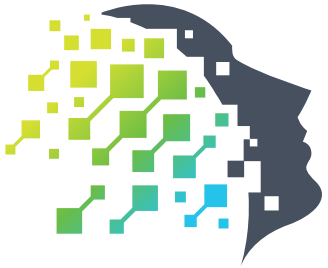


ICLR

International Conference On Learning Representations

Q-RAG: Long Context Multi-Step Retrieval via Value-Based Embedder Training

Artyom Sorokin[1,2] Nazar Buzun[1,3] Alexander Anokhin[2] Egor Vedernikov[2] Petr Anokhin[1]
Mikhail Burtsev[4] Evgeny Burnaev[1,2]

[1]AXXX, Moscow, Russia

[2]Applied AI Institute, Moscow, Russia,

[3]Innopolis University, Innopolis, Russia

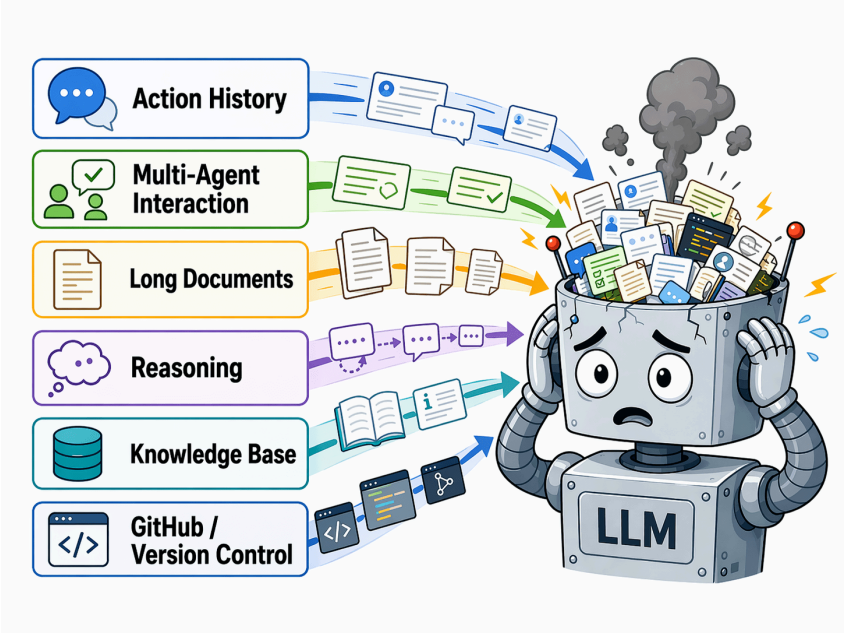
[4]London Institute for Mathematical Sciences, London, UK

Motivation

Long-Context is still challenging for Large Language Models

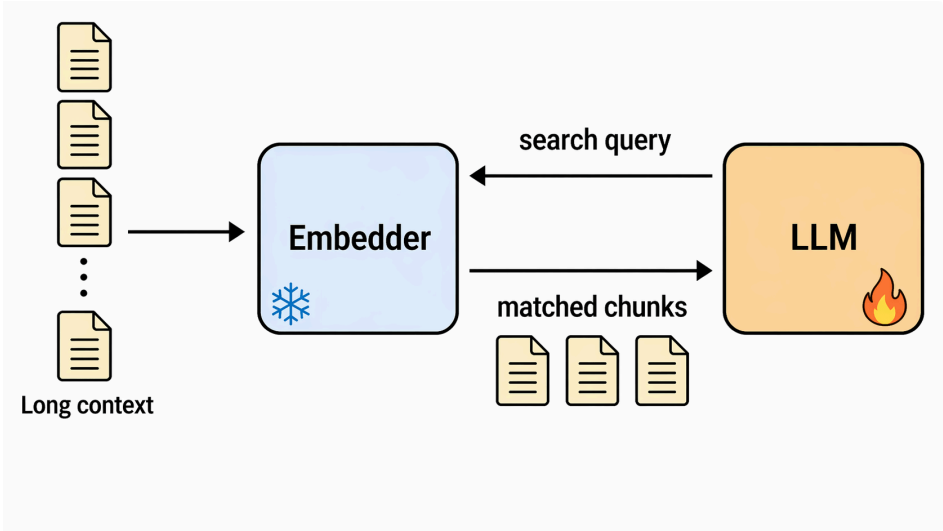
Common approaches to addressing the long-context challenge:

- SSM, Transformers with Recurrence, Linear Attention
  - Can't be combined with best LLMs
- Agents that Knowledge Graphs
  - Slow inference for long context processing
- Multi-step RAG and RAG agents



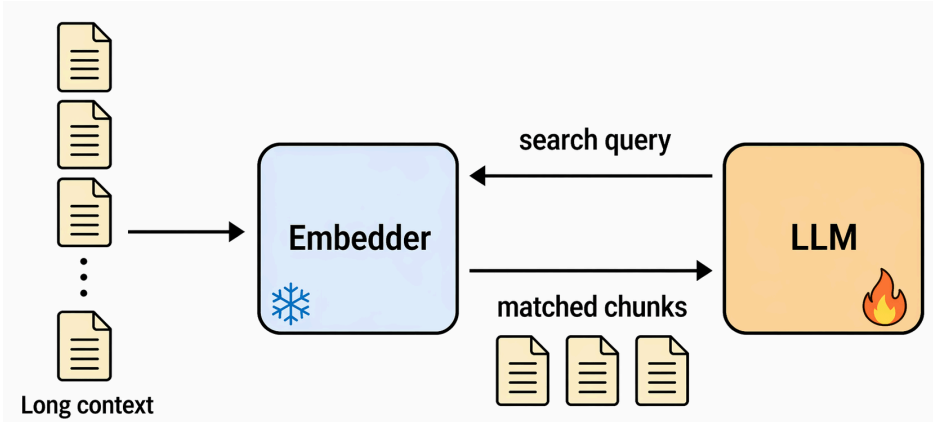
Motivation

Popular direction for multi-step RAG is to fine-tune an LLM to use retrieval as a tool



Motivation

Popular direction for multi-step RAG is to fine-tune an LLM to use retrieval as a tool

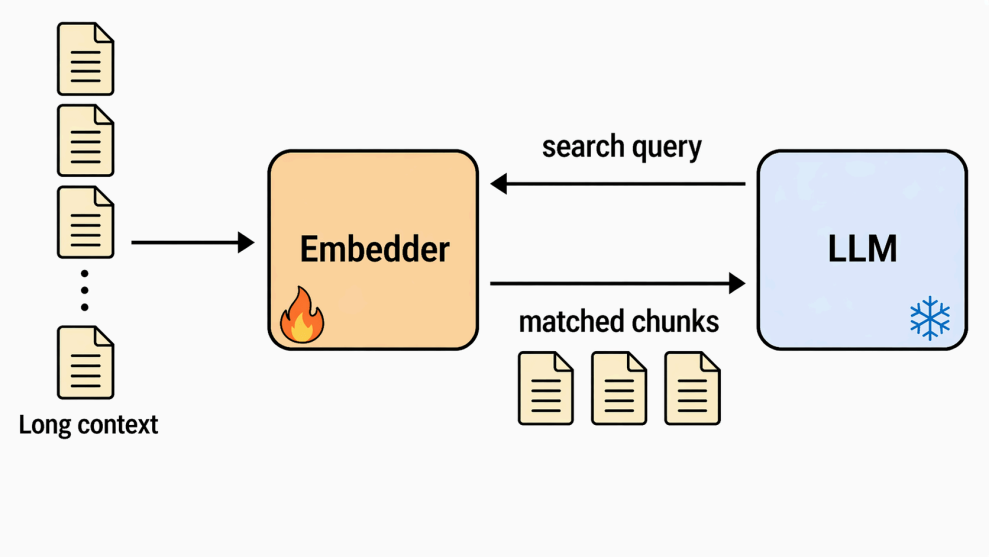


But fine-tuning LLMs can be expensive :(

Q-RAG: Main Idea

Main Idea:

- Train Embedder instead of LLM to multi-step search tasks
- Search queries are generated directly in embedding space
- Formulate multi-step RAG as an MDP



Q-RAG: retrieval as RL problem

Multi-step retrieval as RL problem:

- **State:** query + retrieved information.
- **Actions:** chunks available for retrieval
- **Reward:** 1.0 if all required chunks are found
- **Termination:** exausting retrieval budget

Timestep 0

$$s_0 = \text{prompt}(q)$$

$$\mathbb{A}_0 = \{c^1\ c^2\ c^3\ c^4\ c^5\}$$

Timestep 1

$$s_1 = \text{prompt}(q, c^4)$$

$$\mathbb{A}_1 = \{c^1\ c^2\ c^3\ \times\ c^5\}$$

Q-RAG: Training

We train embedders to approximate Q-function with Inner product beween state embedding and chunk embedding:

$$\langle E_s(s;\theta_1), E_a(a^i, i;\theta_2)\rangle = Q_\theta(s, a^i) \approx Q^*(s, a^i)$$

Q-RAG: Training

We train embedders to approximate Q-function with Inner product beween state embedding and chunk embedding:

$$\langle E_s(s;\theta_1), E_a(a^i, i;\theta_2)\rangle = Q_\theta(s, a^i) \approx Q^*(s, a^i)$$

NCE training: $\langle s, a\rangle \rightarrow$ semantic similarity

Q-value approximation: $\langle s, a\rangle \rightarrow$ usefulness of a given query $s = prompt(q, a_{t-1}..., a_0)$

Q-RAG: Training

We train embedders to approximate Q-function with Inner product beween state embedding and chunk embedding:

$$\langle E_s(s;\theta_1), E_a(a^i, i;\theta_2)\rangle = Q_\theta(s, a^i) \approx Q^*(s, a^i)$$

NCE training: $\langle s, a\rangle \rightarrow$ semantic similarity

Q-value approximation: $\langle s, a\rangle \rightarrow$ usefulness of a given query $s = prompt(q, a_{t-1}..., a_0)$

Max entropy value functions that encourage exploration:

$$Q^\pi(s,a) = r(s,a) + \gamma V^\pi(s' = p(s,a))$$
$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a) - \alpha \log \pi(a|s)\right]$$

Q-RAG: Training

We train embedders to approximate Q-function with Inner product beween state embedding and chunk embedding:

$$\langle E_s(s;\theta_1), E_a(a^i, i;\theta_2)\rangle = Q_\theta(s, a^i) \approx Q^*(s, a^i)$$

NCE training: $\langle s, a\rangle \to$ semantic similarity

Q-value approximation: $\langle s, a\rangle \to$ usefulness of a given query $s = prompt(q, a_{t-1}..., a_0)$

Max entropy value functions that encourage exploration:

$$Q^\pi(s,a) = r(s,a) + \gamma V^\pi(s' = p(s,a))$$
$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a) - \alpha \log \pi(a|s)\right]$$

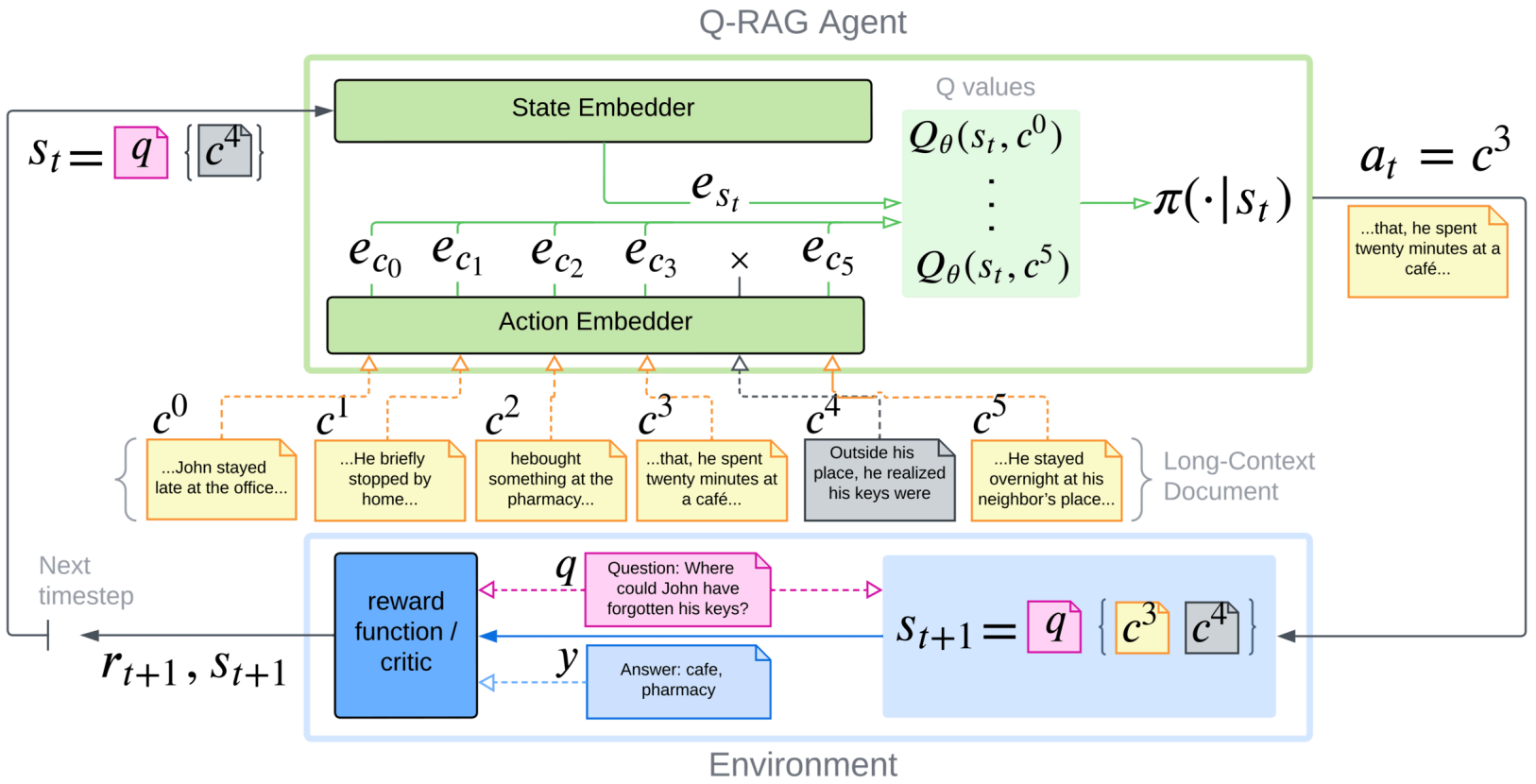
Given Q_θ, the chunk selection probability is computed using a Boltzmann policy:

$$\pi(a_t|s_t) = \frac{\exp\frac{1}{\alpha}\left(Q_\theta(s_t,a_t) - q\right)}{\sum_{a\in\mathcal{A}_t}\exp\frac{1}{\alpha}(Q_\theta(s_t,a) - q)}\ , \quad \text{where } q = \max_{a\in A_t} Q_\theta(s_t, a)$$

Q-RAG: Architecture

Additional Q-RAG Details:

- PQN Backbone:
  - No Replay Buffer
- Separate embedders for states and chunks/actions
- Adding Target Networks improves stability
- λ-return used as target for Q-function



Relative Positional Encoding

Q-RAG assigns each candidate chunk a continuous **relative position** $\rho_t(i)$:

$$\rho_t(i) \;=\; j\,\delta \;+\; \ell\,\frac{i - b_j}{b_{j+1} - b_j},$$

where $b_j < i < b_{j+1}$

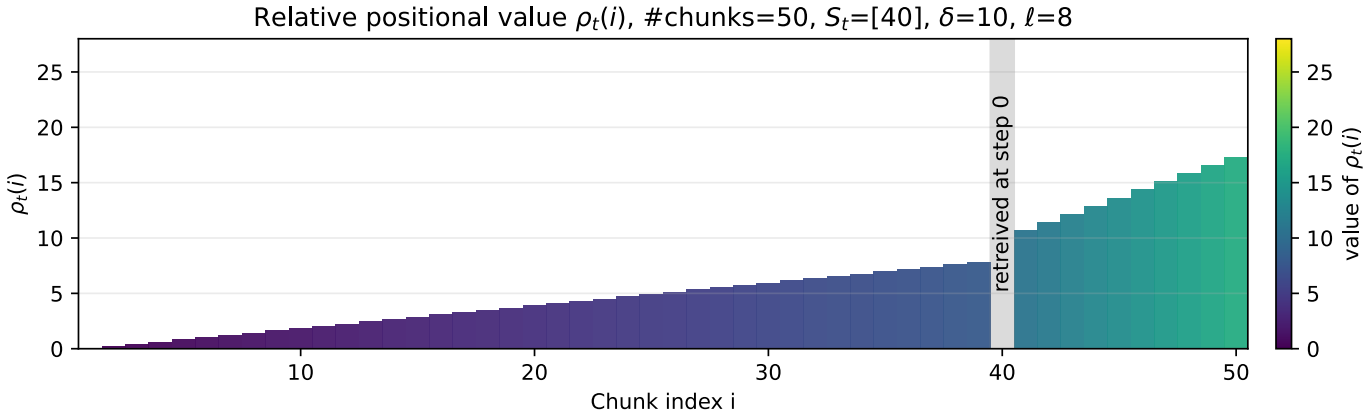
b_j - closest retrieved chunk to the left

b_{j+1} - closest retrieved chunk to the right

$$\langle E_s(s), E_a(a^i)\rangle \Rightarrow \langle E_s(s),\, R_{\rho_t(i)}\, E_a(a^i)\rangle$$

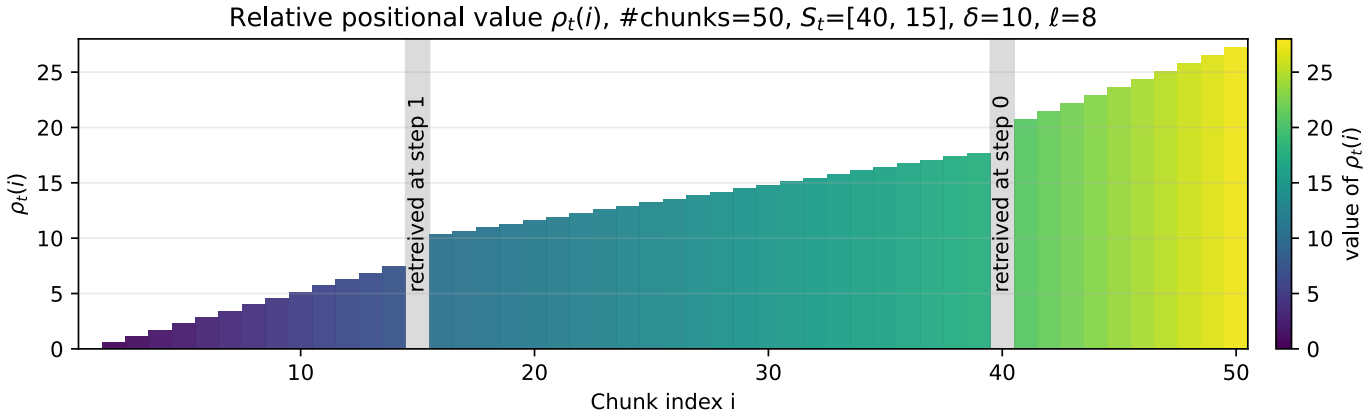
- RoPE + $\rho_t(i)$ is used to rotate chunk embeddings
- encodes position within the document and relative to retrieved chunks

$s_1 = \texttt{[query, chunk\#40]}$

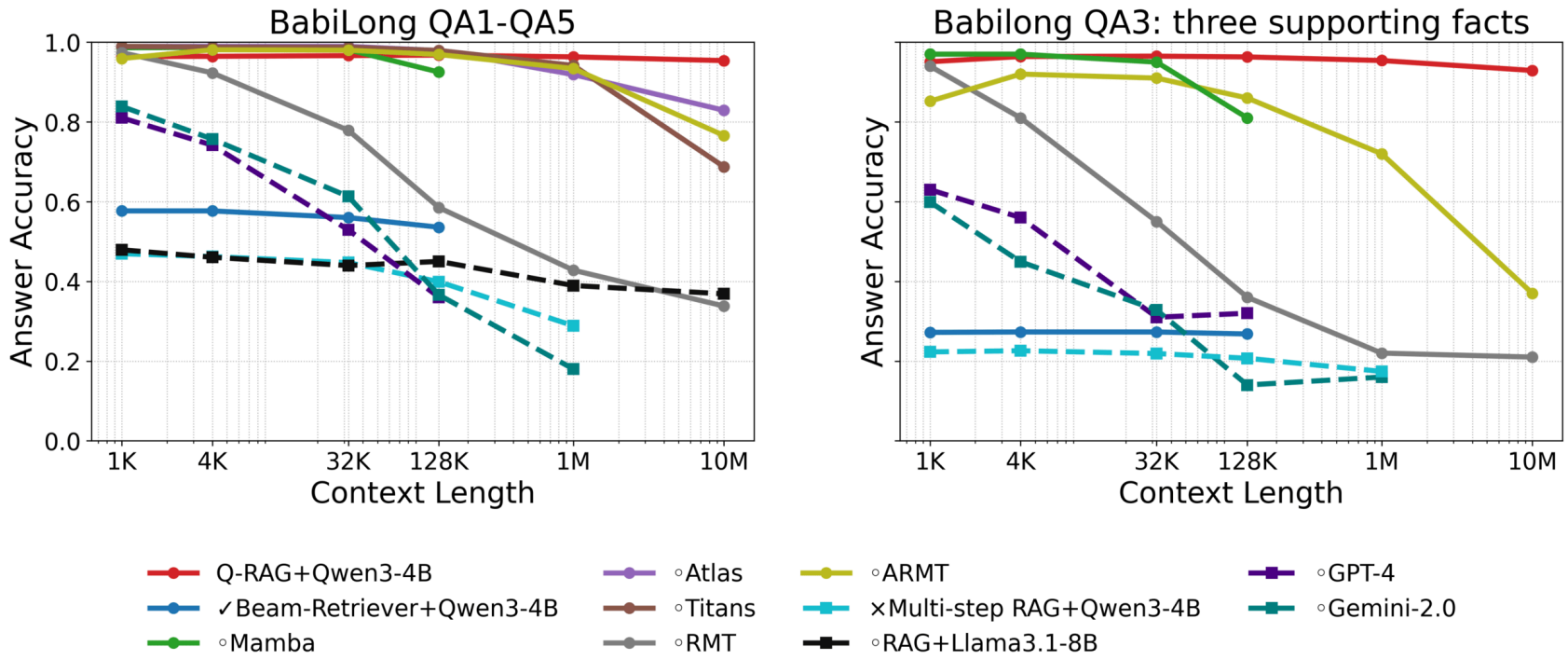


$a_1 = \texttt{chunk\#15}$

$s_2 = \texttt{[query, chunk\#15, chunk\#40]}$



Results: BabiLong



BabiLong is commonsense and temporal reasoning benchmark for ultra-long contexts

- Q-RAG achieves state of the art performance (from 128k to 10M tokens)
- The gap is bigger on harder tasks like (QA3)

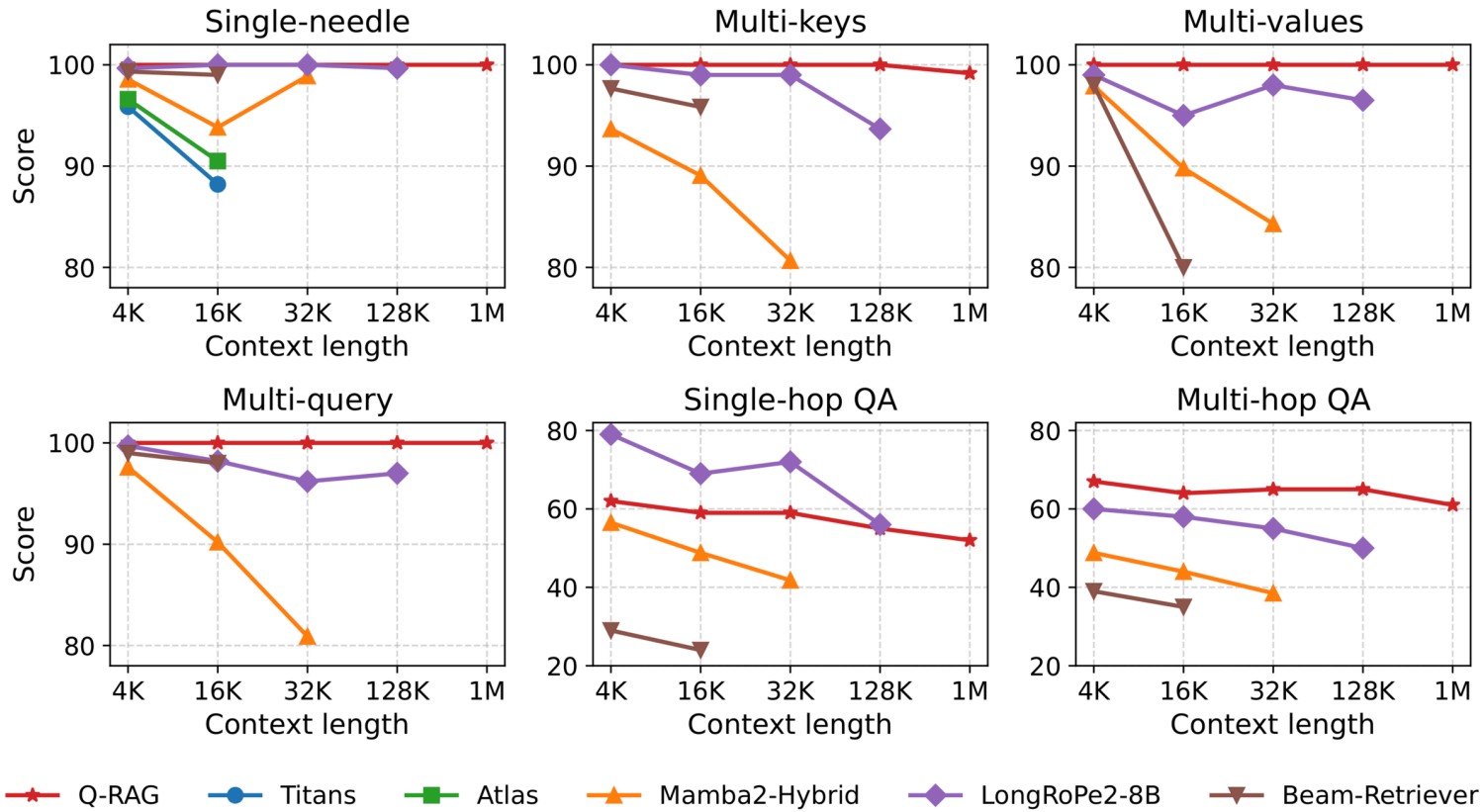
Results: RULER

RULER benchmark:

- different types of NIAH tasks
- couple of Open domain QA tasks

Results:

- Near-perfect retrieval on NIAH
- Generalizes to 1M tokens
- Best on multi-hop QA
- Only mild degradation with length increase



Results: Open Domain QA

Results on Open-Domain QA

- Competitive results on HotPotQA and MuSiQue
- Cheaper training than LLM fine-tuning methods
- Faster inference than Beam Retriever and Graph based methods

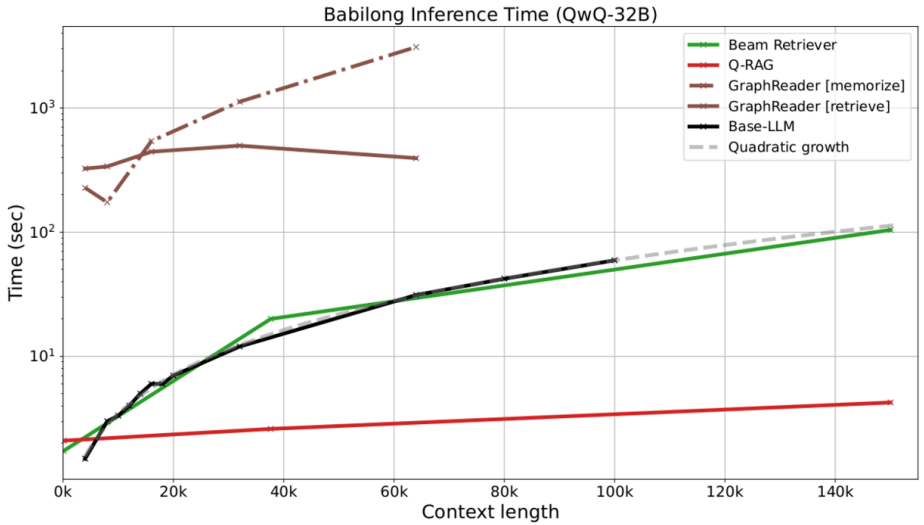
Table 2: Comparison of methods on HotPotQA and Musique benchmarks. Bold text and underline denote the best and second best scores respectively.

Methods	HotPotQA				Musique (OOD)				Avg	
	Fact F1	**Fact EM**	**Ans F1**	**Ans EM**	**Fact F1**	**Fact EM**	**Ans F1**	**Ans EM**	**Ans F1**	**Ans EM**
Finetuned on HotPotQA										
Plan Q-RAG	<u>0.95</u>	<u>0.91</u>	<u>0.76</u>	<u>0.60</u>	0.69	0.53	<u>0.51</u>	0.36	**0.64**	**0.48**
Q-RAG	0.93	0.89	0.76	0.59	**0.71**	**0.55**	**0.52**	<u>0.37</u>	**0.64**	**0.48**
✓Beam-Retriever	**0.97**	**0.94**	**0.77**	**0.61**	0.61	0.36	0.40	0.27	0.59	0.44
✓Search-r1	0.81	0.66	0.65	0.52	**0.71**	**0.55**	0.51	**0.39**	0.58	0.46
°RAG-RL	0.82	–	0.69	–	0.65	–	0.47	–	0.58	–
×Multi-step RAG w.o. FT	0.73	0.54	0.65	0.50	0.51	0.30	0.40	0.27	0.53	0.39
Zero Shot methods										
✓GraphReader	–	–	0.46	0.24	–	–	0.40	0.20	0.43	0.22
✓Single step RAG	–	–	0.53	0.39	–	–	0.28	0.17	0.41	0.28

Training and Inference Cost

Q-RAG fine-tunes only the retriever/embedder and keeps the answering LLM frozen.

- Main experiments fit on a single A100-80GB GPU
- Fine-tuning of one embedder model takes around 12 hours
- Faster inference than Beam Retriever and Graph based methods



Multi-step RAG methods	GPU Memory for fine-tuning
Q-RAG	A100-80GB ×1
Beam Retriever	A100-80GB ×1
Search-R1	H100-80GB ×8
RAG-RL	H100-80GB ×8

Conclusion

- Fine-tuning the embedder for multi-step retrieval is cheaper than LLM
- Competitive Quality on Open Domain QA
- State of the Art on BabiLong and RULER
- The main bottleneck is dependence on support-chunk annotations

An open question for future work:

- can generated data or LLM-based rewards remove dependence on support-chunk annotations?

Thanks For Your Attention!

CODE

PAPER

CHECKPOINTS

Contacts: griver29@gmail.com or n.buzun@seevia.ai