Large Language Diffusion Models

Presented by NanoLLaDA

MARCH 2025

Large Language Diffusion Models

Shen Nie [1 * †] **Fengqi Zhu** [1 * †] **Zebin You** [1 †] **Xiaolu Zhang** [2 ‡] **Jingyang Ou** [1] **Jun Hu** [2 ‡] **Jun Zhou** [2]
Yankai Lin [1 ‡] **Ji-Rong Wen** [1] **Chongxuan Li** [1 ‡ ¶]

Abstract

Autoregressive models (ARMs) are widely regarded as the cornerstone of large language models (LLMs). We challenge this notion by introducing **LLaDA**, a diffusion model trained from scratch under the pre-training and supervised fine-tuning (SFT) paradigm. LLaDA models distributions through a forward data masking process and a reverse process, parameterized by a vanilla Transformer to predict masked tokens. By optimizing a likelihood bound, it provides a principled generative approach for probabilistic inference. Across extensive benchmarks, LLaDA demonstrates strong *scalability*, outperforming our self-constructed ARM baselines. Remarkably, LLaDA 8B is competitive with strong LLMs like LLaMA3 8B in *in-context learning* and, after SFT, exhibits impressive *instruction-following* abilities in case studies such as multi-turn dialogue. Moreover, LLaDA addresses the reversal curse, surpassing GPT-4o in a reversal poem completion task. Our findings establish diffusion models as a viable and promising alternative to ARMs, challenging the assumption that key LLM capabilities discussed above are inherently tied to ARMs. Project page and codes: `https://ml-gsai.github.io/LLaDA-demo/`.

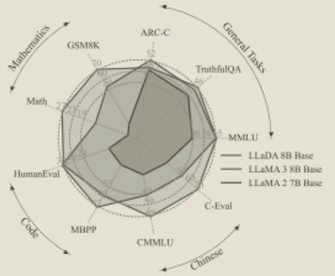


Figure 1. **Zero/Few-Shot Benchmarks.** We scale LLaDA to an unprecedented size of 8B parameters from scratch, achieving competitive performance with strong LLMs (Dubey et al., 2024).

1. Introduction

> *What is now proved was once only imagined.*
> —William Blake

Large language models (LLMs) (Zhao et al., 2023) fall entirely within the framework of *generative modeling*. Specifically, LLMs aim to capture the true but unknown language distribution $p_{\text{data}}(\cdot)$ by optimizing a model distribution $p_\theta(\cdot)$ through maximum likelihood estimation, or equivalently KL divergence minimization between the two distributions:

$$\underbrace{\max_\theta \mathbb{E}_{p_{\text{data}}(x)} \log p_\theta(x) \Leftrightarrow \min_\theta \mathrm{KL}(p_{\text{data}}(x) || p_\theta(x))}_{\text{Generative modeling principles}}. \quad (1)$$

The predominant approach relies on the *autoregressive* modeling (ARM)—commonly referred to as the *next-token prediction* paradigm—to define the model distribution:

$$\underbrace{p_\theta(x) = p_\theta(x^1) \prod_{i=2}^{L} p_\theta(x^i \mid x^1, \ldots, x^{i-1})}_{\text{Autoregressive formulation}}, \quad (2)$$

where x is a sequence of length L, and x^i is the i-th token.

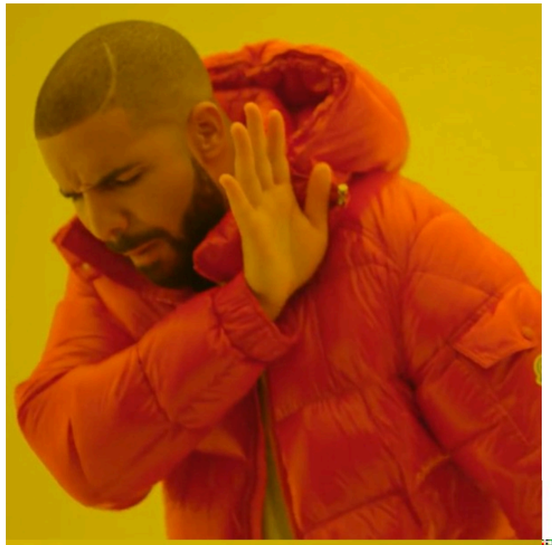
This paradigm has proven remarkably effective (Radford, 2018; Radford et al., 2019; Brown, 2020; OpenAI, 2022) and has become the foundation of current LLMs. Despite its widespread adoption, a fundamental question remains unanswered: *Is the autoregressive paradigm the only viable path to achieving the intelligence exhibited by LLMs?*

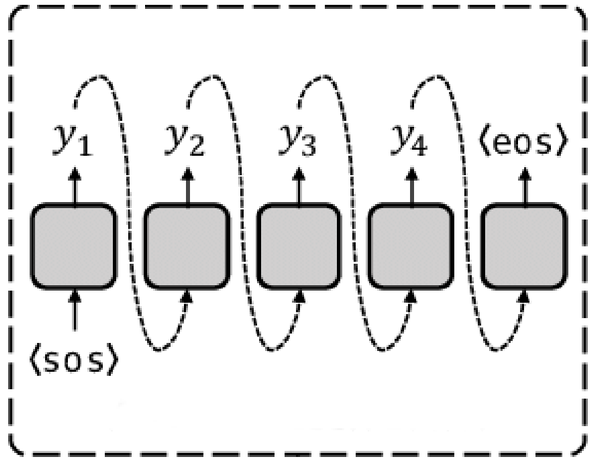
[*]Equal contribution [†]Work done during an internship at Ant Group [‡]Project leaders [1]Gaoling School of Artificial Intelligence, Renmin University of China; Beijing Key Laboratory of Big Data Management and Analysis Methods [2]Ant Group. [¶]Correspondence to: Chongxuan Li <chongxuanli@ruc.edu.cn>.

Preprint.

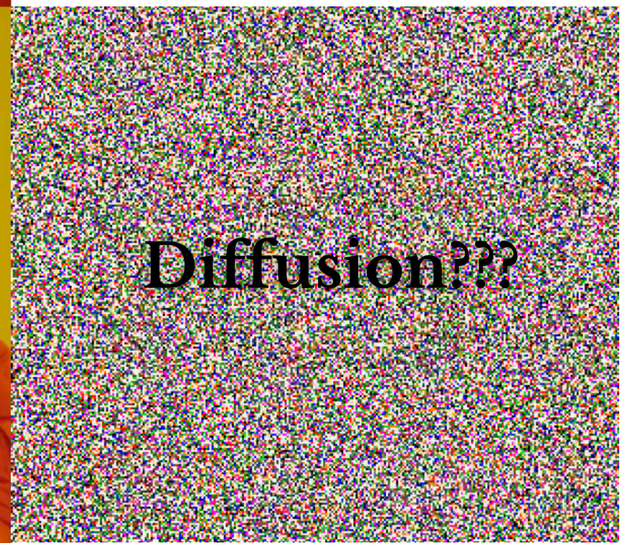
What is being done?

LLaDA: Large Language Diffusion with mAsking





Autoregressive Model



How is it different from previous work?

distribution $p_{\text{data}}(\cdot)$ by optimizing a model distribution $p_\theta(\cdot)$ through maximum likelihood estimation, or equivalently KL divergence minimization between the two distributions:

$$\underbrace{\max_\theta \mathbb{E}_{p_{\text{data}}(x)} \log p_\theta(x) \Leftrightarrow \min_\theta \mathrm{KL}(p_{\text{data}}(x)||p_\theta(x))}_{\text{Generative modeling principles}}. \quad (1)$$

The predominant approach relies on the *autoregressive* modeling (ARM)—commonly referred to as the *next-token prediction* paradigm—to define the model distribution:

$$\underbrace{p_\theta(x) = p_\theta(x^1) \prod_{i=2}^{L} p_\theta(x^i \mid x^1, \ldots, x^{i-1}),}_{\text{Autoregressive formulation}} \quad (2)$$

where x is a sequence of length L, and x^i is the i-th token.

The core of LLaDA is a *mask predictor*, a parametric model $p_\theta(\cdot|x_t)$ that takes x_t as input and predicts all masked tokens (denoted M) simultaneously. It is trained using a cross-entropy loss computed only on the masked tokens:

$$\mathcal{L}(\theta) \triangleq -\mathbb{E}_{t,x_0,x_t} \left[\frac{1}{t} \sum_{i=1}^{L} \mathbf{1}[x_t^i = \mathrm{M}] \log p_\theta(x_0^i|x_t) \right], \quad (3)$$

where x_0 is sampled from the training data, t is sampled uniformly from $[0, 1]$, and x_t is sampled from the forward process. The indicator function $\mathbf{1}[\cdot]$ ensures that the loss is computed only for masked tokens.

How is it being done?

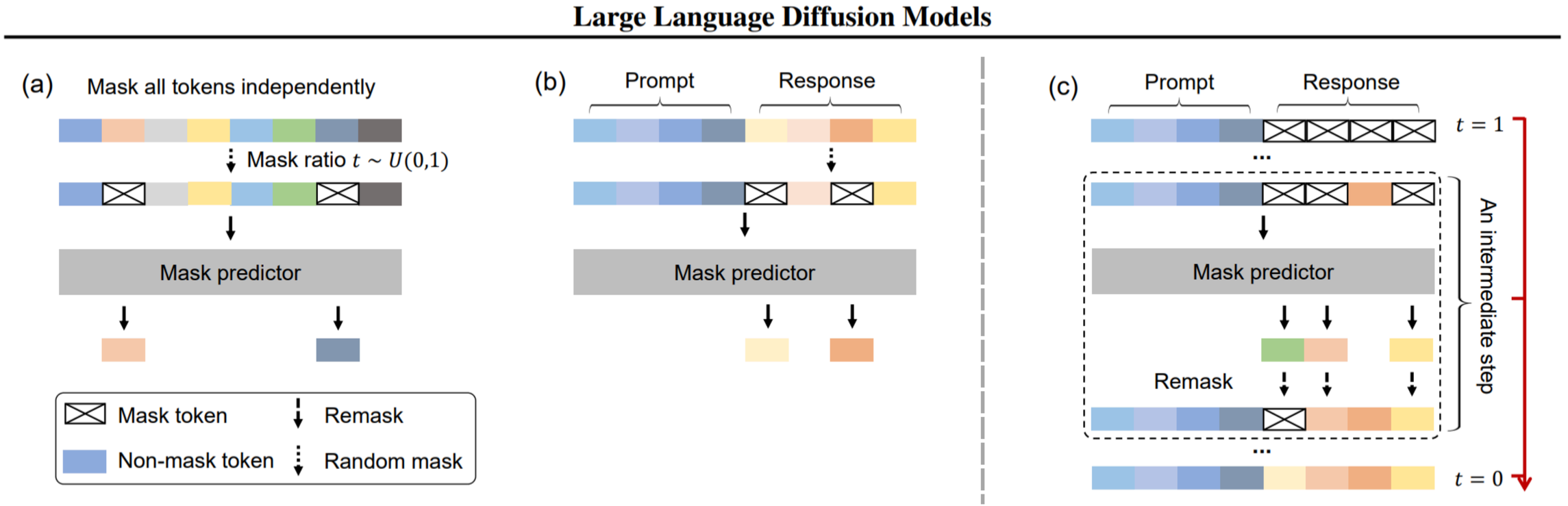


Figure 2. **A Conceptual Overview of LLaDA.** (a) Pre-training. LLaDA is trained on text with random masks applied independently to all tokens at the same ratio $t \sim U[0, 1]$. (b) SFT. Only response tokens are possibly masked. (c) Sampling. LLaDA simulates a diffusion process from $t = 1$ (fully masked) to $t = 0$ (unmasked), predicting all masks simultaneously at each step with flexible remask strategies.

How is it different from previous work?

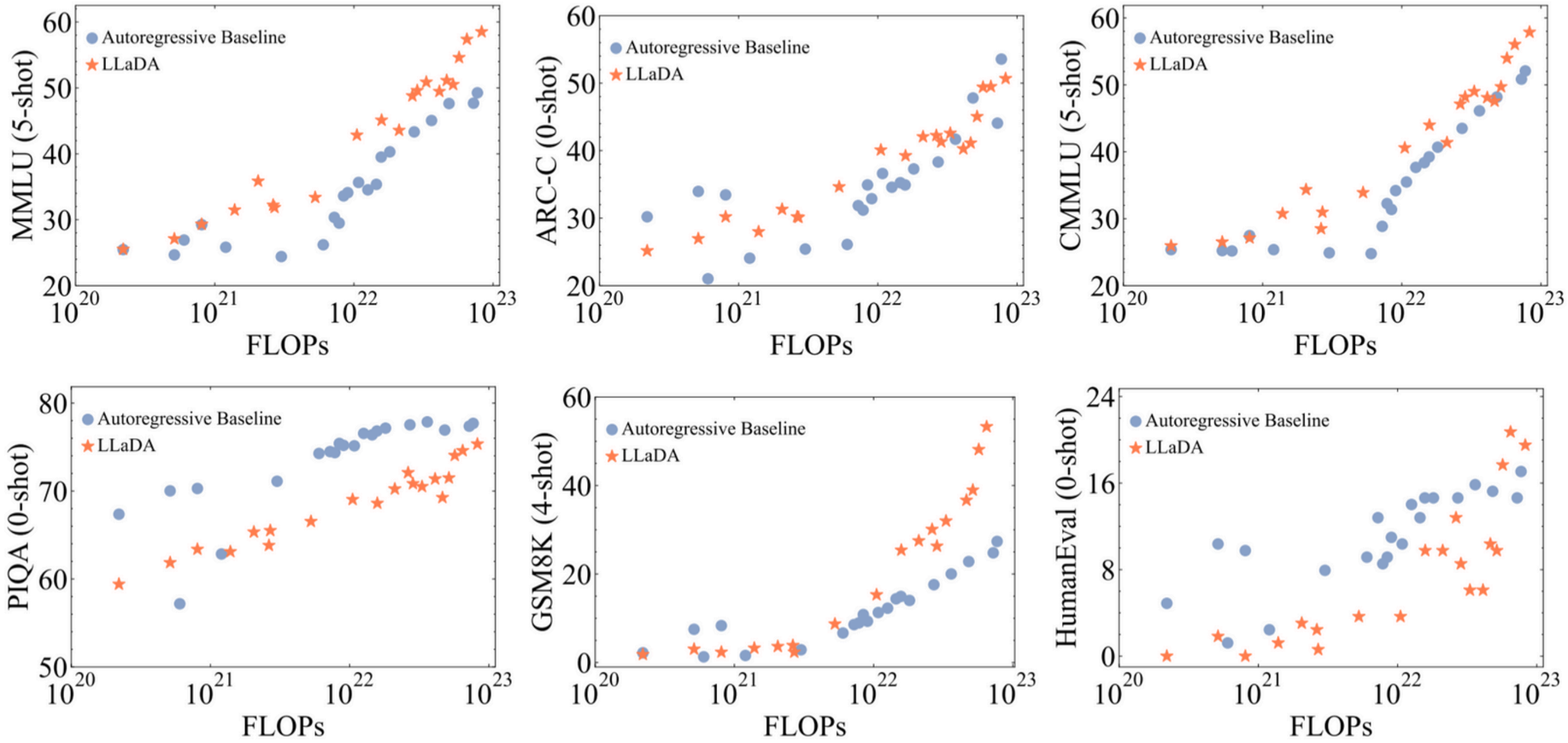


Figure 3. **Scalability of LLaDA.** We evaluate the performance of LLaDA and our ARM baselines trained on the same data across increasing computational FLOPs. LLaDA exhibits strong scalability, matching the overall performance of ARMs on six tasks.

What is the dataset?

- **Pretraining**: 2.3 trillion tokens with
  - low-quality content filtered web corpora
  - high-quality code, math, multilingual data.
- **SFT** (Supervised Fine-Tuning): 4.5 million prompt-response pairs.
- **Sequence length**: up to 4096 tokens.
- **Tokenizer**: Custom BPE tokenizer (vocab size ~126k–128k).
- **No dataset-specific tricks;** follows standard LLM data protocols.

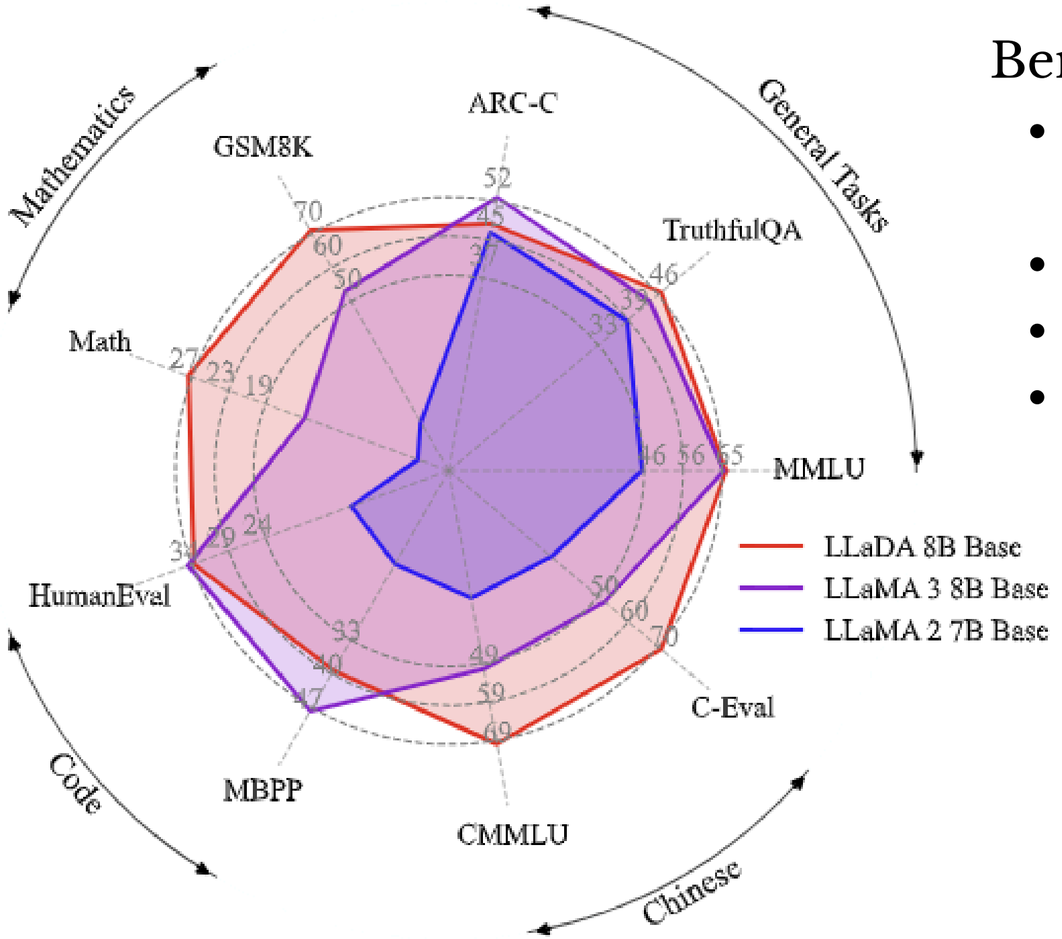
Evaluation

What are the baselines?

	Our ARM Baseline 1B	LLaDA 1B	Our ARM Baseline 7B	LLaDA 8B	LLaMA3 8B
Layers	22	22	28	32	32
Model dimension	2048	2048	4096	4096	4096
Attention heads	32	32	32	32	32
Vocabulary size	126,464	126,464	126,464	126,464	128,000
FFN dimension	5634	5634	13,440	12,288	14,336
Key/Value heads	4	4	8	32	8
Total parameters	1.49 B	1.49 B	6.83 B	8.02 B	8.03 B
Non-embedding parameters	0.97 B	0.97 B	5.80 B	6.98 B	6.98 B

Evaluation

What are the baselines?



Benchmarks:

- General: MMLU, ARC-C, BBH, TruthfulQA, Hellaswag, PIQA, WinoGrande
- Math/Science: GSM8K, Math, GPQA, iGSM
- Code: HumanEval, HumanEval-FIM, MBPP
- Chinese: CMMLU, C-Eval

Evaluation

Table 1. **Benchmark Results of Pre-trained LLMs.** * indicates that LLaDA 8B Base, LLaMA2 7B Base, and LLaMA3 8B Base are evaluated under the same protocol, detailed in Appendix B.5. Results indicated by † and ¶ are sourced from Chu et al. (2024); Yang et al. (2024) and Bi et al. (2024) respectively. The numbers in parentheses represent the number of shots used for evaluation. "-" indicates unknown data.

	LLaDA 8B*	LLaMA3 8B*	LLaMA2 7B*	Qwen2 7B†	Qwen2.5 7B†	Mistral 7B†	Deepseek 7B¶
Model	Diffusion	AR	AR	AR	AR	AR	AR
Training tokens	2.3T	15T	2T	7T	18T	-	2T
General Tasks							
MMLU	**65.9** (5)	65.4 (5)	45.9 (5)	70.3 (5)	74.2 (5)	64.2 (5)	48.2 (5)
BBH	49.8 (3)	**57.6** (3)	37.3 (3)	62.3 (3)	70.4 (3)	56.1 (3)	39.5 (3)
ARC-C	47.9 (0)	**53.1** (0)	46.3 (0)	60.6 (25)	63.7 (25)	60.0 (25)	48.1 (0)
Hellaswag	72.5 (0)	**79.1** (0)	76.0 (0)	80.7 (10)	80.2 (10)	83.3 (10)	75.4 (0)
TruthfulQA	**46.4** (0)	44.0 (0)	39.0 (0)	54.2 (0)	56.4 (0)	42.2 (0)	-
WinoGrande	74.8 (5)	**77.3** (5)	72.5 (5)	77.0 (5)	75.9 (5)	78.4 (5)	70.5 (0)
PIQA	74.4 (0)	**80.6** (0)	79.1 (0)	-	-	-	79.2 (0)
Mathematics & Science							
GSM8K	**70.7** (4)	53.1 (4)	14.3 (4)	80.2 (4)	85.4 (4)	36.2 (4)	17.4 (8)
Math	**27.3** (4)	15.1 (4)	3.2 (4)	43.5 (4)	49.8 (4)	10.2 (4)	6.0 (4)
GPQA	**26.1** (5)	25.9 (5)	25.7 (5)	30.8 (5)	36.4 (5)	24.7 (5)	-
Code							
HumanEval	33.5 (0)	**34.2** (0)	12.8 (0)	51.2 (0)	57.9 (0)	29.3 (0)	26.2 (0)
HumanEval-FIM	**73.8** (2)	73.3 (2)	26.9 (2)	-	-	-	-
MBPP	38.2 (4)	**47.4** (4)	18.4 (4)	64.2 (0)	74.9 (0)	51.1 (0)	39.0 (3)
Chinese							
CMMLU	**69.9** (5)	50.7 (5)	32.5 (5)	83.9 (5)	-	-	47.2 (5)
C-Eval	**70.5** (5)	51.7 (5)	34.0 (5)	83.2 (5)	-	-	45.0 (5)

Evaluation

Table 2. **Benchmark Results of Post-trained LLMs.** LLaDA only employs an SFT procedure while other models have extra reinforcement learning (RL) alignment. * indicates that LLaDA 8B Instruct, LLaMA2 7B Instruct, and LLaMA3 8B Instruct are evaluated under the same protocol, detailed in Appendix B.5. Results indicated by † and ¶ are sourced from Yang et al. (2024) and Bi et al. (2024) respectively. The numbers in parentheses represent the number of shots used for in-context learning. "-" indicates unknown data.

	LLaDA 8B*	LLaMA3 8B*	LLaMA2 7B*	Qwen2 7B†	Qwen2.5 7B†	Gemma2 9B†	Deepseek 7B¶
Model	Diffusion	AR	AR	AR	AR	AR	AR
Training tokens	2.3T	15T	2T	7T	18T	8T	2T
Post-training	SFT	SFT+RL	SFT+RL	SFT+RL	SFT+RL	SFT+RL	SFT+RL
Alignment pairs	4.5M	-	-	0.5M + -	1M + 0.15M	-	1.5M + -
General Tasks							
MMLU	65.5 (5)	**68.4** (5)	44.1 (5)	-	-	-	49.4 (0)
MMLU-pro	37.0 (0)	**41.9** (0)	4.6 (0)	44.1 (5)	56.3 (5)	52.1 (5)	-
Hellaswag	74.6 (0)	**75.5** (0)	51.5 (0)	-	-	-	68.5 (-)
ARC-C	**88.5** (0)	82.4 (0)	57.3 (0)	-	-	-	49.4 (-)
Mathematics & Science							
GSM8K	**78.6** (4)	78.3 (4)	29.0 (4)	85.7 (0)	91.6 (0)	76.7 (0)	63.0 (0)
Math	26.6 (0)	**29.6** (0)	3.8 (0)	52.9 (0)	75.5 (0)	44.3 (0)	15.8 (0)
GPQA	31.8 (5)	**31.9** (5)	28.4 (5)	34.3 (0)	36.4 (0)	32.8 (0)	-
Code							
HumanEval	47.6 (0)	**59.8** (0)	16.5 (0)	79.9 (0)	84.8 (0)	68.9 (0)	48.2 (-)
MBPP	34.2 (4)	**57.6** (4)	20.6 (4)	67.2 (0)	79.2 (0)	74.9 (0)	35.2 (-)

Evaluation

Table 3. **Comparison in the Poem Completion Task**.

	Forward	Reversal
GPT-4o (2024-08-06)	**82.7**	34.3
Qwen2.5 7B Instruct	75.9	38.0
LLaDA 8B Instruct	48.8	**42.4**

Practicality

Prone to parameter tuning?

- No hyperparameter tuning during training.

Computing resource / Runtime (training and testing)

- Pretraining cost: 0.13 million H800 GPU hours for 2.3 trillion tokens

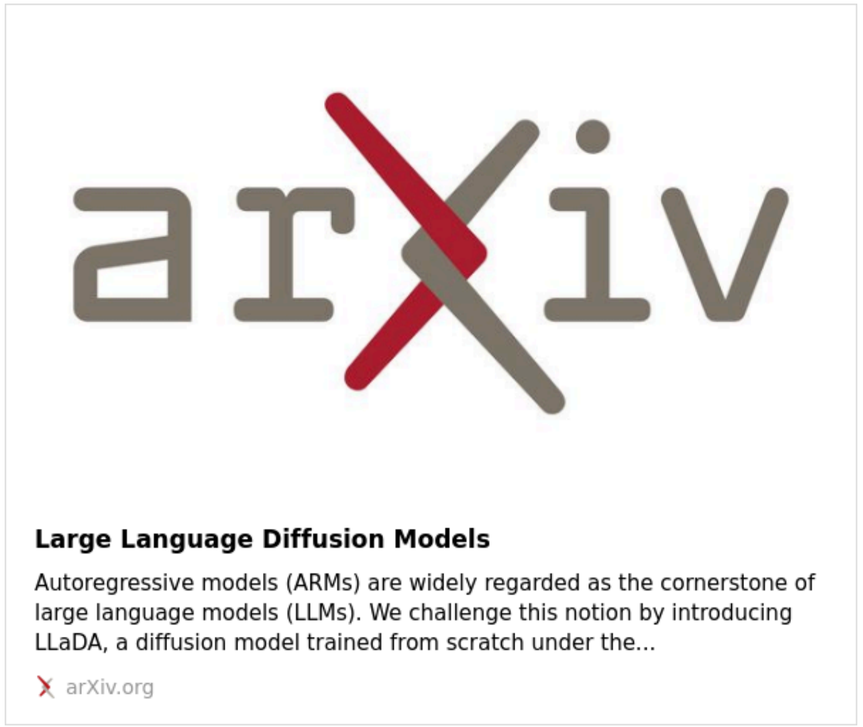
Inference:

- Slower than ARMs (due to no KV caching and multi-step generation)
- Sampling step-dependent, but efficient remasking helps mitigate

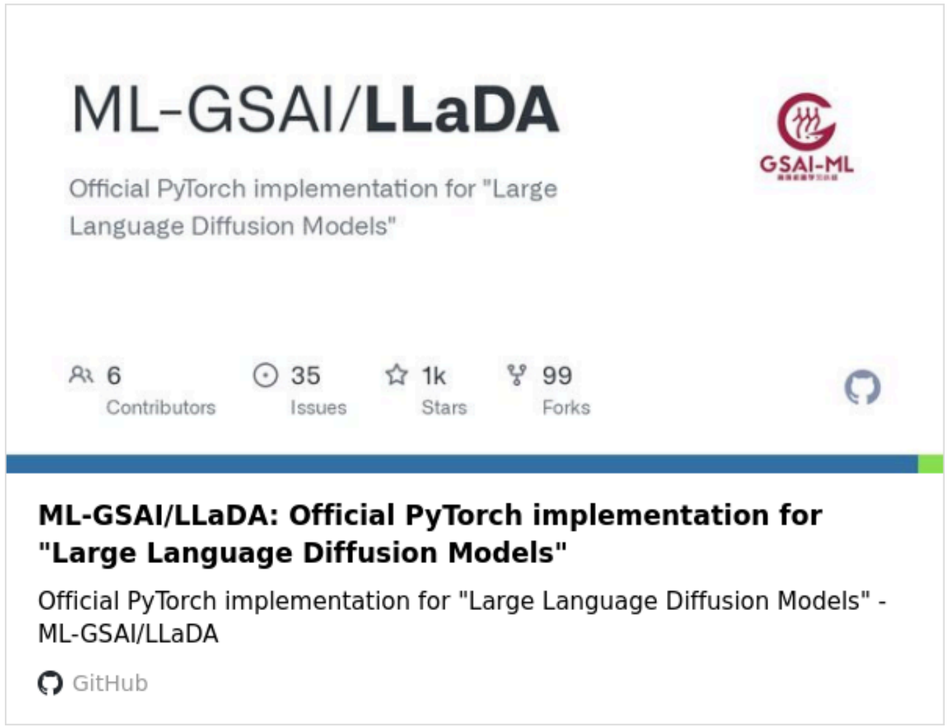
Other assumptions?

- Supervised paradigm: Pretraining + SFT
- No reinforcement learning (unlike many instruct-tuned LLMs)
- Uses Transformer without causal masking — bidirectional modeling
- Trained once without iterative refinement or alignment

References



https://arxiv.org/abs/2502.09992



https://github.com/ML-GSAI/LLaDA.git

Appendices

Algorithm 1 Pre-training of LLaDA

Require: mask predictor p_θ, data distribution p_{data}

repeat
  $x_0 \sim p_{\text{data}}, t \sim \text{U}(0,1]$ # with a probability of 1%, the sequence length of x_0 follows $\text{U}[1, 4096]$
  $x_t \sim q_{t|0}(x_t|x_0)$ # $q_{t|0}$ is defined in Eq. (7)
  Calculate $\mathcal{L} = -\frac{1}{t*L}\sum_{i=1}^{L} \mathbf{1}[x_t^i = \text{M}] \log p_\theta(x_0^i|x_t)$ # L is the sequence length of x_0
  Calculate $\nabla_\theta \mathcal{L}$ and run optimizer.
until Converged
Return p_θ

Algorithm 2 Supervised Fine-Tuning of LLaDA

Require: mask predictor p_θ, pair data distribution p_{data}

repeat
  $p_0, r_0 \sim p_{\text{data}}, t \sim \text{U}(0,1]$ # please refer to Appendix B.1 for details on the SFT data processing.
  $r_t \sim q_{t|0}(r_t|r_0)$ # $q_{t|0}$ is defined in Eq. (7)
  Calculate $\mathcal{L} = -\frac{1}{t*L'}\sum_{i=1}^{L'} \mathbf{1}[r_t^i = \text{M}] \log p_\theta(r_0^i|p_0, r_t)$ # L' is the sequence length of r_0
  Calculate $\nabla_\theta \mathcal{L}$ and run optimizer.
until Converged
Return p_θ

Algorithm 3 Conditional Log-likelihood Evaluation of LLaDA

Require: mask predictor p_θ, prompt p_0, response r_0, the number of Monte Carlo estimations n_{mc}

log_likelihood $= 0$
for $i \leftarrow 1$ to n_{mc} **do**
  $l \sim \{1, 2, \ldots, L\}$ # L is the sequence length of r_0
  Obtain r_l by uniformly sampling l tokens from r_0 without replacement for masking
  log_likelihood $=$ log_likelihood $+ \frac{L}{l}\sum_{i=1}^{L} \mathbf{1}[r_l^i = \text{M}] \log p_\theta(r_0^i|p_0, r_l)$
end for
log_likelihood $=$ log_likelihood$/n_{mc}$
Return log_likelihood

Algorithm 4 Reverse Process of LLaDA

Require: mask predictor p_θ, prompt p_0, answer length L, sampling steps N

Set r_1 is a fully masked sequence of length L.
for $t \leftarrow 1$ **down to** $\frac{1}{N}$ **step** $\frac{1}{N}$ **do**
  $s = t - \frac{1}{N}$
  $r_0 = \arg\max_{r_0} p_\theta(r_0 | p_0, r_t)$ # we employ greedy sampling when predicting masked tokens
  for $i \leftarrow 1$ to L **do**
    if $r_t \neq$ M **then**
      $r_0^i = r_t^i$
    else
      With probability $\frac{s}{t}$, r_0^i is set to M
    end if
  end for
  $r_s = r_0$
end for
Return r_0

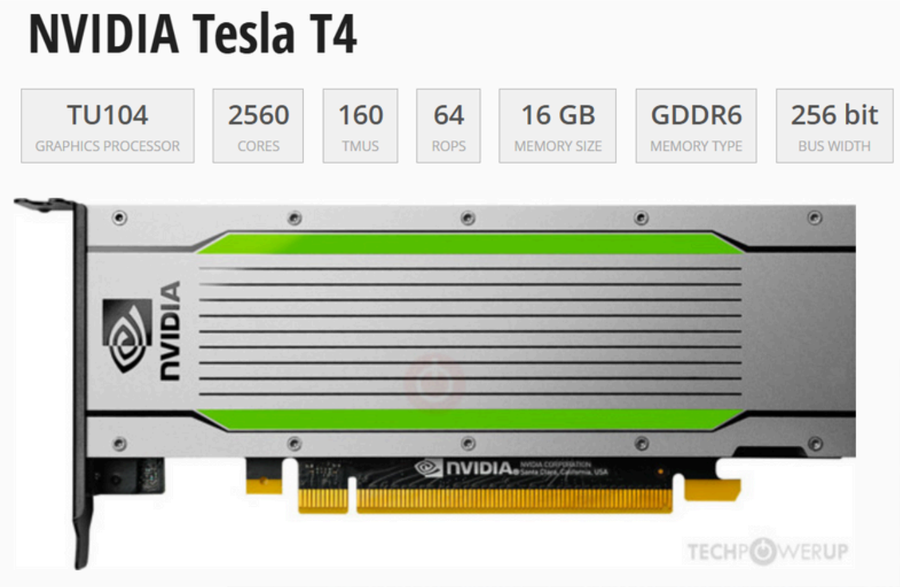
Algorithm 5 Low-confidence Remasking Strategy of LLaDA

Require: mask predictor p_θ, prompt p_0, answer length L, sampling steps N

Set r_1 is a fully masked sequence of length L.
for $t \leftarrow 1$ **down to** $\frac{1}{N}$ **step** $\frac{1}{N}$ **do**
  $s = t - \frac{1}{N}$
  for $i \leftarrow 1$ to L **do**
    if $r_t^i \neq \mathrm{M}$ **then**
      $r_0^i = r_t^i, c^i = 1$
    else
      $r_0^i = \arg\max_{r_0^i} p_\theta(r_0^i | p_0, r_t)$
      $c^i = p_\theta(r_0^i | p_0, r_t)_{r_0^i}$
    end if
  end for
  $n_{un} = \lfloor L(1-s) \rfloor$ # the number of unmasked tokens is n_{un} in timestep s
  for $i \leftarrow 1$ to L **do**
    if $c^i \in \text{Lowest} - n_{un}\left(\{c^i\}_1^L\right)$ **then**
      $r_0^i = \mathrm{M}$ # the n_{un} positions with the least confidence are selected for remasking.
    end if
  end for
  $r_s = r_0$
end for
Return r_0

PROJECT PROGRESS

Colab: CUDA out of memory



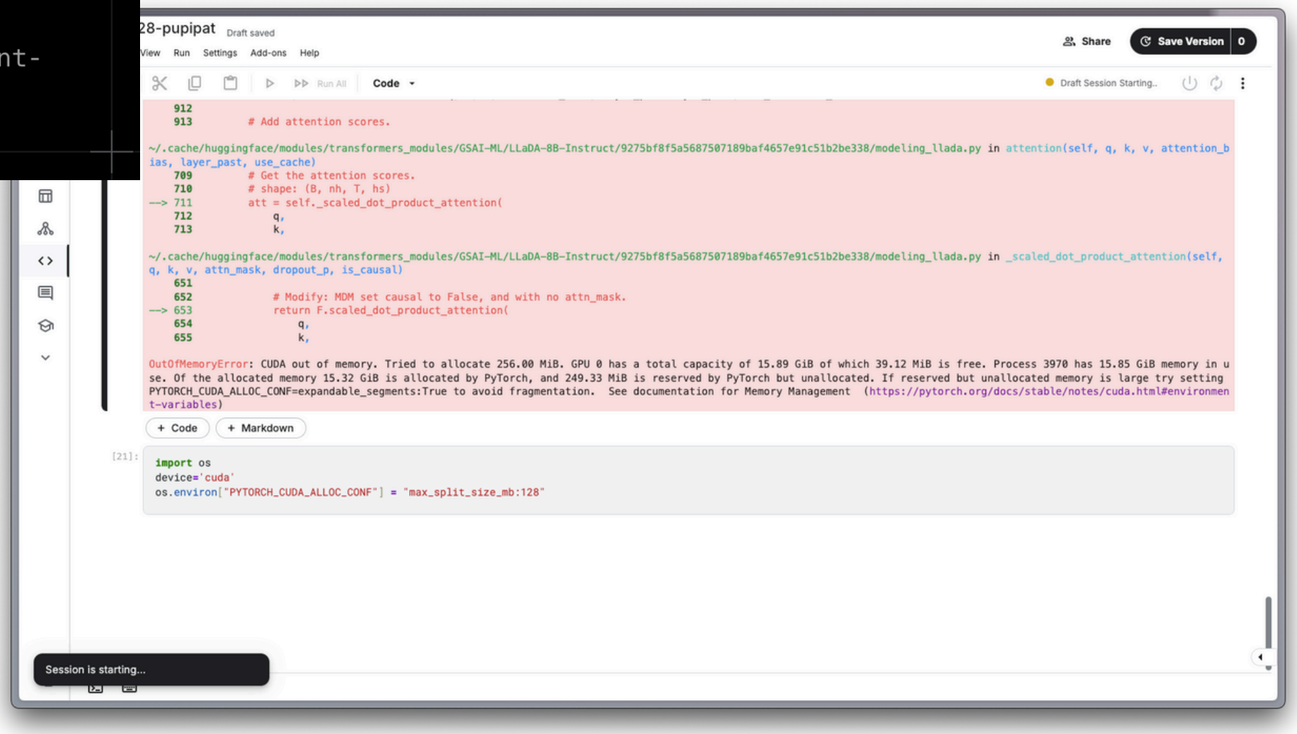

```
                                                  6/6 [00:00<00:00, 25.76it/s]
---------------------------
ack (most recent call last)
()
meters(), lr=learning_rate)
timizer, "iter_num": 0, "step_count": 0}
---> 50 state[ model ], state[ optimizer ] = fabric.setup(state["model"], state["optimizer"])
     51
     52 monitor = SpeedMonitor()

                                  10 frames
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py in convert(t)
   1327                         memory_format=convert_to_format,
   1328                     )
-> 1329                 return t.to(
   1330                     device,
   1331                     dtype if t.is_floating_point() or t.is_complex() else None,

OutOfMemoryError: CUDA out of memory. Tried to allocate 988.00 MiB. GPU 0 has a total capacity of 14.74 GiB of which 690.12 MiB is free. Process
49438 has 14.06 GiB memory in use. Of the allocated memory 13.97 GiB is allocated by PyTorch, and 1.49 MiB is reserved by PyTorch but unallocated. If
reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation.  See documentation for
Memory Management  (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

Kaggle: CUDA out of memory

```
OutOfMemoryError: CUDA out of memory. Tried to allocate 256.00
MiB. GPU 0 has a total capacity of 15.89 GiB of which 39.12 MiB
is free. Process 3970 has 15.85 GiB memory in use. Of the
allocated memory 15.32 GiB is allocated by PyTorch, and 249.33
MiB is reserved by PyTorch but unallocated. If reserved but
unallocated memory is large try setting
PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid
fragmentation.  See documentation for Memory Management
(https://pytorch.org/docs/stable/notes/cuda.html#environment-
variables)
```

- ~~T4x2~~
- ~~P100~~

~~kaggle~~



Modal: A100-40GB is not enough


```
File /usr/local/lib/python3.11/site-packages/torch/optim/adamw.py:171, in AdamW._init_group(sel
th_grad, grads, amsgrad, exp_avgs, exp_avg_sqs, max_exp_avg_sqs, state_steps)
    161 state["step"] = (
    162     torch.zeros(
    163         (),
   (...)    168     else torch.tensor(0.0, dtype=_get_scalar_dtype())
    169 )
    170 # Exponential moving average of gradient values
--> 171 state["exp_avg"] = torch.zeros_like(
    172     p, memory_format=torch.preserve_format
    173 )
    174 # Exponential moving average of squared gradient values
    175 state["exp_avg_sq"] = torch.zeros_like(
    176     p, memory_format=torch.preserve_format
    177 )

OutOfMemoryError: CUDA out of memory. Tried to allocate 96.00 MiB. GPU 0 has a total capacity of 39.49 GiB of whic
h 81.38 MiB is free. Process 68 has 39.41 GiB memory in use. Of the allocated memory 38.86 GiB is allocated by PyT
orch, and 50.43 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try settin
g PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation.  See documentation for Memory Managemen
t  (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

~~A100-40GB~~

Modal: ~~A100-40GB~~ is not enough, upgrading to A100-80GB

🎉 A100-80GB

```
Loading GSAI-ML/LLaDA-8B-Instruct model...
A new version of the following files was downloaded from https://huggingface.co/GSAI-ML/LLaDA-8B-Instruct:
- configuration_llada.py
. Make sure to double-check they do not contain any added malicious code. To avoid downloading new versions of the
code file, you can pin a revision.
A new version of the following files was downloaded from https://huggingface.co/GSAI-ML/LLaDA-8B-Instruct:
- modeling_llada.py
. Make sure to double-check they do not contain any added malicious code. To avoid downloading new versions of the
code file, you can pin a revision.
Fetching 6 files: 100%|██████████| 6/6 [02:08<00:00, 21.35s/it]
Loading checkpoint shards: 100%|██████████| 6/6 [00:00<00:00,  9.43it/s]
GSAI-ML/LLaDA-8B-Instruct model load successfully.
Epoch 1: 100%|██████████| 50/50 [00:38<00:00,  1.32it/s, loss=0]
```

✅

]:

Compute costs — Per hour | Per second

GPU Tasks	
Nvidia H100	$3.95 / h
Nvidia A100, 80 GB	$2.50 / h
Nvidia A100, 40 GB	$2.10 / h
Nvidia L40S	$1.95 / h
Nvidia A10G	$1.10 / h
Nvidia L4	$0.80 / h
Nvidia T4	$0.59 / h

CPU	
Physical core (2 vCPU equivalent)	$0.0473 / core / h *minimum of 0.125 cores per container

Memory	
	$0.0080 / GiB / h

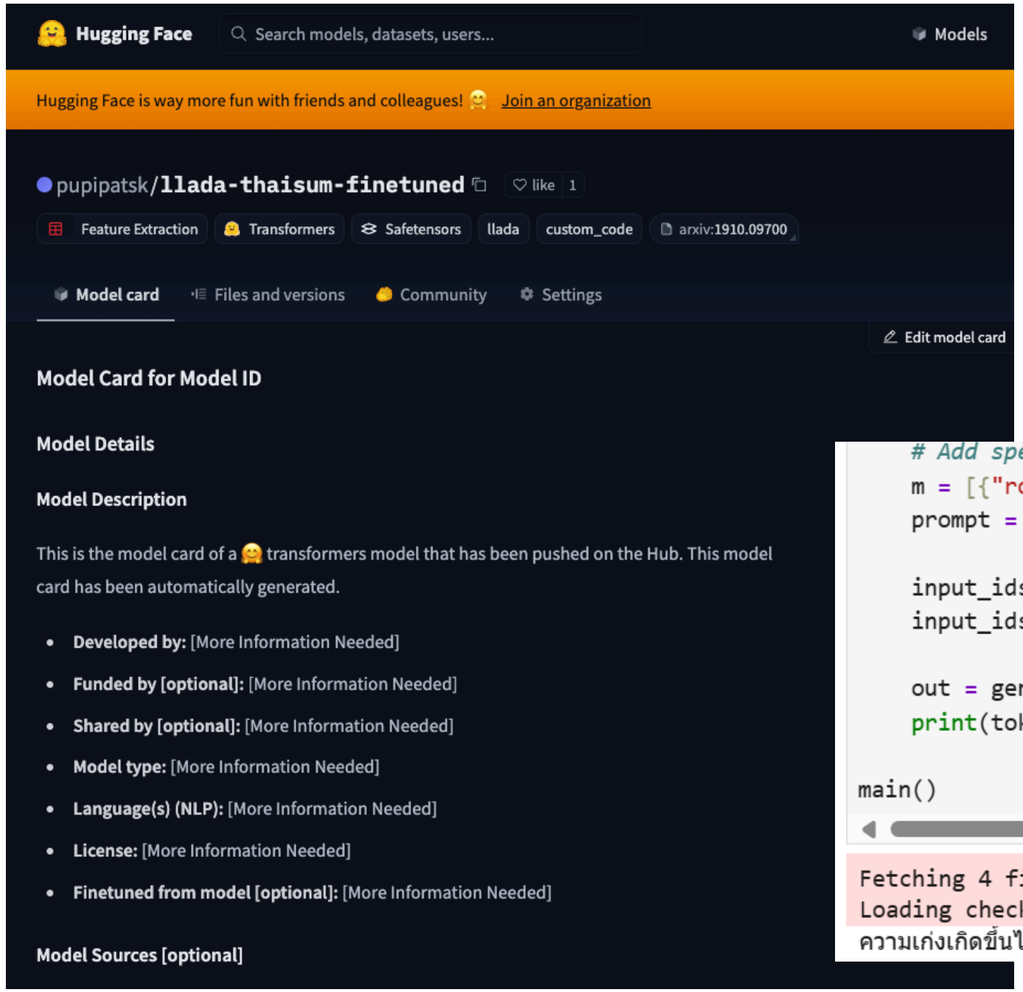
Test Inference run

Summarize following text in less than 3 sentences: ความเก่ง เกิดขึ้นได้หลายแบบไม่ว่าจะ ความหมั่นเพียร(ฝึกซ้อม), ประสบการณ์, สิ่งแวดล้อมเกื้อหนุน, มีต้นทุนบางอย่างดี เหมือนคนเกิดมาร่างกายสูงใหญ่มีโอกาสเก่งในกีฬาหลายประเภท นี่ก็ถือว่าต้นทุนดี แต่เหล่านี้เองจึงย้อนไปบั่นทอนคนที่คิดว่าตนไม่เก่ง เช่น เราขี้เกียจ-ไม่มีเวลาซ้อม, เราไม่เคยทำมาก่อน, ยังไม่พร้อม, ต้นทุนไม่ดีเหมือนเขา ส่วนหนึ่งก็ใช่ว่าผิด แต่แน่นอนไม่ถูก และกลายเป็นถ่วงอนาคตอย่างมาก

```
    # Add special tokens for the Instruct model. The Base model does not require the following two lines.
    m = [{"role": "user", "content": prompt}, ]
    prompt = tokenizer.apply_chat_template(m, add_generation_prompt=True, tokenize=False)

    input_ids = tokenizer(prompt)['input_ids']
    input_ids = torch.tensor(input_ids).to(device).unsqueeze(0)

    out = generate(model, input_ids, steps=128, gen_length=128, block_length=32, temperature=0., cfg_scale=0., rema
    print(tokenizer.batch_decode(out[:, input_ids.shape[1]:], skip_special_tokens=True)[0])

main()
```

```
ความเก่งเกิดขึ้นได้หลายแบบไม่ว่าจะ ความหมั่นเพียร(ฝึกซ้อม), ประสบการณ์, สิ่งแวดล้อมเกื้อหนุน, มีต้นทุนบางอย่างดี และกลายมาก
```

Upload to huggingface




```
    # Add special tokens for the Instruct model. The Base model does not require the following two lines
    m = [{"role": "user", "content": prompt}, ]
    prompt = tokenizer.apply_chat_template(m, add_generation_prompt=True, tokenize=False)

    input_ids = tokenizer(prompt)['input_ids']
    input_ids = torch.tensor(input_ids).to(device).unsqueeze(0)

    out = generate(model, input_ids, steps=128, gen_length=128, block_length=32, temperature=0., cfg_sca
    print(tokenizer.batch_decode(out[:, input_ids.shape[1]:], skip_special_tokens=True)[0])

main()
```

```
Fetching 4 files: 100%|██████████| 4/4 [03:17<00:00, 49.45s/it]
Loading checkpoint shards: 100%|██████████| 4/4 [00:00<00:00,  7.12it/s]
ความเก่งเกิดขึ้นได้หลายแบบไม่ว่าจะ ความหมั่นเพียร(ฝึกซ้อม), ประสบการณ์, สิ่งแวดล้อมเกื้อหนุน, มีต้นทุนบางอย่างดี และกลายมาก
```

Concern: context length

```python
# -- Tokenizer --
MODEL_NAME = "GSAI-ML/LLaDA-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, trust_remote_code=True)

# -- Formatting --
def format_llada_prompt(example):
    instruction = f"<start_id>user<end_id>\n{example['body']}<eot_id><start_id>assistant<end_id>\n{example['summary
    tokenized = tokenizer(
        instruction, padding="max_length", truncation=True, max_length=1024
    )
    prompt_end = instruction.find("<start_id>assistant<end_id>")
    prompt_tokens = tokenizer(instruction[:prompt_end])["input_ids"]
    return {"input_ids": tokenized["input_ids"], "prompt_length": len(prompt_tokens)}
```

Some data has > 20,000 characters!

เมื่อวันที่ 6 ม.ค.60 ที่ทำเนียบรัฐบาล นายวิษณุ เครืองาม รองนายกรัฐมนตรี กล่าวถึงกรณี ที่ นายสุรชัย เลี้ยงบุญเลิศชัย รองประธานสภานิติบัญญัติแห่งชาติ (สนช.) ออกมาระบุว่า การเลือกตั้งจะถูกเลื่อนออกไปถึงปี 2561 ว่า ขอให้ไปสอบถามกับ สนช. แต่เชื่อว่าคงไม่กล้าพูดอีก เพราะทำให้คนเข้าใจผิด ซึ่งที่ สนช.พูดเนื่องจากผูกกับกฎหมายของกรรมการร่างรัฐธรรมนูญ(กรธ.) ตนจึงไม่ขอวิพากษ์วิจารณ์ แต่รัฐบาลยืนยันว่ายังเดินตามโรดแม็ป ซึ่งโรดแม็ปมองได้สองแบบ คือ มียดังกล่าวล่าช้ากว่ากำหนด ส่งผลให้เกิดข้อสงสัยจนถึงทุกวันนี้ ส่วนกรณีที่ สนช. ระบุว่า มีกฎหมายเข้าสู่การพิจารณาของ สนช.เป็นจำนวนมาก ทำให้ส่งผลกระทบต่อโรดแม็ปนั้น .....

Q & A