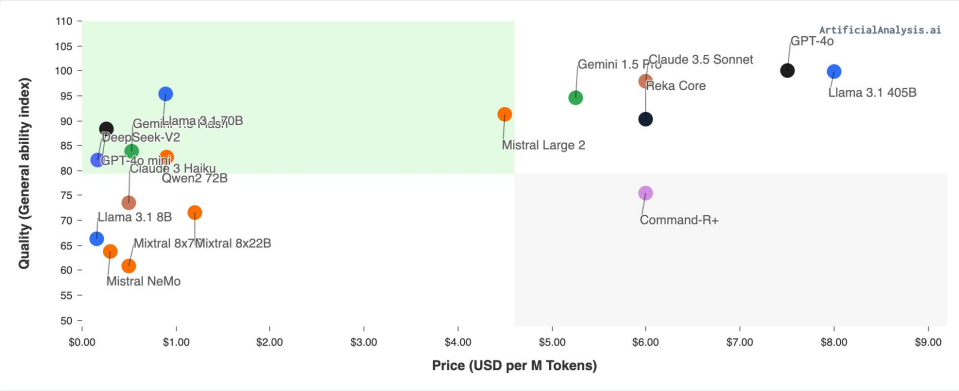
LLM 2024 H1 Rewind

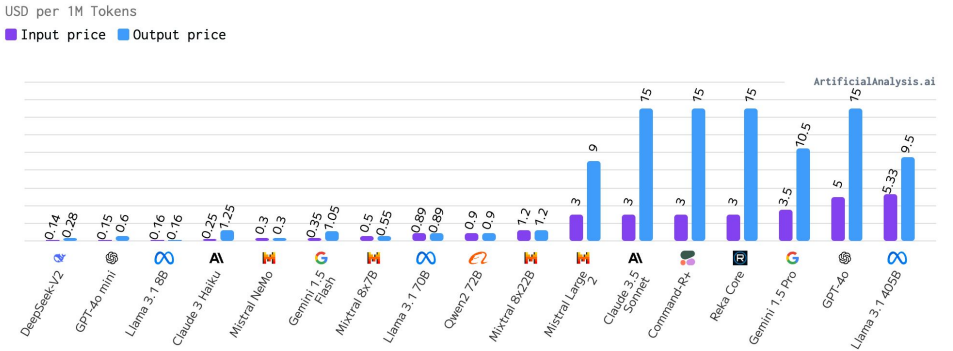
Kai Wang

2024.08.02

LLM in 2024 H1

1. User experience >> Leaderboard
2. Big Model: Higher Performance, longer context
3. Small Model: Powerful enough, faster, cheaper
4. Multimodal

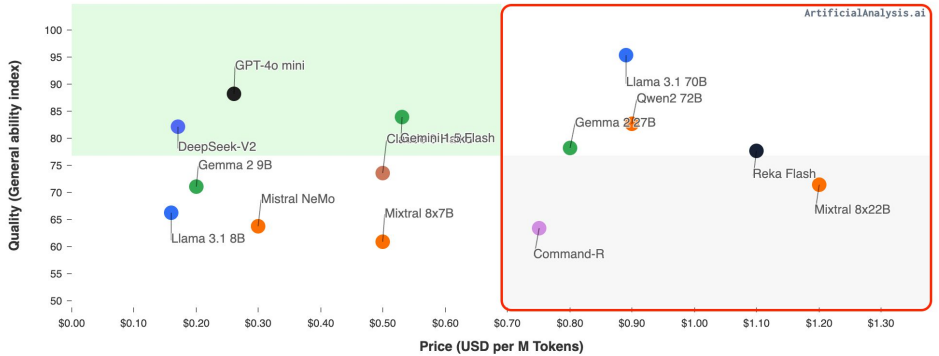


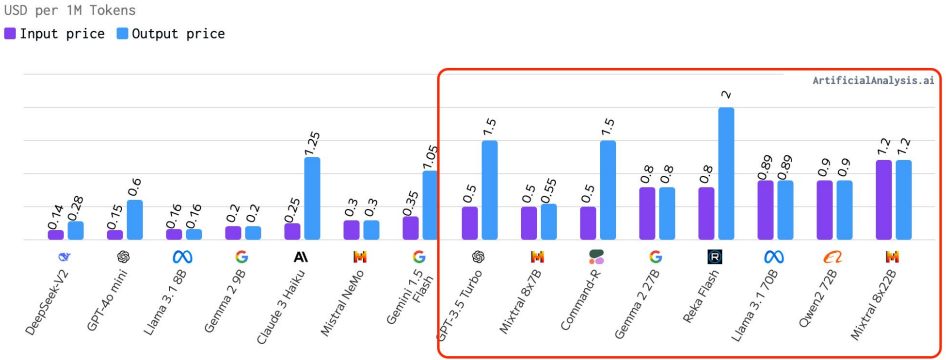


Quality: Index represents normalized average relative performance across Chatbot arena, MMLU & MT-Bench.

LLM in 2024 H1

1. User experience >> Leaderboard
2. Big Model: Higher Performance, longer context
3. Small Model: Powerful enough, faster, cheaper
4. Multimodal



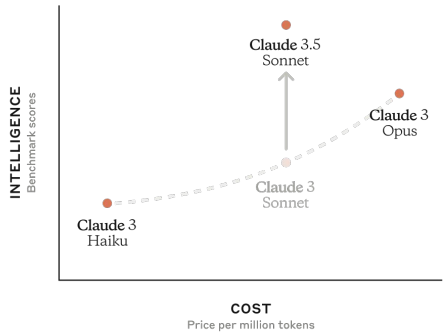


Quality: Index represents normalized average relative performance across Chatbot arena, MMLU & MT-Bench.

Claude AI

Sonnet 3.5 - Sonnet

- Extremely strict account banning policy



	Claude 3.5 Sonnet	Claude 3 Opus	GPT-4o	Gemini 1.5 Pro	Llama-400b (early snapshot)
Graduate level reasoning *GPQA, Diamond*	59.4%* 0-shot CoT	50.4% 0-shot CoT	53.6% 0-shot CoT	–	–
Undergraduate level knowledge *MMLU*	88.7%** 5-shot 88.3% 0-shot CoT	86.8% 5-shot 85.7% 0-shot CoT	– 88.7% 0-shot CoT	85.9% 5-shot –	86.1% 5-shot –
Code *HumanEval*	92.0% 0-shot	84.9% 0-shot	90.2% 0-shot	84.1% 0-shot	84.1% 0-shot
Multilingual math *MGSM*	91.6% 0-shot CoT	90.7% 0-shot CoT	90.5% 0-shot CoT	87.5% 8-shot	–
Reasoning over text *DROP, F1 score*	87.1 3-shot	83.1 3-shot	83.4 3-shot	74.9 Variable shots	83.5 3-shot Pre-trained model
Mixed evaluations *BIG-Bench-Hard*	93.1% 3-shot CoT	86.8% 3-shot CoT	–	89.2% 3-shot CoT	85.3% 3-shot CoT Pre-trained model
Math problem-solving *MATH*	71.1% 0-shot CoT	60.1% 0-shot CoT	76.6% 0-shot CoT	67.7% 4-shot	57.8% 4-shot CoT
Grade school math *GSM8K*	96.4% 0-shot CoT	95.0% 0-shot CoT	–	90.8% 11-shot	94.1% 8-shot CoT

* Claude 3.5 Sonnet scores 67.2% on 5-shot CoT GPQA with maj@32
** Claude 3.5 Sonnet scores 90.4% on MMLU with 5-shot CoT prompting

OpenAI

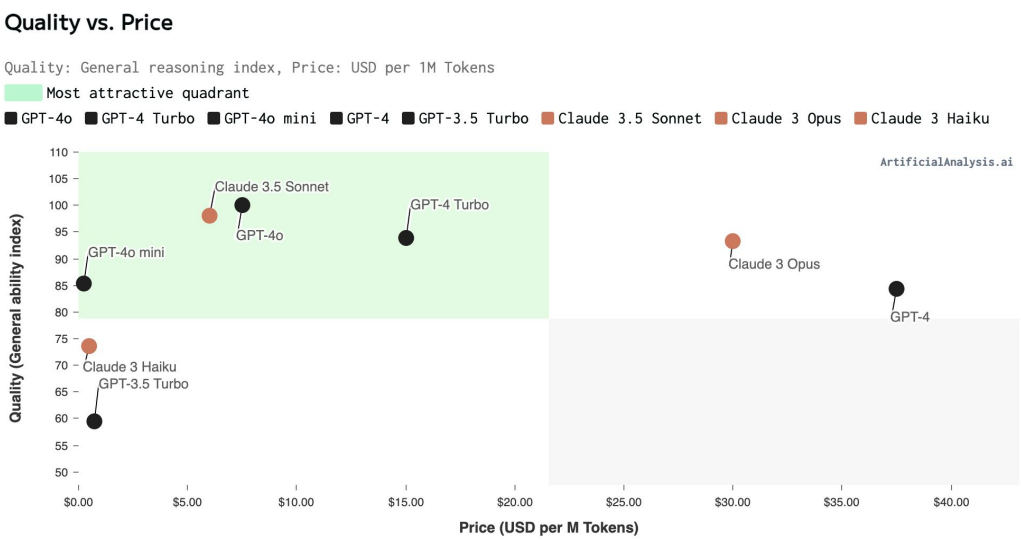
GPT4o

- Multimodal
- Fast

GPT4o mini

- Low price

Easily use API from Azure



Google

Gemini 1.5 Ultra / Pro / Nano

- Sparse mixture-of-expert（MOE）
- Long context Support (Pro 10 million tokens, Nano 2 million tokens)
- Nano
  - int4 online distilled from Pro

- **Parallel Layers** – We use a "parallel" formulation in each Transformer block (Wang and Komatsuzaki, 2021), rather than the standard "serialized" formulation. Specifically, the standard formulation can be written as:

$$y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x))))$$

Whereas the parallel formulation can be written as:

$$y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$$

The parallel formulation results in roughly 15% faster training speed at large scales,

https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf

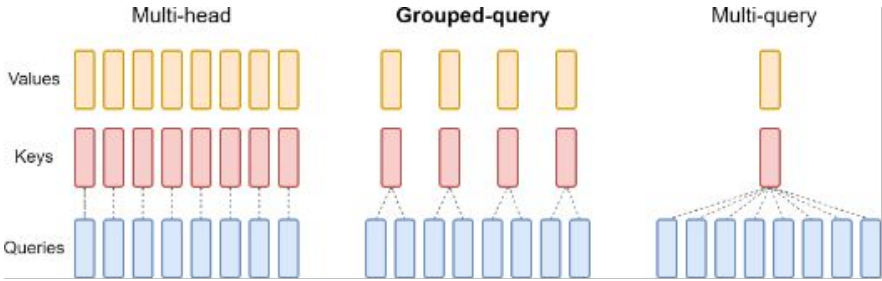
Google

Gemma 2B/7B

- Models are trained on a context length of 8192 tokens.
- Multi-Query Attention for 2 B
  - Each Query for 1 attention head
  - Share K / V for Query

Parameters	2B	7B
*d*_model	2048	3072
Layers	18	28
Feedforward hidden dims	32768	49152
Num heads	8	16
Num KV heads	1	16
Head size	256	256
Vocab size	256128	256128

Table 1 | Key model parameters.



https://arxiv.org/pdf/2403.08295

Google

Gemma 2B/7B

- RoPE (Rotary Position Embeddings)
  - Steps:
    - View as complex
    - Multiply Rotary matrix
  - decaying inter-token dependency, improve perf
  - Bring relative position encoding

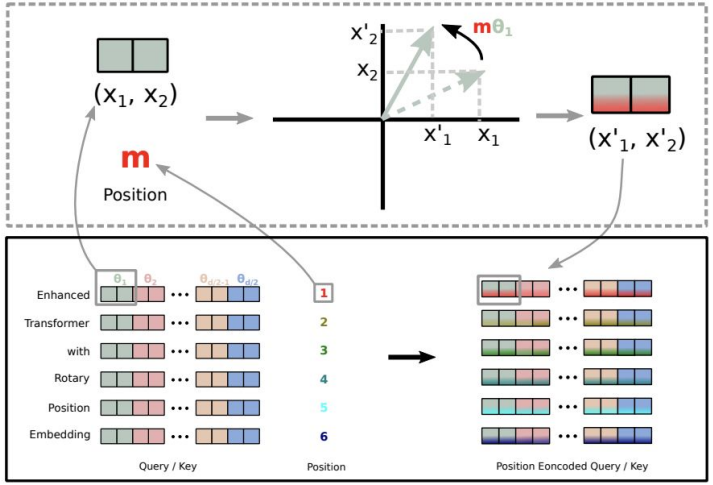


Figure 1: Implementation of Rotary Position Embedding(RoPE).

```python
def precompute_freqs_cis(dim: int,
                         end: int,
                         theta: float = 10000.0) -> torch.Tensor:
    """Precomputes the frequency cis."""
    freqs = 1.0 / (theta**(torch.arange(0, dim, 2)[:(dim // 2)].float() / dim))
    t = torch.arange(end, device=freqs.device)
    freqs = torch.outer(t, freqs).float()
    freqs_cis = torch.polar(torch.ones_like(freqs), freqs)  # complex64
    return freqs_cis


def apply_rotary_emb(x: torch.Tensor, freqs_cis: torch.Tensor) -> torch.Tensor:
    """Applies the rotary embedding to the query and key tensors."""
    x_ = torch.view_as_complex(
        torch.stack(torch.chunk(x.transpose(1, 2).float(), 2, dim=-1),
                    dim=-1))
    x_out = torch.view_as_real(x_ * freqs_cis).type_as(x)
    x_out = torch.cat(torch.chunk(x_out, 2, dim=-1), dim=-2)
    x_out = x_out.reshape(x_out.shape[0], x_out.shape[1], x_out.shape[2],
                          -1).transpose(1, 2)
    return x_out
```

https://arxiv.org/pdf/2403.08295

Google

Gemma 2B/7B

- RMSNorm, replace LayerNorm
  - Better for variable-length sequences
  - Stablize and converge faster

$$\mathrm{RMS}(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

$$\hat{x} = \frac{x}{\mathrm{RMS}(x)}$$

$$y = \gamma\hat{x} + \beta$$

```python
class RMSNorm(torch.nn.Module):

    def __init__(
        self,
        dim: int,
        eps: float = 1e-6,
        add_unit_offset: bool = True,
    ):
        super().__init__()
        self.eps = eps
        self.add_unit_offset = add_unit_offset
        self.weight = nn.Parameter(torch.zeros(dim))

    def _norm(self, x):
        return x * torch.rsqrt(x.pow(2).mean(-1, keepdim=True) + self.eps)

    def forward(self, x):
        # Llama does x.to(float16) * w whilst Gemma2 is (x * w).to(float16)
        # See https://github.com/huggingface/transformers/pull/29402
        output = self._norm(x.float())
        if self.add_unit_offset:
            output = output * (1 + self.weight.float())
        else:
            output = output * self.weight.float()
        return output.type_as(x)
```

https://arxiv.org/pdf/2403.08295

Google

Gemma 2 2B/9B/27B

- Context length of 8192 tokens with RoPE
- Post-norm and pre-norm with RMSNorm for sublayer I/O
- Grouped-Query Attention (num group 2)

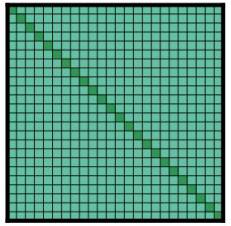
Parameters	**2.6B**	**9B**	**27B**
d_model	2304	3584	4608
Layers	26	42	46
pre-norm	yes	yes	yes
post-norm	yes	yes	yes
Non-linearity	GeGLU	GeGLU	GeGLU
Feedforward dim	18432	28672	73728
Head type	GQA	GQA	GQA
Num heads	8	16	32
Num KV heads	4	8	16
Head size	256	256	128
global att. span	8192	8192	8192
sliding window	4096	4096	4096
Vocab size	256128	256128	256128
Tied embedding	yes	yes	yes

Table 1 | Overview of the main model parameters and design choices. See the section on model architectures for more details.

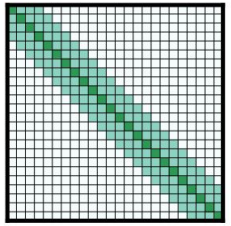
https://arxiv.org/pdf/2403.08295
https://storage.googleapis.com/deepmind-media/gemma/gemma-2-report.pdf
https://arxiv.org/pdf/2004.05150

Google

```python
def get_config_for_9b() -> GemmaConfig:
    return GemmaConfig(
        architecture=Architecture.GEMMA_2,
        num_hidden_layers=42,
        num_attention_heads=16,
        num_key_value_heads=8,
        hidden_size=3584,
        intermediate_size=14336,
        use_pre_ffw_norm=True,
        use_post_ffw_norm=True,
        final_logit_softcapping=30.0,
        attn_logit_softcapping=50.0,
        head_dim=256,
        attn_types=[AttentionType.LOCAL_SLIDING, AttentionType.GLOBAL] * 21,
        sliding_window_size=4096,
    )
```



(a) Full n^2 attention



(b) Sliding window attention

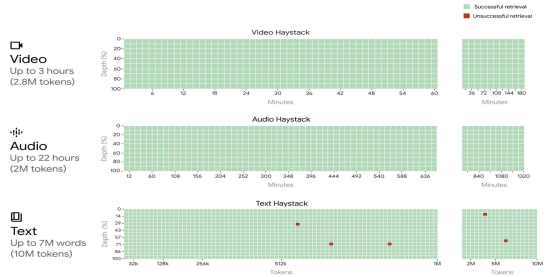
Gemma 2 2B/9B/27B

- Logits Soft-capping
  - scaling logits to a fixed range, improving training.
- Interleaved Local Sliding Window and Global Attention
  - sliding window of local attention layers 4096 tokens
  - span of the global attention layers 8192 tokens
  - Pro for needle in haystack（大海捞针实验）

```python
# [batch_size, n_local_heads, input_len, max_seq_len]
q.mul_(self.scaling)
scores = torch.matmul(q, k.transpose(2, 3))
if (
    self.attn_type == gemma_config.AttentionType.LOCAL_SLIDING
    and self.sliding_window_size is not None
):
    all_ones = torch.ones_like(mask)
    sliding_mask = torch.triu(
        all_ones, -1 * self.sliding_window_size + 1
    ) * torch.tril(all_ones, self.sliding_window_size - 1)
    mask = torch.where(sliding_mask == 1, mask, -2.3819763e38)
if self.attn_logit_softcapping is not None:
    scores = scores / self.attn_logit_softcapping
    scores = torch.tanh(scores)
    scores = scores * self.attn_logit_softcapping
scores = scores + mask
scores = F.softmax(scores.float(), dim=-1).type_as(q)

# [batch_size, n_local_heads, input_len, head_dim]
output = torch.matmul(scores, v)

# [batch_size, input_len, hidden_dim]
output = (output.transpose(1, 2).contiguous().view(
    batch_size, input_len, -1))
output = self.o_proj(output)
return output
```



https://arxiv.org/pdf/2403.08295
https://storage.googleapis.com/deepmind-media/gemma/gemma-2-report.pdf
https://arxiv.org/pdf/2004.05150

Meta

RMSNorm/RoPE/SwiGLU

LLama

- Contexts 2k

LLama2

- Contexts 4k
- GQA for 34 B+

LLama3

- Contexts 8k
- GQA

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

	Training Data	Params	Context Length	GQA
LLAMA 1	*See Touvron et al. (2023)*	7B	2k	✗
		13B	2k	✗
		33B	2k	✗
		65B	2k	✗
LLAMA 2	*A new mix of publicly available online data*	7B	4k	✗
		13B	4k	✗
		34B	4k	✓
		70B	4k	✓

	Training Data	Params	Context length	GQA	Token count	Knowledge cutoff
Llama 3	A new mix of publicly available online data.	8B	8k	Yes	15T+	March, 2023
		70B	8k	Yes		December, 2023

Meta

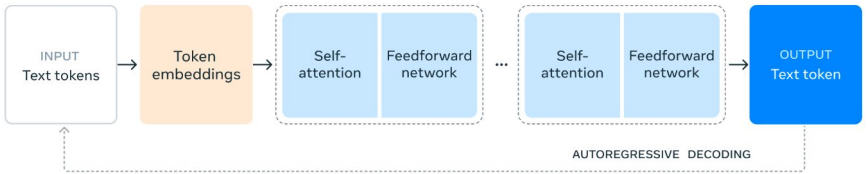
Llama 3.1 405B

- 128K context length
- GQA (8 k/v heads)
- 15 T data
- 16K H100
- Dense transformer, Not MoE
- Need Quantization & Parallel Inference

Category	Benchmark	Llama 3 8B	Gemma 2 9B	Mistral 7B	Llama 3 70B	Mixtral 8x22B	GPT 3.5 Turbo	Llama 3 405B	Nemotron 4 340B	GPT-4 (0125)	GPT-4o	Claude 3.5 Sonnet
General	MMLU (5-shot)	69.4	**72.3**	61.1	**83.6**	76.9	70.7	87.3	82.6	85.1	89.1	**89.9**
	MMLU (0-shot, CoT)	**73.0**	72.3△	60.5	**86.0**	79.9	69.8	88.6	78.7◁	85.4	**88.7**	88.3
	MMLU-Pro (5-shot, CoT)	**48.3**	–	36.9	**66.4**	56.3	49.2	73.3	62.7	64.8	74.0	**77.0**
	IFEval	**80.4**	73.6	57.6	**87.5**	72.7	69.9	**88.6**	85.1	84.3	85.6	88.0
Code	HumanEval (0-shot)	**72.6**	54.3	40.2	**80.5**	75.6	68.0	89.0	73.2	86.6	90.2	**92.0**
	MBPP EvalPlus (0-shot)	**72.8**	71.7	49.5	**86.0**	78.6	82.0	88.6	72.8	83.6	87.8	**90.5**
Math	GSM8K (8-shot, CoT)	**84.5**	76.7	53.2	**95.1**	88.2	81.6	**96.8**	92.3◇	94.2	96.1	96.4◇
	MATH (0-shot, CoT)	**51.9**	44.3	13.0	**68.0**	54.1	43.1	73.8	41.1	64.5	**76.6**	71.1
Reasoning	ARC Challenge (0-shot)	83.4	**87.6**	74.2	**94.8**	88.7	83.7	**96.9**	94.6	96.4	96.7	96.7
	GPQA (0-shot, CoT)	32.8	–	28.8	**46.7**	33.3	30.8	51.1	–	41.4	53.6	**59.4**
Tool use	BFCL	**76.1**	–	60.4	84.8	–	**85.9**	88.5	86.5	88.3	80.5	**90.2**
	Nexus	**38.5**	30.0	24.7	**56.7**	48.5	37.2	**58.7**	–	50.3	56.1	45.7
Long context	ZeroSCROLLS/QuALITY	81.0	–	–	90.5	–	–	**95.2**	–	**95.2**	90.5	90.5
	InfiniteBench/En.MC	65.1	–	–	78.2	–	–	**83.4**	–	72.1	82.5	–
	NIH/Multi-needle	98.8	–	–	97.5	–	–	98.1	–	**100.0**	**100.0**	90.8
Multilingual	MGSM (0-shot, CoT)	**68.9**	53.2	29.9	**86.9**	71.1	51.4	**91.6**	–	85.9	90.5	**91.6**

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	6,144	12,288	20,480
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 ($\theta = 500,000$)		

Table 3 Overview of the key hyperparameters of Llama 3. We display settings for 8B, 70B, and 405B language models.



https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Meta

Scaling Law $C \sim 6ND$

Given FLOPs,

determine model size

And datasets

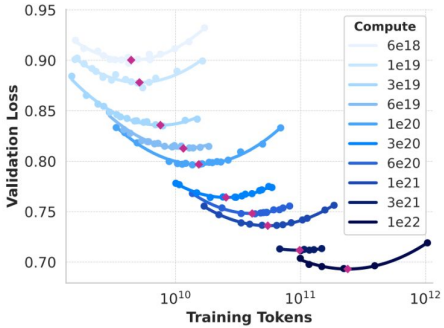


Figure 2 Scaling law IsoFLOPs curves between 6×10^{18} and 10^{22} FLOPs. The loss is the negative log-likelihood on a held-out validation set. We approximate measurements at each compute scale using a second degree polynomial.

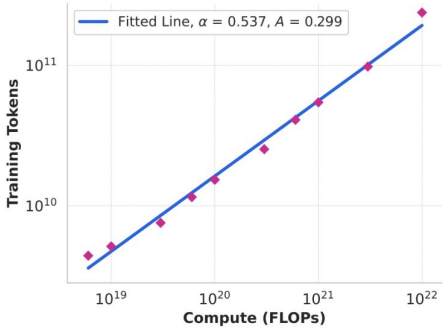


Figure 3 Number of training tokens in identified compute-optimal models as a function of pre-training compute budget. We include the fitted scaling-law prediction as well. The compute-optimal models correspond to the parabola minimums in Figure 2.

We use the compute-optimal models we identified this way to predict the optimal number of training tokens for a specific compute budget. To do so, we assume a power-law relation between compute budget, C, and the optimal number of training tokens, $N^\star(C)$:

$$N^\star(C) = AC^\alpha.$$

We fit A and α using the data from Figure 2. We find that $(\alpha, A) = (0.53, 0.29)$; the corresponding fit is shown in Figure 3. Extrapolation of the resulting scaling law to 3.8×10^{25} FLOPs suggests training a 402B parameter model on 16.55T tokens.

An important observation is that IsoFLOPs curves become *flatter* around the minimum as the compute budget increases. This implies that performance of the flagship model is relatively robust to small changes in the trade-off between model size and training tokens. Based on this observation, we ultimately decided to train a flagship model with 405B parameters.

https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Meta

Infra

- 16 GPUs / 2 servers
- 3072 GPUs /192 racks
- 24K GPUs/8 pods

Only Use 16K GPUs

- Scaling law
- Backup for training

Compute. 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 (Matt Bowman, 2022). Each server is equipped with eight GPUs and two CPUs. Within a server, the eight GPUs are connected via NVLink. Training jobs are scheduled using MAST (Choudhury et al., 2024), Meta's global-scale training scheduler.

Storage. Tectonic (Pan et al., 2021), Meta's general-purpose distributed file system, is used to build a storage fabric (Battey and Gupta, 2024) for Llama 3 pre-training. It offers 240 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. A major challenge is supporting the highly bursty checkpoint writes that saturate the storage fabric for short durations. Checkpointing saves each GPU's model state, ranging from 1 MB to 4 GB per GPU, for recovery and debugging. We aim to minimize GPU pause time during checkpointing and increase checkpoint frequency to reduce the amount of lost work after a recovery.

Network. Llama 3 405B used RDMA over Converged Ethernet (RoCE) fabric based on the Arista 7800 and Minipack2 Open Compute Project[4] OCP rack switches. Smaller models in the Llama 3 family were trained using Nvidia Quantum2 Infiniband fabric. Both RoCE and Infiniband clusters leverage 400 Gbps interconnects between GPUs. Despite the underlying network technology differences between these clusters, we tune both of them to provide equivalent performance for these large training workloads. We elaborate further on our RoCE network since we fully own its design.

- **Network topology.** Our RoCE-based AI cluster comprises 24K GPUs[5] connected by a three-layer Clos network (Lee et al., 2024). At the bottom layer, each rack hosts 16 GPUs split between two servers and connected by a single Minipack2 top-of-the-rack (ToR) switch. In the middle layer, 192 such racks are connected by Cluster Switches to form a pod of 3,072 GPUs with full bisection bandwidth, ensuring no oversubscription. At the top layer, eight such pods within the same datacenter building are connected via Aggregation Switches to form a cluster of 24K GPUs. However, network connectivity at the aggregation layer does not maintain full bisection bandwidth and instead has an oversubscription ratio of 1:7. Our model parallelism methods (see Section 3.3.2) and training job scheduler (Choudhury et al., 2024) are all optimized to be aware of network topology, aiming to minimize network communication across pods.

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	4	131,072	16	16M	380	38%

Table 4 Scaling configurations and MFU for each stage of Llama 3 405B pre-training. See text and Figure 5 for descriptions of each type of parallelism.

https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Meta

Inference

- Do not quantize parameters in the self-attention layers of the model
- Do not perform quantization in the first and last Transformer layers
- Row-wise quantization

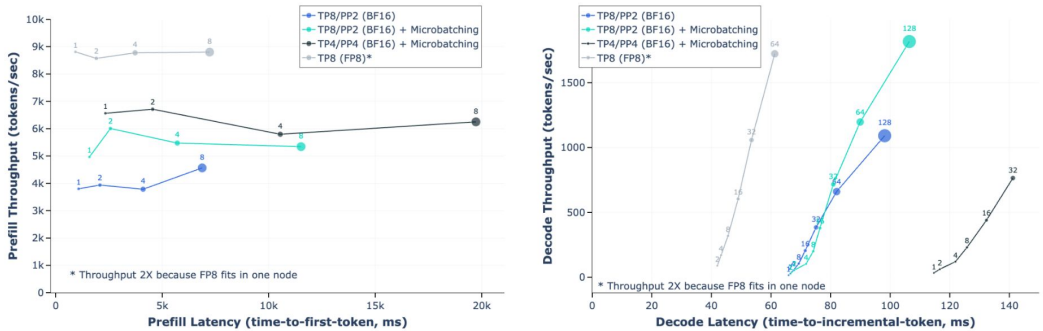


Figure 27 Throughput-latency trade-off in FP8 inference with Llama 3 405B compared with BF16 inference using different pipeline parallelization setups. *Left:* Results for pre-filling. *Right:* Results for decoding.

https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Meta

Multimodal

- Image encoder 850 M
  - ViT-H/14
    - 630M/224x224/16x16 patches
  - Multi-layers features extract
    - Like SSD
  - 8 gated self-attention layers (40 transformer block)
- Image adaptor
  - Cross-attention every 4 layers
  - 100 B
  - Use GQA

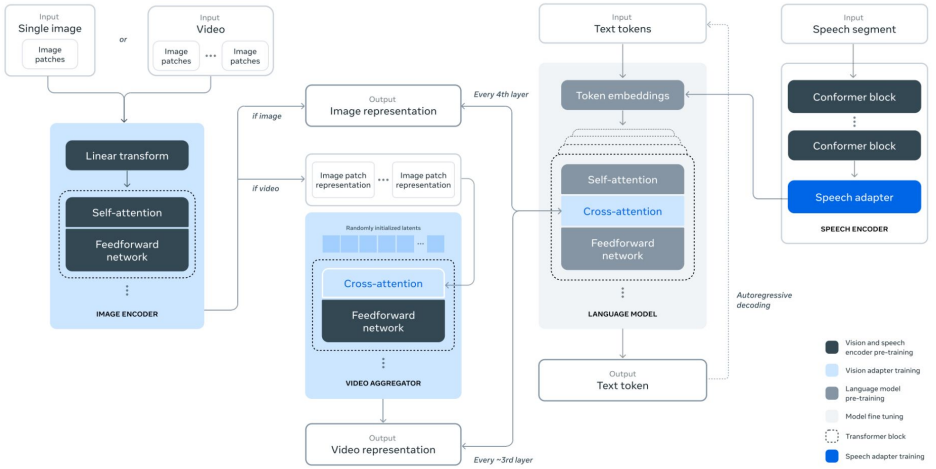
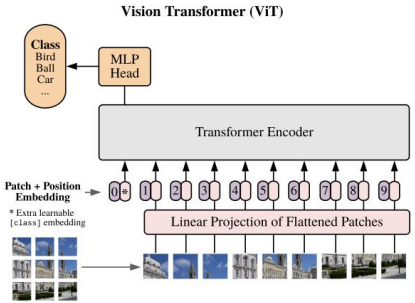


Figure 28 Illustration of the compositional approach to adding multimodal capabilities to Llama 3 that we study in this paper. This approach leads to a multimodal model that is trained in five stages: **(1)** language model pre-training, **(2)** multi-modal encoder pre-training, **(3)** vision adapter training, **(4)** model finetuning, and **(5)** speech adapter training.



https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Meta

Multimodal

- Video Adaptor
  - 64 frames
  - Frame -> image encoder
- temporal aggregator
  - merges 32 consecutive frames into 1
- video cross attention before every fourth image cross attention layer

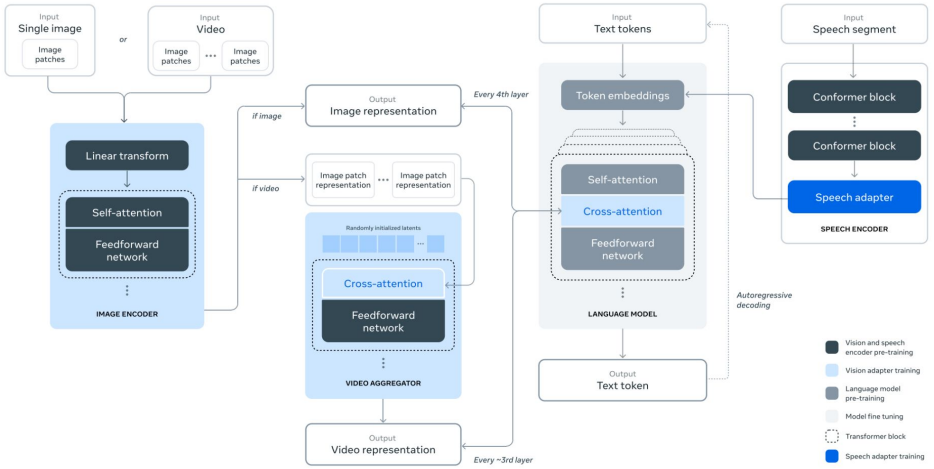


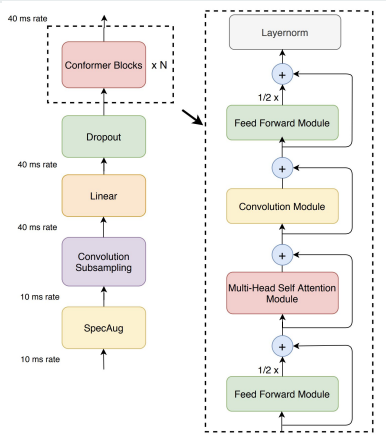
Figure 28 Illustration of the compositional approach to adding multimodal capabilities to Llama 3 that we study in this paper. This approach leads to a multimodal model that is trained in five stages: **(1)** language model pre-training, **(2)** multi-modal encoder pre-training, **(3)** vision adapter training, **(4)** model finetuning, and **(5)** speech adapter training.

https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Meta

Multimodal

- Speech Encoder 1 B Conformer
  - Input 80-dimensional mel-spectrogram features
  - 24 Conformer layers with Latent dimension of 1536
    - 1 convolution module with kernel size 7
    - 1 rotary attention module with 24 attention heads
    - 2 Macron-net style feed-forward networks with dimension 4096
- Speech adapter 100M
  - 1 convolution layer
  - 1 rotary Transformer layer latent dimension 3072
  - 1 linear layer 4096



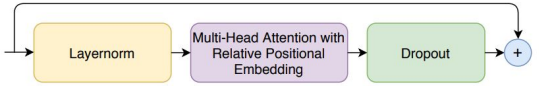


Figure 3: ***Multi-Headed self-attention module.*** *We use multi-headed self-attention with relative positional embedding in a pre-norm residual unit.*

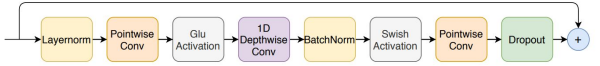


Figure 2: ***Convolution module.*** *The convolution module contains a pointwise convolution with an expansion factor of 2 projecting the number of channels with a GLU activation layer, followed by a 1-D Depthwise convolution. The 1-D depthwise conv is followed by a Batchnorm and then a swish activation layer.*

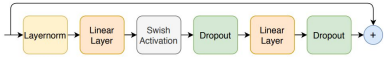


Figure 4: ***Feed forward module.*** *The first linear layer uses an expansion factor of 4 and the second linear layer projects it back to the model dimension. We use swish activation and a pre-norm residual units in feed forward module.*

https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

Mistral

- Nemo 12B 128 K contexts
- **Large 2** 128 K contexts, pro on math & code
- (legacy) Mixtral 32 K contexts 2/8 experts

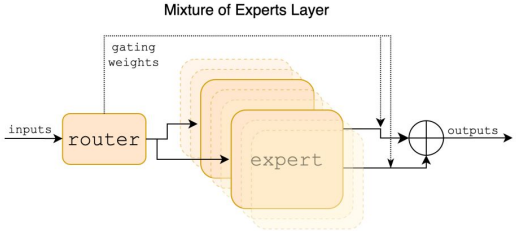
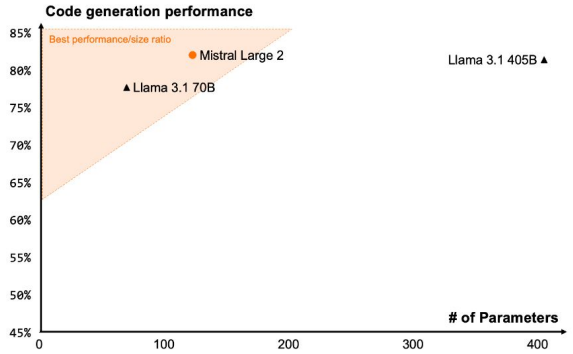
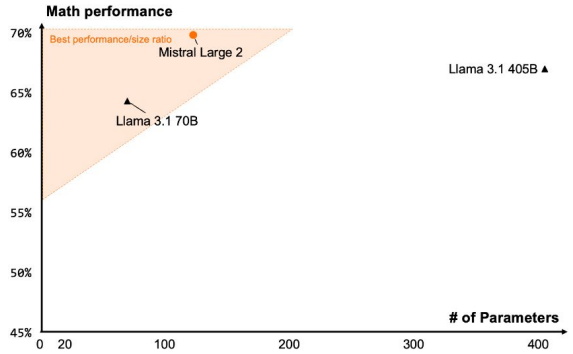


Figure 1: Mixture of Experts Layer. Each input vector is assigned to 2 of the 8 experts by a router. The layer's output is the weighted sum of the outputs of the two selected experts. In Mixtral, an expert is a standard feedforward block as in a vanilla transformer architecture.

Parameter	Value
dim	4096
n_layers	32
head_dim	128
hidden_dim	14336
n_heads	32
n_kv_heads	8
context_len	32768
vocab_size	32000
num_experts	8
top_k_experts	2

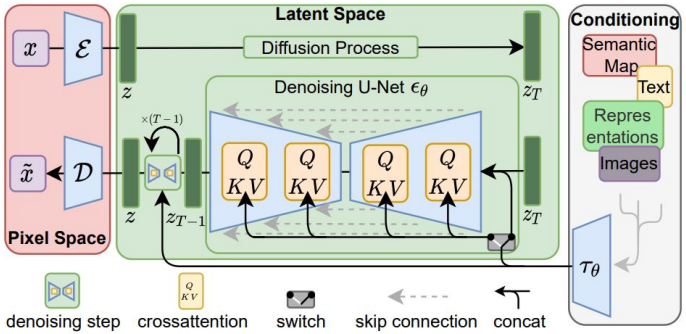
Table 1: Model architecture.

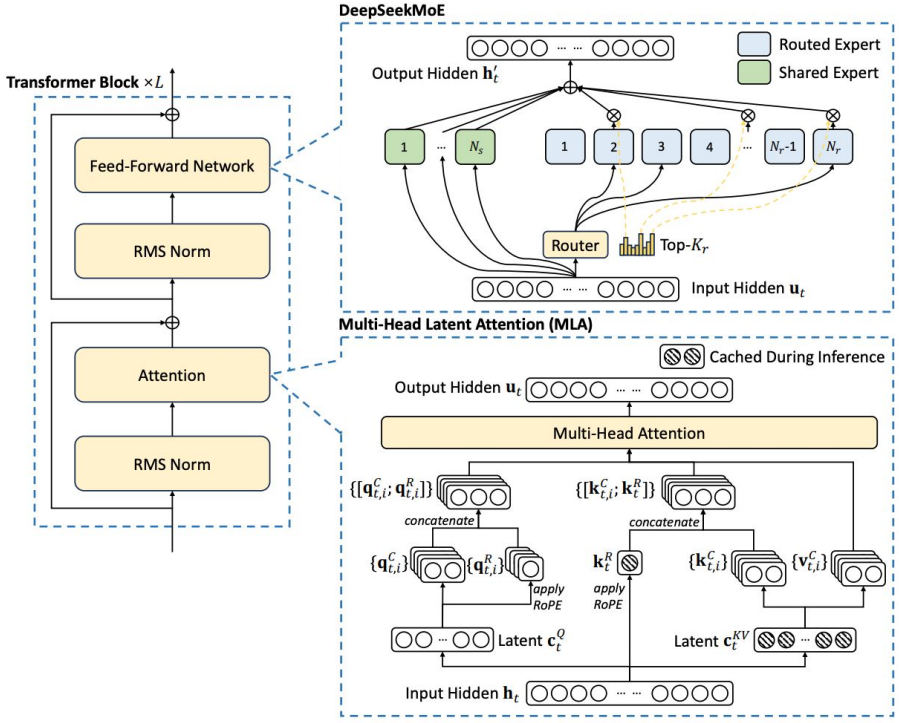




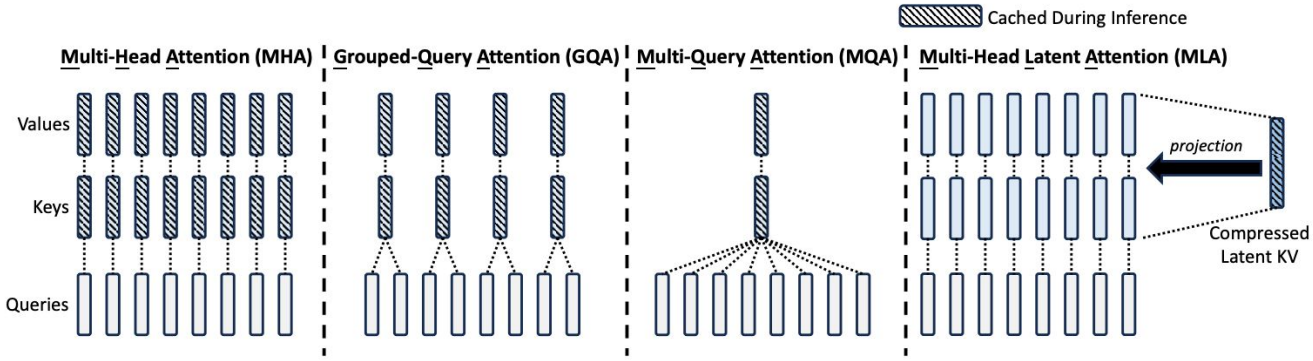
DeepSeek

- DeepSeek-V2-Lite (2.4B/token total 15.7B)
- DeepSeek V2
  - 21B/token, total 236B
  - 128K contexts
  - Shared + routed MoE
  - MLA (Latent Space)





DeepSeek

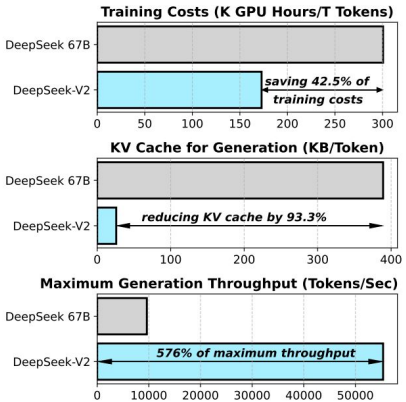


$$\mathbf{q}_t = W^Q \mathbf{h}_t,$$
$$\mathbf{k}_t = W^K \mathbf{h}_t,$$
$$\mathbf{v}_t = W^V \mathbf{h}_t,$$

$$\mathbf{c}_t^{KV} = W^{DKV} \mathbf{h}_t,$$
$$\mathbf{k}_t^C = W^{UK} \mathbf{c}_t^{KV},$$
$$\mathbf{v}_t^C = W^{UV} \mathbf{c}_t^{KV},$$

$$\mathbf{c}_t^Q = W^{DQ} \mathbf{h}_t,$$
$$\mathbf{q}_t^C = W^{UQ} \mathbf{c}_t^Q,$$

$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; ...; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \mathrm{RoPE}(W^{QR} \mathbf{c}_t^Q),$$
$$\mathbf{k}_t^R = \mathrm{RoPE}(W^{KR} \mathbf{h}_t),$$
$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R],$$
$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R],$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right) \mathbf{v}_{j,i}^C,$$
$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; ...; \mathbf{o}_{t,n_h}],$$

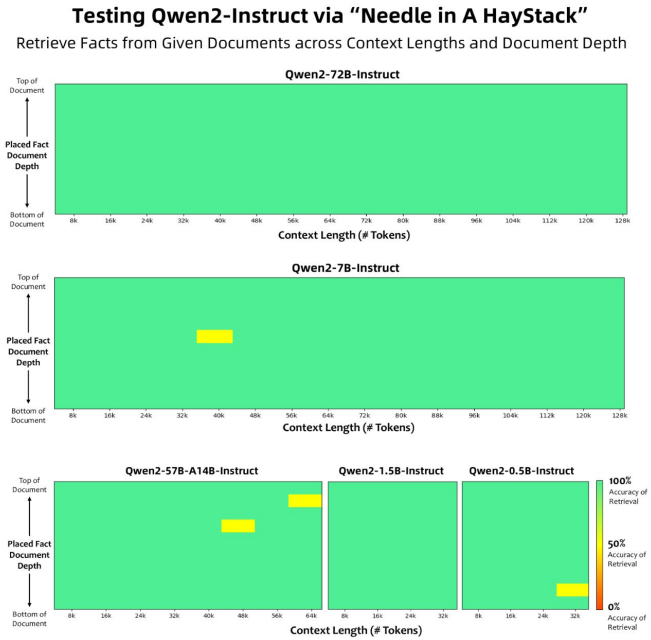


Alibaba

Qwen 2

Table 1: **Architecture of Qwen2 dense and MoE models.** For MoE models, 57B-A14B denotes that the model has 57B parameters in total and for each token 14B parameters are active, the Intermediate size denotes that of each expert, and # Activated Experts excludes the shared experts.

Configuration	0.5B	1.5B	7B	72B	57B-A14B
Hidden Size	896	1,536	3,584	8,192	3,584
# Layers	24	28	28	80	28
# Query Heads	14	12	28	64	28
# KV Heads	2	2	4	8	4
Head Size	64	128	128	128	128
Intermediate Size	4,864	8,960	18,944	29,568	2,560
# Routed Experts	-	-	-	-	64
# Activated Experts	-	-	-	-	8
# Shared Experts	-	-	-	-	8
Embedding Tying	True	True	False	False	False
Vocabulary Size	151,646	151,646	151,646	151,646	151,646
# Trained Tokens	12T	7T	7T	7T	4.5T



Alibaba

Qwen 2

Dual Chunk Attention

- Low computation complexity
- Low memory
- Pro for long context

```
while remain_len > 0:
    flash_per_chunk = []
    begin = kv_seq_len - remain_len
    curr_chunk_len = min(chunk_len, remain_len)
    end = begin + curr_chunk_len

    q_states_intra = apply_rotary_pos_emb(query_states[:, :, begin:end, :], q_cos, q_sin,
                                          position_ids[:, begin:end])

    k_states_intra = key_states[:, :, begin:end, :]
    v_states_intra = value_states[:, :, begin:end, :]
    flash_result = do_flash_attn(q_states_intra, k_states_intra, v_states_intra)
    flash_per_chunk.append(flash_result)

    q_states_succ = apply_rotary_pos_emb(query_states[:, :, begin:end, :], qc_cos, qc_sin,
                                         position_ids[:, begin:end])
    flash_result = do_flash_attn(q_states_succ, k_states_prev, v_states_prev, False)
    flash_per_chunk.append(flash_result)

    if begin - (k_states_prev.size(-2)) > 0:
        prev_len = k_states_prev.size(-2)
        q_states_inter = apply_rotary_pos_emb(query_states[:, :, begin:end, :], qc_cos, qc_sin,
                                              position_ids[:, chunk_len - 1][:, None].repeat(1, curr_chunk_len
        k_states_inter = key_states[:, :, :begin - prev_len, :]
        v_states_inter = value_states[:, :, :begin - prev_len, :]
        flash_result = do_flash_attn(q_states_inter, k_states_inter, v_states_inter, False)
        flash_per_chunk.append(flash_result)

    flash_results.append(flash_per_chunk)
    k_states_prev = k_states_intra
    v_states_prev = v_states_intra
    remain_len = remain_len - chunk_len

attn_output = merge_attn_outputs(flash_results)
```

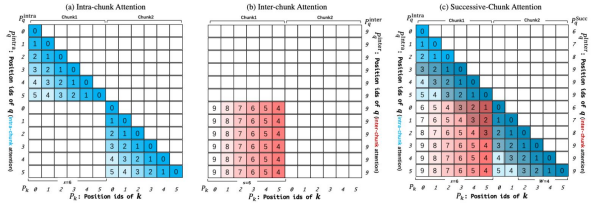


Figure 2. Visualization of the Relative Position Matrix M employing Dual Chunk Attention (DCA), with chunk size $s = 6$, pretraining window size $c = 10$, and local window size $w = 4$ noted by the shadow in (c). The sequence is segmented into chunks to ensure that relative positions do not exceed 9. The matrix element $M[i][j]$ represents the relative position between the i-th query vector $\mathbf{q}$ and the j-th key vector $\mathbf{k}$. Unlike the original position indices for $\mathbf{q}, \mathbf{k}$ in RoPE, DCA utilizes distinct position index sets $P_{\mathbf{k}}$, $P_{\mathbf{q}}^{\text{Intra}}$ (defined in Eq. 2), $P_{\mathbf{q}}^{\text{Inter}}$ (defined in Eq. 5), $P_{\mathbf{q}}^{\text{Succ}}$ (defined in Eq. 7) to compute the relative distances within different sections of M.

ByteDance

- SeedTTS

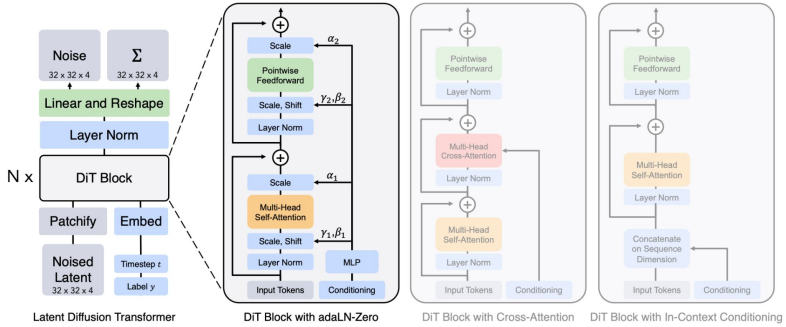


Figure 3. **The Diffusion Transformer (DiT) architecture.** *Left:* We train conditional latent DiT models. The input latent is decomposed into patches and processed by several DiT blocks. *Right:* Details of our DiT blocks. We experiment with variants of standard transformer blocks that incorporate conditioning via adaptive layer norm, cross-attention and extra input tokens. Adaptive layer norm works best.

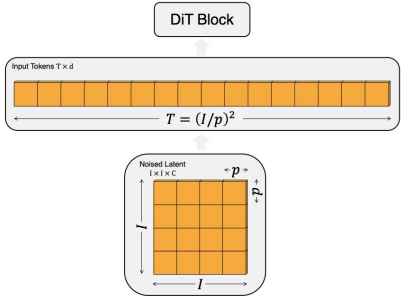


Figure 4. **Input specifications for DiT.** Given patch size $p \times p$, a spatial representation (the noised latent from the VAE) of shape $I \times I \times C$ is "patchified" into a sequence of length $T = (I/p)^2$ with hidden dimension d. A smaller patch size p results in a longer sequence length and thus more Gflops.

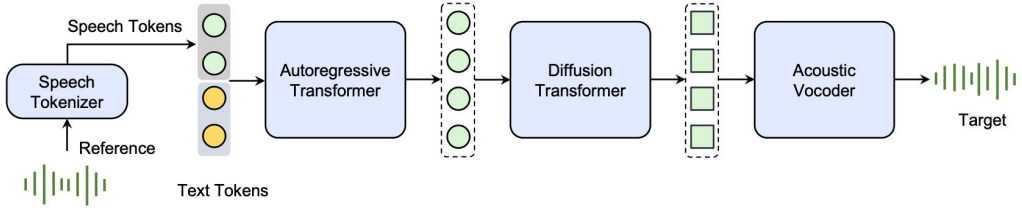


Figure 1. An overview of the Seed-TTS inference pipeline. (1) The speech tokenizer learns tokens from reference speech. (2) The autoregressive language model generates the speech tokens based on the condition text and speech. (3) The diffusion transformer model generates continuous speech representations given generated speech tokens in a coarse-to-fine manner. (4) The acoustic vocoder yields higher-quality speech from the diffusion output.

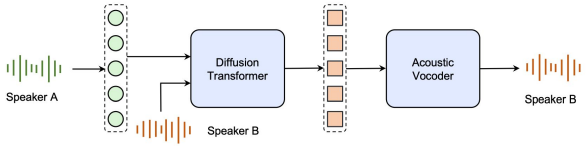


Figure 4: The diagram for zero-shot voice conversion in Seed-TTS system.

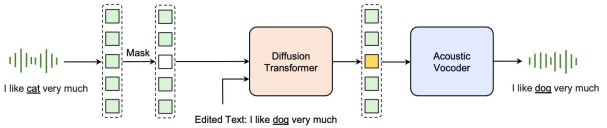


Figure 5. Fully diffusion-based model Seed-TTS$_{\text{DiT}}$, supporting speech content editing. In this example, we replace the word "cat" in the original speech with the word "dog".

MoonShot

Mooncake

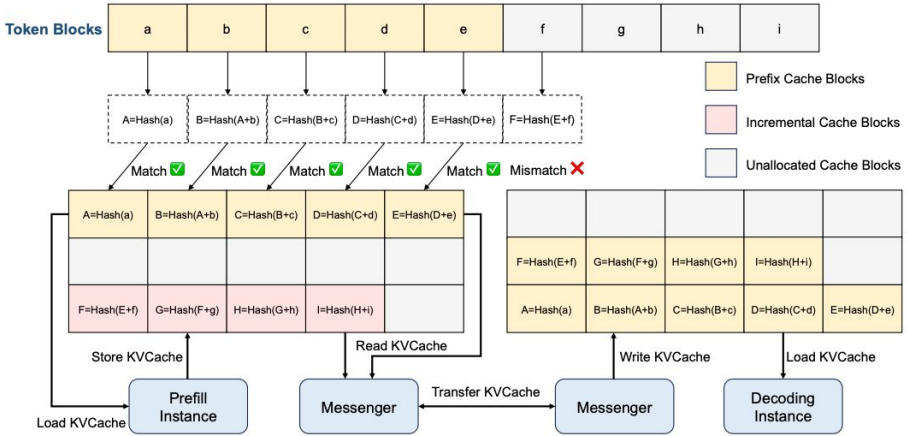


Figure 3: The KVCache pool in CPU memory. Each block is attached with a hash value determined by both its own hash and its prefix for deduplication.

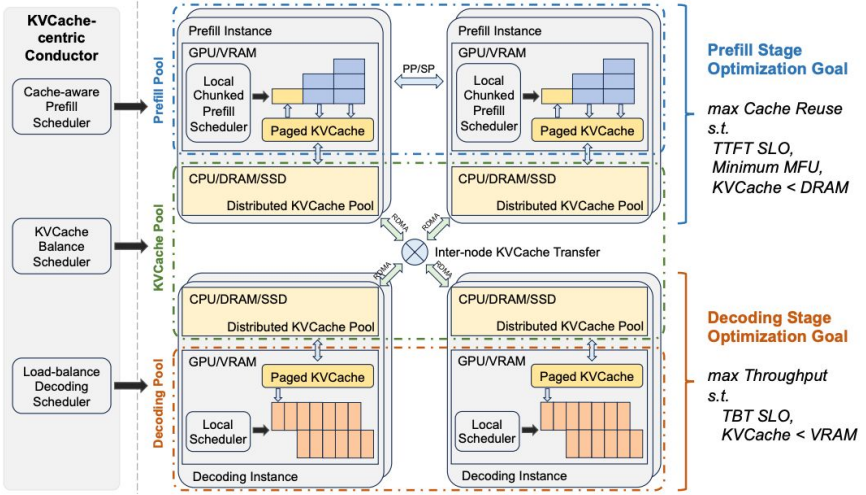


Figure 1: Mooncake Architecture.

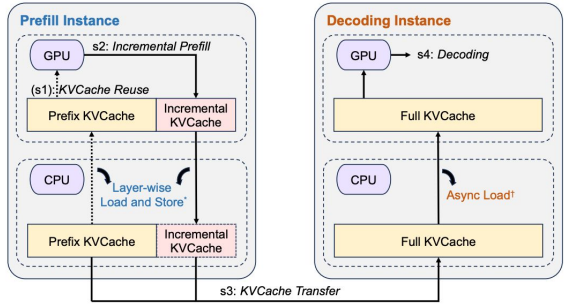


Figure 4: Workflow of inference instances. (∗) For prefill instances, the load and store operations of the KVCache layer are performed layer-by-layer and in parallel with the prefill computation to mitigate transmission overhead (see §5.2). (†) For decoding instances, asynchronous loading is performed concurrently with GPU decoding to prevent GPU idle time.