LLaMa LLM

•••

Adrien Dubois

09/13/2023

Managing Expectations

Managing Expectations

Cut me off and ask questions!

General Overview

LLaMa (Large Language Model Meta AI)

Authors	● Meta AI
	● LLaMa1: Feb 2023, LLaMa2: Jul 2023
Purpose	● Foundational Model for LLM work
	● LLaMa2: Research and Commercial Free Use
Specs	● LLaMa1: 7B, 13B, 33B, 65B
	● LLaMa2: 7B, 13B, 70B
	● M.P.: 1, 2, 8
Other info	● Trained on 2T Tokens (vs. 13T in GPT4)
	● Batch size: 4M tokens
	● Context Length 4096 (vs. up to 32,768 in GPT4)
	● Training time: 3.3M GPU hours (A100-80GB)

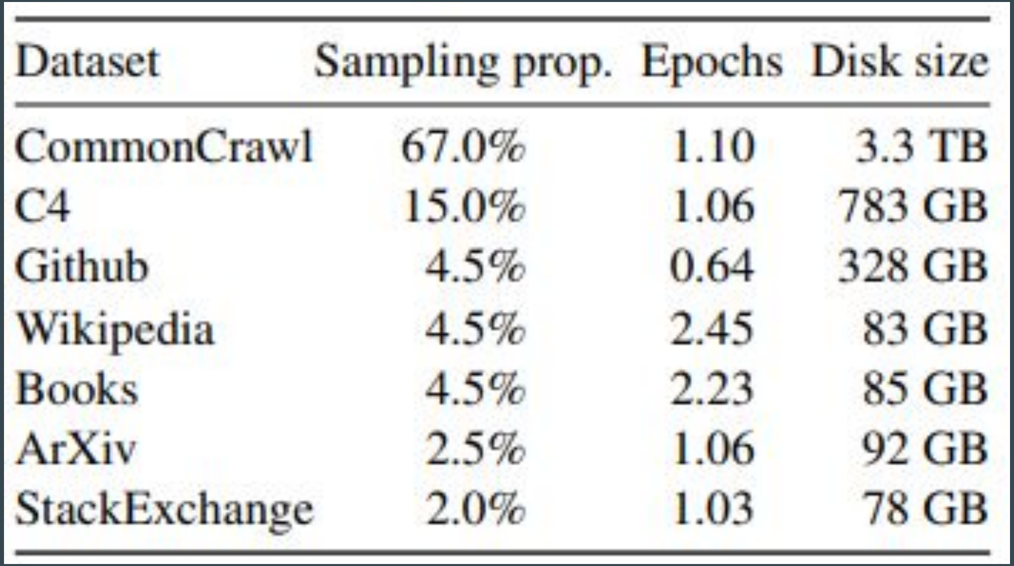
Dataset

Dataset Overview:

LLaMa focused on increased dataset instead of increasing parameters

- Argument 1: LLMs are undertrained, so they kept parameter size small and focused on increasing dataset size. [1]
- Argument 2: Inference cost is much higher than training cost [2]

2T tokens are split from these sources



Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

CommonCrawl

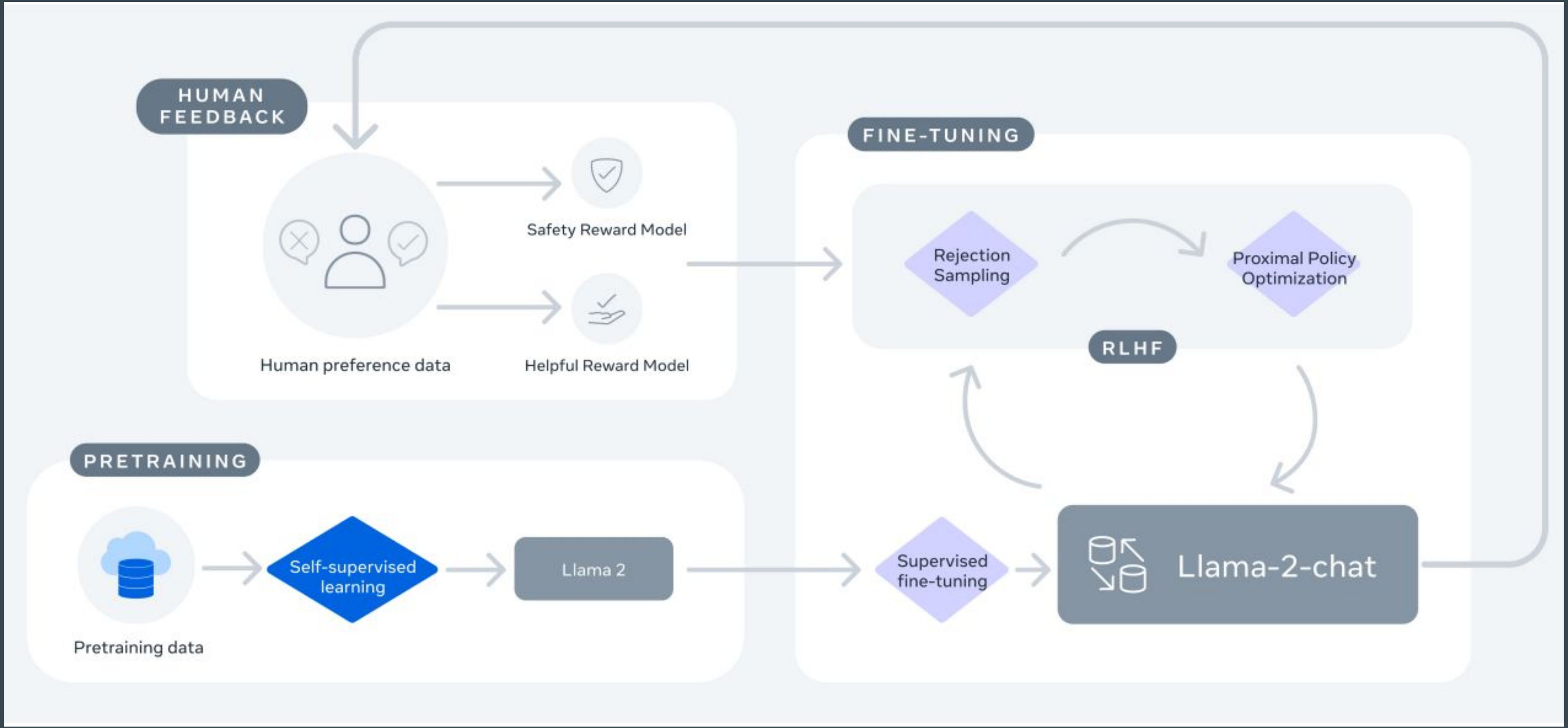
- Taken from 2017-2020
- Tools used:
    - CCNet pipeline to de-duplicate information at the line level
    - FastText linear classifier to remove non-English pages
    - N-Gram model to filter low quality pages
    - Trained Linear Classifier to remove non-“reference” Wikipedia pages

Other contributions

- C4 Dataset (cleaned up common crawl)
    - “During exploratory experiments, we observed that using diverse pre-processed CommonCrawl data sets improves performance”[2]
    - Same pre-processing except the CCNet de-duplication also does quality filtering
        - This is done heuristically by looking at the presence of punctuation, and the number of words per page (basically quality of grammar)
- GitHub Code:
    - Public repositories using Google BigQuery
    - Filtered based on:
        - License: Apache, MIT and BSD
        - Deduplication at the file level for exact matches
        - Low quality files: heuristics based on line length and proportion of alphanumeric chars.
        - Removed boilerplate, headers, etc. using regex

Model Architecture

LLaMa2 Pipeline



LLaMa2 Code Architecture - generation.py [3]

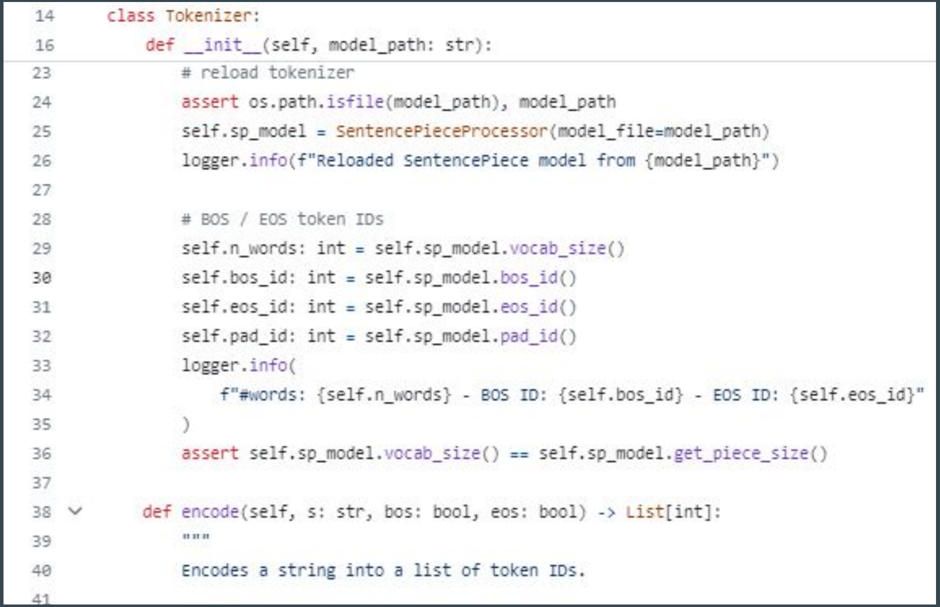
Init:

1. Initialize Hyper-parameters
2. Set up Model Parallelism
3. Initialize Tokenizer
4. Initialize Transformer

Generation:

1. Scan through input prompt by context length
2. Tokenizes input
3. Runs the transformer
4. Optional calculation of log probabilities etc

Tokenizer - SentencePiece [4]

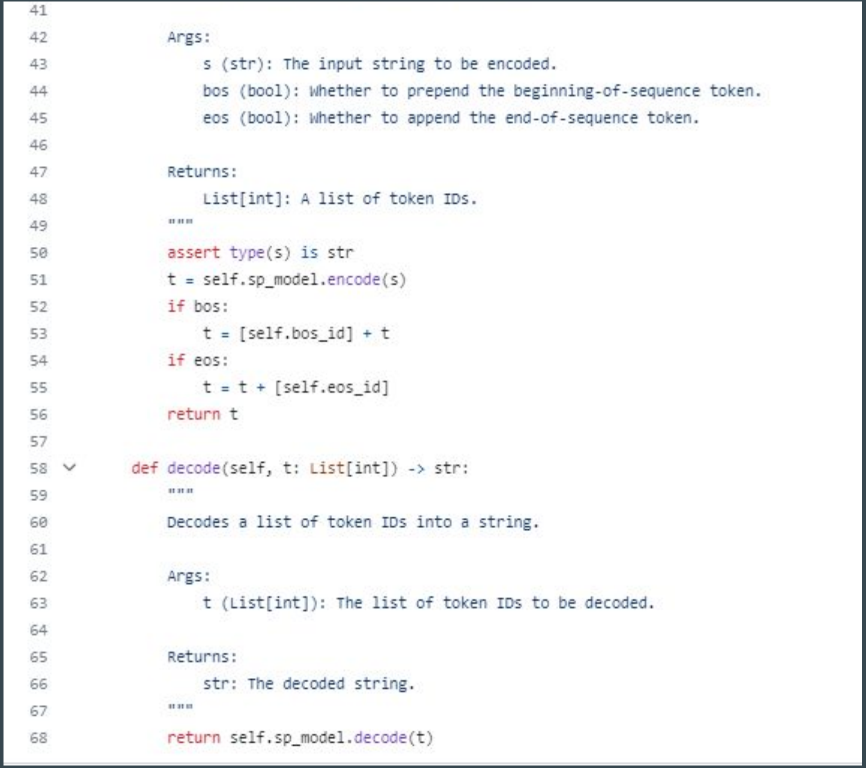


```python
class Tokenizer:
    def __init__(self, model_path: str):
        # reload tokenizer
        assert os.path.isfile(model_path), model_path
        self.sp_model = SentencePieceProcessor(model_file=model_path)
        logger.info(f"Reloaded SentencePiece model from {model_path}")

        # BOS / EOS token IDs
        self.n_words: int = self.sp_model.vocab_size()
        self.bos_id: int = self.sp_model.bos_id()
        self.eos_id: int = self.sp_model.eos_id()
        self.pad_id: int = self.sp_model.pad_id()
        logger.info(
            f"#words: {self.n_words} - BOS ID: {self.bos_id} - EOS ID: {self.eos_id}"
        )
        assert self.sp_model.vocab_size() == self.sp_model.get_piece_size()

    def encode(self, s: str, bos: bool, eos: bool) -> List[int]:
        """
        Encodes a string into a list of token IDs.

```



```python

        Args:
            s (str): The input string to be encoded.
            bos (bool): Whether to prepend the beginning-of-sequence token.
            eos (bool): Whether to append the end-of-sequence token.

        Returns:
            List[int]: A list of token IDs.
        """
        assert type(s) is str
        t = self.sp_model.encode(s)
        if bos:
            t = [self.bos_id] + t
        if eos:
            t = t + [self.eos_id]
        return t

    def decode(self, t: List[int]) -> str:
        """
        Decodes a list of token IDs into a string.

        Args:
            t (List[int]): The list of token IDs to be decoded.

        Returns:
            str: The decoded string.
        """
        return self.sp_model.decode(t)
```

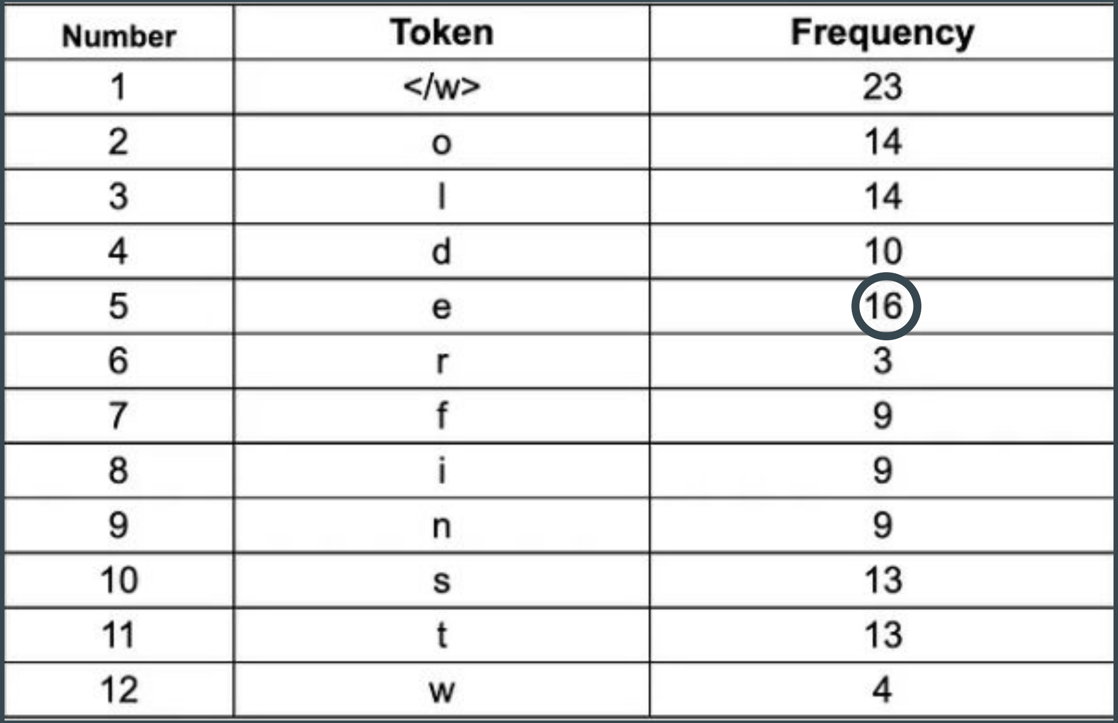
Tokenizer: How does Byte Pair Encoding Work?

- Split words into uni-grams
- Add <\w> at the end of each word
- Create table that holds occurrence for uni-grams
- Recursively join the most frequent uni-gram pairs together until you have attained the required vocabulary size

- A naive-scan method is O(N^2) but SentencePiece includes an implementation based on BinaryHeaps to get O(Nlog(n))
- BPE is useful to reduce the required size of the vocabulary

Tokenizer: BPE Example

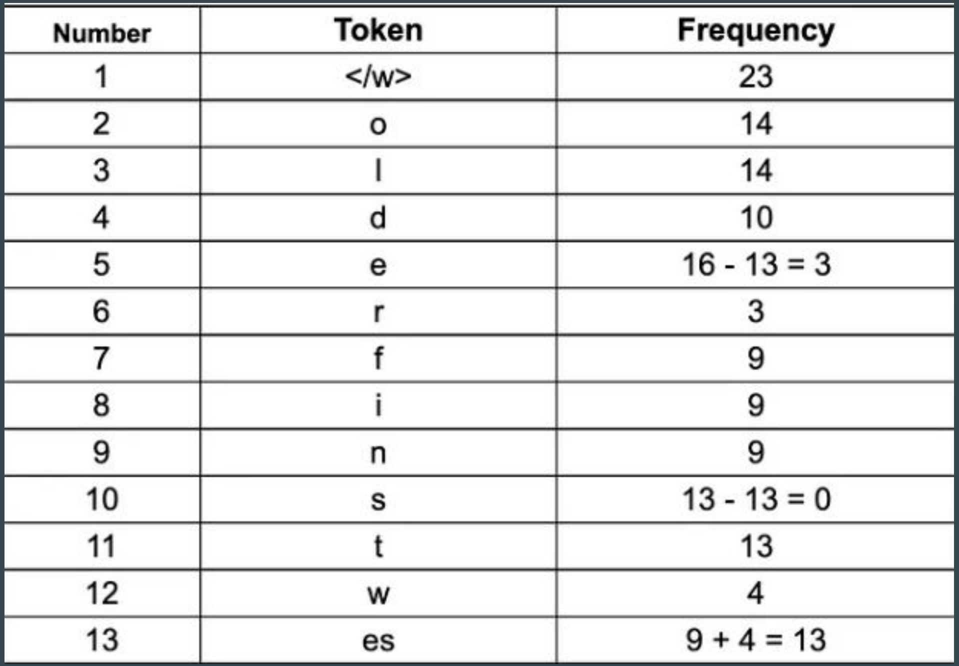
- Input sentence is: {“old”: 7, “older”: 3, “finest”: 9, “lowest”: 4}



Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13
11	t	13
12	w	4

Tokenizer: BPE Example

- The most common bi-gram with e is “es” so we merge them into an “es” token



Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13
12	w	4
13	es	9 + 4 = 13

Tokenizer: SentencePiece

- Whitespace is counted as a separate token
- Vocabulary size is only 8k
- Involves a normalizer, trainer, encoder and decoder
- The normalizer is a module to normalize semantically-equivalent unicode characters into canonical forms (Unicode NFKC here)
- On top of BPE to create subwords, the subwords are also regularized and randomly sampled.
    - Hello will thus be tokenized in such a way that maximizes the likelihood estimate.
        1. Initialize the unigram probabilities
        2. Compute the most probable unigram sequence given the current probabilities.
        3. Given the current tokenization, recompute the unigram probabilities by counting the occurrence of all subwords in the tokenization.

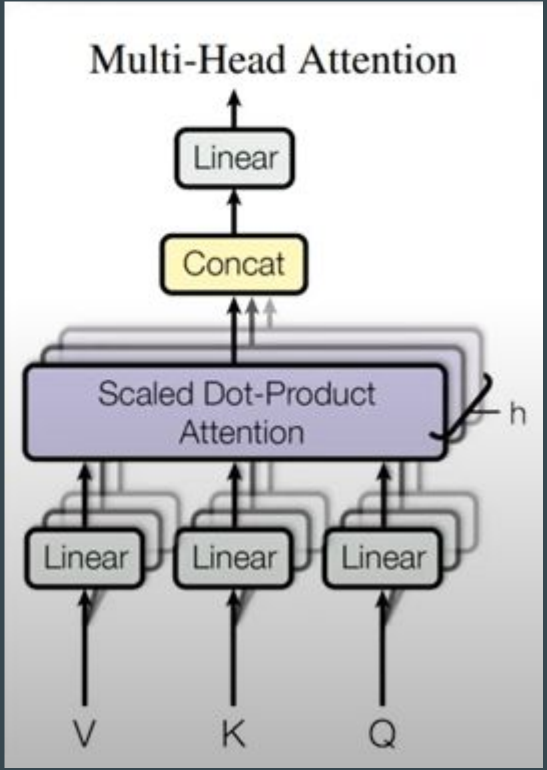
Transformer:

- LLaMa is built through 4 main classes:
    - Rotational Embeddings
        - A form of positional encodings to solve sequence invariance issue
        - Joins absolute and relative positional encodings into one framework
    - Feed-forward
        - Linear Layers running in parallel on multi-gpu
    - RMSNorm
    - Multi-Headed attention

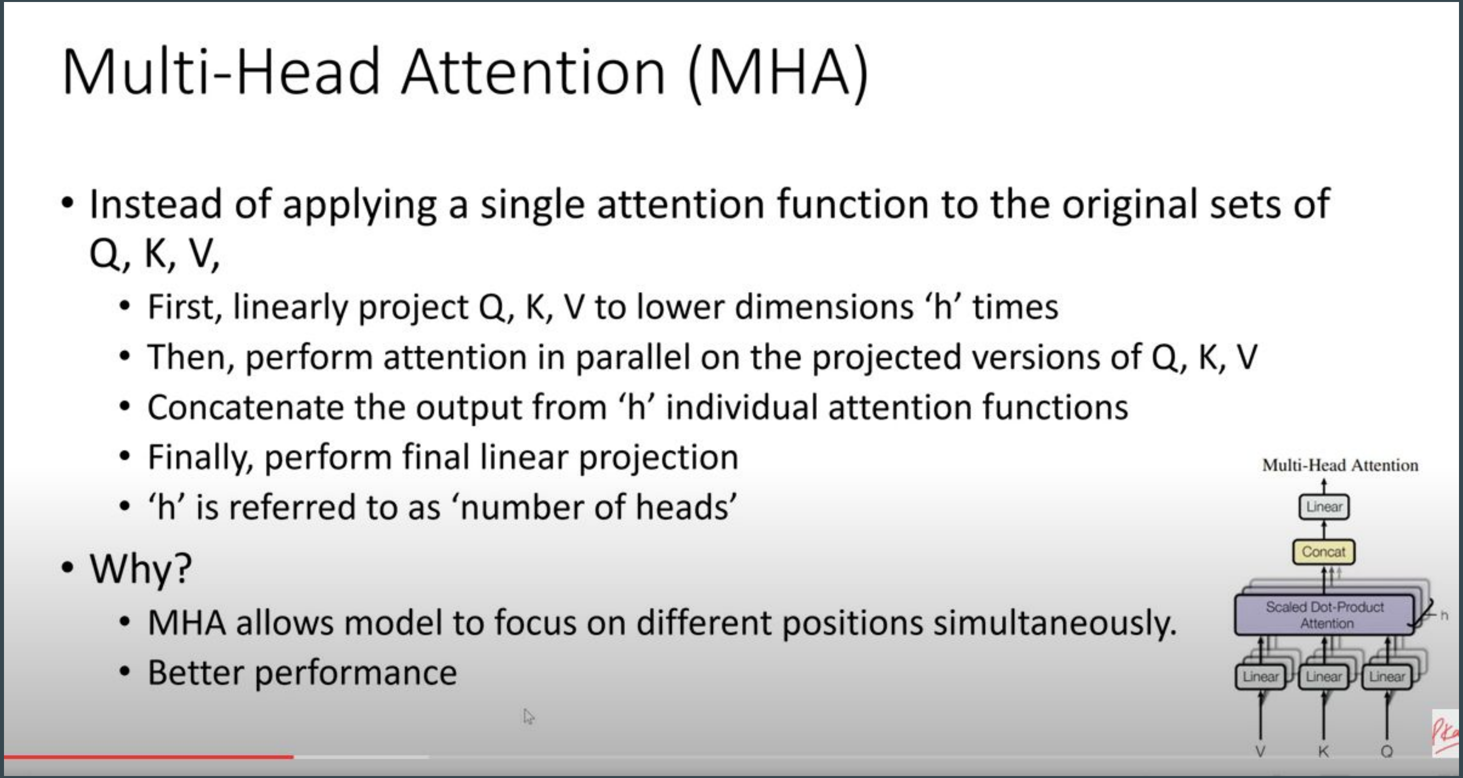
Transformer: RMS Norm

- Replaces the commonly used LayerNorm
- We use LayerNorm because:
  - LayerNorm helps the Attention component craft an attention query such that all keys are equally accessible. It does this by projecting the key vectors onto the same hyperplane, thus enabling the model to align the queries to be orthogonal to the keys. And in doing so, obviates the need for the Attention component to learn how to do this on it's own. [5]
  - Scales input such that every key can have the highest attention and no key can be in the un-selectable zone.
- Instead it regularizes the summed inputs to a neuron in one layer according to root mean square (RMS)
- Advertised to give the model re-scaling invariance and implicit learning rate adaptation ability

Transformer: Multi-Headed Attention

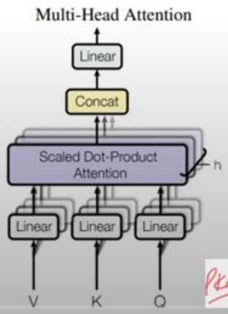


Transformer: Multi-Headed Attention



Multi-Head Attention (MHA)

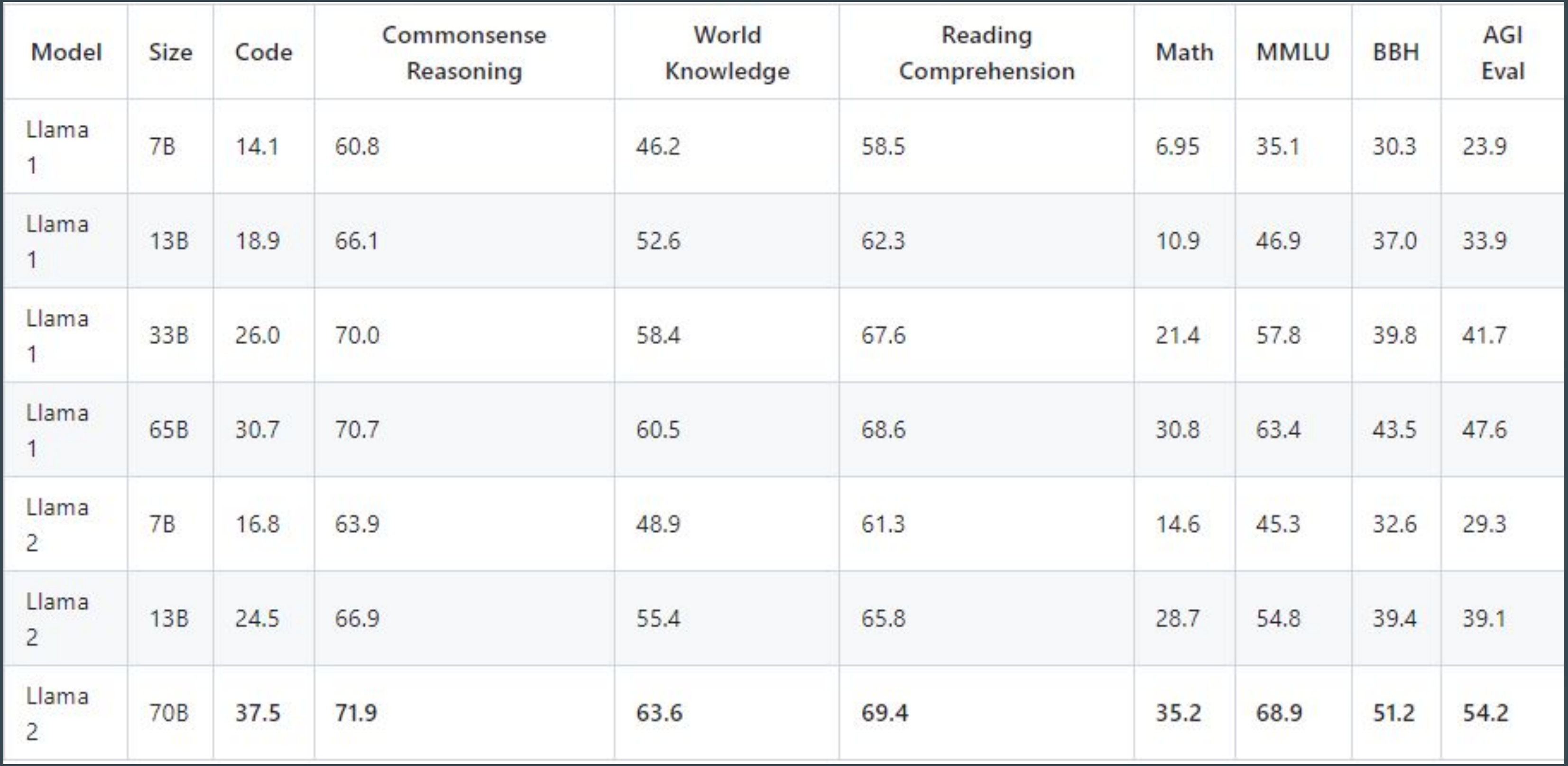
- Instead of applying a single attention function to the original sets of Q, K, V,
  - First, linearly project Q, K, V to lower dimensions ‘h’ times
  - Then, perform attention in parallel on the projected versions of Q, K, V
  - Concatenate the output from ‘h’ individual attention functions
  - Finally, perform final linear projection
  - ‘h’ is referred to as ‘number of heads’
- Why?
  - MHA allows model to focus on different positions simultaneously.
  - Better performance



For more information, look at slides 9-23:
https://engineering.purdue.edu/DeepLearn/pdf-kak/Transformers.pdf

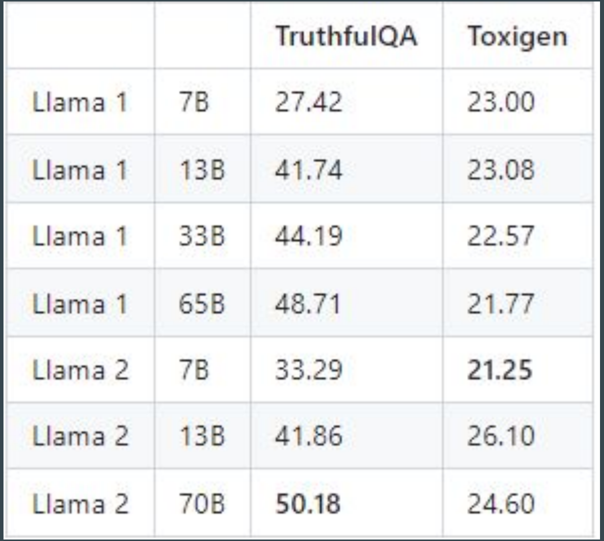
Results

Results:

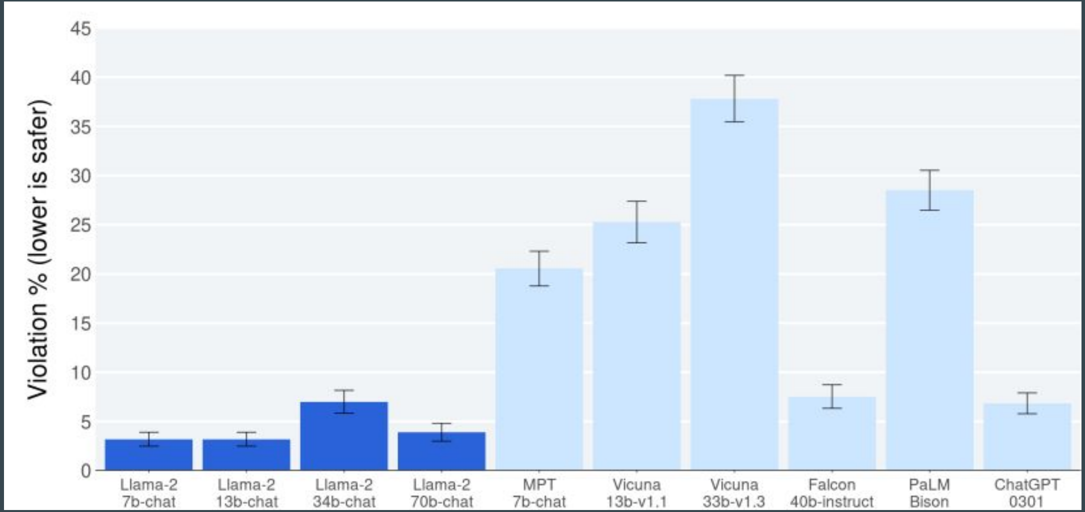


Model	Size	Code	Commonsense Reasoning	World Knowledge	Reading Comprehension	Math	MMLU	BBH	AGI Eval
Llama 1	7B	14.1	60.8	46.2	58.5	6.95	35.1	30.3	23.9
Llama 1	13B	18.9	66.1	52.6	62.3	10.9	46.9	37.0	33.9
Llama 1	33B	26.0	70.0	58.4	67.6	21.4	57.8	39.8	41.7
Llama 1	65B	30.7	70.7	60.5	68.6	30.8	63.4	43.5	47.6
Llama 2	7B	16.8	63.9	48.9	61.3	14.6	45.3	32.6	29.3
Llama 2	13B	24.5	66.9	55.4	65.8	28.7	54.8	39.4	39.1
Llama 2	70B	**37.5**	**71.9**	**63.6**	**69.4**	**35.2**	**68.9**	**51.2**	**54.2**

Results:



		TruthfulQA	Toxigen
Llama 1	7B	27.42	23.00
Llama 1	13B	41.74	23.08
Llama 1	33B	44.19	22.57
Llama 1	65B	48.71	21.77
Llama 2	7B	33.29	**21.25**
Llama 2	13B	41.86	26.10
Llama 2	70B	**50.18**	24.60



Safety Evaluation

"Human raters judged model generations for safety violations"

Next Steps

Next Steps

1. **Increase Accuracy**
   - Instruction Tuning
   - Improvements in RLHF

2. **Safety improvements**
   - Most of the effort in LLaMa2 was to improve safety and truthfulness
   - Seems to be the big research space they are investing into.

Thank You

References

[1] https://arxiv.org/abs/2203.15556

[2] https://arxiv.org/pdf/2302.13971.pdf

[3] https://github.com/facebookresearch/llama/blob/main/llama/generation.py

[4] https://github.com/facebookresearch/llama/blob/main/llama/tokenizer.py

[5] https://arxiv.org/abs/2305.02582