MEMORY-VQ: Compression for Tractable Internet-Scale Memory

Yury Zemlyanskiy[*†], **Michiel de Jong**[*†], **Luke Vilnis**
Santiago Ontañón, **William W. Cohen**, **Sumit Sanghai**, **Joshua Ainslie**

Google Research

Abstract

Retrieval augmentation is a powerful but expensive method to make language models more knowledgeable about the world. Memory-based methods like LUMEN (de Jong et al., 2023a) pre-compute token representations for retrieved passages to drastically speed up inference. However, memory also leads to much greater storage requirements from storing pre-computed representations.

We propose MEMORY-VQ, a new method to reduce storage requirements of memory-augmented models without sacrificing performance. Our method uses a vector quantization variational autoencoder (VQ-VAE) to compress token representations. We apply MEMORY-VQ to the LUMEN model to obtain LUMEN-VQ, a memory model that achieves a 16x compression rate with comparable performance on the KILT benchmark. LUMEN-VQ enables practical retrieval augmentation even for extremely large retrieval corpora.

	FiD	**LUMEN**	**L-VQ**
		Inference cost in TFLOPs	
Per sample	28.0	12.5	12.5
		Storage cost	
Per token	2 bytes	8 KB	0.5 KB
For Wikipedia	8 GB	30 TB	2 TB
For 1T tokens	2 TB	7 PB	0.5 PB
		KILT valid in % exact match	
Average	72.80	72.66	72.42
NaturalQuestions	61.47	62.64	62.74
TriviaQA	83.40	82.84	82.61
FEVER	93.47	92.77	92.18
TREX	83.58	83.78	83.42
ZeroShot RE	72.77	72.85	72.61
HotpotQA	42.09	41.09	41.00

Table 1: **Main results: LUMEN-VQ (L-VQ) nearly matches Fusion-in-Decoder in quality while benefiting from LUMEN compute savings without impractical LUMEN storage requirements.**

1 Introduction

Retrieval augmentation is a common method to improve the factual knowledge of language models (Izacard and Grave, 2021; Borgeaud et al., 2022; Lewis et al., 2020; Khandelwal et al., 2020; Guu et al., 2020; Izacard et al., 2022). Retrieval provides a model with additional context in the form of text passages relevant to an input query. However, retrieval augmentation comes at an increased computational cost, as the model must process the retrieved passages on-the-fly.

A recent line of work (Zemlyanskiy et al., 2021; de Jong et al., 2022; Chen et al., 2022; Li et al., 2022; de Jong et al., 2023a) speeds up retrieval augmentation by pre-encoding passages from the corpus in advance. This way, the model can retrieve representations instead of raw text, which avoids the cost of reading retrieved passages from scratch. One such model, LUMEN, stands out for its strong performance, achieving 3x faster inference than standard Fusion-in-Decoder (Izacard and Grave, 2021) (FiD) with minimal loss in quality.

However, these pre-encoding memory models use much more storage than traditional retrieval-augmented models - LUMEN saves an embedding for each token in the corpus, which takes up much more space than token IDs. Table 1 compares storage requirements for T5 XXL-sized models. FiD requires 2 bytes to store an ID of each token, while LUMEN uses a 4096-dimensional vector of `bfloat16` values, summing to 8KB per token. Wikipedia contains around 4 billion tokens, which means LUMEN token representations take up 30TB. For an internet-scale corpus of 1 trillion tokens, disk requirements balloon to an impractical 7PB.

[*] Equal contribution. Correspondence to {yury,michiel}@augmentcode.com
[†] Augment Computing. Work done at Google Research.

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers), pages 737–744
June 16-21, 2024

This work combines product quantization (Jégou et al., 2011) and VQ-VAE method (van den Oord et al., 2017) to significantly reduce storage requirements for memory-based methods with limited loss in quality. In particular, LUMEN-VQ achieves a 16x compression rate, meaning we only need 2TB to store memories for all of Wikipedia and 500TB for a 1 trillion token corpus. Moreover, LUMEN-VQ suffers minimal loss in performance on the KILT benchmark (Petroni et al., 2021) of knowledge intensive tasks.

Our contribution is the first paper on compressing pre-encoded token memory representations. This compression makes memory methods such as LUMEN practical even for extremely large retrieval corpora. Previous works (e.g., (Santhanam et al., 2022; Yang et al., 2022b; Cohen et al., 2022; Yang et al., 2022a)) have focused on token representation compression for late-interaction reranking models. In contrast, our approach compresses the intermediate representations of a language model. These compressed representations are used as inputs into an LLM, and the compression layers' parameters are trained alongside the rest of the model.

2 Background

We aim to match FiD and LUMEN performance in quality while reducing LUMEN storage requirements. We first describe FiD and LUMEN, methods on which MEMORY-VQ is built, and their storage requirements. For an in-depth analysis, please see de Jong et al. (2023a). We follow up with background on vector quantization, including product quantization and VQ-VAE used for MEMORY-VQ.

2.1 Retrieval and memory augmented models

2.1.1 Fusion-in-Decoder

Fusion-in-Decoder (FiD) (Izacard and Grave, 2021) builds upon the T5 (Raffel et al., 2020) encoder-decoder model. It retrieves relevant text passages, appends them to the input Q, and processes each input-passage pair with the encoder. The resulting token representations are merged and attended by the decoder. We highlight **live** components in blue and **pre-computed** in orange. FiD does not have any pre-computed components.

$$G = \textbf{Dec}\Big[\textbf{Enc}(Q; \text{Passage}_1); \ldots \textbf{Enc}(Q; \text{Passage}_k)\Big]$$

FiD storage needs are low since we only need to store token IDs. Each ID can be encoded with 16 bits, so the storage cost for a retrieval corpus with N tokens is

$$S_{\text{FiD}} = 16 \cdot N$$

2.1.2 LUMEN

LUMEN (de Jong et al., 2023a) reduces inference cost by partially pre-computing encoder representations for retrieved passages. Instead of retrieving the actual text, LUMEN retrieves intermediate layer representations during inference.

LUMEN is initialized from a pre-trained T5 encoder-decoder model, with a **memory encoder** containing the initial $1-\alpha$ proportion of layers and a **live encoder** with the remaining α proportion of layers. The memory encoder is applied offline to pre-compute memory representations for passages in the corpus. Later, these representations are dynamically updated with the fine-tuned live encoder based on the input and task. To ensure compatibility, MEMORY-VQ applies the memory encoder to the input before concatenating the question representation with the memory representation.

$$H_i = \Big[\textbf{MemEnc}(Q); \quad \textbf{MemEnc}(\text{Passage}_i)\Big]$$

$$G = \textbf{Dec}\Big[Q; \textbf{LiveEnc}(H_1); \ldots \textbf{LiveEnc}(H_k)\Big]$$

Choosing $\alpha = 1$ yields a model very close to FiD while $\alpha = 0$ is a full memory model. One of the insights of the LUMEN paper is that one can match FiD performance while using small α, reducing inference cost to a fraction α of FiD encoder FLOPs for any given model size.

LUMEN keeps d-dimensional **MemEnc** output representations for every token. With `bfloat16` format, the total storage cost becomes

$$S_{\text{LUMEN}} = 16d \cdot N$$

2.2 Vector quantization

Vector quantization (VQ) is a classical compression technique for vector data. The general idea is to prepare a set of vectors known as "codes" and then represent each input vector with the nearest code. The approach significantly reduces storage requirements as we only need to store the integer ID of the code instead of the entire high-dimensional input vector. VQ is a lossy compression method since decompression returns the value of the nearest code (by looking up the ID) instead of the original vector. Usually, codes are generated by clustering the input vectors, for example, using `kmeans`-like methods.

2.2.1 Product quantization

A popular variant of vector quantization is product quantization (Jégou et al., 2011; Ge et al., 2013). The method involves partitioning high-dimensional vectors into subspaces and independently quantizing each subspace using a vector quantization subroutine. The product quantization is frequently used in modern approximate nearest neighbor search engines (Guo et al., 2020; Johnson et al., 2021) to speed up lookup.

2.2.2 VQ-VAE

The VQ-VAE approach (van den Oord et al., 2017) is a variant of variational autoencoders that utilizes vector quantization for obtaining a discrete latent representation. Notably, the VQ-VAE compression layer allows joint training with the rest of the model due to a straight-through estimator for gradient backpropagation. The method is commonly used in creating discrete representations of continuous objects such as images or audio (van den Oord et al., 2017; Razavi et al., 2019).

3 MEMORY-VQ

We propose MEMORY-VQ, an efficient method for reducing storage requirements for memory-based models. The high-level idea is to compress memories using vector quantization techniques and store integer codes instead of the original memory vectors. Codes are decompressed into vectors on the fly. Applying the method to LUMEN yields the following LUMEN-VQ model.

$$\text{codes}_i = \textbf{CompressVQ}(\textbf{MemEnc}(\text{Passage}_i))$$

$$H_i = \Big[\textbf{MemEnc}(Q_i);\ \ \textbf{DecompressVQ}(\text{codes}_i)\Big]$$

$$G = \textbf{Dec}\Big[Q; \textbf{LiveEnc}(H_1); \ldots \textbf{LiveEnc}(H_k)\Big]$$

To perform **CompressVQ** and **DecompressVQ** we apply product quantization, splitting each vector into subspaces and independently quantizing each subspace using VQ-VAE. Codes are an exponential moving average of memory vectors assigned to the code in each batch. Appendix A in van den Oord et al. (2017) contains a detailed description.

For training the compression layer jointly with the model, we follow the VQ-VAE recipe, but we avoid using the commitment loss in our experiments as it led to model divergence.

To initialize the codebooks, we use a procedure similar to `kmeans++` initialization (Arthur and Vassilvitskii, 2007). Additionally, we perform codebook reset (Williams et al., 2020) using the same procedure to re-initialize infrequently used codes.

We divide memories into g subspaces, and if needed, pad memories with zeros to ensure divisibility. Each subspace has C codes. Therefore the storage requirement for each quantized vector is the number of subspaces multiplied by the number of bits required to represent each ID, which is the logarithm of the number of codes.

$$S_{\text{LUMEN-VQ}} = g \cdot \lceil \log_2 C \rceil \cdot N$$

4 Experiments

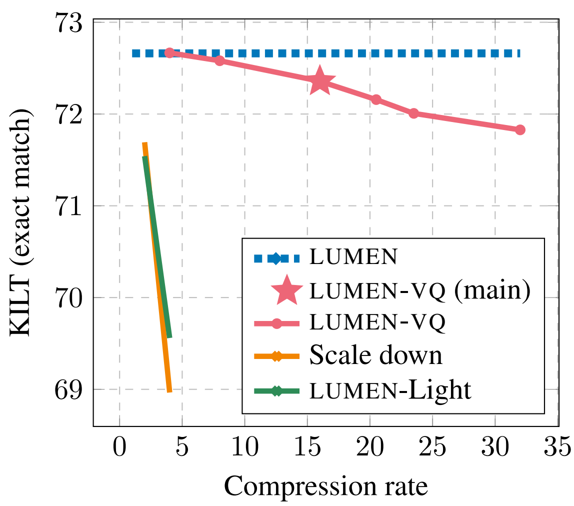

Figure 1: **LUMEN-VQ achieves a strongly improved trade-off between performance and compression.** The plot shows average exact match on dev sets of KILT tasks as a function of compression rate. We compare LUMEN-VQ with baselines Scale down (LUMEN XL and LUMEN Large) and LUMEN-*Light* (FiD-Light from Hofstätter et al. (2022a) adapted for LUMEN).

Model	KILT, EM
LUMEN-VQ	72.43
initialize from fine-tuned LUMEN	72.42
+ freeze memory encoder	72.33
+ freeze whole model	71.79

Table 2: Performance comparison of different approaches for initializing and training the LUMEN-VQ.

4.1 Experimental setup

Model configuration LUMEN-VQ and LUMEN are built on the T5.1.1 architecture (Raffel et al., 2020) and implemented in JAX using Flax (Heek et al., 2020) and Flaxformer. All models fine-tune public T5.1.1 XXL checkpoints. We train FiD using the recipe from Izacard and Grave (2021).

The training setup for LUMEN and LUMEN-VQ is based on de Jong et al. (2023b). We initialize the memory encoder with the first 1 - α proportion of layers from the T5 encoder and the live encoder with the last α proportion of layers, where α is the given proportion of live layers. We set $\alpha = \frac{1}{3}$ in our main experiments.

We train and evaluate on a subset of knowledge-intensive task datasets from the KILT benchmark (Petroni et al., 2021). We adopt the retrieval procedure from Hofstätter et al. (2022b) and use GTR-Base model (Ni et al., 2021) as the retriever. See Appendix A and de Jong et al. (2023b) for details.

4.2 Main results

In our main experiments, we compress LUMEN-XXL's 4096-dimensional memories using $g = 256$ subspaces and $C = 65536$ codes per subspace, allowing us to store code IDs in `int16` format. We need 512 bytes to store each token vector instead of 8192 bytes for the original memories. As a result, LUMEN-VQ achieves a compression rate of 16 with minimal performance loss, as shown in Table 1.

4.3 Quality-compression rate trade-off

We investigate the quality-compression tradeoff for LUMEN-VQ by varying the number of subspaces.

We compare against several naive baselines; the first involves scaling down the model (e.g., LUMEN-XL or LUMEN-Large). This reduces d from 4096 to 2048 or 1024, respectively. The second baseline, called LUMEN-Light, is inspired by the FiD-Light approach (Hofstätter et al., 2022a). In LUMEN-Light, we retain memories of the first K tokens, varying K from $\frac{1}{2}$ to $\frac{1}{4}$ of the passage length, achieving compression rates of 2 and 4.

Figure 1 presents the performance results. Both baselines exhibit significant performance losses as compression rates increase. In contrast, the LUMEN-VQ measure shows a gradual decline in performance, with a loss of approximately 0.2 performance points at a compression rate of 16.

4.4 Ablations

We investigate if initializing VQ-VAE training from a fine-tuned LUMEN model yields better results. The results in Table 2 show that fine-tuning LUMEN-VQ from scratch achieves similar performance to initializing from a fine-tuned LUMEN model.

We also analyze which model components benefit most from joint fine-tuning with VQ-VAE. Freezing the memory encoder during joint training, starting with a fine-tuned LUMEN model, has little impact on performance. However, updating only VQ-VAE codes while freezing the entire model leads to decreased performance, indicating the model's need to adapt to decompression layer errors.

5 Related work

Memory models Retrieval augmentation can be computationally expensive due to the additional context that language models need to process. To mitigate this, memory models like LUMEN (de Jong et al., 2023a), GLIMMER (de Jong et al., 2023b), and others (Zemlyanskiy et al., 2021; de Jong et al., 2022; Wu et al., 2022a; Li et al., 2022; Zhong et al., 2022; Chen et al., 2022; Wu et al., 2022b; Bertsch et al., 2023; Milbauer et al., 2023) store pre-computed representations in memory. MEMORY-VQ focuses on improving the storage requirements for memory-based models. While our experiments involve the LUMEN (de Jong et al., 2023a) model due to its strong performance, the method applies to a broader range of models.

Compression for late-interaction reranking MEMORY-VQ focuses on compression for late-interaction memory models, while other works have explored compression for late-interaction reranking. For instance, SDR (Cohen et al., 2022) employs an autoencoder to reduce token representation dimensionality, followed by product quantization. BECR (Yang et al., 2022a) utilizes locality-sensitive hashing for token representation compression. CQ (Yang et al., 2022b) learns vector quantization parameters by treating codes as learnable weights and uses Gumbel-Softmax for differentiable nearest code determination. Finally, ColBERTv2 (Santhanam et al., 2022) proposes a custom compression scheme combining PQ and integer quantization to handle reconstruction residuals.

6 Conclusion

We introduced MEMORY-VQ, a novel approach for reducing the storage requirements of memory-augmented language models without compromising performance. By employing VQ-VAE to compress token representations, we obtain a LUMEN model with 16x compression, denoted as LUMEN-VQ. Remarkably, LUMEN-VQ maintains performance close to LUMEN and FiD and benefits from LUMEN inference speed-ups with sharply reduced storage cost. Using MEMORY-VQ, memory augmentation is a practical solution for drastic inference speedups with extensive retrieval corpora.

7 Limitations

This work concerns a memory compression and speedup method for language models augmented with retrieved passages. The goal of a retrieval-augmented language model is often to enhance factuality by grounding generations in a specific corpus of text. Of course, this pushes the burden of factuality on to the curation of text, and without a good corpus can still result in model confabulations and propagation of harmful biases. Especially in the context of search-result-augmented language models, retrieved web data has no guarantee of factuality or unbiasedness. Secondly, when looking at compression-quality tradeoffs, it is important to consider the measures of quality. In our work we evaluate the compressed model on a variety of knowledge-intensive benchmarks, but those wishing to use our method in contexts requiring other capabilities or safeguards will need to evaluate the compression-quality tradeoff in those specific domains.

References

David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035. SIAM.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. 2023. Unlimiformer: Long-range transformers with unlimited length input. *CoRR*, abs/2305.01625.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR.

Wenhu Chen, Pat Verga, Michiel de Jong, John Wieting, and William W. Cohen. 2022. Augmenting pre-trained language models with qa-memory for open-domain question answering. *CoRR*, abs/2204.04581.

Nachshon Cohen, Amit Portnoy, Besnik Fetahu, and Amir Ingber. 2022. SDR: efficient neural re-ranking using succinct document representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6624–6637. Association for Computational Linguistics.

Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Joshua Ainslie, Sumit Sanghai, Fei Sha, and William W. Cohen. 2023a. Pre-computed memory or on-the-fly encoding? A hybrid approach to retrieval augmentation makes the most of your compute. *CoRR*, abs/2301.10448.

Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Sumit Sanghai, William W. Cohen, and Joshua Ainslie. 2023b. GLIMMER: generalized late-interaction memory reranker. *CoRR*, abs/2306.10231.

Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William W. Cohen. 2022. Mention memory: incorporating textual knowledge into transformers through entity mention attention. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Hady ElSahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon S. Hare, Frédérique Laforest, and Elena Simperl. 2018. T-rex: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA).

Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pages 2946–2953. IEEE Computer Society.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3887–3896. PMLR.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: retrieval-augmented language model pre-training. *CoRR*, abs/2002.08909.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. 2020. Flax: A neural network library and ecosystem for JAX.

Sebastian Hofstätter, Jiecao Chen, Karthik Raman, and Hamed Zamani. 2022a. Fid-light: Efficient and effective retrieval-augmented text generation. *CoRR*, abs/2209.14290.

Sebastian Hofstätter, Jiecao Chen, Karthik Raman, and Hamed Zamani. 2022b. Multi-task retrieval-augmented text generation with relevance sampling. *CoRR*, abs/2207.03030.

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, EACL 2021, Online, April 19 - 23, 2021*, pages 874–880. Association for Computational Linguistics.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot learning with retrieval augmented language models. *CoRR*, abs/2208.03299.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1601–1611. Association for Computational Linguistics.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Vancouver, Canada, August 3-4, 2017*, pages 333–342. Association for Computational Linguistics.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Zonglin Li, Ruiqi Guo, and Sanjiv Kumar. 2022. Decoupled context processing for context augmented language modeling. *CoRR*, abs/2210.05758.

Jeremiah Milbauer, Annie Louis, Mohammad Javad Hosseini, Alex Fabrikant, Donald Metzler, and Tal Schuster. 2023. LAIT: efficient multi-segment encoding in transformers with layer-adjustable interaction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10251–10269. Association for Computational Linguistics.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y. Zhao, Yi Luan, Keith B. Hall, Ming-Wei Chang, and Yinfei Yang. 2021. Large dual encoders are generalizable retrievers. *CoRR*, abs/2112.07899.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick S. H. Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2523–2544. Association for Computational Linguistics.

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. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Ali Razavi, Aäron van den Oord, and Oriol Vinyals. 2019. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14837–14847.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 3715–3734. Association for Computational Linguistics.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 809–819. Association for Computational Linguistics.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6306–6315.

Will Williams, Sam Ringer, Tom Ash, David MacLeod, Jamie Dougherty, and John Hughes. 2020. Hierarchical quantized autoencoders. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022a. Memorizing transformers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Yuxiang Wu, Yu Zhao, Baotian Hu, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2022b. An efficient memory-augmented transformer for knowledge-intensive NLP tasks. *CoRR*, abs/2210.16773.

Yingrui Yang, Yifan Qiao, Jinjin Shao, Xifeng Yan, and Tao Yang. 2022a. Lightweight composite re-ranking for efficient keyword search with BERT. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, pages 1234–1244. ACM.

Yingrui Yang, Yifan Qiao, and Tao Yang. 2022b. Compact token representations with contextual quantization for efficient document re-ranking. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 695–707. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. 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, Brussels, Belgium, October 31 - November 4, 2018*, pages 2369–2380. Association for Computational Linguistics.

Yury Zemlyanskiy, Joshua Ainslie, Michiel de Jong, Philip Pham, Ilya Eckstein, and Fei Sha. 2021. Readtwice: Reading very large documents with memories. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5189–5195. Association for Computational Linguistics.

Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. Training language models with memory augmentation. *CoRR*, abs/2205.12674.

A Experimental setup

Model configuration The original LUMEN implementation employed a separate question encoder, but de Jong et al. (2023b) showed we can re-use the memory encoder as long as it is fine-tuned.

Fine-tuning During fine-tuning, we utilize the Adafactor optimizer (Shazeer and Stern, 2018) with a constant learning rate of 0.0001, a batch size of 128, and a dropout rate of 0.1 for all tasks. When performing multi-task training, we uniformly sample from the tasks. We allocate 48 and 304 tokens for question and passage inputs, respectively. LUMEN-VQ is using 0.999 as an EMA factor for code updates.

Data We train and evaluate on a subset of knowledge-intensive task datasets from the KILT benchmark (Petroni et al., 2021). The datasets include question-answering datasets such as Natural Questions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and HotPotQA (Yang et al., 2018), along with the fact verification dataset FEVER (Thorne et al., 2018), and the slot-filling datasets Zero Shot RE (Levy et al., 2017) and T-REx (ElSahar et al., 2018). To address imbalanced dataset issues, we apply the relevance filtering procedure introduced by Hofstätter et al. (2022b).

For the retrieval corpus, we use a Wikipedia dump provided by the KILT benchmark http://dl.fbaipublicfiles.com/BLINK/enwiki-pages-articles.xml.bz2 containing approximately 4B tokens.

Retrieval We adopt the retrieval procedure introduced by Hofstätter et al. (2022b), where Wikipedia articles are segmented into chunks, each containing up to 200 words. The dense retriever, a pre-trained GTR-Base model (Ni et al., 2021), is utilized to identify the most relevant chunks for each query, with 20 retrieved passages for each query.

B Experiments

B.1 Smaller codebook

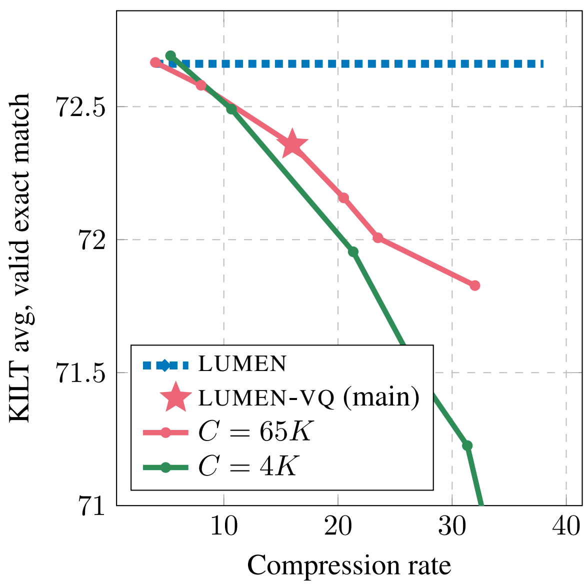

Figure 2: The plot shows average exact match on validation sets of KILT tasks as a function of compression rate. We compare LUMEN-VQ with the codebook of size $C = 65536$ and $C = 4096$.

We study the effect of using a smaller codebook of size $C = 4096$ instead of $C = 65536$. Results in Figure 2 show that using a smaller codebook has similar quality-compression trade-offs for lower compression rates but leads to worse trade-offs when we increase the compression rate.

B.2 Can we compress code IDs even further?

Integer data, like token IDs, might exhibit regularities, enabling additional data compression by using fewer bits for frequent patterns. For instance, applying standard compression tools like `gzip` or `zstd` to Wikipedia token IDs resulted in a compression factor of around 1.5. However, using the same tools on LUMEN-VQ codes of Wikipedia passages yielded a more modest compression rate of 1.1.

Compression was performed independently on each subspace, with most subspaces being incompressible. Around 5% of the subspaces showed compression rates ranging from 2 to 6. Notably, no compression was achieved when attempting to compress codes from all subspaces together.