

Training and Applications of Large Language Models


Chengyue Jiang

2024.4

Large Language Models

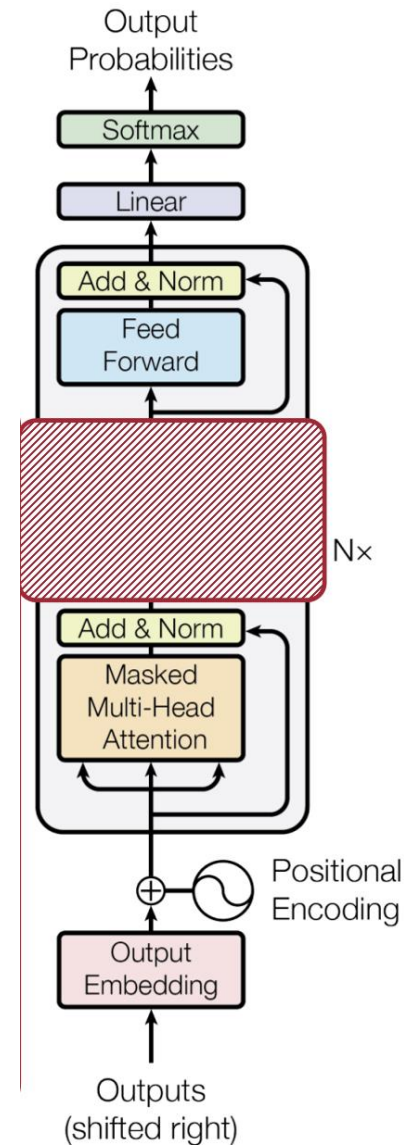
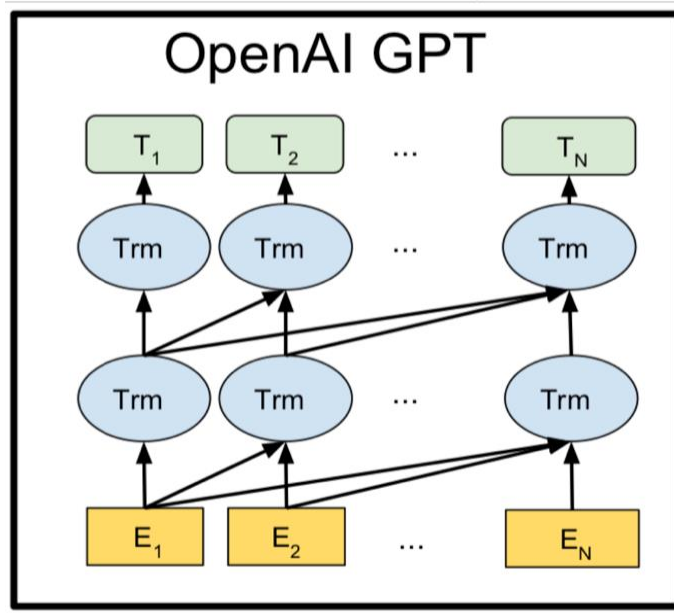
- Have you used some LLMs?
- You use them for?

Topics

- LLM Model and Training
 - Pretraining
 - Data, Infra, Evaluation
 - Supervised Fine-tuning (SFT)
 - Preference Optimization
 - Rejection Sampling, DPO, PPO
- Some LLM Applications
 - “Helpful Assistant” – GPT4  , Kimi, Qwen
 - “Role Play” – 星野, Glow
 - “Embodied Agent (Virtual)” – Generative Agent, Voyager, Cradle

Recap: LLM Basics

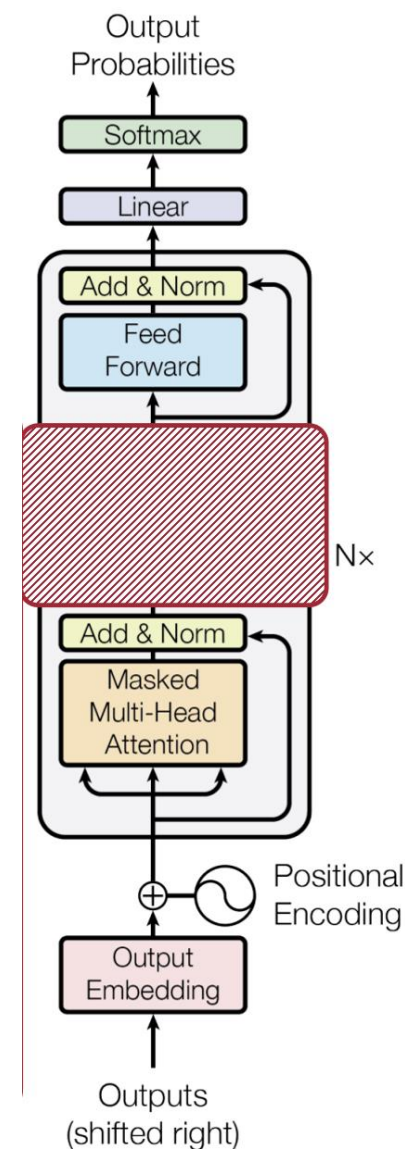
- ▶ Autoregressive
 - ▶ Transformer with attention mask
- ▶ Pre-training loss: Next word prediction
 - ▶ $-\sum_i \log P(w_i | w_{i-k}, \dots, w_{i-1}; \theta)$



$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$-\infty$	$-\infty$	$-\infty$
		$-\infty$	$-\infty$
			$-\infty$

LLM Model Architecture

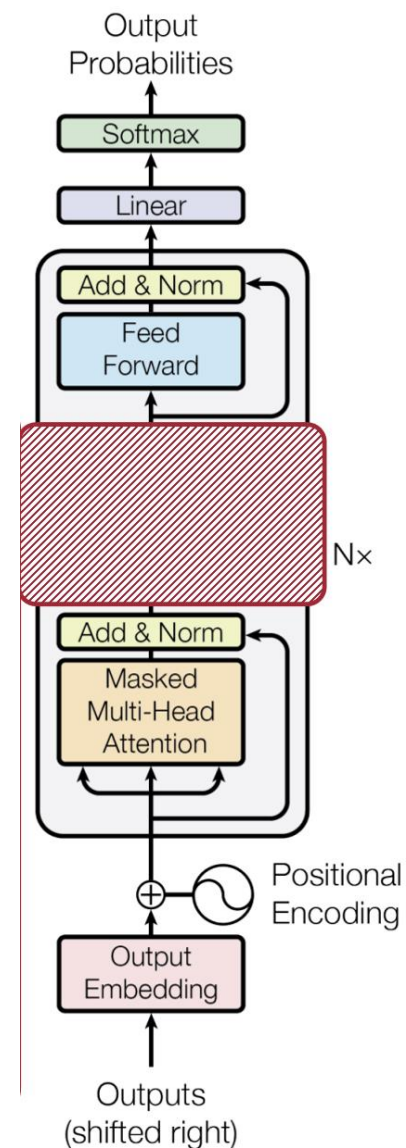
- ▶ Llama2 as example (most popular open source model!) ∞
 - ▶ Lot's of model build on this!
- ▶ Huggingface 🤗 transformer
 - ▶ Most popular lib for LLM model training and inference.
- ▶ Llama2 model architecture in reality?
 - ▶ Is ***almost same*** as we introduced in the lecture!
 - ▶ [huggingface transformer llama2 implementation](#)



```

910 class LlamaModel(LlamaPreTrainedModel):
940     def forward(
992         # embed positions
993         hidden_states = inputs_embeds
994
995         # decoder layers
996         all_hidden_states = () if output_hidden_states else None
997         all_self_attns = () if output_attentions else None
998         next_decoder_cache = None
999
1000         for decoder_layer in self.layers:
1001             if output_hidden_states:
1002                 all_hidden_states += (hidden_states,)
1003
1004             if self.gradient_checkpointing and self.training:
1005                 layer_outputs = self._gradient_checkpointing_func(
1006                     decoder_layer.__call__,
1007                     hidden_states,
1008                     causal_mask,
1009                     position_ids,
1010                     past_key_values,
1011                     output_attentions,
1012                     use_cache,
1013                     cache_position,
1014                 )
1015             else:
1016                 layer_outputs = decoder_layer(
1017                     hidden_states,
1018                     attention_mask=causal_mask,
1019                     position_ids=position_ids,
1020                     past_key_value=past_key_values,
1021                     output_attentions=output_attentions,
1022                     use_cache=use_cache,
1023                     cache_position=cache_position,
1024                 )
1025
1026             hidden_states = layer_outputs[0]

```



```
class LlamaDecoderLayer(nn.Module):
```

```
def forward(
```

```
    hidden_states, self_attn_weights, present_key_value = self.self_attn(
```

```
        hidden_states=hidden_states,
```

```
        attention_mask=attention_mask,
```

```
        position_ids=position_ids,
```

```
        past_key_value=past_key_value,
```

```
        output_attentions=output_attentions,
```

```
        use_cache=use_cache,
```

```
        cache_position=cache_position,
```

```
        **kwargs,
```

```
)
```

```
    hidden_states = residual + hidden_states
```

```
    # Fully Connected
```

```
    residual = hidden_states
```

```
    hidden_states = self.post_attention_layernorm(hidden_states)
```

```
    hidden_states = self.mlp(hidden_states)
```

```
    hidden_states = residual + hidden_states
```

```
    outputs = (hidden_states,)
```

```
    if output_attentions:
```

```
        outputs += (self_attn_weights,)
```

```
    if use_cache:
```

```
        outputs += (present_key_value,)
```

```
    return outputs
```

Self-Attention

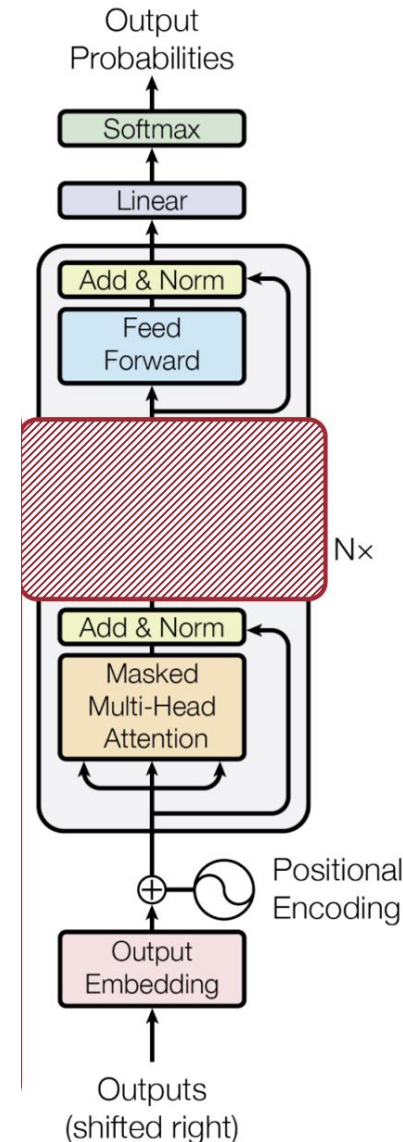
Residual

LayerNorm

MLP

Residual

output for next layer



- ▶ **Almost same**
- ▶ Positional Encoding
 - ▶ RoPE
- ▶ MLP
 - ▶ SwiGLU
- ▶ Normalization
 - ▶ Pre/Post Norm
 - ▶ RMSNorm

LLM Model



- ▶ Huggingface prepared the code for me
 - ▶ I have code for **Model Architecture** and teacher taught me!
 - ▶ I have code for a Trainer and I know the training objective is just **next word prediction!**
 - ▶ I have code for **Inference** and I know it's just **beam/greedy search!**
- ▶ Too simple! I can be another OpenAI !
- ▶ **What is the real hard part for developing a good LLM?**
 - ▶ A good LLM consumes tooooooo much **data** for **pretraining**.
 - ▶ A good LLM requires **at least** hundreds of **Nvidia A100/A800/H100/H200** for **pretraining**.
 - ▶ It's Really hard to tell **what is a good LLM**.

LLM Model Training!

- ▶ Three Phases of LLM model Training.
- ▶ Pretrain
 - ▶ “Read books”
 - ▶ Most of the knowledge is obtained from the pre-training phase.
- ▶ Supervised Fine-tuning
 - ▶ “Answer questions”
 - ▶ Learns to utilize the knowledge for applications.
 - ▶ Helpful Assistant
- ▶ Preference Optimization
 - ▶ “正面教材与反面教材”
 - ▶ Teach models what is bad, what is better for applications
 - ▶ Helpful Assistant: Helpful(Lazy) Safety(Evil)
 - ▶ 伏地魔Bot: Evil(Helpful, Safety)

Llama 2 was trained on **40% more data** than Llama 1,
and has double the context length.

Llama 2

MODEL SIZE (PARAMETERS)	PRETRAINED	FINE-TUNED FOR CHAT USE CASES
7B	<div>Model architecture:</div> <div>Pretraining Tokens: 2 Trillion</div> <div>Context Length: 4096</div>	Data collection for helpfulness and safety:
13B		Supervised fine-tuning: Over 100,000
70B		Human Preferences: Over 1,000,000

LLM Model Pretraining: Method

- ▶ “Read Books” => ...

- ▶ Books, Web data

- ▶ Training Objective

- ▶ **Next word prediction**

- ▶ Data Format

- ▶ Format all text in

- ▶ How? Simply concatenate

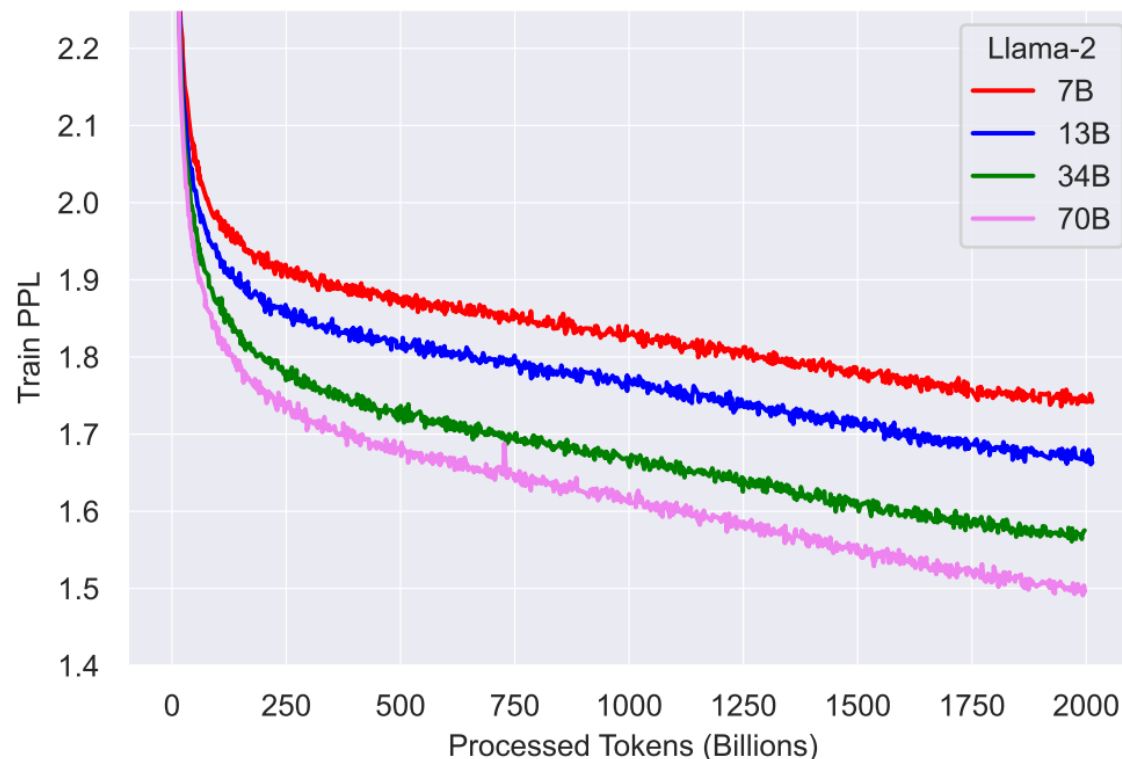
- ▶ Why?

- ▶ Faster training!

- ▶ Model learn to

- ▶ Different Model Size (Number of Parameters)

- ▶ 7B 13B 34B 70B for Llama2 The “scaling law”



LLM Model Pretraining: Data and Computational Cost

		Time (GPU hours)
LLAMA 2	7B	184320
	13B	368640
	34B	1038336
	70B	1720320
Total		3311616

- ▶ More details (Llama2 as example)
- ▶ If you have only **1 A100, 8-10W ¥, 80GB GPUMEM**
- ▶ You need to runs **196 years to finish pretraining Llama2.** on **2T tokens**
- ▶ Actually you **cannot** because
 - ▶ 70B model cannot be placed in 1 GPU
 - ▶ Needs to **split the model parameters into multiple GPUs** to enable training (tensor parallel, TP), or You may need to **offload your parameters / gradients into CPU Mem.**
 - ▶ **Your device will broken** 😊
 - ▶ Single A100's broken rate in 1 month: 8%
 - ▶ optimistic estimate: **2/3 of the A100s will break at least once within a year.**
 - ▶ You cannot live that long. Even if you can, your data will be **196 years behind.**

LLM Model Pretraining: Infrast

- ▶ What you need to train a Llama2 (within 3 month?)
 - ▶ You ***have enough clean data.***
 - ▶ You should have around ***2000 A100s.***
 - ▶ You need to have a ***strong training system (software&hardware)***
 - ▶ You make sure that your training job will not interrupted by the break down of a single card. Because everyday there's a 95% chance you'll find that one of your cards has broken.
 - ▶ You can efficiently utilize 2000 A100s.
 - ▶ accelerate 1000 times is a really good performance for a 2000 card cluster.
 - ▶ Split data / model onto different devices requires ***inter-device Communications.***
 - ▶ ***Distributed Training frameworks ...***
 - ▶ Pytorch
 - ▶ Nvidia-Megatron
 - ▶ Deepspeed
 - ▶ Colossal AI

Llama 2 was trained on **40% more data** than Llama 1,
and has double the context length.

Llama 2

MODEL SIZE (PARAMETERS)	PRETRAINED	FINE-TUNED FOR CHAT USE CASES
7B	Model architecture: Pretraining Tokens: 2 Trillion Context Length: 4096	Data collection for helpfulness and safety: Supervised fine-tuning: Over 100,000 Human Preferences: Over 1,000,000
13B		
70B		

LLM Model Supervised Finetuning: Method

- ▶ “Answer Questions”
 - ▶ Over 100000 (instruction, response) pair. (maybe not enough)
 - ▶ SFT can be more specific
- ▶ Training Objective
 - ▶ **Next word prediction (usually only for the response)**
- ▶ Data Format (**ChatML** format as example)

<|im_start|>system

Assistant is a large language model
trained by OpenAI.

<|im_end|>

<|im_start|>user

Who were the founders of Microsoft?

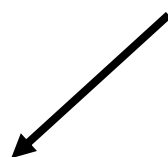
<|im_end|>

<|im_start|>assistant

The founders of Microsoft are xxx

<|im_end|>

Learns to **generate response** and to **STOP**

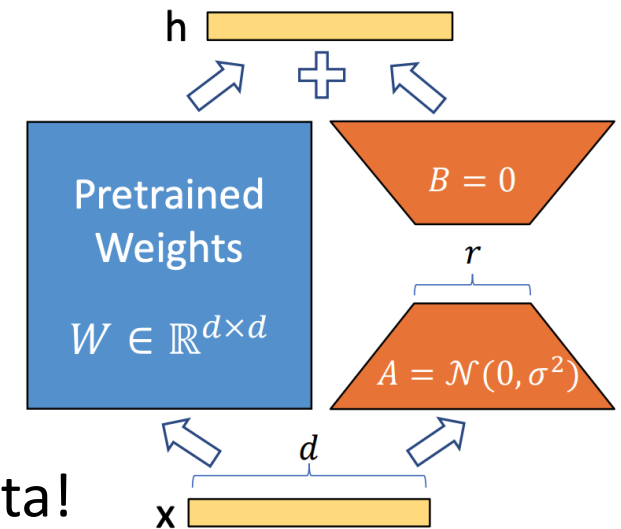


LLM Model Supervised Finetuning: Data

- ▶ Data source
 - ▶ Human annotations
 - ▶ From other open source data
 - ▶ Synthetic data (e.g., the chat log of ChatGPT / GPT4)
- ▶ Meta find: quality of data really matters.
- ▶ Fun facts:
 - ▶ *Quality matters for both stage (pretrain, SFT, preference optimization)*
 - ▶ *Qwen1.5 **improve their SFT and pretrain data** to have a big performance gain.*
 - ▶ *Actually the system prompt (although fixed) is also important, especially you need domain specific application, e.g., train a 伏地魔*
- ▶ What makes a good data? [**Selection** and **Mixing**]
 - ▶ Diversity (domain)
 - ▶ Quality (factuality ...)
 - ▶ No inter-conflicts (consistency)

LLM Model Supervised Finetuning: Infra

- ▶ SFT is much easier
- ▶ **Libs:** Huggingface Transformers, LlamaFactory, etc
- ▶ 1 to 8 4090 / A100/ A800 is enough depend on
 - ▶ Model size
 - ▶ Update all the parameters or use adapter like **LoRA**.
 - ▶ LoRA is a very good choice if you don't have too much data!
 - ▶ Depends on How to represent parameters
 - ▶ **32bit** float, **16bit** float, **8bit** float, or even **4bit** float?
 - ▶ Depends on if use split gradients/model parameters onto multiple GPUs or offload to CPUs.



Llama 2 was trained on **40% more data** than Llama 1,
and has double the context length.

Llama 2

MODEL SIZE (PARAMETERS)	PRETRAINED	FINE-TUNED FOR CHAT USE CASES
7B	Model architecture:	Data collection for helpfulness and safety:
13B	Pretraining Tokens: 2 Trillion	Supervised fine-tuning: Over 100,000
70B	Context Length: 4096	Human Preferences: Over 1,000,000

LLM Model Preference Optimization

- ▶ *Teach Model what is bad, what is better*
- ▶ Preference Optimization *Require Model to have basic ability to respond.*
 - ▶ *Typically After the model is well finetuned*
 - ▶ *Why?*
 - ▶ Preference are from the samples generated by model.
 - ▶ If all responses has low quality, having preference to these response is meaningless
- ▶ *What is Preference*
 - ▶ Given Instruction, Model sample two responses
 - ▶ **Preference:** The response A is better than response B.
 - ▶ Given Instruction, Model sample **k** responses
 - ▶ **Preference:** The response X is the best among K responses.
 - ▶ **Preference:** The rank of k response. (too hard to decide)

LLM Model Preference Optimization

▶ ***Preference Optimization Method***

▶ ***Rejection Sampling (Applied in Llama2 multiple rounds)***

- ▶ Given an instruction, Model outputs *k* samples.
- ▶ ***Human annotator choose the best response.***
- ▶ ***SFT the best response!***
- ▶ *Simple but pretty effective!*

▶ ***Direct Preference Optimization (DPO)***

- ▶ Given an instruction, Model outputs *k* samples.
- ▶ *Find a **good one** and a **bad one**.*

Intuitively:

- ▶ ***Use a loss to encourage increasing the likelihood of good sample***
- ▶ ***And decreasing the likelihood of bad sample***

▶ ***Other method: PPO, KTO, RRHF***

LLM Model Preference Optimization

▶ ***Why we need preference optimization?***

- ▶ 1. Teach model what to not generate.
- ▶ 2. Key reason (In my opinion)
 - ▶ It's ***hard to write the whole good response*** by human!
 - ▶ But is ***easy for a human to judge*** the responses.
 - ▶ Which one is better / the best?
 - ▶ Model can already generate good enough response (even better than human)after SFT
 - ▶ GPT4 is ***too helpful***
 - ▶ Will your classmate be that helpful if you ask him/her some code question?
 - ▶ Maybe Not! Human is very busy and lazy.

Base LLM evaluation (Really Hard)

▶ **Automatic Evaluation**

- ▶ **Code**
- ▶ **Factual Knowledge**
- ▶ **Logic**
- ▶ **Safety**
- ▶ **Commonsense Knowledge (multiple choice)**
- ▶ **Math (gold answer)**
- ▶ **Instruction following**
 - ▶ *E.g., 20字以内, json format*

▶ **Human evaluation**

- ▶ **Chat arena, ELO rating**
- ▶ **Blind !**

▶ **Rating by a very strong model**

- ▶ **GPT4**
- ▶ **Questionable**

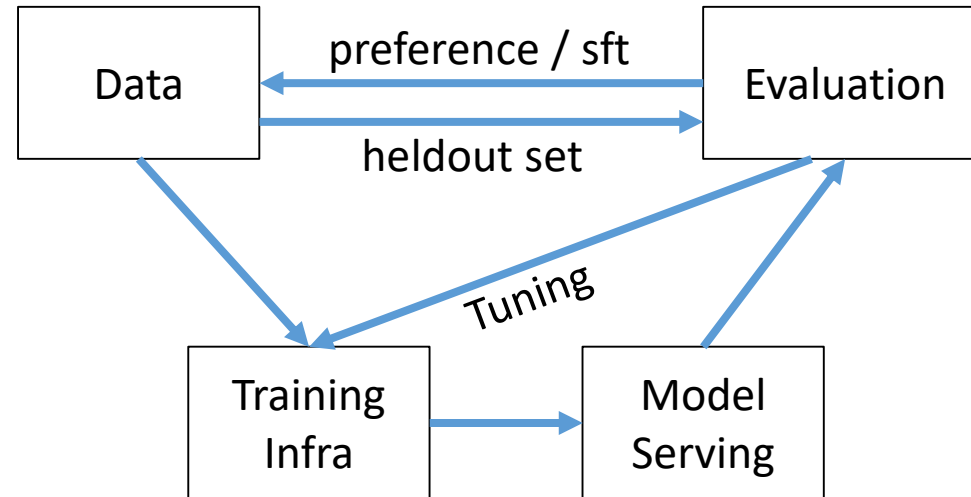
Benchmark (Higher is better)	MPT (7B)	Falcon (7B)	Llama-2 (7B)	Llama-2 (13B)	MPT (30B)	Falcon (40B)	Llama-1 (65B)
MMLU	26.8	26.2	45.3	54.8	46.9	55.4	63.4
TriviaQA	59.6	56.8	68.9	77.2	71.3	78.6	84.5
Natural Questions	17.8	18.1	22.7	28.0	23.0	29.5	31.0
GSM8K	6.8	6.8	14.6	28.7	15.2	19.6	50.9
HumanEval	18.3	N/A	12.8	18.3	25.0	N/A	23.7
AGIEval (English tasks only)	23.5	21.2	29.3	39.1	33.8	37.0	47.6
BoolQ	75.0	67.5	77.4	81.7	79.0	83.1	85.3
HellaSwag	76.4	74.1	77.2	80.7	79.9	83.6	84.2
OpenBookQA	51.4	51.6	58.6	57.0	52.0	56.6	60.2
QuAC	37.7	18.8	39.7	44.8	41.1	43.3	39.8
Winogrande	68.3	66.3	69.2	72.8	71.0	76.9	77.0

LLM evaluation depends on your application

▶ 伏地魔 *Bot*

- ▶ *Safety ?*
- ▶ *Able to code?*
- ▶ *Able to answer factual knowledge in China?*

Summary of LLM training



Applications: “You are a helpful assistant ...”



CLAUDE 3



GEMINI

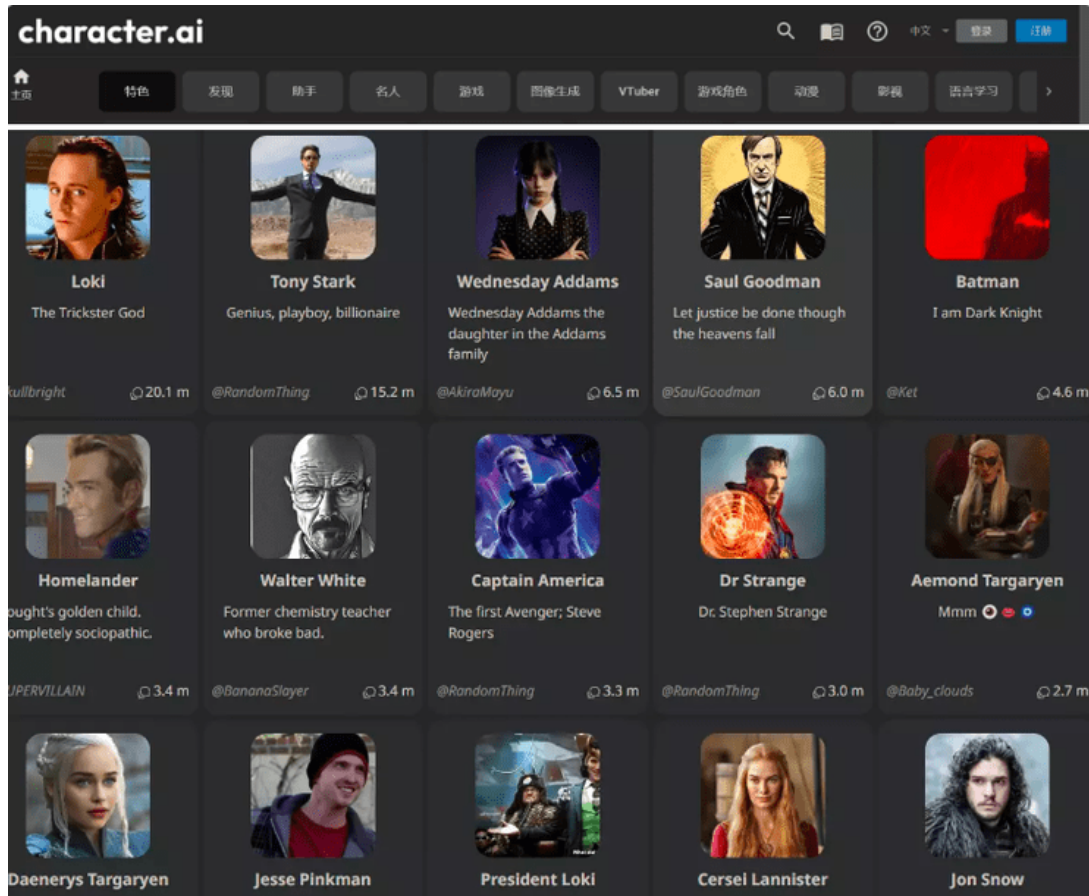


GPT-4

Coding
Doing Homework
Translate
Summarization

Applications: Emotional Companionship

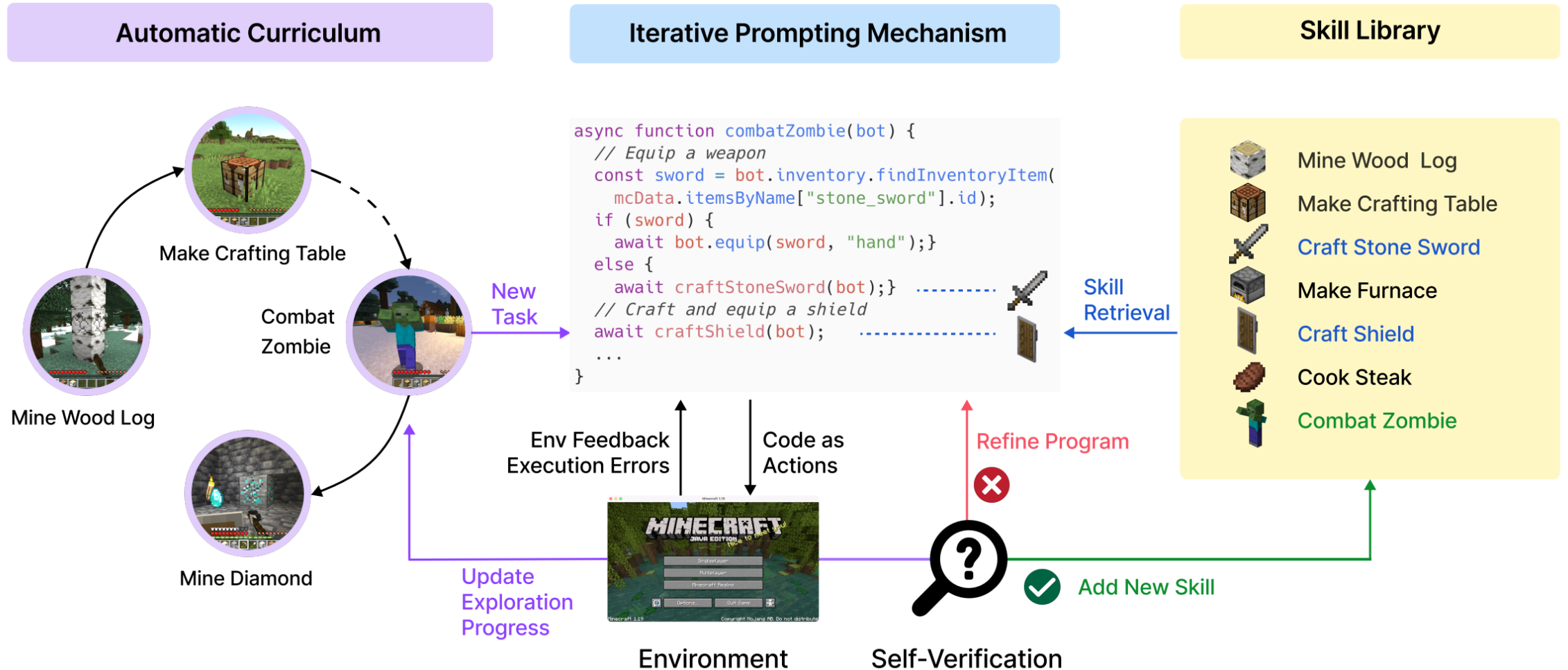
“Roleplay”



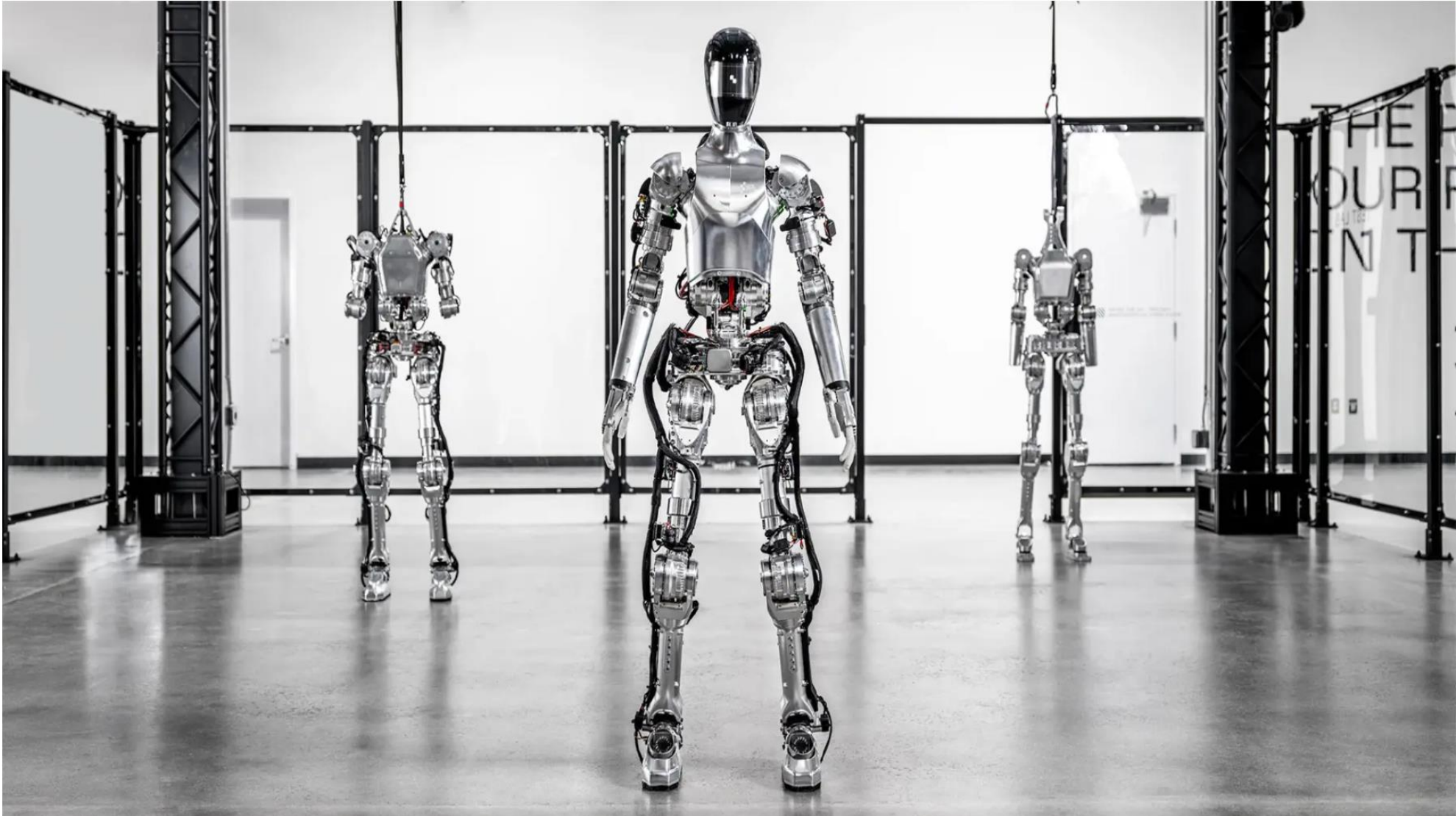
Applications: Brain of Game NPC



Applications: Brain of Game NPC



Applications: Embodied AI In Reality / Virtual World



Multi-modality as Input & Output

▶ *Inputs: “Environment”*

- ▶ *Text*
- ▶ *Image / Video*
- ▶ *Sound*
- ▶ *...*

▶ *Outputs: “Actions / Decisions”*

- ▶ *Text*
- ▶ *Image / Video*
- ▶ *Sound*
- ▶ *Skeleton Animation*
- ▶ *...*

Suno.AI: ? -> Sound

OpenAI Sora: ? -> Video

GPT4V/Claude3: ? -> Text

Thanks