Memory Layers at Scale

Vincent-Pierre Berges*, Barlas Oğuz*, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, Gargi Ghosh

Meta FAIR

고경빈

2025.01.31

Background

- Larger LMs recall information more accurately
- Increasing parameter size in dense networks increases computational and energy costs, raising efficiency concerns for information storage
- While neural networks can learn lookup tables, associative memory is more efficient for storing associations
- Attempts to integrate memory layers into neural networks failed due to memory bandwidth limits and lack of hardware optimization
- Thus, the focus moved to scalable alternatives like MOE, similar to dense layers

Background

- MOE(Mixture of Experts)
  - Use multiple experts, activating a specific set for each input to maximize efficiency
- Memory augmented models
  - Product-key networks: use sparse activation to select experts based on product keys
  - PEER: replaces vector values with rank-one matrices

$$u = \begin{bmatrix} 2 \\ 3 \end{bmatrix},\ v^{\mathsf{T}} = \begin{bmatrix} 1 & 4 \end{bmatrix} \Rightarrow\ A = u \cdot v^{\mathsf{T}} = \begin{bmatrix} 2 & 8 \\ 3 & 12 \end{bmatrix}$$

    - Like a memory layer but needs 2x parameters

Memory Augmented Architectures

- Trainable memory layers work like attention mechanisms
  - Given q, K, and V, the output is a weighted sum of values based on query similarity
- Two key differences
  - The keys and values in memory layers are trainable parameters, not activations
  - Memory layers usually have a larger scale, requiring sparse lookups and updates
- How it works

$$I = \mathrm{SelectTopkIndices}(Kq), \qquad s = \mathrm{Softmax}(K_I q), \qquad y = sV_I$$

  - Each token embedding performs a memory lookup independently, selecting only the necessary information for more efficient processing

Product-key lookup

- Inefficient query-key retrieval for large memories
- Approximate vector similarity techniques are difficult to use due to continuous training and re-indexing of keys

→ Trainable product-quantized keys

- Split the keys and query sets in half
- Perform top-k on each half and combine the results to find the most similar key

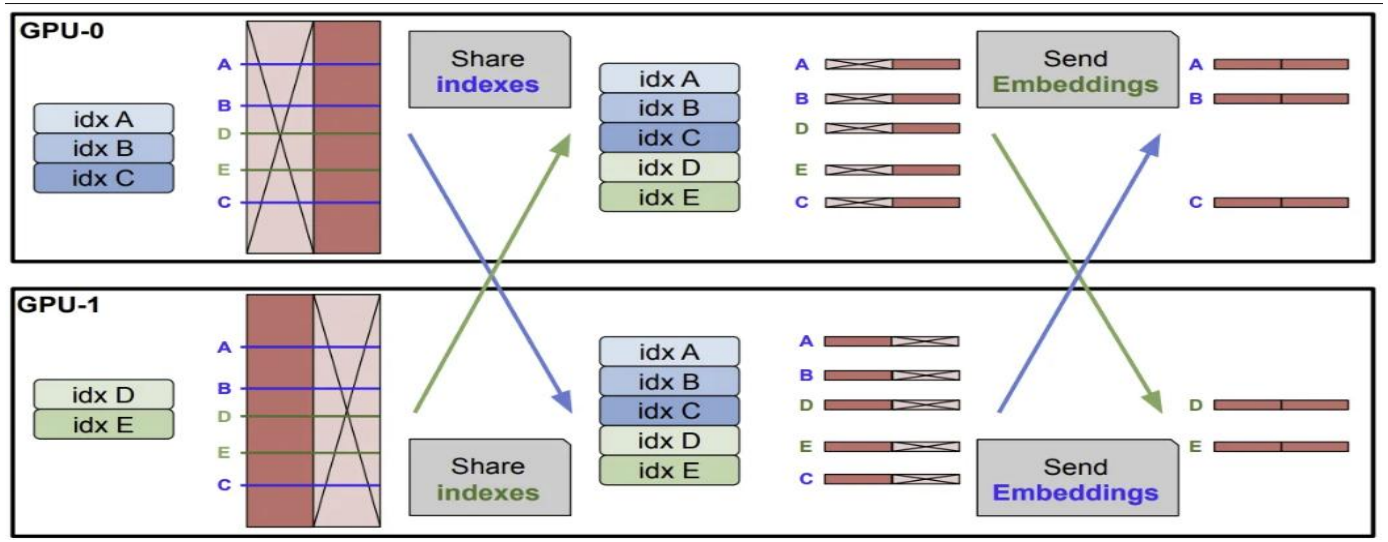
$$argmax_{i_1 \in I_1, i_2 \in I_2} s_1[i_1] + s_2[i_2]$$

- Efficiency: Reduces computation and memory by searching in smaller key spaces and combining results.

Parallel memory

- Memory layers are memory-intensive due to the many trainable parameters

→ Parallelize the embedding lookup and aggregation across multiple GPUs.



- Fast and efficiently manages memory usage
- Works well without conflict with other parallelization methods

Shared memory

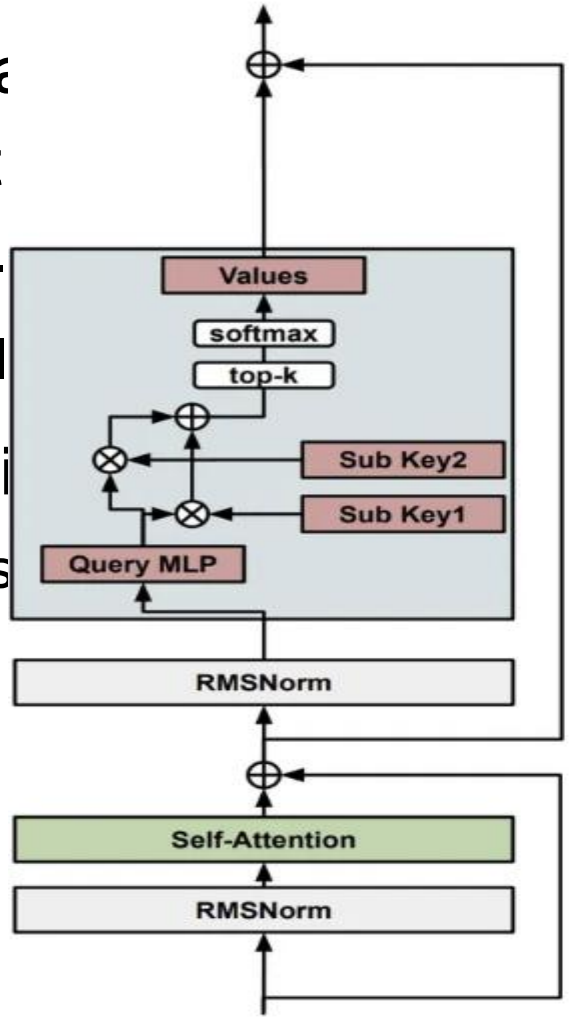
- Sharing the same memory pool across layers optimizes usage
- Adding memory improves performance up to a point, then it decreases
  - Sparse and dense layers are both needed and likely complementary

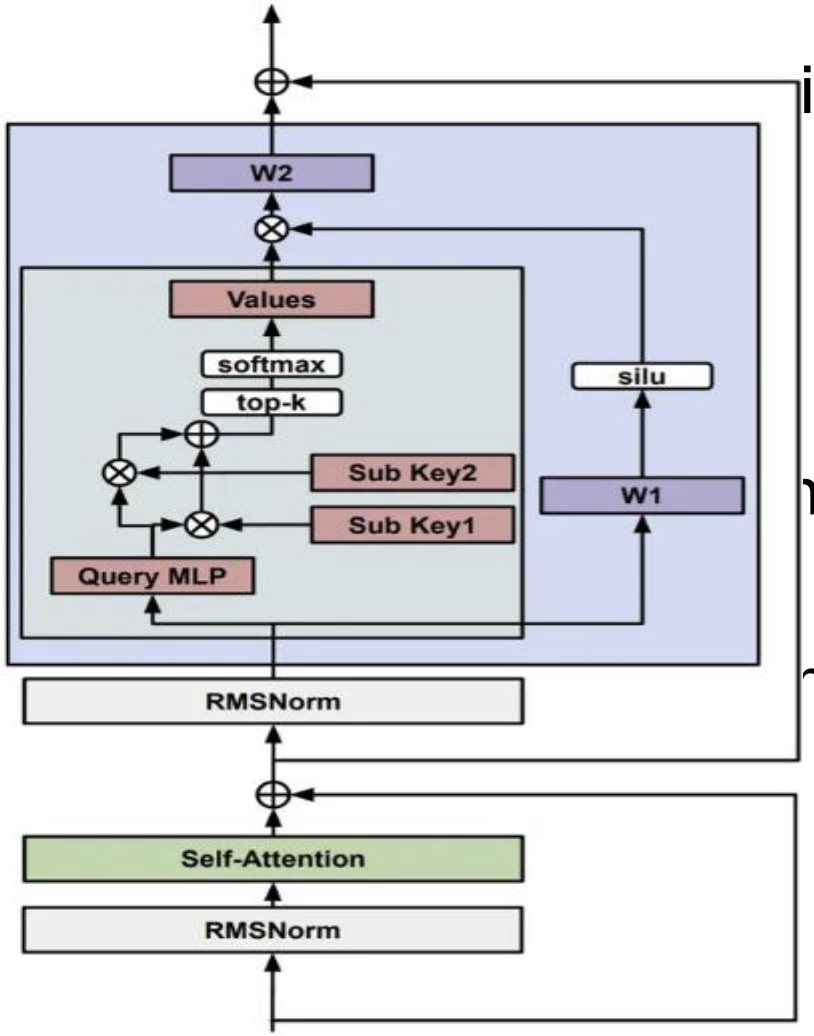
New effective CUDA kernels

- The computation is GPU-light, but memory bandwidth limits performance
- EmbeddingBag is inefficient due to poor memory access optimization and low bandwidth usage
- Forward: optimizes memory access, reaching 3TB/s bandwidth
- Backward pass is complex because multiple output gradients share the same weight gradient
  - Atomics: Accumulation using atomic additions, x5 faster than Pytorch
  - Lock: Row-level atomic locking amortizes memory lock
  - Reverse_indices: Inverses token-to-embedding mapping for atomic-free gradient updates

Input-dependent gating

- Improve tra… input-dependent
  - SiLU = x ·
  - Memory l…
- Qk-normali… mory layers
  - Stabilizes…
  - Prevents … rmalization and scaling





Experiments Setup

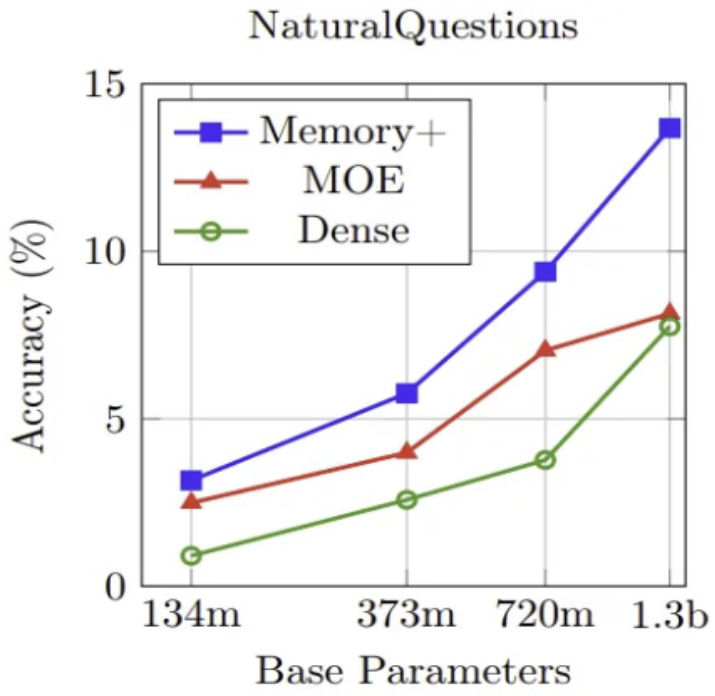
- Base model: Dense transformers, replacing some feed-forward layers with a shared memory layer
  - Scaling law: model size(134m, 373m, 720m, 1.3b) ← Llama2 tokenizer(32k), 1T tokens
  - 8B Base: Llama3 configuration and tokenizer(128k)
- Baselines
  - Dense Transformers
  - MOE
  - PEER
- Evaluation Benchmarks
  - Factual question answering: NaturalQuestions, TriviaQA
  - Multi-hop question answering: HotpotQA
  - Scientific and common sense world knowledge: MMLU, HellaSwag, OBQA, PIQA
  - Coding: HumanEval, MBPP

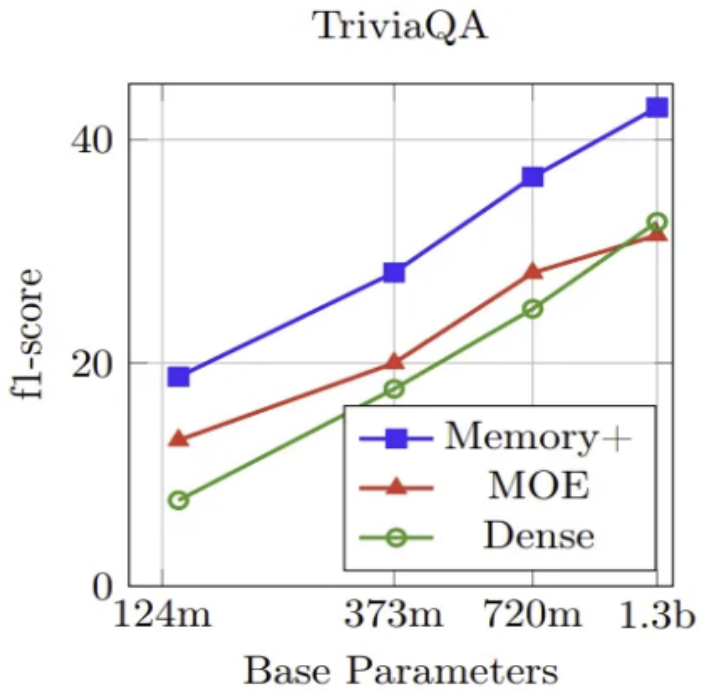
With fixed memory size

	Model Configuration	Total Params	NQ	TQA	PIQA	OBQA	HotPot
134m	Dense	134m	0.91	7.7	62.13	16.40	5.18
	MOE	984m	2.49	13.08	65.78	18.80	7.80
	PEER	1.037b	2.46	16.34	**67.25**	17.40	8.82
	Memory	937m	2.1	16.31	66.65	**17.80**	9.28
	Memory+	937m	**3.16**	**18.77**	65.94	17.60	**9.35**
373m	Dense	373m	2.58	17.68	67.47	18.80	10.06
	MOE	1.827b	3.99	19.94	68.88	**22.20**	12.50
	PEER	1.575b	5.1	26.39	70.19	21.60	12.96
	Memory	1.441b	4.95	24.24	69.37	20.40	12.53
	Memory+	1.434b	**5.76**	**28.10**	**71.22**	22.00	**13.34**
720m	Dense	720m	3.77	24.85	71.33	22.60	12.90
	MOE	2.768b	7.04	28.08	70.08	20.80	14.10
	PEER	2.517b	7.92	33.26	71.98	**25.00**	14.03
	Memory	2.316b	7.2	34.8	71.82	24.40	**14.94**
	Memory+	2.316b	**9.39**	**36.67**	**72.42**	24.00	14.92
1.3b	Dense	1.3b	7.76	32.64	72.74	23.40	13.92
	MOE	3.545b	8.14	31.46	73.72	25.20	15.15
	PEER	3.646b	12.33	42.46	73.34	26.60	15.39
	Memory	3.377b	9.83	39.47	72.29	25.80	15.46
	Memory+	3.377b	**13.68**	**42.89**	**75.35**	**26.80**	**16.72**
	Memory+ 4m	9.823b	14.43	51.18	75.03	27.80	18.59
	Memory+ 16m	35.618b	20.14	58.67	76.39	26.80	**20.65**
	Memory+ 64m	138.748b	**20.78**	**62.14**	**77.31**	**30.00**	20.47
llama2 7B (2T)	Dense	7b	25.10	64.00	78.40	33.20	25.00

- Dense<MOE<Memory<PEER<Memory+
- Memory+ 1.3b ≈ Llama2 7B

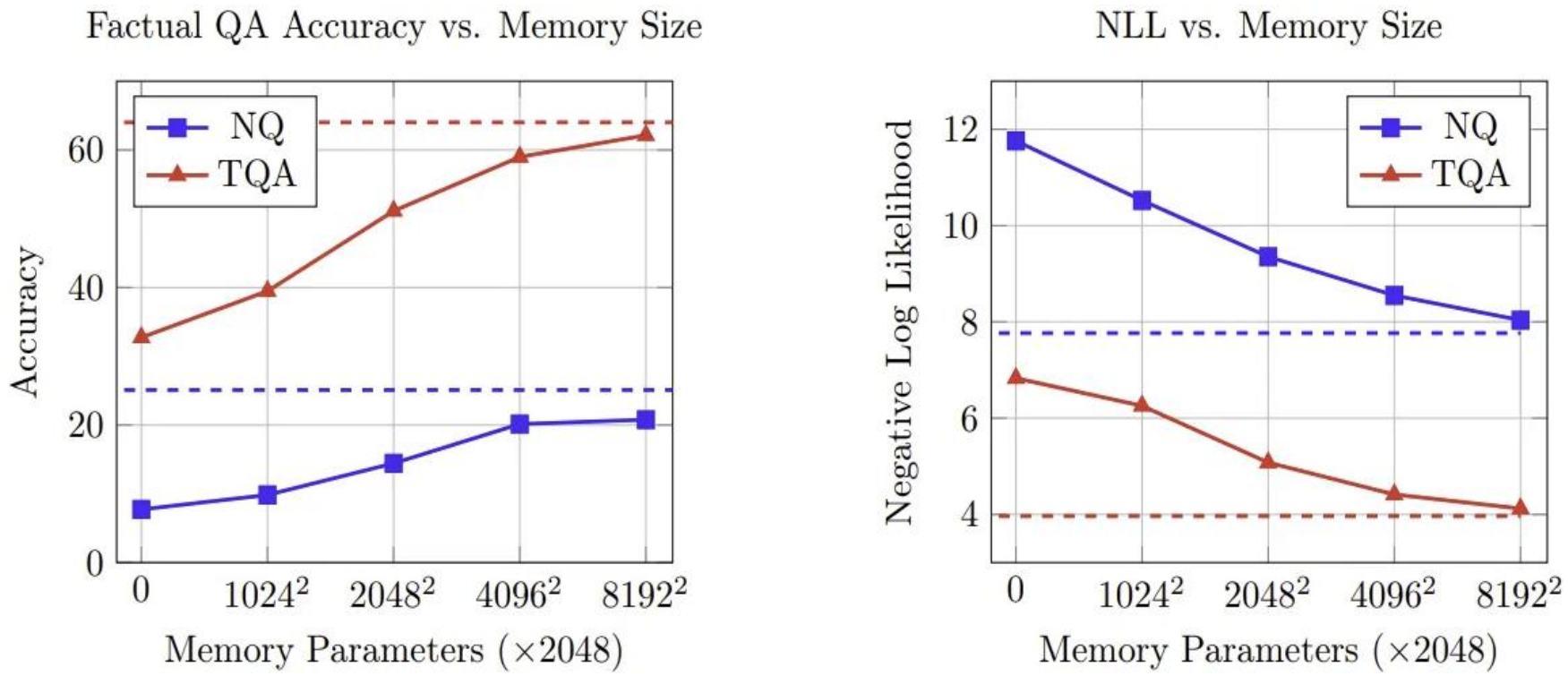
With fixed memory size





- Memory+ model has better scaling performance than MOE and Dense models

Scaling memory size with a fixed base mode



- Memory models train more efficiently than dense models, achieving high performance with fewer resources

Results at 8B scale

Model	HellaS.	Hotpot	HumanE.	MBPP	MMLU	NQ	OBQA	PIQA	TQA
llama3.1 8B (15T)	60.05	27.85	37.81	48.20	66.00	29.45	34.60	79.16	70.36
dense (200B)	53.99	20.41	21.34	**30.80**	41.35	18.61	**31.40**	78.02	51.741
Memory+ (200B)	**54.33**	**21.75**	**23.17**	29.40	**50.14**	**19.36**	30.80	**79.11**	**57.64**
dense (1T)	58.90	25.26	29.88	**44.20**	59.68	25.24	34.20	**80.52**	63.62
Memory+ (1T)	**60.29**	**26.06**	**31.71**	42.20	**63.04**	**27.06**	**34.40**	79.82	**68.15**

- Memory model enables faster learning of factual information
- Memory model matches Llama 3.1 8B performance with less data

Model ablations

- Memory layer placement

layer #	nll	NQ nll	TQA nll
12	2.11	12.13	8.34
12,16,20	2.08	11.60	7.54
8,12,16	2.07	11.79	7.64
4,12,20	**2.06**	**11.32**	**7.20**
5,8,11,14,17,21	2.11	11.79	7.73

- Memory layer variants

Model	nll	NQ nll	TQA nll
PK base	2.11	12.13	8.34
+gated	2.11	12.24	8.17
+swilu	2.11	12.05	8.09
+random values	2.11	12.36	8.09
+softmax sink	2.11	12.19	8.04

Model ablations

- Value dimension

		nll	NQ nll	TQA nll
v_dim	#values			
64	16m	2.15	12.86	8.75
256	4m	2.14	12.63	8.49
1024	1m	**2.11**	**12.13**	**8.34**
2048	512k	2.14	12.49	8.53

- Key dimension

	nll	NQ nll	TQA nll
key_dim			
256	2.11	12.13	8.34
512	2.12	12.32	8.15
1024	2.11	12.37	8.25
2048	**2.09**	**11.98**	**7.83**

Conclusion

- Improving and scaling memory layers can greatly complement dense neural networks, providing significant advantages
- Memory layers improved performance across sizes of memory and model
- In QA benchmarks, memory layers increased factual accuracy by over 100% and boosted coding and general knowledge performance
- Models with memory layers rivaled dense models with more computation and surpassed MOE architectures on fact-based tasks

My Review

- The concept of memory layers seems simple but is truly revolutionary
- If the Scaling Law holds true, then wealthy companies are bound to create the best AI, but recent developments like Deepseek suggest that there are new approaches that could break free from the Scaling Law

Open Question

- Can memory layers be applied to all LLMs? Would there be performance improvements in other transformer-based LLMs?