Inference

Monday, 14 August 2023 12:57 AM

https://hamel.dev/notes/llm/inference/03_inference.html

Llama-v2-7b benchmark: *batch size = 1, max output tokens = 200*

			avg tok/sec	avg time (seconds)	avg output token count
platform	**options**	**gpu**			
CTranslate2	float16 quantization	A6000	44.8	4.5	200.0
	int8 quantization	A6000	62.6	3.2	200.0
HF Hosted Inference Endpoint	-	A10G	30.4	6.6	202.0
HuggingFace Transformers (no server)	-	A6000	24.6	7.5	181.4
	nf4 4bit quantization bitsandbytes	A6000	24.3	7.6	181.4
TGI	-	A6000	21.1	9.5	200.0
	quantized w/ GPTQ	A6000	23.6	8.8	200.0
	quantized w/ bitsandbytes	A6000	1.9	103.0	200.0
text-generation-webui	exllama	A6000	77.0	1.7	134.0
vllm	-	A100 (on Modal Labs)	41.5	3.4	143.1
		A6000	46.4	3.8	178.0

Framework	Tokens Per Second	Query per second	Latency	Adapters	Quantisation	Variable precision	Batching	Distributed Inference	Custom models	Token streaming	Prometheus metrics
vLLM	115	0.94	4.8 s	✗	✗	✗	✓	✓	✓	✓	✗
Text generation inference	50	0.26	4.8 s	✗	✓	✓	✓	✓	✓	✓	✓
CTranslate2	93	0.55	4.5 s	✗	✓	✓	✓	✓	✓	✓	✗
DeepSpeed-MII	80	0.47	2.5 s	✗	✗	✓	✓	✓	✗	✗	✗
OpenLLM	30	0.15	6.6 s	✓	✓	✓	✗	✗	✓	✗	✓
Ray Serve	28	0.15	7.1 s	✗	✓	✓	✓	✓	✓	✗	✓
MLC LLM	25	0.13	7.2 s	✗	✓	✗	✗	✗	✓	✓	✗

Comparison of frameworks for LLMs inference

https://kipp.ly/transformer-inference-arithmetic/

https://finbarr.ca/how-is-llama-cpp-possible/

https://betterprogramming.pub/frameworks-for-serving-llms-60b7f7b23407

Framework for serving LLM:
https://betterprogramming.pub/frameworks-for-serving-llms-60b7f7b23407

Inference Arithmetic:
https://kipp.ly/transformer-inference-arithmetic/

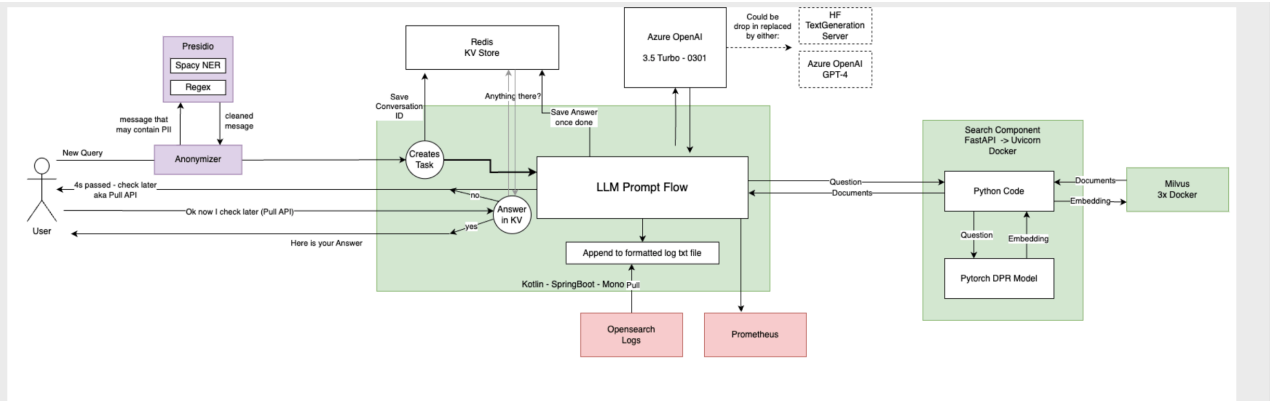
Speed up Inference :
https://betterprogramming.pub/speed-up-llm-inference-83653aa24c47

LLM Inference :
https://github.com/huggingface/text-generation-inference

Building LLM applications for production:
https://huyenchip.com/2023/04/11/llm-engineering.html#prompt_evaluation

Caching:
https://gptcache.readthedocs.io/en/latest/usage.html

Run LLM Locally:
https://medium.com/@meta_heuristic/how-to-use-private-llm-gpt4all-with-langchain-9f890e6960f3
https://lightning.ai/courses/deep-learning-fundamentals/unit-8.0-natural-language-processing-and-large-language-models/8.7-a-large-language-model-for-classification/



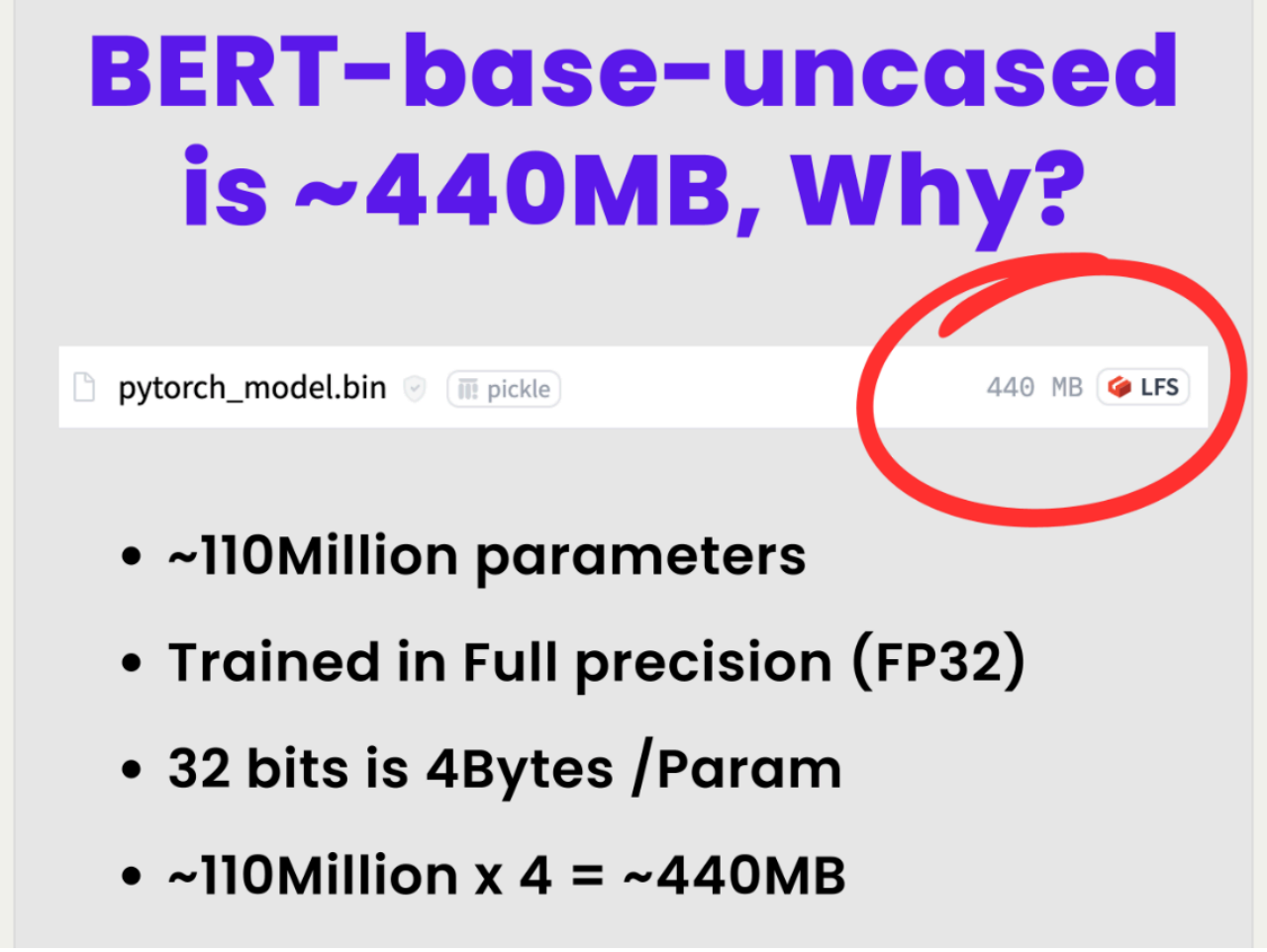
Sentence Bert
https://www.sbert.net/docs/training/overview.html

https://www.sbert.net/docs/training/overview.html

Sentence Bert:

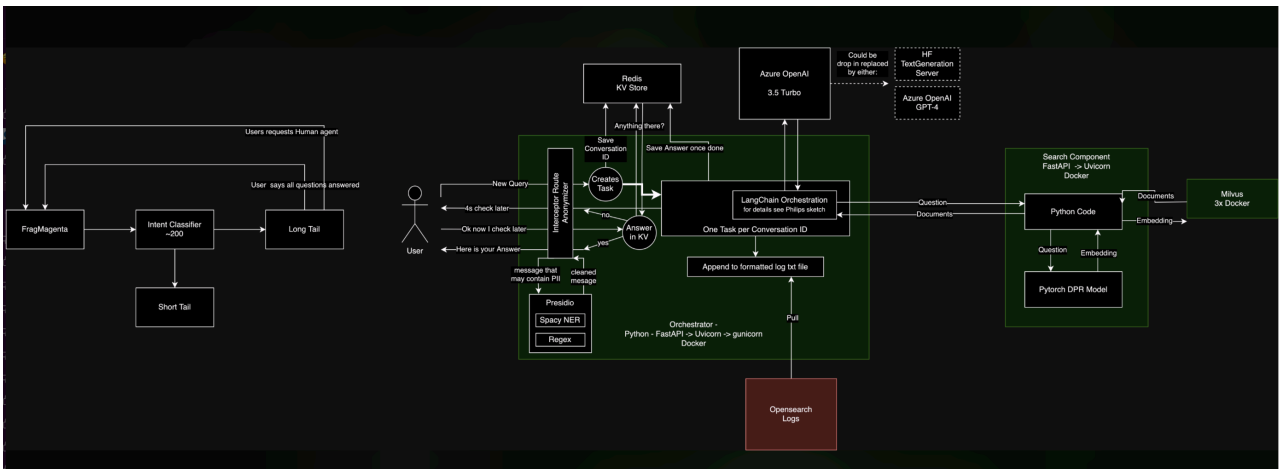
Retrieve and Re-rank:
https://www.sbert.net/examples/applications/retrieve_rerank/README.html

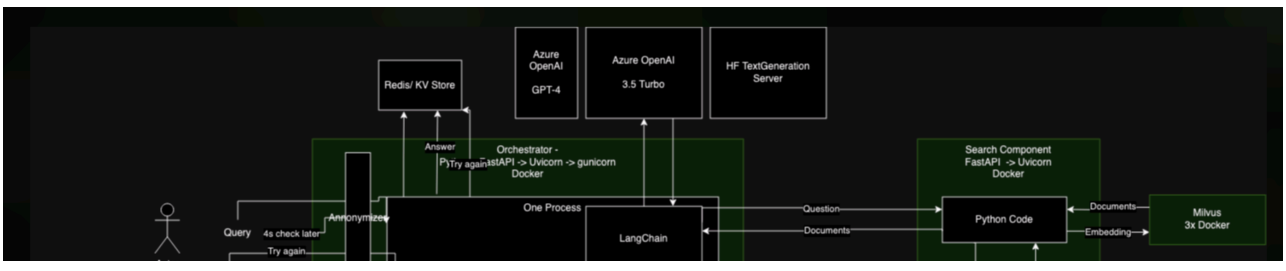
Training Overview:
https://github.com/UKPLab/sentence-transformers/blob/master/docs/training/overview.md

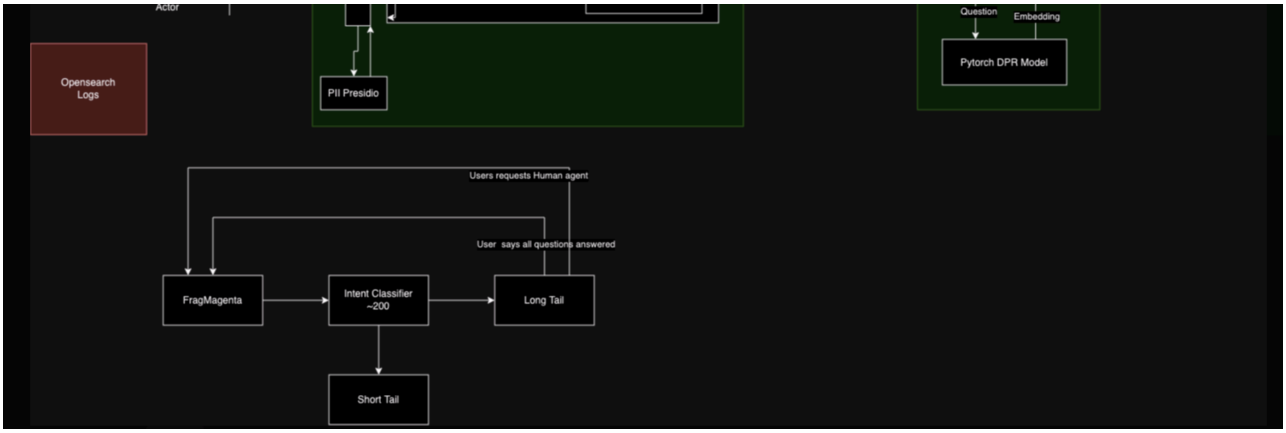


pytorch_model.bin 892 MB LFS

- ~220Million parameters
- Trained in Full precision (FP32)
- 32 bits is 4Bytes /Param
- ~220Million x 4 = ~880MB







EVAluation:

https://explodinggradients.com/all-about-evaluating-large-language-models

https://huyenchip.com/2023/04/11/llm-engineering.html#prompt_evaluation

https://www.vellum.ai/landing-pages/14-day-trial

RAGAS evaluation:
https://medium.aiplanet.com/evaluate-rag-pipeline-using-ragas-fbdd8dd466c1

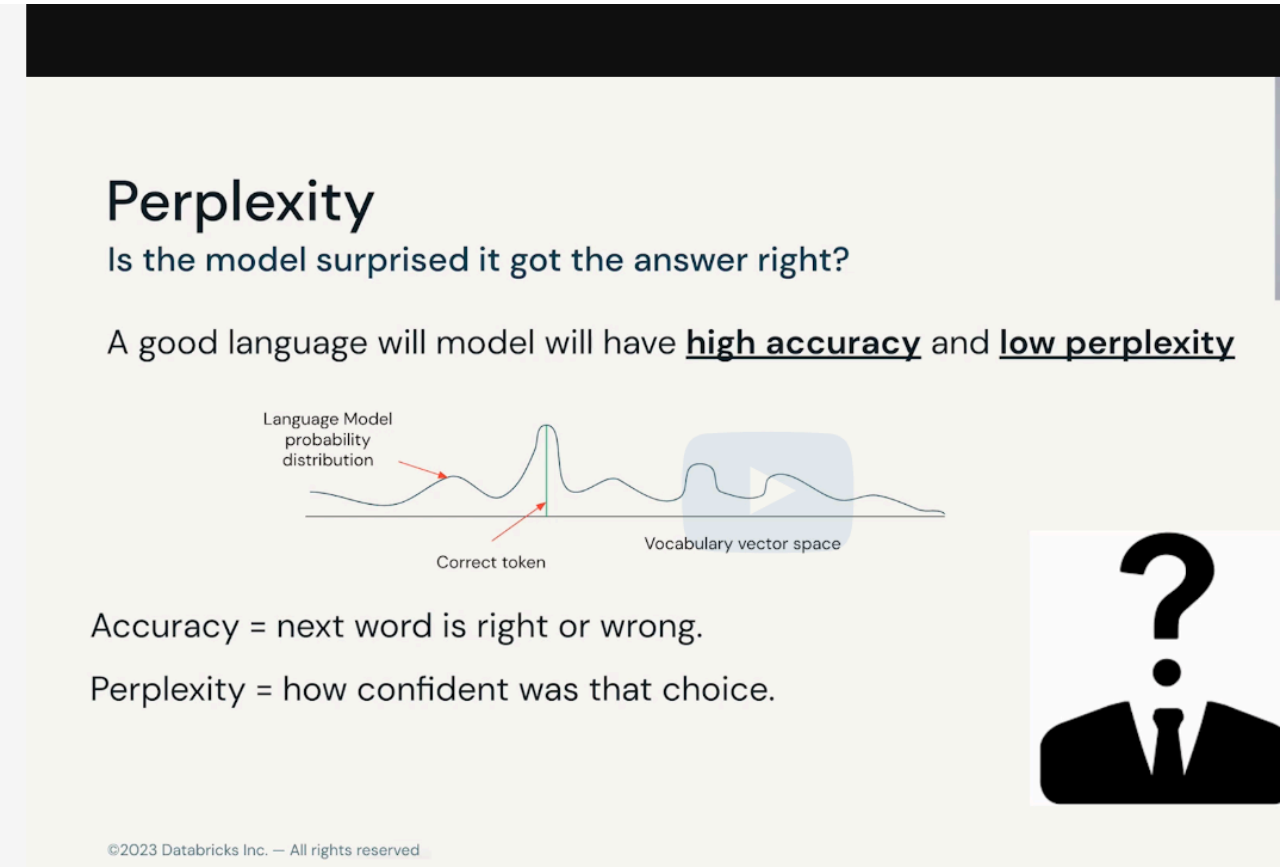
Information retrieval metrics:
https://amitness.com/2020/08/information-retrieval-evaluation/

SBERT:
https://medium.com/mlearning-ai/semantic-search-with-s-bert-is-all-you-need-951bc710e160

RAG evaluation:
https://github.com/explodinggradients/ragas
https://sangeethavenkatesan.substack.com/p/ragas-retrieval-augmented-generation?sd=pf

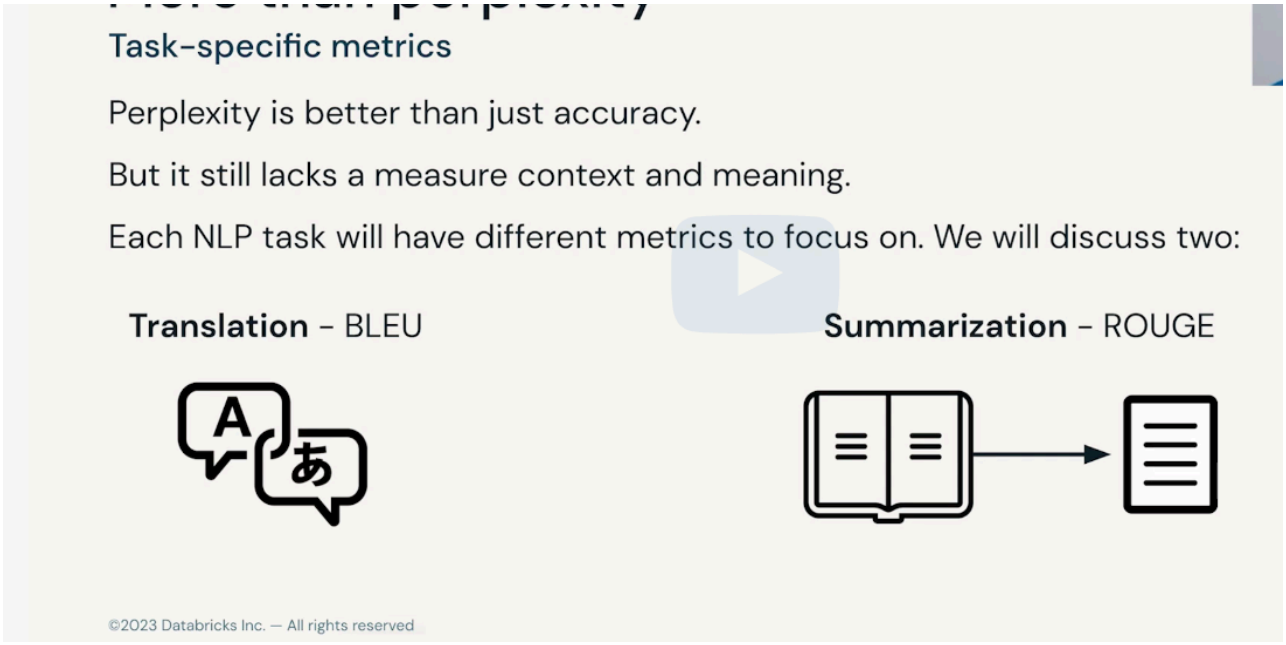
Factuality evaluation:
https://github.com/GAIR-NLP/factool?trk=public_post_comment-text

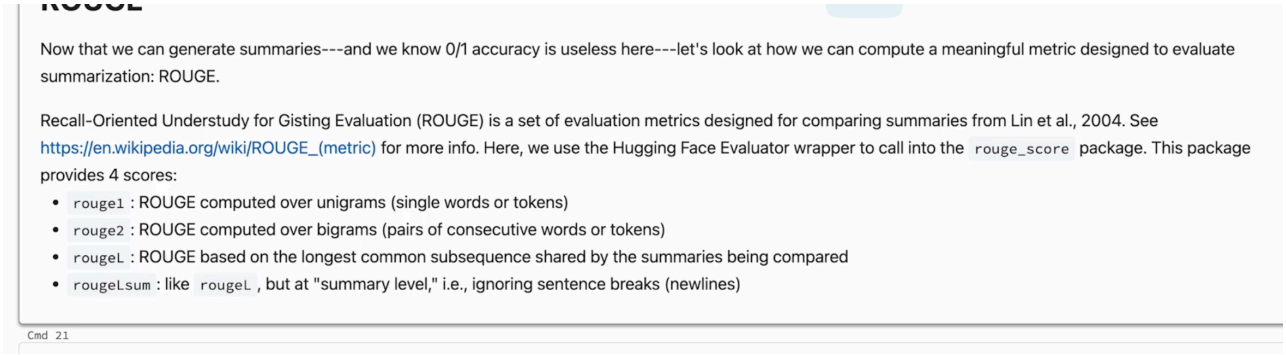
Data Cleaner PII removal :
https://lilacml.com/index.html



1. Perplexity is really just how the spread of the probability distribution is over the tokens that its trying to

   predict. If we have a very sharp probability distribution of our tokens that means it has a very low perplexity and it knows
2. then it's very confident that that is
3. the token it should be picking. Whether
4. or not that token is correct or not
5. depends on the accuracy so really what a
6. good language model will have is high
7. **accuracy and very low perplexity**





RAG :

The so called BM25 method is the baseline for all these retrieval methods. It is a method based on "keyword density". It is good, very old and very fast, but it can be better.

Our models (so called DPR - dense passage retriever) must beat this baseline.

BM25 vs TF-IDF:

https://abishek21.medium.com/building-your-favourite-tv-series-search-engine-information-retrieval-using-bm25-ranking-8e8c54bcdb38

But! what TF-IDF misses is the length of the document.

For an example, A million word document obviously has more chance capturing all the words, So it has

more chance of getting ranked at the top always for any given set of keyword

BM25

BM25 ranking solves this issue by introducing a parameter for relative length of document.

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)},$$

The formula looks quite Complex, but let's break it down

The BM25 for the query "Machine Learning" would be summation of BM25Score(Machine)+BM25Score(Learning)

The first part in the formula is the IDF of the term.

The second part in the formula represents the TF of the word which is normalized by the length

SELECT ENCODER FOR Semantic search:

https://pakodas.substack.com/p/llm-chronicles-8-how-to-select-encoder?trk=public_post_comment-text

Fine tuning embedding for RAG:

https://medium.com/llamaindex-blog/fine-tuning-embeddings-for-rag-with-synthetic-data-e534409a3971

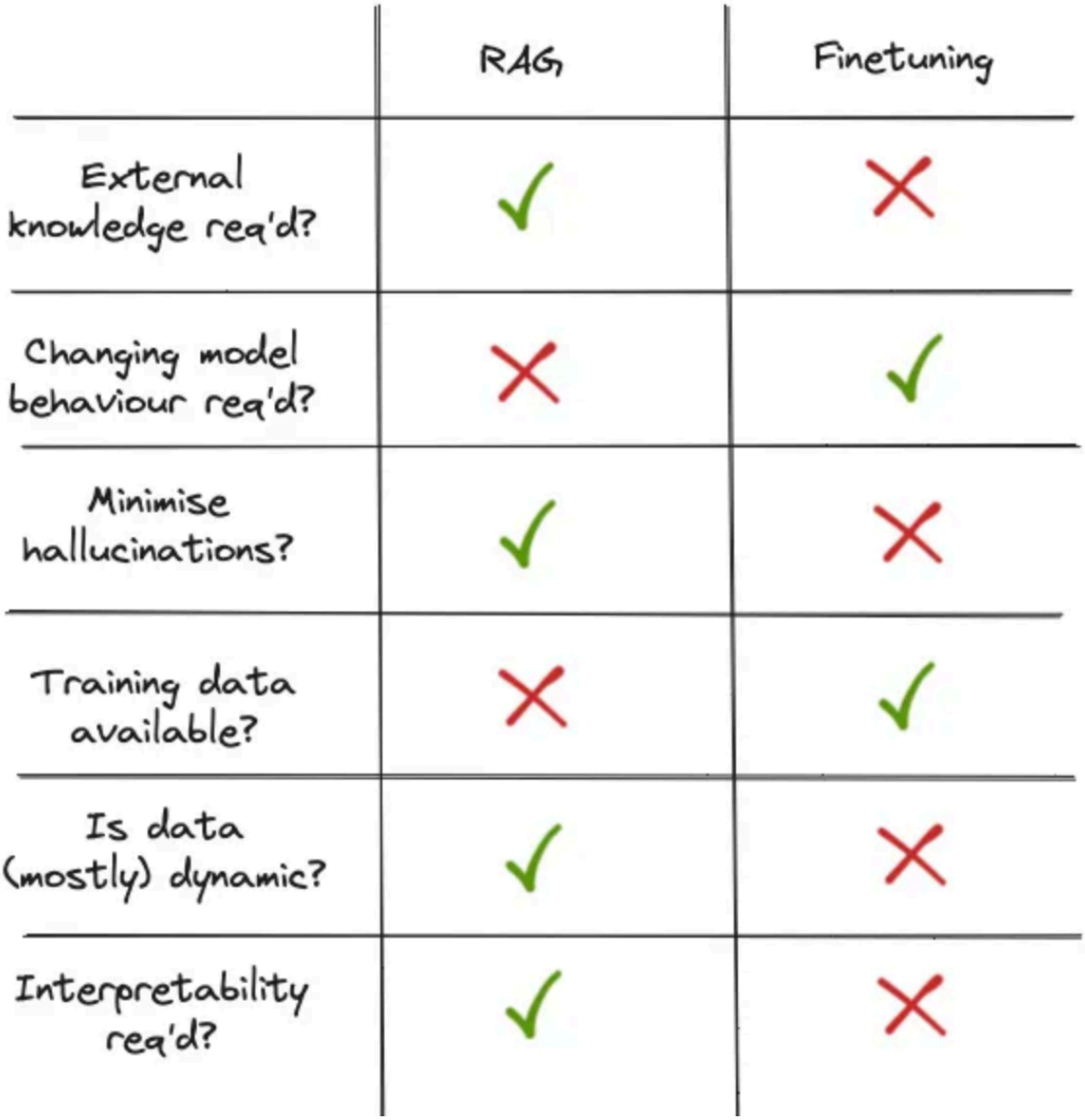
https://betterprogramming.pub/fine-tuning-your-embedding-model-to-maximize-relevance-retrieval-in-rag-pipeline-2ea3fa231149

SCALING RAG:

https://medium.com/@alcarazanthony1/optimizing-dialog-llm-chatbot-retrieval-augmented-generation-with-a-swarm-architecture-f41749d80919

RAG vs FT:

https://towardsdatascience.com/rag-vs-finetuning-which-is-the-best-tool-to-boost-your-llm-application-94654b1eaba7?source=user_profile---------0----------



Pascal Biese • Following
Daily LLM Research Nuggets 📲🤗 | AI/ML Engineer | NLP | LLMOps
2d •

👩‍🚒 Retrieve-and-Generate: Beyond Concatenation

Basically everyone is using Retrieval-augmented Generation (RAG) by now for leveraging Large Language Models (LLMs) on some kind of external data.

But many - including myself - are wondering how to best design their LLM workflows. Especially when it to the details and the quantity of prompts. Is one query enough? Do I want a summary first to reduce complexity step-by-step? Would a voting system work better? Not to mention all the X-of-Thought strategies that evolved from Chain-of-Thought (CoT).

The base method for RAG is usually simple concatenation - before or after the instructions. This new paper from **Salesforce** is exploring a few other options:

1. Post-Fusion (PF): Querying an answer for each retrieved passage. Then choosing from the answer pool.
2. Pruning Prompt: Instructing the model to omit passages when it doesn't "know" an answer.
3. Summary Prompt: Querying for a summary of all retrieved passages before answering the question.
4. PF + Concat/Concat + PF: Combination of the base method and PF with different implementations depending on the order of steps.

They found 4.) to work best for their data. %Unk stands for the %-age of retrieved passages that don't include an answer - which is pretty high with 20% for the base RAG method. I personally have used summarization prompts in the past and it seems that my intuition about them hasn't been totally wrong.

I might look into PF + Concat though since it's using up to 50% less tokens than Concat + PF and summarization prompting at similar performance.

[arXiv] **https://lnkd.in/daHJMJFe**

Questions:

1) Which encoder we are using in DPR?
2) Did we try to evaluate (custom evaluation for different encoders against

2) DID we try to evaluate /custom evaluation for different encoders against BM25?

3) Are we using multilingual sentence encoder ? Or user qury ==> english ==> response ==> user query language

4) How we are evaluating RAG system?