Large Language Diffusion Models

태그	구현코드 작성중
논문	https://arxiv.org/abs/2502.09992
눈문 출판일	@2025년 2월 18일
짧은 소개	Diffusion 기법을 LLM에 도입한 모델

코드 구현

https://github.com/tigerak/LLaDA

소개

1. 기존 대형 언어 모델의 한계

현재 LLM은 대부분 생성 모델링(generative modeling) 프레임워크를 따른다.

가장 일반적인 접근 방식은 자기회귀 모델(Autoregressive Model, ARM)을 사용하는 것인데, 이는 다음 토큰을 예측하는 방식으로 학습함.

(eq. 2) ⇒ $P_\theta(x) = P_\theta(x^1) \prod_{i=2}^{L} P_\theta(x^i \mid x^1, ..., x^{i-1})$

이 방식은 매우 효과적임에도 불구하고 몇 가지 근본적인 한계가 존재함.

- 토큰을 하나씩 예측하는 구조이기 때문에 속도가 느림.
- 문장을 반대로 읽거나 재구성하는 작업에서 성능이 떨어짐.
- 길이가 긴 문장을 처리할 때 비용이 증가.

2. LLaDA (Large Language Diffusion with Masking)

따라서 자기회귀(AR) 방식 대신 Diffusion을 적용한 새로운 LLM 제안.

LLaDA는 Masking을 사용하여 입력 데이터를 변형하고, 이를 복원하는 방식으로 모델을 학습.

3. LLADA의 주요 특징

양방향 종속성(Bidirectional Dependency)을 활용하여 기존의 단방향 정보만 이용할 수 있던 학습 방법의 대안 제공.

한 번에 여러 개의 마스킹된 토큰을 예측할 수 있어 속도가 향상

특히 역방향 추론(Reverse Reasoning)과 같은 작업에서 GPT-4o보다 뛰어난 성능을 보였다.

▼ 역방향 추론

순방향 추론 : 사실로부터 결론을 추론.
역방향 추론 : 목표를 설정하고 증명하는 추론.

4. 실험 및 성능

8B 크기의 모델로 확장하여 실험.

- LLaDA는 10^{28}FLOPs까지 확장 가능하며, 이로써 ARM과 비슷한 성능을 낼 수 있었다.

  ▼ FLOPs(부동소수점 연산)

  모델이 수행하는 연산량을 의미하며, 모델 크기와 훈련 데이터 크기가 커질수록 필요 연산량도 증가.

  즉, LLaDA는 거대한 데이터와 연산 비용을 투입할수록 성능이 향상됨.
- In-context learning에서 LLaDA는 LLaMa3 8B와 동등한 성능을 보임.
- SFT 후 instruction-following 능력이 크게 향상 됨.
- 순방향 / 역방향 추론 모두 일관된 성능을 나타냄.

방법

1. 확률적 공식

1.1. LLM의 일반적 확률 정의

일반적으로 LLM은 특정 데이터 분포 $p_{data}(x)$를 근사하는 모델 분포 $p_\theta(x)$를 학습한다.

(eq. 1)⇒ $max_\theta \mathbb{E}_{Pdata(x)} log p_\theta(x)$

이는 KL Divergence를 최소화 하는 공식과 비슷하다.

(eq. 1)⇒ $min_\theta KL(p_{data}(x) \parallel p_\theta(x))$

1.2. 자기 회귀 모델(ARM)과 비교

ARM은 아래와 같이 이전 토큰이 주어졌을 때, 다음 토큰을 예측하는 방식.

(eq. 2) ⇒ $P_\theta(x) = P_\theta(x^1) \prod_{i=2}^{L} P_\theta(x^i \mid x^1, ..., x^{i-1})$

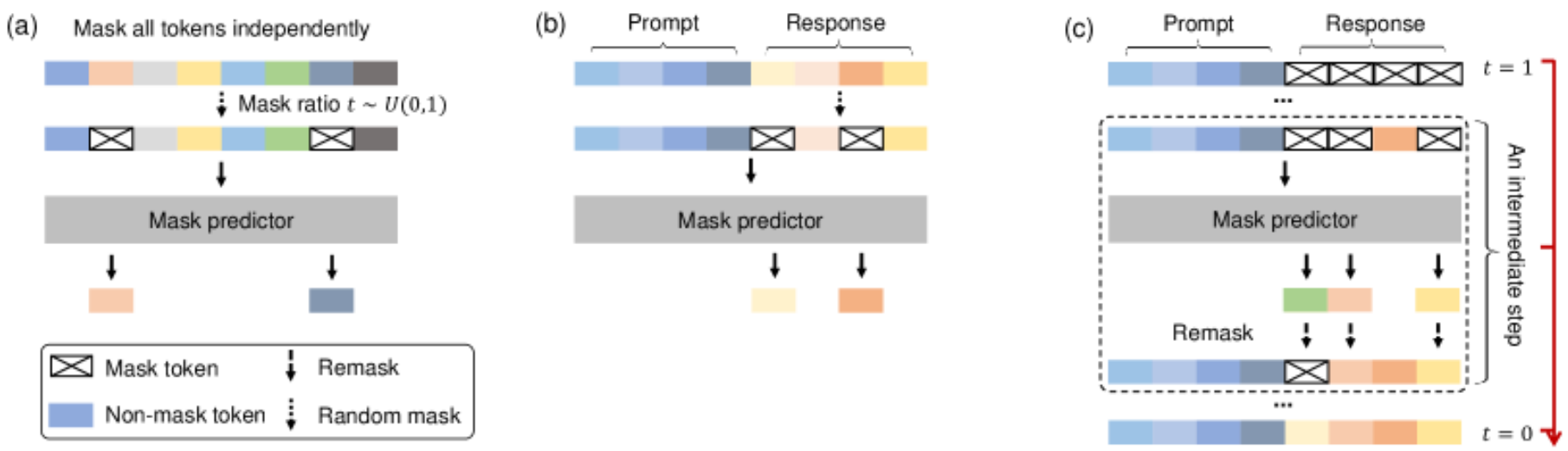
→ L은 문장 길이, x^i는 i번 째 토큰, p_θ는 모델의 확률분포

문제점

- 병렬 연산이 어렵고 연산 속도가 느림.
- 문장을 거꾸로 읽거나 중간에서 예측하는 능력이 제한적.
- 데이터 압축 관점에서 보면 ARM은 완전한 정보 손실 없는 데이터 압축을 수행하지만, 모든 가능한 문장을 탐색할 수 없음.

1.3. LLaDA의 Diffusion 기반 확률 정의

(eq. 2)와 달리 LLaDA는 정방향 프로세스와 역방향 프로세스를 통해 $p_\theta(x_0)$로 모델 분포를 정의한다.



1.3.1. 정방향 프로세스

$t \in (0, 1)$일 때, t확률만큼 토큰을 마스킹한다. (Training은 1-step masking learning만으로 충분하다)

```
def mask_tokens_with_random_t(input_ids, prompt_len, tokenizer, max_t=1.0, min_t=0.01):
    """
    - `t`를 [min_t, max_t] 범위에서 랜덤하게 샘플링하여 각 샘플마다 다른 마스킹 비율 적용
    """
    seq_len = input_ids.shape[0]

    # Monte Carlo 방식으로 `t`를 샘플링
    t = torch.rand(1).item() * (max_t - min_t) + min_t  # 0.01~1.0 사이 랜덤한 float 값

    # 프롬프트 & 특수 토큰 제외한 마스킹 가능한 토큰 찾기
    valid_tokens = torch.ones(seq_len, dtype=torch.bool)  # 모든 위치를 True로 초기화
    valid_tokens[:prompt_len] = False  # 프롬프트 제외
    # valid_tokens[input_ids == tokenizer.pad_token_id] = False  # 패딩 제외
    valid_tokens[input_ids == tokenizer.bos_token_id] = False  # BOS 제외
    # valid_tokens[input_ids == tokenizer.eos_token_id] = False  # EOS 제외

    # `t` 확률로 마스킹 여부 결정
    valid_indices = valid_tokens.nonzero(as_tuple=True)[0]  # 마스킹 가능한 인덱스
    num_to_mask = max(1, round(t * valid_indices.shape[0]))  # 최소 1개 이상 마스킹

    # num_to_mask 개수만큼 선택하여 마스킹
```

```
    mask_indices = valid_indices[torch.randperm(valid_indices.size(0))[:num_to_mask]]

    # 마스킹 적용
    masked_input_ids = input_ids.clone()
    masked_input_ids[mask_indices] = tokenizer.mask_token_id  # [MASK] 적용

    return masked_input_ids, mask_indices, t
```

1.3.2. 마스크 예측기

- 손실 함수

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

  x_0 : 원본 문장

  x_t : t step에서 마스킹 된 문장 (training은 1-step) (t확률만큼 마스킹)

  $log p_\theta(x_0^i \mid x_t)$: 모델이 x_t에서 x_0를 복원할 확률. (이 논문에서는 Cross-Entropy Loss 사용)

  $1[x_t^i = M]$: 마스킹 된 토큰들에 대해서만 손실을 계산

  $\frac{1}{t}$: 샘플마다 다른 t로 인한 스케일 조정

```
def calculate_loss(self, logits, labels, t):
    """
      - (B, L, V) 로짓 & (B, L) 레이블
      - CrossEntropy(reduction='none')로 모든 토큰 위치별 손실값 계산
      - 마스킹되지 않은 위치(-100)는 무시
      - 각 샘플별 마스킹 토큰 평균 후 1/t 곱
      - 배치 평균
    """
    # 손실 계산을 위한 CrossEntropyLoss
    loss_fn = torch.nn.CrossEntropyLoss(ignore_index=-100, reduction="none")

    B, S, vocab_size = logits.shape
    # logits: (B,S,vocab_size) → (B*S, vocab_size)
    # labels: (B,S) → (B*S,)
    ce_1d = loss_fn(logits.view(-1, vocab_size), labels.view(-1))  # (B*S)

    # 2D로 reshape (B,S)
    ce_2d = ce_1d.view(B, S)  # (B, S)

    # 유효 토큰 마스크: ignore_index=-100이 아닌 위치만 1.0
    mask = (labels != -100).float()      # (B, S)
    # 샘플별 "유효 토큰" 갯수
    valid_counts = mask.sum(dim=1)       # (B,)

    # 샘플별 "유효 토큰"들의 CE 총합
    # (mask로 무효 토큰 위치는 0이 됨)
    sum_ce = (ce_2d * mask).sum(dim=1)   # (B,)

    # 샘플별 평균 CE = 총합 / 갯수 (갯수가 0인 경우를 대비해 clamp)
    mean_ce_per_sample = sum_ce / valid_counts.clamp(min=1)  # (B,)

    # 각 샘플별 1/t 곱
    scaled_loss_per_sample = mean_ce_per_sample * (1.0 / t)  # (B,)

    # 배치 평균
    final_loss = scaled_loss_per_sample.mean()
```

```
    return final_loss
```

2. 사전 학습

- 데이터

  일반 텍스트 뭉치 외에 고품질 코드, 수학 및 다국어 데이터가 포함.

  총 2조 3천억 토큰.

  Label의 Masking 되지 않은 부분은 모두 -100으로 처리하여 loss 계산에서 배제 시킨다.

- 학습

  4096 토큰의 고정 시퀀스 길이를 사용.

  Optimizer : AdamW

  사다리꼴 스케줄러 : learning rate를 초반에 점진적으로 증가 시키다가 중반에 멈추고 후반에 다시 줄이는 방법.

  Weight Decay : 0.1

  Batch ize : 1280

- 모델

  Mask 예측기를 만들기 위해 Transformer 적용 (논문 2.2)

  Vanilla Multi Head Attention 적용

  Causal Mask는 적용하지 않는다.

```
class LLaDA_TransformerBlock(nn.Module):
    """
    RoPE 적용
    SwiGLU 적용
    """
    def __init__(self, config):
        super(LLaDA_TransformerBlock, self).__init__()
        self.hidden_dim = config.hidden_dim
        self.num_heads = config.num_attention_heads
        self.head_dim = config.head_dim
        self.dropout = config.dropout

        self.rn1 = nn.RMSNorm(config.hidden_dim, eps=1e-5)

        self.q_peoj = nn.Linear(self.hidden_dim, self.hidden_dim, bias=False)
        self.k_peoj = nn.Linear(self.hidden_dim, self.hidden_dim, bias=False)
        self.v_peoj = nn.Linear(self.hidden_dim, self.hidden_dim, bias=False)
        self.o_peoj = nn.Linear(self.hidden_dim, self.hidden_dim, bias=False)

        self.rn2 = nn.RMSNorm(config.hidden_dim, eps=1e-5)

        self.ffn = nn.Sequential(
                    nn.Linear(config.hidden_dim, config.intermediate_size*2, bias=False),
                    SwiGLU(config.intermediate_size*2, config.intermediate_size),
                    nn.Linear(config.intermediate_size, config.hidden_dim, bias=False)
                    )

def forward(self, hidden_states, attention_mask, rope_cache=None, offset=0):
    B, S, d = hidden_states.size()

    attn_in = self.rn1(hidden_states)

    q = self.q_peoj(attn_in)
```

```
        k = self.k_peoj(attn_in)
        v = self.v_peoj(attn_in) # (B, S, d)

        q = q.view(B, S, self.num_heads, self.head_dim)
        k = k.view(B, S, self.num_heads, self.head_dim)
        v = v.view(B, S, self.num_heads, self.head_dim)

        if rope_cache is not None:
            q, k = apply_rotary_pos_emb(q=q,
                                        k=k,
                                        cos_sin=rope_cache,
                                        offset=offset)
        q = q.permute(0, 2, 1, 3)
        k = k.permute(0, 2, 1, 3)
        v = v.permute(0, 2, 1, 3) # (B, n_head, S, head_dim)

        attn_scores = torch.matmul(q, k.transpose(-2, -1)) / math.sqrt(self.head_dim)

        if attention_mask is not None: # 이 논문에서는 사용 안 함
            if attention_mask.dim() == 2: # (B, S) → (B, 1, 1, S)
                attention_mask = attention_mask.unsqueeze(1).unsqueeze(2)
            attn_scores = attn_scores + attention_mask

        attn_props = F.softmax(attn_scores, dim=-1)
        attn_props = F.dropout(input=attn_props,
                               p=self.dropout,
                               training=self.training)
        attn_out = torch.matmul(attn_props, v) # (B, n_head, S, head_dim)

        attn_out = attn_out.permute(0, 2, 1, 3).contiguous() # (B, S, n_head, head_dim)
        attn_out = attn_out.view(B, S, self.hidden_dim) # (B, S, d)

        attn_out = self.o_peoj(attn_out)

        residual1 = hidden_states + attn_out

        ffn_in = self.rn2(residual1)
        ffn_out = self.ffn(ffn_in)

        residual2 = residual1 + ffn_out


        return residual2
```

- 활성화 함수 SwiGLU 적용 (부록 B.2)

```
class SwiGLU(nn.Module):
    def __init__(self, in_dim, out_dim):
        super().__init__()
        self.fc1(in_dim, out_dim)
        self.fc2(in_dim, out_dim)

    def forward(self, x):
        return F.silu(self.fc1(x)) * self.fc2(x)
```

- 위치 임베딩 RoPE 적용 (부록 B.2)

```
def rotate_half(x):
    """
```

```
    x: (batch, seq_len, n_heads, head_dim)
      head_dim이 짝수일 것것.
    """
    x1, x2 = x.chunk(2, dim=-1)
    return torch.cat([-x2, x1], dim=-1)

def apply_rotary_pos_emb(q, k, cos_sin, offset=0):
    """
    q, k: (B, seq_len, n_heads, head_dim)
    cos_sin: (seq_len, head_dim) 형태의 cos/sin 사전 계산 텐서
    offset: 캐시 사용시 현재 시퀀스 시작 위치 등 (간단 예시는 0)
    """
    seq_len = q.shape[1]
    # (seq_len, head_dim) → (1, seq_len, 1, head_dim)
    cos, sin = cos_sin
    cos = cos[offset:offset+seq_len, :].unsqueeze(0).unsqueeze(2)  # (1, seq_len, 1, head_dim)
    sin = sin[offset:offset+seq_len, :].unsqueeze(0).unsqueeze(2)

    q_ = (q * cos) + (rotate_half(q) * sin)
    k_ = (k * cos) + (rotate_half(k) * sin)
    return q_, k_

def build_rope_cache(seq_len, head_dim, base=10000.0):
    """
    seq_len 길이만큼 사용할 cos/sin 테이블을 미리 만들어놓기 위한 함수.
    base=10000 등은 표준 transformer의 PE를 따름.
    """
    # head_dim의 절반에 해당하는 차원까지 각기 다른 주파수를 구성
    # 예: rope_dim = head_dim // 2
    freqs = 1.0 / (base ** (torch.arange(0, head_dim, 2).float() / head_dim))
    # pos는 [0..seq_len-1]
    positions = torch.arange(0, seq_len, dtype=torch.float)
    # 외적을 구해 각 위치(pos)에 대해 cos(pos * freq), sin(pos * freq) 계산
    angles = torch.einsum("i,j→ij", positions, freqs)  # (seq_len, rope_dim)
    cos = torch.cos(angles)
    sin = torch.sin(angles)
    # cos, sin을 head_dim 크기에 맞춰 expand
    # 중간에 2배 차원이지만, cos/sin을 (cos[even], cos[odd])와 같은 식으로 재배치
    # 여기서는 간단히 interleave 없이 half-chunk 기준으로 사용
    cos_ = torch.zeros(seq_len, head_dim)
    sin_ = torch.zeros(seq_len, head_dim)
    cos_[:, 0::2] = cos
    sin_[:, 0::2] = sin
    return cos_, sin_
```

3. SFT

- 데이터

  Prompt와 Response로 쌍을 이룬 데이터 (p_0, r_0) 450만 개.

  $-\mathbb{E}_{t,p_0,r_0,r_t}\left[\frac{1}{t}\sum_{i=1}^{L}1[r_t^i = M]logp_\theta(r_0^i \mid p_0, r_t)\right]$ 에서 볼 수 있듯

  Prompt는 마스킹하지 않을 채로, Respons는 $t \in (0,1)$일 때, t확률로 마스킹 되어 입력된다.

  Label의 Masking 되지 않은 부분은 모두 -100으로 처리하여 loss 계산에서 배제 시킨다.

  모든 데이터에서 동일한 시퀀스 길이를 보장하기 위해 |eos| 토큰을 padding 토큰으로 사용하며, 모든 |eos| 토큰은 예측 대상이 된다. (부록 B.1)

- 학습

Weight Decay : 0.1

Batch ize : 256

4. 추론

4.1. Prompt 입력

고정된 시퀀스 길이에서, Prompt 외에는 모두 |mask| 토큰으로 채운다.

위 방법으로 완전히 마스킹 된 상태에서 시작한다.

```
def llada_inference(cfg, model, prompt_ids, num_step=128):
    model.eval()

    prompt_len = prompt_ids.size(1)
    total_len = cfg.max_seq_len
    response_len = total_len - prompt_len

    prompt_ids = prompt_ids.unsqueeze(0).to(device)

    mask_token_id = cfg.mask_token_id

    init_input_ids = torch.full(
        (1, total_len),
        fill_value=mask_token_id,
        dtype=torch.long,
        device=device
    )
    init_input_ids[0, :prompt_len] = prompt_ids[0]
```

4.2. 역방향 프로세스

- 역방향 프로세스 이산화

  몬테 카를로 방법에 따라 샘플링 하기 위해, 샘플링 할 회수를 Hyperparameter(num_step = 128)로 정했다.

  $t \in (0, 1], s \in [0, t)$일 때, $\frac{s}{t}$만큼의 예측된 토큰을 재 마스킹 해주어야한다.

  이 구현 코드에서는, 0-1 사이의 값을 num_step 개로 균일하게 나눠서 각 step마다 t가 선형적으로 줄어들도록 코딩했다.

  ```
  # 스텝 균등 분할
     steps_t = torch.linspace(start=1.0, end=0.0, steps=num_step + 1).tolist()
  ```

- 예측

  input의 모든 mask token에 대해 한번에 추론한다.

```
### Denoising ###
current_ids = init_input_ids.clone()

for step_idx in range(num_step):
    with torch.no_grad():
        torch.cuda.empty_cache()
        logits, _ = model(input_ids=current_ids,
                          attention_mask=None,
                          labels=None,
                          t=None)

    # 마스킹 된 위치에 대한 예측
    masked_positions = (current_ids == mask_token_id)
    if masked_positions.any():
        masked_logits = logits[masked_positions]
```

```
            predicted_ids = torch.argmax(masked_logits, dim=-1)
            current_ids[masked_positions] = predicted_ids
```

- 재 마스킹

  원칙적으로 재 마스킹은 무작위여야한다.

  하지만 저신뢰도 재마스킹(low-confidence remasking) 방법이 효과적이기 때문에 이를 따른다.

  https://arxiv.org/abs/2005.14165

  예측기가 예측한 logit에 softmax를 적용한 확률 순서에 따라 낮은 것부터 재 마스킹 한다.

  마지막 step에서는 더이상 masking 하지 않도록하고 최종 결과를 출력한다.

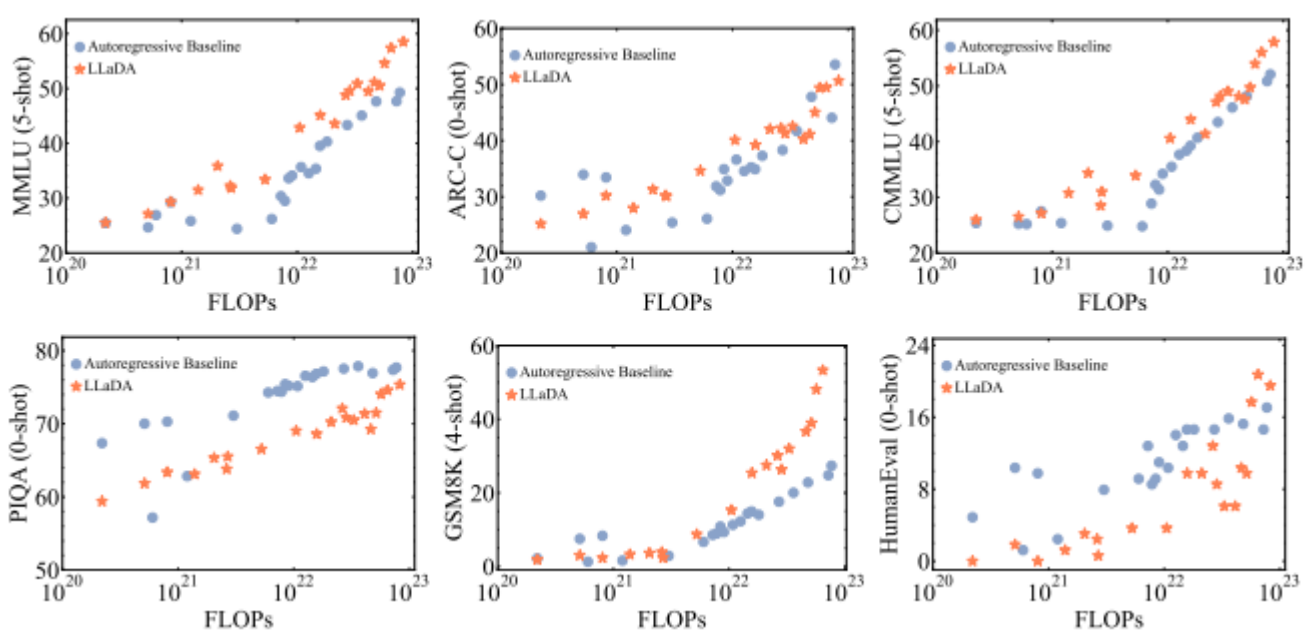
```
        # Remasking
        remask_ratio = steps_t[step_idx + 1]
        if step_idx + 1 < num_step: # 마지막 스텝 제외
            with torch.no_grad():
                probs = F.softmax(masked_logits, dim=-1)
                gather_conf = probs[range(len(predicted_ids)), predicted_ids]
                sorted_conf, sorted_idx = torch.sort(gather_conf) # 오름차순 정렬

                # remaskin 비율
                n_low = int(remask_ratio * response_len)
                low_conf_idx = sorted_idx[:n_low]

                masked_pos_flat = masked_positions.view(-1).nonzero(as_tuple=True)[0]

                global_remask_ids = masked_pos_flat[low_conf_idx ]

                current_ids.view(-1)[global_remask_ids] = mask_token_id

        else : # 마지막 step
            return current_ids.squeeze(0).tolist()
```

실험

1. 언어 과제에 대한 LLaDA의 확장성

표준 벤치 마크에서 LLaDA의 확장 모델에 대한 성능을 확인하였다.

LLaDA 모델을 1B에서 8B까지 크기를 키웠을 때 성능이 증가하는 지에 대한 실험.



모델이 커질 수록 성능이 증가함을 알 수 있다.

MMLU나 GSM8K의 경우에는 ARM보다 더 나은 성능을 보이며, PIQA에서도 성능 격차를 좁힘을 볼 수 있다.

	LLaDA 8B*	LLaMA3 8B*	LLaMA2 7B*
Model	Diffusion	AR	AR
Training tokens	2.3T	15T	2T
General Tasks			
MMLU	**65.9** (5)	65.4 (5)	45.9 (5)
BBH	49.8 (3)	**57.6** (3)	37.3 (3)
ARC-C	47.9 (0)	**53.1** (0)	46.3 (0)
Hellaswag	72.5 (0)	**79.1** (0)	76.0 (0)
TruthfulQA	**46.4** (0)	44.0 (0)	39.0 (0)
WinoGrande	74.8 (5)	**77.3** (5)	72.5 (5)
PIQA	74.4 (0)	**80.6** (0)	79.1 (0)
Mathematics & Science			
GSM8K	**70.7** (4)	53.1 (4)	14.3 (4)
Math	**27.3** (4)	15.1 (4)	3.2 (4)
GPQA	**26.1** (5)	25.9 (5)	25.7 (5)
Code			
HumanEval	33.5 (0)	**34.2** (0)	12.8 (0)
HumanEval-FIM	**73.8** (2)	73.3 (2)	26.9 (2)
MBPP	38.2 (4)	**47.4** (4)	18.4 (4)

LLaDA는 2.3T 개의 토큰만 학습했음에도 LLaMA 3와도 경쟁할만한 성능을 보여주고 있다.

2. 벤치 마크 결과

LLaDA는 강화 학습 없이 SFT만 하였음에도 LLaMA 3와 비교할만한 성능을 보인다.

	LLaDA 8B*	LLaMA3 8B*	LLaMA2 7B*
Model	Diffusion	AR	AR
Training tokens	2.3T	15T	2T
Post-training	SFT	SFT+RL	SFT+RL
Alignment pairs	4.5M	-	-
General Ta			
MMLU	65.5 (5)	**68.4** (5)	44.1 (5)
MMLU-pro	37.0 (0)	**41.9** (0)	4.6 (0)
Hellaswag	74.6 (0)	**75.5** (0)	51.5 (0)
ARC-C	**88.5** (0)	82.4 (0)	57.3 (0)
Mathematics &			
GSM8K	**78.6** (4)	78.3 (4)	29.0 (4)
Math	26.6 (0)	**29.6** (0)	3.8 (0)
GPQA	31.8 (5)	**31.9** (5)	28.4 (5)
Code			
HumanEval	47.6 (0)	**59.8** (0)	16.5 (0)
MBPP	34.2 (4)	**57.6** (4)	20.6 (4)

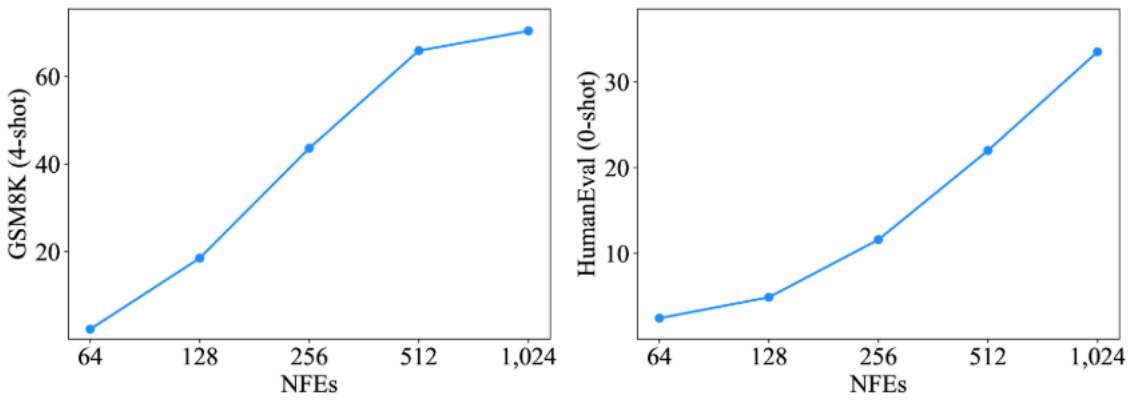
3. 역방향 추론 및 분석

	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**

LLaDA는 “역전의 저주” 문제를 효과적으로 해결한다.

실험은 496개의 유명한 중국 시를 이용하여, 시의 한 문장이 주어지면 다음 줄 또는 이전 줄을 생성하는 작업을 수행하여 진행하였다.

GPT-4o에 비해서도 역방향 추론 능력이 높음을 볼 수 있다.



추론 문장의 길이가 1024인 문장을 이용하여, 몬테 카를로 방식의 샘플링을 적용하는 횟수를 늘릴 수록 성능이 어떻게 변하는 지를 실험한 결과이다.

횟수가 늘어날 수록 성능이 올라감을 볼 수 있다.

4. 사례 연구

LLaDA의 응답에서 어두운 색상은 샘플링의 후반 단계에서 예측된 토큰을 나타내고 밝은 색상일 수록 이른 시기의 예측이다.

Sample Process는 Instruction-falowing능력을 보여주고, Nulti-round Dialogue Case는 ARM에서 벗어난 최초의 채팅 기능임이 인상적이다.

Sampling Process	
User	Lily can run 12 kilometers per hour for 4 hours. After that, she runs 6 kilometers per hour. How many kilometers can she run in 8 hours?
LLaDA	In 4 hours , Lily runs 12 * 4 = 48 kilometers . After 4 hours she runs 6 * 4 = 24 kilometers . In total she runs 48 + 24 = 72 kilometers in 8 hours . Result 72
Multi-round Dialogue Case	
User	Could you please share with me what the initial two lines of the renowned poem 'The Road Not Taken'?
LLaDA	Certainly! The initial two lines of the renowned poem "The Road Not Taken" by Robert Frost are: "Two roads diverged in a yellow wood, And sorry I could not travel both." These lines set the stage for the poem, introducing the idea of a speaker facing a choice between two paths.
User	Please help me translate into Chinese.
LLaDA	Sure! The first two lines of "The Road Not Taken" by Robert Frost can be translated into Chinese as: "两条路分岔在黄色的树林中，遗憾我不能同时走"
User	Please translate into German.
LLaDA	Sure! The first two lines of "The Road Not Taken" by Robert Frost can be translated into German as: "Zwei Wege trennten sich im gelben Wald, und leider konnte ich nicht beide tun."
User	Please also write a poem about life choices. I require 5 sentences, and the first word of each sentence starts with C as the first letter.
LLaDA	Certainly! Here's a poem about life choices, with each sentence starting with the letter "C": Choices cascade through life's journey, Creating connections with each decision, Careful consideration leads to better paths, Courageous choices shape our destiny, Choosing wisely guides us through the years.

결론

LLaDA는 현재 Hyperparameter에 민감하며, 아직 최적화와 강화 학습이 충분히 적용되지 아니하였음에도 좋은 성능을 보이고 있다.

모델이 더 커질 수 있음을 생각해본다면 앞으로의 발전 가능성은 충분하다.

- Diffusion 을 활용한 LLM 상업 모델 Mercury.

Inception Labs

We are leveraging diffusion technology to develop a new generation of LLMs. Our dLLMs are much faster and more efficient than traditional auto-regressive LLMs. And diffusion models are more accurate, controllable, and performant on multimodal tasks.

https://www.inceptionlabs.ai/news