An overview of Llama 3.1

Edwin Brown, Ryan Chan

Technical Report

Overview

- Meta's new set of open-source foundation models called **Llama 3** (specifically, Llama 3.1)
- Release a "herd" of new language models at three sizes: **8B**, **70B** and **405B**
- Focused on three key areas for improving the development of high-quality foundation models:
  - **Data**: improving quantity and quality of data in pre-training and post-training stages (training on 15.6T tokens, compared to 1.8T for Llama 2)
  - **Scale**: train model at much larger scale (3.8 x 10^25 FLOPs, 50x more than Llama 2)
  - **Managing complexity**: make design choices to maximise ability to scale the model development process
- Resulting models have good capabilities on wide range of tasks
  - writing quality code, solving complex reasoning tasks, using tools, multilingual (8 languages)
- Develop **multimodal** extensions for image, video and speech understanding - still under development

Overview

Development of Llama 3 (and generally other LLMs) comprise of two main stages:

1. Language model pre-training
    - Convert a large (multilingual) text corpus to discrete tokens and pre-train a LLM to perform **next-token prediction**
    - Resulting model has rich understanding of language
2. Language model post-training
    - Train the model to **follow instructions** and behave in a way we expect assistants to
    - Involves several rounds of supervised fine-tuning and alignment (with DPO)

In Llama 3, there is a third stage to add image, video and speech capabilities to the trained model

Overview

	Finetuned	Multilingual	Long context	Tool use	Release
Llama 3 8B	✗	✗[1]	✗	✗	April 2024
Llama 3 8B Instruct	✓	✗	✗	✗	April 2024
Llama 3 70B	✗	✗[1]	✗	✗	April 2024
Llama 3 70B Instruct	✓	✗	✗	✗	April 2024
Llama 3.1 8B	✗	✓	✓	✗	July 2024
Llama 3.1 8B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 70B	✗	✓	✓	✗	July 2024
Llama 3.1 70B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 405B	✗	✓	✓	✗	July 2024
Llama 3.1 405B Instruct	✓	✓	✓	✓	July 2024

Table 1 Overview of the Llama 3 Herd of models. All results in this paper are for the Llama 3.1 models.

Pre-training I: Data

Extraction

- Rejection list of Domains with unsafe content, adult content or personal information. **This list isn't published.**
- Custom HTML Parser. Removal of boilerplate and content recall. **This parser isn't published despite popular request.**
- Special care is taken for mathematic and code content to preserve structure: Often use image alt attribute text for mathematical content
- Remove all markdown content - Find it harms model poorly

Pre-training I: Data

De-Duplication and Cleaning

- Three different levels of deduplication: URLs, Document and Line
  - URL: Keep most recent version of duplicated urls
  - Document: Near duplicate documents using something called MinHash.
  - Line: Remove lines that appear more than 6 times in 30M documents.
- Removes navigation menus, cookie warnings, etc.
- Additional effort to remove logs or error messages. They often slip by line level duplication removal because they are long and unique.
- Remove articles with large amounts of “dirty words”.

Pre-training I: Data

Extra Processing

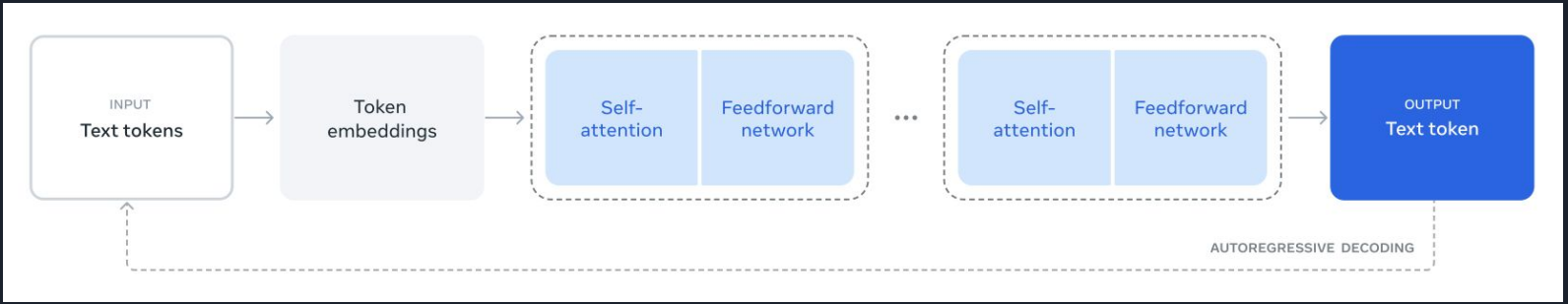
- Model based classifier: I.e. LLama2 Model is trained to determine if text is high quality
- Code and reasoning data: Custom HTML Parsers
- Multilingual data: fasttext-based language identification model to categorize documents into 176 languages.

Pre-training I: Data

Annealing

- Pick a high quality subset of the training data and continuously update the model whilst reducing the learning rate.
- Helps performance on benchmarks. It is specifically stated that benchmark data is not included in the annealing training dataset. .

Pre-training II: Model architecture



- Llama 3 uses a standard (dense) Transformer architecture [Vaswani et al., 2017] with four key modifications
  - Does not deviate much from previous Llama models
  - Argue that performance gains come primarily from **data quality/diversity** improvements and increased **training scale**

Pre-training II: Model architecture

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

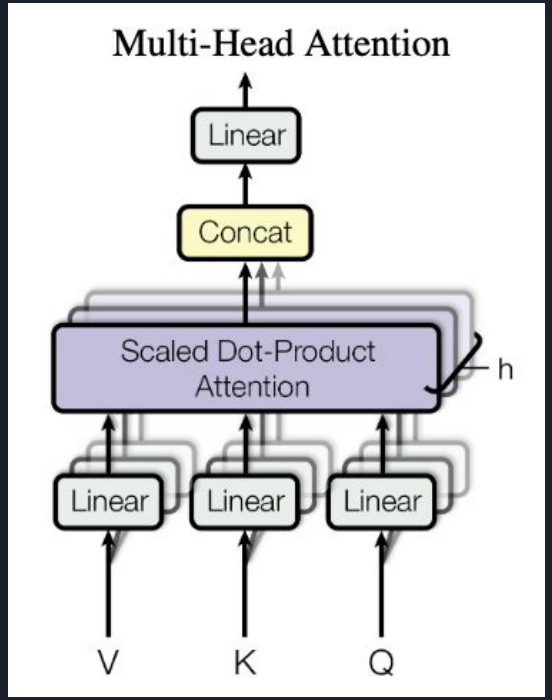
Pre-training II: Model architecture modifications

Small modifications to standard Transformer architecture:

- **Grouped Query Attention** (GQA) with 8 K-V heads
  - Improve inference speed and to reduce size of key-value caches during decoding
- Use an attention mask that **prevents** self-attention between **different documents** within the same sequence
  - Improve performance on very long sequence lengths
- Use vocabulary of **128K tokens** (100K tokens from “tiktoken” tokenizer + 28K tokens to support non-English languages)
  - Better compression rates from 3.17 characters per token in Llama 2 to 3.94 characters per token on sample of English data (model can use fewer tokens to express the same sequence on average)
- Use **Rotary Position Embeddings** (RoPE) with base frequency hyperparameter 500000
  - Improved performance over longer contexts

Grouped Query Attention (GQA)

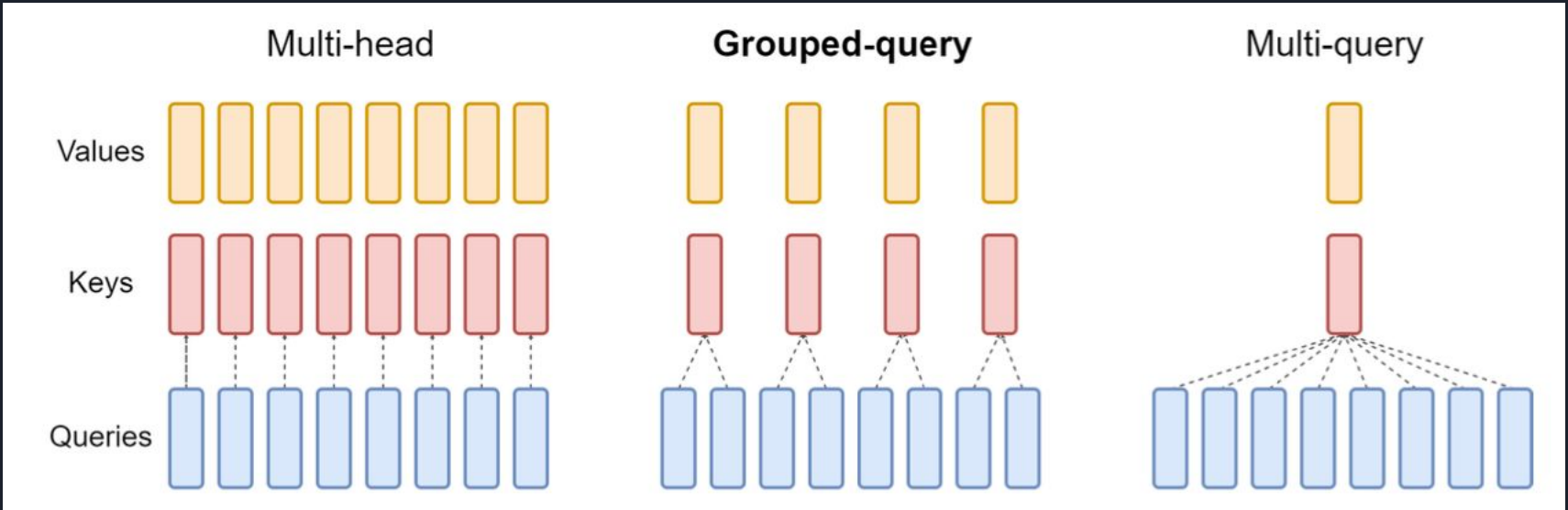
- In **Multi-Head Attention (MHA)** [Vaswani et al., 2017], there are ***H* different attention heads** in parallel:
  1. Each head has different learned linear projections for the key, value and query vectors
  2. Attention is performed on each head in parallel
  3. Outputs from each head are concatenated and projected
- For decoders, each query produced at each position attend to key-value pairs produced at all positions including that position
- During training, it's more efficient to batch together multiple queries and can prevent prevent tokens attending to future by using a triangular matrix and masking



Grouped Query Attention (GQA)

- However, during autoregressive decoder inference, you need to sequentially generate new tokens
  - This is a severe bottleneck for Transformers due to the memory bandwidth overhead from loading decoder weights and all previous attention key and values at every decoding step (key-value caching)
- In **Multi-Query Attention (MQA)** [Shazeer, 2019], different heads share a single set of keys and values to sharply reduce memory bandwidth
  - Going from MHA to MQA reduces *H* key and value heads to a single key and value
  - Reduces size of key-value cache and amount of data to load by factor of *H*
  - This drop off can be too aggressive to large models - lead to quality degradation and training instability
- **Grouped Query Attention** [Ainslie et al., 2023] is an interpolation between MHA and MQA with a single key and value head *per subgroup of query heads*
  - For this approach, you define a number *N* of key-value heads and **mean-pool** groups of size *H/N*

Grouped Query Attention (GQA)

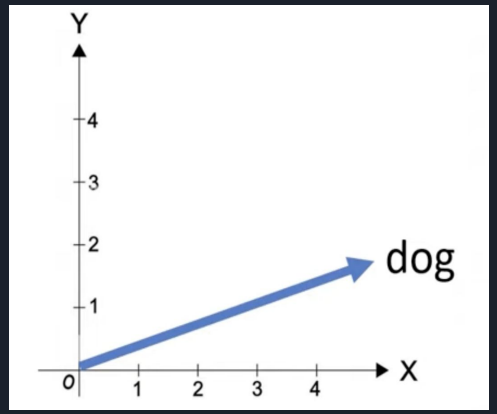


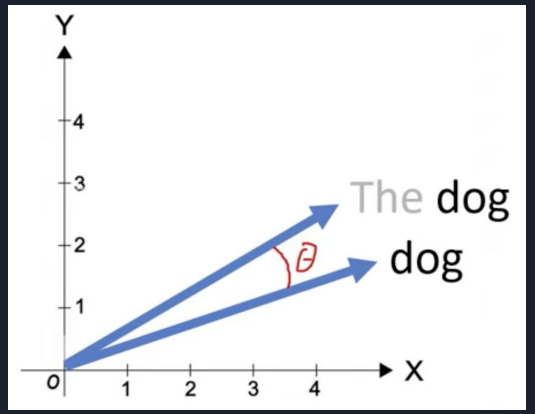
Position Embeddings

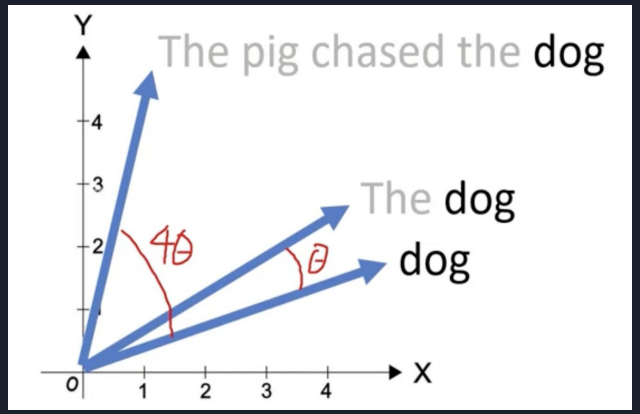
- Transformers have no recurrence or convolution - there's no notion of relative or absolute positions of tokens in the input (attention is position invariant to order)
  - So **position embeddings** are added to the input embeddings to preserve order
- Simple way is to **randomly initialise learnable embeddings** corresponding to each input position (example of an **absolute** position embedding)
  - Limited to maximum sequence length seen during training
  - Might not be ideal for capturing relative relationships between words
- Another way is to have **relative** position embeddings where embeddings capture the relative distance between pairs of tokens
  - Instead of having an embedding for each position, you have an embedding for the distance between two tokens
  - Bit more flexible for handling varying sequence lengths
  - Cannot simply just add position embedding to token - you must modify the attention mechanism in some other way to incorporate ordering

Rotary Position Embeddings (RoPE)

- Instead of adding a position embedding to encode a position of a token in a sequence, RoPE [Su et al., 2023] proposes to apply a **rotation** to the vector instead
  - The angle of the rotation is determined by the position of the token





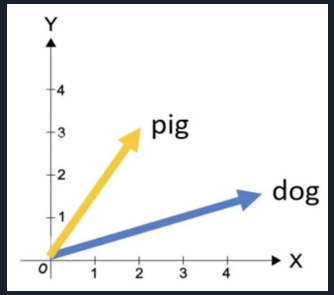


(Illustrations taken from “Efficient NLP” Youtube explainer on RoPE)

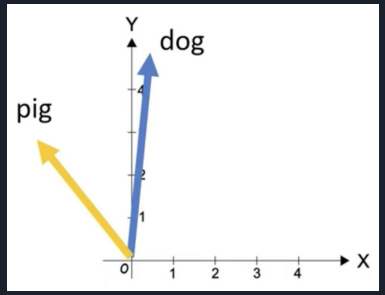
Rotary Position Embeddings (RoPE)

- Offers good **flexibility** and can be applied to sentences of any length
- It naturally incorporates **relative** positional information
  - Relative position of words are preserved - if we preserve the relative distance between tokens, the token embeddings will be rotated by the same amount (i.e. angle between them are preserved)
  - Dot product will be the same if we shift the absolute position of the tokens in the sequence

The **pig** chased the **dog**



Once upon a time, the **pig** chased the **dog**



- Has a **long-term decay property**: inner product between two tokens will decay as relative position increases (i.e. tokens close together, more likely to have larger dot-product)

(Illustrations taken from “Efficient NLP” Youtube explainer on RoPE)

Training Details

- “Llama 3 405B is trained on up to 16K H100 GPUs, each running at 700W TDP with 80GB HBM3, using Meta’s Grand Teton AI server platform”
- 405B Model took 54 days to train
- “40 PB of storage out of 7,500 servers equipped with SSDs, and supports a sustainable throughput of 2 TB/s and a peak throughput of 7 TB/s. “
- 38-43% GPU utilization

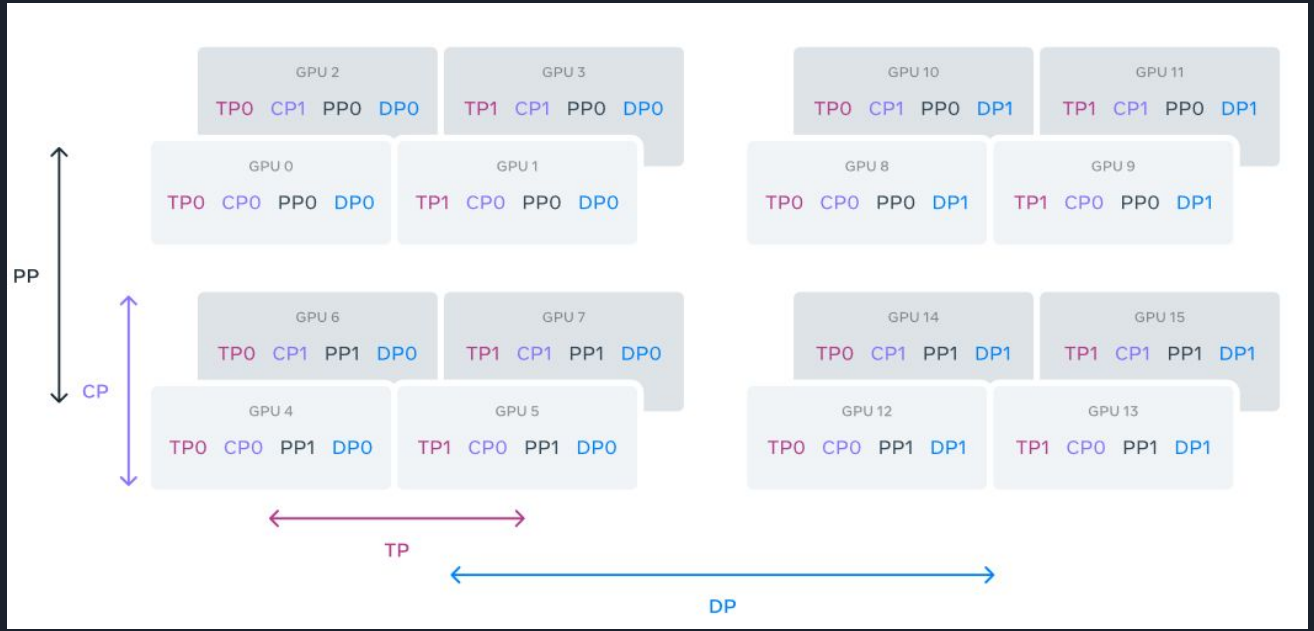
Training Infrastructure

TP: Tensor Parallelisation

PP: Pipeline Parallelisation

DP: Data Parallelisation

CP: Context Parallelisation



Can our power grids continue to support this?

“During training, tens of thousands of GPUs **may increase or decrease power consumption at the same time**, for example, due to all GPUs waiting for checkpointing or collective communications to finish, or the startup or shutdown of the entire training job. When this happens, it can result in instant fluctuations of power consumption across the data center on the order of tens of megawatts, **stretching the limits of the power grid.** This an ongoing challenge for us as we scale training for future, even larger Llama models.”

Post-training

- To fine-tune LLMs for human-AI interaction, there needs to be a chat dialog format introduced for the model to understand human instructions and to perform conversational tasks
- The latest Llama 3.1 chat template facilitates tool use capabilities
    - This is now quite long, so here is the original Llama 3 template earlier this year for an example:

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

{{ system_prompt }}<|eot_id|><|start_header_id|>user<|end_header_id|>

{{ user_message }}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Post-training

- Apply six rounds of post-training to the pre-trained Llama 3 models
- Each round consists of:
  - Training a **reward model (RM)** covering different capabilities on top of a pre-trained checkpoint
  - **Supervised fine-tuning (SFT**) from several sources:
    - Prompts from human annotation collection with rejection-sampled responses (i.e. use RM to select best out of a sample from latest model)
    - Synthetic data targeting specific capabilities
    - Small amounts of human-curated data
  - **Direct Preference Optimisation (DPO)** [Rafailov et al., 2024]
    - DPO optimizes for human preferences while avoiding reinforcement learning (like in RLHF)
- At the end of each stage, they collect new preference annotations and SFT data, and sample new synthetic data from the latest models
- They average models obtained from experiments using various versions of data and hyperparameters at each RM, SFT or DPO stage

Inference

Model	Quantization	Required VRAM (Estimate)	Options
405B	FP16 (default)	810GB	16xH100s or 8 x AMD MI300
405B	FP8	405GB	8 x H100s
405B	Int4	203GB	4 x H100s?
70B	FP8	70GB	2 x H100s (1 probably isn't enough)
70B	Int4	35GB	1 x H100
8B	FP8	16GB	Laptop (probably)

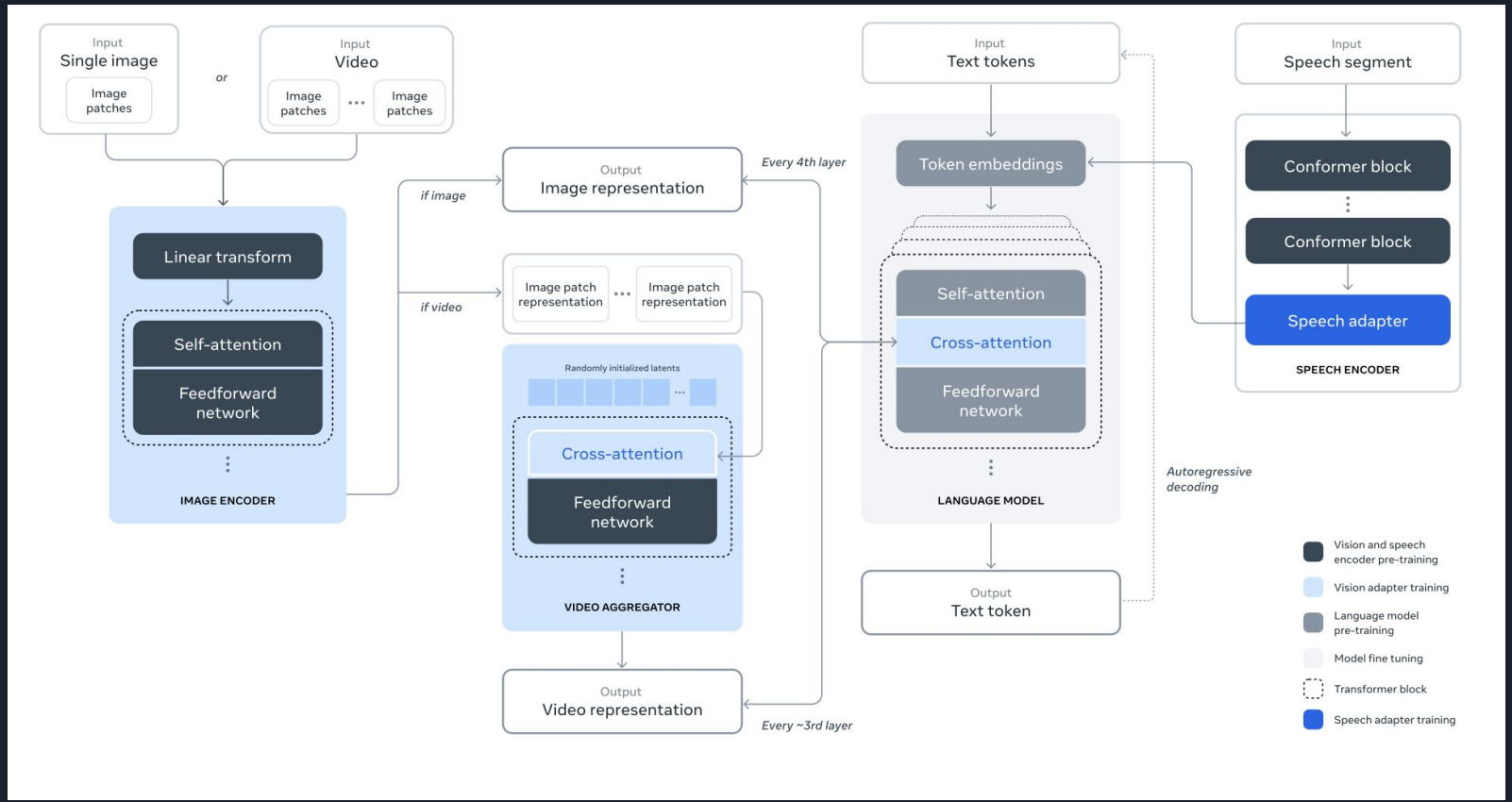
https://huggingface.co/blog/llama31

VRAM values are just to load weights into GPU memory

Multimodal capabilities

- Started work on incorporating visual- and speech-understanding capabilities into Llama 3 using a **compositional** approach
  - Compose a pre-trained image-/video-/audio-encoder with an adapter to the post-trained Llama 3 model
- Multimodal Llama 3 models are not released yet and still under development
- They prefer this approach for several reasons:
  - Allows them to parallelise development of vision, speech/audio and language capabilities
  - Circumvents complexities of jointly pre-training visual and language data
  - Guarantees that model performance on text-only tasks are not affected by the introduction of visual-recognition capabilities

Multimodal capabilities

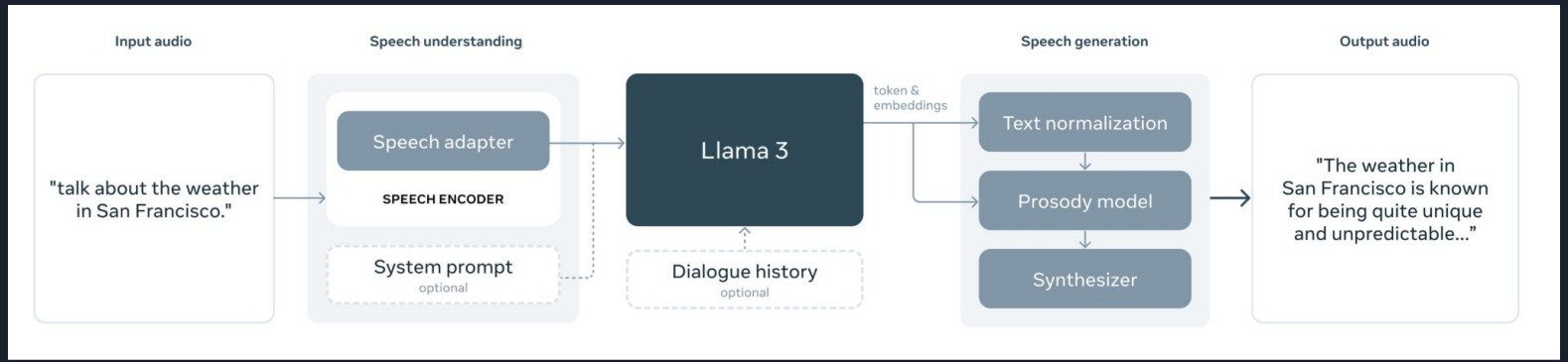


Multimodal capabilities: Vision

- Incorporate visual-recognition capabilities into Llama 3 using a compositional approach:
  - Compose a pre-trained image-encoder and the pre-trained language model by introducing a set of cross-attention layers between two models
  - Train new layer on a large number of image-text pairs
- For video, introduce temporal aggregator layers and additional video cross-attention layers
  - Train specific video cross-attention layers with video-text pairs to train the model to recognise and process temporal information from videos

Multimodal capabilities: Speech

- Use an encoder and introduce an adapter to process speech signals
  - The output of the speech module is directly fed into the language model as token representations enabling direct interaction between speech and text tokens
- Instead of using cross-attention layers to feed in multimodal information to Llama (as in vision), the speech module generates embeddings which are integrated with text tokens
  - Enabling the speech interface to leverage all the capabilities of the Llama 3 language model
- The outputs of Llama 3 are then fed to a speech generation module for audio response



References

- Llama Team. 2024. *The Llama 3 Herd of Models*. Meta AI.
- Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., & Sanghai, S. (2023). *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. arXiv preprint arXiv:2305.13245.
- Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., & Finn, C. (2024). *Direct preference optimization: Your language model is secretly a reward model*. Advances in Neural Information Processing Systems, 36.
- Shazeer, N. 2019. Fast transformer decoding: One write-head is all you need. arXiv preprint arXiv:1911.02150
- Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., & Liu, Y. 2024. *RoFormer: Enhanced transformer with rotary position embedding*. Neurocomputing, 568, 127063.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, 30.
- Efficient NLP YouTube. *Rotary Positional Embeddings: Combining Absolute and Relative*: https://youtu.be/o29P0Kpobz0?si=NcOAXwQeDIhnCuYK