, and their combination (task-level compression with instance-level retrieval), all built from the same demonstration pool. Each one-sided configuration falls short on tasks that depend on the information it discards, while the combination retains much of the full many-shot performance at lower latency.

We further construct two synthetic tasks, each isolating one kind of information. Global Rule Induction requires inferring a rule spread across many demonstrations, while Local-Exception Handling requires recalling sparse overrides of per-key defaults. Neither one-sided method matches full many-shot prompting on both tasks: retrieval only falls short on the former and compression only fails on the latter, while the combination performs on par with it on both.

Our contributions are: (1) we distinguish task-level and instance-level information in many-shot demonstrations and study where each should be placed; (2) on ManyICLBench, we show that combining compressed memory with a few retrieved demonstrations preserves much of the many-shot performance at lower latency, especially on similar-sample learning tasks; and (3) with synthetic tasks, we show that compression loses exceptions first, and that even retrieved exceptions can be overridden by default labels.

2 Placing Task- and Instance-Level Information

We now formalize how each existing efficiency method relates to the two information types introduced in Section 1. Given a demonstration pool $D = \{(x_i, y_i)\}_{i=1}^{n}$ and a query x, standard many-shot ICL conditions on

$$p_\theta(y \mid \mathrm{Text}(D), x). \tag{1}$$

Retrieval is exact but local. Demonstration retrieval (Liu et al., 2022) keeps a query-dependent subset $S_k(x, D) \subset D$ as raw text:

$$p_\theta(y \mid \mathrm{Text}(S_k(x, D)), x). \tag{2}$$

The selected demonstrations enter verbatim, so the exact content of those k examples, including any rare exception retrieved, is preserved at full fidelity. The remaining $n - k$ demonstrations are dropped, so task-level regularities distributed across the full pool must be inferred from only k examples and are recovered at most partially.

Compression is global but lossy. Context compression methods replace D with a compact memory $M(D)$, either a textual summary $M_{\text{text}}(D)$ or a compressed KV representation $M_{\text{lat}}(D)$:

$$p_\theta(y \mid M(D), x). \tag{3}$$

Here $M(D)$ does not keep any single demonstration as-is; it is a coarsened representation derived from all of D. Because task-level information is expressed redundantly across many demonstrations, we expect it to degrade more gracefully under such lossy aggregation than instance-level information, which is concentrated in a single demonstration and is therefore likely to be lost first.

Combining the two placements. Since the two placements lose different information first, combining them may retain more of both. We therefore also consider a compressed memory of the full pool with a few retrieved demonstrations as raw text:

$$p_\theta(y \mid M(D), \mathrm{Text}(S_k(x, D)), x). \tag{4}$$

The memory carries task-level information aggregated from all n demonstrations, while the retrieved demonstrations restore the instance-level evidence most relevant to the query.

3 Experimental Setup

3.1 Models

We use Qwen3-14B (Yang et al., 2025) and Gemma3-27B (Team, 2025). Both models are run in bfloat16 with greedy decoding. For Qwen3-14B, we disable the thinking mode so that the model directly outputs the final answer.

3.2 Demonstration Pool and Budgets

For each seed and context length $L \in \{1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}, 16\text{k}\}$ tokens, we sample a demonstration pool D that fills L tokens as raw text. All methods start from the same pool D and the same test queries, and differ only in how the information in D is placed in the context. Unless otherwise noted, we report scores averaged over the five context lengths.

3.3 Latency

We measure wall-clock latency per query on a single NVIDIA A100 GPU. Operations performed once per pool, namely prefilling and compressing the demonstration prefix, are amortized over the test queries that share the pool; query-dependent

	SSL					ASL					
	BANKING77	**CLINC150**	**TREC50**	**DialogRE**	**BBH-geometric**	**BBH-salient**	**BBH-languages**	**BBH-word**	**GSM8K**	**ARC-Challenge**	**XLSUM**
Qwen3-14B											
Many-shot	0.788	0.862	0.505	0.168	0.599	0.591	0.126	**0.336**	**0.929**	0.924	**0.221**
Retrieve	0.796	0.869	0.501	0.168	**0.602**	0.583	0.165	0.309	0.909	0.922	0.218
Summ.	0.652	0.756	0.442	0.168	0.376	0.578	0.082	0.250	0.706	0.713	0.202
Latent	0.748	0.868	0.503	0.161	0.489	0.579	0.126	0.329	0.925	**0.929**	0.202
Summ. + Retrieve	0.753	0.816	0.530	**0.190**	0.547	0.589	0.099	0.306	0.745	0.772	0.220
Latent + Retrieve	**0.803**	**0.885**	**0.556**	0.176	0.551	**0.605**	**0.188**	0.314	0.918	0.925	0.206
Gemma3-27B											
Many-shot	0.780	0.900	0.547	0.354	0.572	0.644	0.270	**0.552**	0.905	0.904	**0.257**
Retrieve	0.799	0.890	0.546	0.345	0.542	0.610	0.180	0.382	0.892	0.903	0.244
Summ.	0.740	0.861	0.528	0.331	0.450	0.614	0.238	0.526	0.701	**0.913**	0.210
Latent	0.772	0.848	0.526	0.304	0.446	**0.658**	0.278	0.542	**0.934**	0.898	0.221
Summ. + Retrieve	0.816	0.902	0.574	**0.364**	0.554	0.612	**0.302**	0.412	0.769	0.878	0.238
Latent + Retrieve	**0.835**	**0.905**	**0.620**	0.360	**0.576**	0.606	0.284	0.536	0.925	0.873	**0.257**

Table 1: Results on ManyICLBench using Qwen3-14B and Gemma3-27B. SSL (similar-sample learning) tasks rely on query-relevant demonstrations, whereas ASL (all-sample learning) tasks rely on information aggregated across the pool. We average scores over context lengths of 1k, 2k, 4k, 8k, and 16k tokens. We use $k = 8$ and $c = 0.75$ where applicable. Scores are reported on a 0–1 scale.

prefill, retrieval, and decoding are measured per query. Summary generation for Summ. is performed offline once per pool and is excluded from the reported latency.

3.4 Methods

We instantiate the placements in Section 2 as six methods. **Many-shot** places all of D as raw text. Since every other method sees either a subset or a compressed form of the same D, we treat Many-shot as a reference for how much of the pool's information is available. **Retrieve** keeps the top-k demonstrations ranked by BM25. **Summ.** replaces D with a summary generated once per pool by the evaluated model (Honda et al., 2025). **Latent** compresses the KV cache of D with ExpectedAttentionPress (Devoto et al., 2025) at compression ratio c (the fraction of KV entries removed), leaving the system prompt and query uncompressed. **Summ.+Retrieve** and **Latent+Retrieve** append the k retrieved demonstrations as raw text to the same memory. We use $k = 8$ and $c = 0.75$. See Appendix B.3 for details.

4 Experiments on Real Tasks

4.1 Task and Evaluation

We evaluate on ManyICLBench (Zou et al., 2025), which groups many-shot ICL tasks into similar-sample learning (SSL), where query-relevant demonstrations matter, and all-sample learning (ASL), where information aggregated across the pool matters. Following the benchmark setup, we use eleven tasks. The SSL tasks are BANKING77 (Casanueva et al., 2020) and CLINC150 (Larson et al., 2019) for intent classification, TREC50 (Li and Roth, 2002) for question classification, DialogRE (Yu et al., 2020) for relation extraction, and BBH-geometric (Suzgun et al., 2023) for geometric reasoning. The ASL tasks are BBH-salient, BBH-languages, and BBH-word from BIG-Bench Hard (Suzgun et al., 2023), GSM8K (Cobbe et al., 2021) for math reasoning, ARC-Challenge (Clark et al., 2018) for science question answering, and XLSUM (Hasan et al., 2021) for summarization. We report macro-F1 for classification and relation extraction, ROUGE-L for XLSUM, and accuracy otherwise (Appendix B).

4.2 Main Results

Table 1 shows the results. Latent alone stays close to Many-shot on some tasks, such as GSM8K and ARC-Challenge, but drops on others, such as BBH-geometric (0.489 vs. 0.599 for Qwen3-14B; 0.446 vs. 0.572 for Gemma3-27B) and XLSUM. This suggests that compressed latent memory can retain task-level information while exact local evidence is less reliably preserved. On SSL tasks, adding retrieved demonstrations to the latent memory improves over Latent on all five tasks for both models. Latent+Retrieve outperforms Many-shot on BANKING77, CLINC150, TREC50, and DialogRE for both models, and achieves the best score among all methods on three of the five SSL tasks for Qwen3-14B and four for Gemma3-27B. On ASL tasks, the picture is

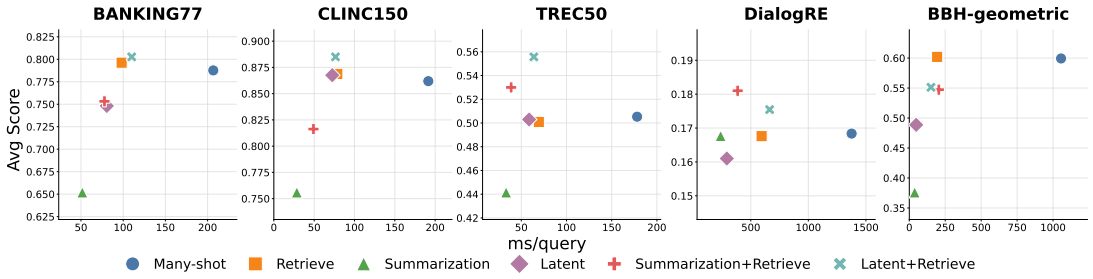


Figure 2: Performance–latency trade-off on SSL tasks in ManyICLBench using Qwen3-14B.

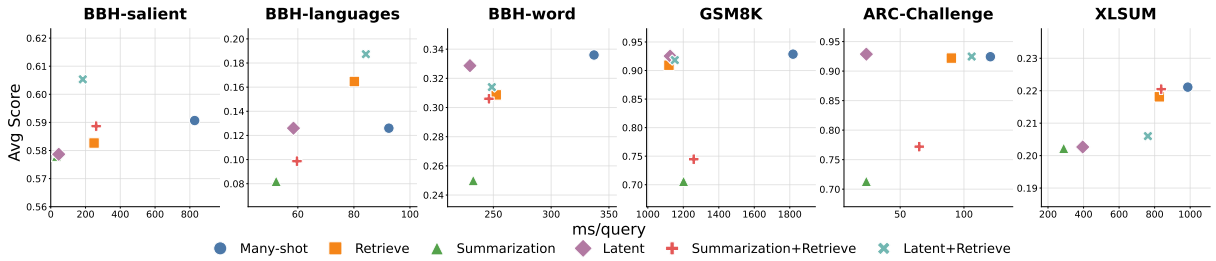


Figure 3: Performance–latency trade-off on ASL tasks in ManyICLBench using Qwen3-14B.

mixed. Latent+Retrieve improves over Latent on BBH-languages and XLSUM for both models, but Many-shot remains the best on BBH-word for both models, and for Gemma3-27B, Latent alone is the best on BBH-salient and GSM8K. When a task relies on information aggregated across the pool, a few retrieved demonstrations add little and can even distract from the signal already carried by the full context or its compressed memory.

4.3 Performance–Budget Trade-offs

Figures 2 and 3 show the performance–latency trade-off on SSL and ASL tasks, respectively. Summarization is fast but often loses accuracy, while full Many-shot prompting is strong but costly. On SSL tasks, Latent+Retrieve achieves a favorable trade-off, often matching or exceeding Many-shot with lower latency. On ASL tasks, the trend is more mixed: Latent+Retrieve improves some tasks, but full Many-shot or Latent remains stronger when globally aggregated information or formatting cues are sensitive to compression.

5 Synthetic Diagnostic Tasks

The ManyICLBench results suggest that compressed memory and retrieved demonstrations provide complementary benefits. However, real tasks mix many factors, including task format, label semantics, reasoning difficulty, and retrieval quality. To isolate the source of these gains, we construct two synthetic diagnostic tasks that separately test task-level and instance-level information.

5.1 Task Design

Global Rule Induction. This task tests whether a method can infer task-level structure distributed across many demonstrations. Internally, each example is generated from a six-attribute record

$$\mathbf{r}_i = (a_i, b_i, c_i, d_i, e_i, m_i),$$

and its label is produced by a hidden seed-specific rule,

$$\ell_i = f_s(\mathbf{r}_i).$$

The model observes only serialized input-output demonstrations. Test queries do not repeat records from the demonstration pool, so success requires inferring the global rule rather than copying a local match. The full data-generation rule is described in Appendix A.1.

Local-Exception Handling. This task tests whether a method can recall sparse exceptions that override dense key-specific evidence. Demonstrations define a base assignment function $f_{\text{base}}(k)$ and sparse override assignments $f_{\text{override}}(k)$. The prediction rule is as follows:

$$y = \begin{cases} f_{\text{override}}(k) & \text{if an override for } k \text{ is present,} \\ f_{\text{base}}(k) & \text{otherwise.} \end{cases}$$

Thus, *default queries* require a key's base entry, while *exception queries* require its sparse override, which takes priority. The full data-generation procedure is described in Appendix A.2.

	Method	
Global rule	Many-shot	**0.572**
	Retrieve	0.440
	Summ.	0.335
	Latent	0.391
	Summ.+Retrieve	0.475
	Latent+Retrieve	0.566
Local exception	Many-shot	0.841
	Retrieve	**1.000**
	Summ.	0.345
	Latent	0.158
	Summ.+Retrieve	0.610
	Latent+Retrieve	0.871

Table 2: The synthetic diagnostic results using Qwen3-14B. We use $k = 8$ and $c = 0.75$ where applicable.

5.2 Evaluation

We evaluate each task with three random seeds under the context budgets in Section 3, with 256 test queries per seed-budget setting. Accuracy is measured by exact match against the gold label.

5.3 Results

Table 2 shows the results. On Global Rule Induction, Retrieve is substantially worse than Many-shot, confirming that a small set of local demonstrations is insufficient to recover structure distributed across the full pool. Latent alone also drops (0.391), but it retains about two-thirds of Many-shot accuracy, far more than on Local-Exception Handling (less than a fifth). Latent+Retrieve nearly matches Many-shot, improving over Retrieve by 0.126; since the two differ only in the compressed memory, this gain reflects task-level information carried by the memory. On Local-Exception Handling, Retrieve achieves the best score, showing that explicit local evidence is crucial for rare exceptions. In contrast, Latent performs poorly, indicating that compressed memory can lose sparse instance-level information. Latent+Retrieve largely recovers this loss by restoring exception evidence as raw text.

Overall, the synthetic results support our hypothesis: task-level information benefits from access to the full demonstration pool, while instance-level information needs to remain explicit. Combining compressed memory with retrieved demonstrations performs on par with Many-shot on both tasks, whereas neither one-sided method does.

5.4 Effect of compression

Figure 4 varies the compression ratio c with $k = 8$ and splits Local-Exception Handling into default and exception queries. For Latent alone, exception

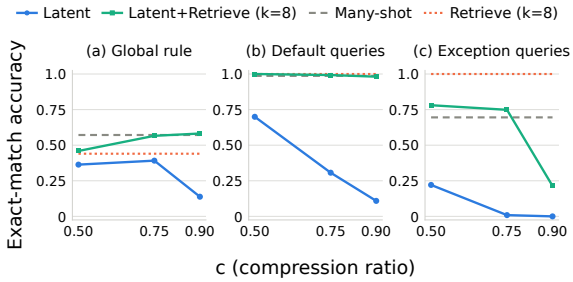


Figure 4: Accuracy under increasing compression ratio c on the synthetic tasks. We report Global Rule Induction in (a), and split Local-Exception Handling into default queries (b) and exception queries (c). Many-shot and Retrieve do not depend on c and are shown as horizontal lines.

accuracy collapses already at $c = 0.75$ (0.221 $\rightarrow$ 0.009), while Global Rule Induction remains stable (0.364 $\rightarrow$ 0.391) and degrades only at $c = 0.90$, confirming that compression removes instance-level information first. Adding retrieval restores default queries ($\geq$0.98), but exception accuracy still falls from 0.781 to 0.214 as c increases, far below Retrieve alone (1.000). This is not a retrieval failure: the correct override is always retrieved, and 89.6–100% of errors return the key's default value. The same errors dominate whenever default-supporting context grows, as in Retrieve with $k = 16$ (0.812) and Many-shot (0.695), suggesting that retrieved exceptions compete with the prevalence of base labels, a task-level prior that compression may strengthen by discarding overrides first.

6 Conclusion

We studied where task-level and instance-level information in many-shot demonstrations should be placed under a limited inference budget. On ManyICLBench, each one-sided placement fell short on tasks that depend on the information it discards, while combining a compressed KV memory of the full pool with a few retrieved demonstrations as raw text retained much of the many-shot performance at lower latency, particularly on similar-sample learning tasks. Synthetic tasks further showed that compression loses exceptions first, and that even retrieved exceptions can be overridden by default labels. Future work could adaptively balance compressed memory and raw text.

Limitations

This study has several limitations. First, our evaluation is based on ManyICLBench, which covers

a range of classification, reasoning, question answering, relation extraction, and summarization tasks, but does not cover all possible many-shot ICL settings. More open-ended generation, tool-use, domain-specific, and multilingual scenarios may require different allocations of task-level and instance-level information.

Second, the combination method depends on both compression quality and retrieval quality. If compression removes task-relevant global structure, or if retrieval selects irrelevant or distracting demonstrations, the combination may provide limited gains.

Third, we do not fully explore adaptive allocation strategies. In this work, the number of retrieved demonstrations and the compression configuration are fixed in most settings. Future work could dynamically choose how much information to store in compressed memory and how much to preserve as raw retrieved demonstrations.

Finally, our synthetic diagnostic tasks are intentionally simplified to isolate task-level and instance-level information. While they support our interpretation, real tasks often combine global structure, local exceptions, formatting constraints, and reasoning difficulty in more complex ways.

References

Rishabh Agarwal, Avi Singh, Lei 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. In *Advances in Neural Information Processing Systems*, volume 37, pages 76930–76966. Curran Associates, Inc.

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. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 12119–12149, Albuquerque, New Mexico. Association for Computational Linguistics.

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. 2020. 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. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 38–45, Online. Association for Computational Linguistics.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846, Singapore. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

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:2110.14168v2.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. 2025. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*.

Tahmid Hasan, Abhik Bhattacharjee, Md. Saiful Islam, Kazi Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. 2021. XL-sum: Large-scale multilingual abstractive summarization for 44 languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4693–4703, Online. Association for Computational Linguistics.

Ukyo Honda, Soichiro Murakami, and Peinan Zhang. 2025. Distilling many-shot in-context learning into a cheat sheet. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 17158–17178, Suzhou, China. Association for Computational Linguistics.

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.

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

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. 2024. Dynamic memory compression: retrofitting llms for accelerated inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Gemma Team. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Dian Yu, Kai Sun, Claire Cardie, and Dong Yu. 2020. Dialogue-based relation extraction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4927–4940, Online. Association for Computational Linguistics.

Kaijian Zou, Muhammad Khalifa, and Lu Wang. 2025. On many-shot in-context learning for long-context evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25605–25639, Vienna, Austria. Association for Computational Linguistics.

A Synthetic Diagnostic Datasets Details

A.1 Global Rule Generation

The global rule induction task is generated from a structured record

$$\mathbf{r} = (a, b, c, d, e, m).$$

The model does not observe the rule directly. It only observes serialized input-output pairs (x, y), where $x = \mathrm{Text}(\mathbf{r})$ and $y = \mathrm{Text}(\ell)$.

For each seed s, the record is mapped to one of four labels through two hidden bits. First, attribute a determines the first bit:

$$\beta_0 = g_a(a),$$

where g_a assigns each value of a to a hidden group in $\{0, 1\}$.

The second bit is determined by two trigger conditions:

$$t_{bc} = \mathbb{1}[b \in B_s^\star \wedge c \in C_s^\star],$$

$$t_{dem} = \mathbb{1}[d = d_s^\star \wedge g_e(e) = 1 \wedge m \equiv 1 \pmod 2].$$

Then,

$$\beta_1 = t_{bc} \vee t_{dem}.$$

The two bits define a code

$$q = \beta_0 + 2\beta_1,$$

which is mapped to a surface label by a seed-specific label mapping:

$$\ell = L_s(q), \qquad q \in \{0, 1, 2, 3\}.$$

The code assignment is summarized as follows:

β_1	β_0	q
0	0	0
0	1	1
1	0	2
1	1	3

An example serialized prompt is shown below:

```
Input: item a=a01; b=b3; c=cC; d=d0; e=e04; m
=3.
Label: GB9

Input: item a=a07; b=b2; c=cD; d=d1; e=e01; m
=0.
Label: GU9

Input: item a=a06; b=b0; c=cA; d=d0; e=e05; m
=8.
Label: GU9

Input: item a=a04; b=b1; c=cB; d=d0; e=e02; m
=4.
Label:
```

A.2 Local-Exception Generation

For the nominal context settings of 1k, 2k, 4k, 8k, and 16k, pools contain $50, 99, 198, 396$, and 792 distinct opaque keys, respectively. For each seed, $f_{\mathrm{base}}(k)$ assigns each key pseudorandomly to one

of four default labels. Each base assignment appears once; keys have no observable group structure.

We designate $5, 11, 22, 44,$ and 88 keys as exceptions, respectively (10% of keys at 1k and approximately 11.11% otherwise). Each exception key receives one additional override demonstration, with a value from four override labels disjoint from the default labels. Overrides appear 8–32 base assignments after their corresponding base entries and take priority over them.

For each seed and context setting, we evaluate 256 queries, equally split between default and exception queries. All queried keys occur in the pool; keys are reused when necessary. Targets are $f_{\text{base}}(k)$ and $f_{\text{override}}(k)$, respectively. We use seeds 0, 1, and 2.

An abridged example follows; ellipses indicate omitted demonstrations. The correct answer is `LEO-NEXA-997-135`.

```
Input: base assignment key=KEY-D2873B5BF0
Label: LEB-EMBER-00-418

...

Input: override assignment key=KEY-D2873B5BF0
Label: LEO-NEXA-997-135

...

Input: key=KEY-D2873B5BF0
Label:
```

B ManyICLBench Details

B.1 Task Groups, Metrics, and Evaluation Sizes

ManyICLBench categorizes tasks into two broad groups: similar-sample learning (SSL) and all-sample learning (ASL). SSL tasks are designed so that query-relevant demonstrations are especially important, while ASL tasks require using information distributed across the demonstration pool. This distinction is useful for our study because it roughly separates settings where retrieved demonstrations should help from settings where broader many-shot context may be needed.

Table 3 summarizes the tasks used in our real-task experiments. For classification and relation-extraction tasks, we report macro-F1. For BBH tasks, GSM8K, and ARC-Challenge, we report accuracy. For XLSUM, we report ROUGE-L.

B.2 Prompting and Parsing Details

We follow the ManyICLBench evaluation protocol whenever available and keep the task formatting fixed across allocation methods. For classification-style tasks, the model is instructed to output only the final label or intent. For BANKING77 and TREC_50, we include the allowed label set in the query-side prompt to reduce invalid outputs. For BBH-LANGUAGES, we use a compact backtick-delimited format and strip backticks during answer parsing.

All methods are evaluated with the same test examples, random seeds, context-length budgets, decoding settings, and task-specific parsers.

B.3 Retrieval and Compression Details

We implement KV-cache compression using `ExpectedAttentionPress` from the KVPress library (Devoto et al., 2025). We evaluate compression ratios $c \in \{0.50, 0.75, 0.90\}$, corresponding to retained KV ratios of $1 - c$. We divide each prompt into the system prompt, the demonstration prefix, retrieved demonstrations, and the query. We apply KV-cache compression only to the demonstration prefix, while leaving the system prompt and query uncompressed. For Latent+Retrieve, we insert the retrieved demonstrations as additional raw text without compression. Thus, Latent and Latent+Retrieve share the same compressed representation of the demonstration prefix, and differ only in whether they additionally expose query-specific demonstrations as raw text.

Task	Group	Task Type	Metric	Eval. Size
BANKING77	SSL	intent classification	macro-F1	600
CLINC150	SSL	intent classification	macro-F1	600
TREC50	SSL	question classification	macro-F1	600
DialogRE	SSL	relation extraction	macro-F1	600
BBH-geometric	SSL	geometric reasoning	accuracy	300
BBH-salient	ASL	translation error detection	accuracy	300
BBH-languages	ASL	formal-language reasoning	accuracy	300
BBH-word	ASL	algorithmic reasoning	accuracy	300
GSM8K	ASL	math reasoning	accuracy	600
ARC-Challenge	ASL	science question answering	accuracy	600
XLSUM	ASL	summarization	ROUGE-L	600

Table 3: ManyICLBench task details used in our experiments. Eval. Size denotes the typical number of evaluated examples per task and context-length cell across three seeds.