Prelim - SpecDec

[Fast Inference from Transformers via Speculative Decoding]

- specdec发明的motivation：大模型在decoding阶段经常是memory bound的，计算资源是有富余，提高计算强度基本不会带来额外开销。因此可以想一种办法来提高token生成过程中的并行度——specdec中的大模型verification过程就相当于并行验证多个tokens，这就有效提升了大模型每次forward时的计算强度。

  本质上来说：小模型多次解码是在为大模型组织一个大的batch，使得更多tokens能够搭乘大模型加载1次weight计算的顺风车

- 流程：
  - 给定prefix，先用小模型 M_q 自回归采样 γ 个draft tokens： $x_1, \cdots, x_\gamma$ 。每个token是由 M_q 采样得到： $x_i \sim q_i(x) = M_q(prefix + [x_1, \cdots, x_{i-1}])$ ，该过程中同时也会保留各个分布： $q_1(x), \cdots, q_\gamma(x)$
  - 然后，将 $prefix + [x_1, \cdots, x_\gamma]$ 输入大模型 M_p ，推理一次得到分布 $p_1(x), \cdots, p_\gamma(x), p_{\gamma+1}(x)$ （注意在最后一个token x_γ 后还额外推出一个 $x_{\gamma+1}$ ）
  - 然后，从前往后逐次验证每个新token，以 $\frac{p_i(x)}{q_i(x)}$ 的概率接受小模型给出的 x_i ，直到某个token被拒绝或 $x_1, \cdots, x_\gamma$ 都被接受，设此时总共接受了n个tokens，也即 $x_1, \cdots, x_n$ 被接受
    - 如果所有token都被接受，那么就按照刚才大模型推理时得到的 $p_{\gamma+1}(x)$ 采样最后一个token t （或者说 $t = x_{\gamma+1}$ ），然后返回 $prefix + [x_1, \cdots, x_\gamma, t]$
    - 如果只有 $x_1, \cdots, x_n$ 被接受且 $n < \gamma$ ，则构造一个新分布： $p'(x) = norm(\max(0, p_{n+1}(x) - q_{n+1}(x)))$ ，然后从中采样最后一个token t （或者说 x_{n+1} ），然后返回 $prefix + [x_1, \cdots, x_n, t]$

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i-1 \mid 1 \le i \le \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

注：若使用greedy sampling，则直接对llm输出的logits统计argmax，然后和draft tokens直接做相等判断，相等的话就接受

收益分析：

1、平均每轮speddec能够产生多少个tokens（平均接收长度）

设接收率 $\beta_{<x_t}$ 为接收 $x_t \sim q(x_t|x_{<t})$ 的概率，则其均值 $\alpha = E(\beta)$ 可以体现（第t个token处） M_q 对于 M_p 的拟合程度。假设 β 是iid的，也即每一步接收token的概率都是 α，则运行一轮SpecDec能够产生的有效token数量服从上限为 $\gamma + 1$ 且参数（success prob）为 $1 - \alpha$ 的几何分布（也即从前往后每个token都有 $1 - \alpha$ 的概率被拒绝从而停止该轮specdec）。则一轮specdec能够产生的有效token数量的期望为：

$$1 + \alpha + \alpha^2 + \cdots + \alpha^\gamma = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha}$$

其中第一项的1表示主模型多生成的那个token， α 表示第1个draft token被接收， α^2 表示第2个draft token被接收.....

$$E(\#\ generated\ tokens) = \frac{1-\alpha^{\gamma+1}}{1-\alpha}$$

下面推导 α

首先，定义一个natural divergence D_{LK}，其对于分别p,q是对称的，用于衡量p或者q分布和二者均值之间的距离（也即反映p和q之间的距离）：

Definition 3.2. $D_{LK}(p,q) = \sum_x |p(x) - M(x)| = \sum_x |q(x) - M(x)|$ where $M(x) = \frac{p(x)+q(x)}{2}$.

然后可以用min(p,q)表示 D_{LK}：

Lemma 3.3. $D_{LK}(p,q) = 1 - \sum_x \min(p(x), q(x))$

Proof. $D_{LK}(p,q) = \sum_x |p(x) - M(x)| = \sum_x \frac{|p-q|}{2} = 1 - \sum_x \frac{p+q-|p-q|}{2} = 1 - \sum_x \min(p(x), q(x))$ □

再结合 β 和min(p,q)的关系，可得 β, α 和 D_{LK} 的关系：

Theorem 3.5. $\beta = 1 - D_{LK}(p,q)$

Proof. $\beta = E_{x\sim q(x)} \begin{cases} 1 & q(x) \le p(x) \\ \frac{p(x)}{q(x)} & q(x) > p(x) \end{cases} = E_{x\sim q(x)} \min(1, \frac{p(x)}{q(x)}) = \sum_x \min(p(x), q(x))$ □

Finally we get:

Corollary 3.6. $\alpha = 1 - E(D_{LK}(p,q)) = E(\min(p,q))$

其中：

$$
\begin{aligned}
\beta &= E_{x\sim q(x)} \min(1, \frac{p(x)}{q(x)}) \\
&= \sum_x q(x) \min(1, \frac{p(x)}{q(x)}) \\
&= \sum_x \min(q(x), p(x)) \\
&= 1 - D_{LK}(p, q)
\end{aligned}
$$

$$
\alpha = E(\beta) = E(\sum_x \min(q(x), p(x))) = E(\min(p, q))
$$

2、平均产生一个新token的延时减小倍数

假设算力充足，也即大模型运行一次forward的时间总是一样的（也即：大模型decode出1个新token的延时和验证 γ 个tokens得到 $\gamma + 1$ 个分布的延时是一样的，不会受到compute bound的影响）

设cost coefficient（小模型运行一次forward和大模型运行一次forward的延时比例）为 c，其通常~0.05甚至接近0。则运行一次specdec可以使得平均生成一个token的延时减少的倍数为：

$$
\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}
$$

如果平均接收率大于推理开销系数： $\alpha > c$，则存在 γ 可以带来收益，且延时收益系数至少是 $\frac{1+\alpha}{1+c}$

3、平均产生一个新token的计算量增加倍数

设小模型产生一个新token所需计算量是大模型产生一个token计算量的 $\hat{c}$ 倍（这里假设序列足够长，kv cache量不变），每轮specdec会使得小模型付出推理 γ 个tokens的计算量+大模型付出推理 $\gamma + 1$ 个tokens的计算量，则一轮specdec使得平均产生每个新token的计算量增大倍数为：

$$
\frac{(1-\alpha)(\gamma\hat{c}+\gamma+1)}{1-\alpha^{\gamma+1}}
$$

4、平均产生一个新token的访存数减小倍数就是 $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$，也即大模型原先每decode出一个token就要load一遍weight+kv cache，现在产生这么多tokens才需要load一遍weight+kv cache

[MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding]

Motivation：

在batch size较大的情况下，模型变成compute bound，因此不适合使用specdec（过大的batch size会导致大模型的verification step代价很大，因为它是compute hungry的。如果小模型align不好的话就会需要更多verification，使得情况更恶劣）（bsz增大会增大linear layer的计算强度，最终变成compute bound）。这就导致specdec虽然有利于减小per token latency，但由于不宜使用大batch导致throughput受损。文章希望能够利用specdec同时优化latency和throughput。

文章通过观察发现：

- 在长文本和大bsz的情境下，KV Cache是主要的bottleneck：KV Cache的开销随bsz增长，其内存开销会大于权重。bsz增大确实会使得计算量增大，但由于GPU的峰值计算访存比很大，所以bsz增大导致KV加载延时变长的影响是远大于前者（计算量增长）的
- 当序列长度超过一定程度时，specdec会给throught带来增益：之前的文章认为sd在大bsz下由于过高的verification costs导致低效，但他们的结论都是基于短序列来的，在短序列情景下增大bsz会使得计算量变成主要的瓶颈，导致verification过程负担过大。然而，当序列长度超过一定程度后，即使对于大bsz来说，KV loading也会变成主要瓶颈，此时sd又变得有效，因为verification带来的开销相比之下又没那么显著了，其可以被均摊到要verify的tokens上
- 压缩后的kv cache可以让speculation更高效：研究发现，相比于压缩权重，压缩draft model的kv cache能够获得更高的accept rate

由此提出：通过压缩kv cache，sd可以为大bsz提供加速

关于LLM inference performance和sd的理论分析

设大模型（target）和小模型（draft）对bsz=B, seq_len=S的序列进行一次decode（也即普通decoding下生成一个新token）的延时为 $T_T(B,S), T_D(B,S)$ ，大模型验证 γ 个tokens的延时为 $T_V(B,S,\gamma)$ 。设每个draft token的接受率为 α ，则每轮verification预计获得的有效token数量 $\Omega(\gamma,\alpha)$ 为：

$$\Omega(\gamma,\alpha) := \mathbb{E}[\#generated tokens] = \frac{1-\alpha^{\gamma+1}}{1-\alpha}$$

一轮specdec的总时间（包括小模型生成 γ 个draft tokens + 大模型验证它们） T^{SD}_{Total} 为：

$$T^{SD}_{Total} = \gamma \cdot T_D(B,S) + T_V(B,S,\gamma)$$

则每个新token平均所需时间为： $T^{SD}_{avg} = \frac{T^{SD}_{Total}}{\Omega(\gamma,\alpha)}$

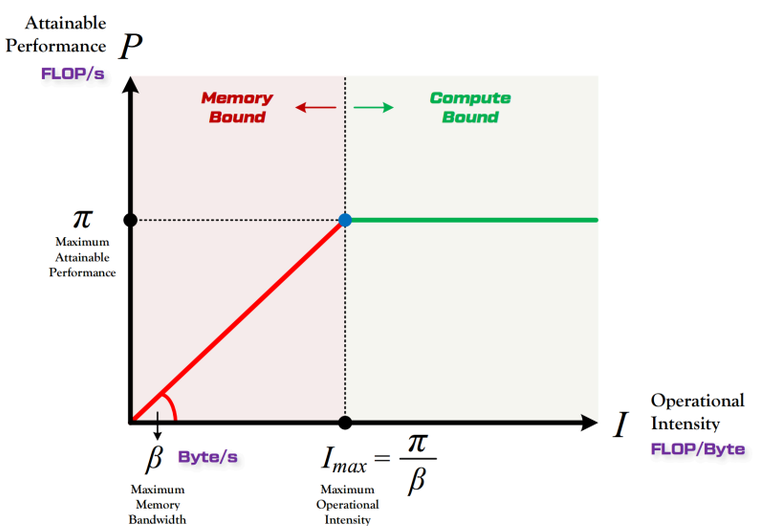
则平均每个新token生成的加速比为：

$$\frac{T_{Avg}^{SD}}{T_T} = \frac{1}{\Omega(\gamma,\alpha)}\left(\frac{\gamma \cdot T_D}{T_T} + \frac{T_V(\gamma)}{T_T}\right)$$

可见，影响平均加速比的因素有：

- 大模型verify γ 个draft token和大模型普通decode出1个token的延时之比：$\frac{T_V(\gamma)}{T_T}$（在前一篇文章中，认为这个比例是1，因为都是一次forward）。希望它尽可能小并接近1
- 小模型和大模型decode出1个token的延时之比：$\frac{T_D}{T_T}$。希望它尽可能小并接近0
- 每轮sd预期产生的有效token数量：$\Omega(\gamma,\alpha)$。希望它尽可能大

长文本下KV loading使得由compute bound变为了memory bound：



大模型verify γ 个draft token和大模型普通decode出1个token的kv loading开销是完全一样的，大bsz下由于前者计算强度过大（加载一份weight+kv cache执行的计算量很大）导致被计算资源限制了（变成了compute bound），使得前者的延时超过了后者很多。但在kv loading为主要瓶颈（占据了过多memory带宽）的情境下，又变成了memory bound（加载weight+很大一份kv cache执行的计算量相比之下比较小），此时verification带来的并行化又能够拉高计算强度。

设存在一个序列长度threshold $S_{inflection}$，当长度大于它时，sd可以为大bsz带来加速，且其加速效果和bsz成正相关

- 当序列长度小于该阈值时：bsz增大使得decoding变得更加compute-bound，占满可用计算资源，使得verification相对地更加昂贵，$\frac{T_V(\gamma)}{T_T}$ 增大显著。在序列长度小于阈值的情况下，sd的期望加速会随bsz增大而减小
- 当序列长度大于该阈值时：kv memory bound随bsz而增长，由于decode和verify的kv loading是一致的，因此 $\frac{T_V(\gamma)}{T_T}$ 还会接近1，然而它仍随着bsz单调递增，这就使得bsz增大的情况下仍然难以带来speed up。另外，如果小模型的kv cache size增长比大模型的曼，则小模型和大模型decode出1个token的延时之比 $\frac{T_D}{T_T}$ 会随着bsz增大而减小，这是因为大模型的latency会更被kv cache所主导

因此，如果能压缩小模型的kv cache，会带来较大好处（较低的draft cost和较高的接受率）

考虑kv cache压缩（kv selection方法）后的draft model，则加速收益可以如下建模：

设kv selection的budget为K（确保kv cache一直不超过这些），记kv selection带来的延时开销为 $T_{select}(B,S,K)$，则draft model相当于在一个长度为K的序列上做decode，因此draft model生成一个draft token的延时为：

$$T_{D,select_K}(B,S)=T_D(B,K)+T_{select}(B,S,K)$$

想要找到最优参数 $T_{select},K,\gamma,\alpha$，来得到最小的平均每个新token生成的加速比：

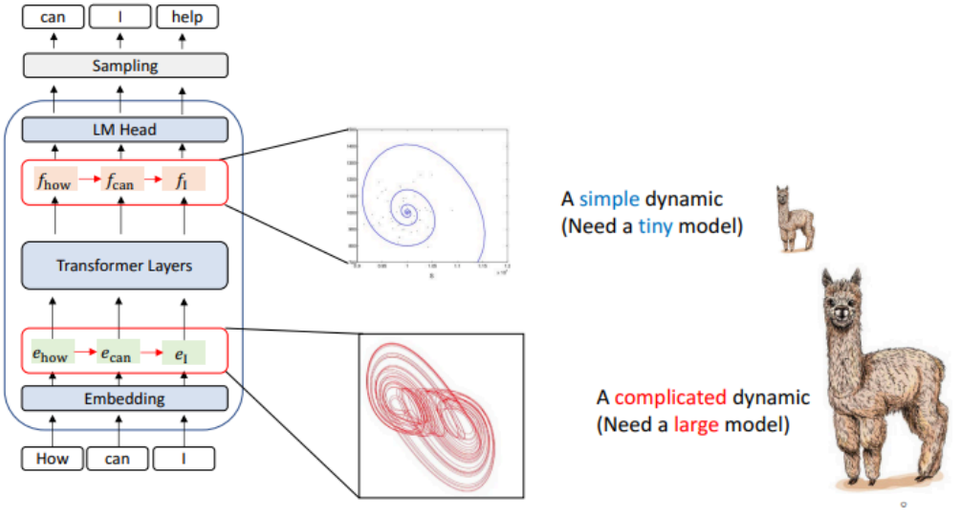
$$\min_{T_{select},K,\gamma,\alpha}\left[\frac{T_{Avg}^{SD}}{T_T}\right]=\min_{T_{select},K,\gamma,\alpha}\left[\frac{1}{\Omega(\gamma,\alpha)}\left(\frac{\gamma\cdot(T_D(B,K)+T_{select}(B,S,K))}{T_T(B,S)}+\frac{T_V(B,S,\gamma)}{T_T(B,S)}\right)\right]$$

[EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty]

目标：相比于重训整个draft model来说更小的训练开销（TinyLLaMA等draft model需要巨量token训练，而eagle的draft model只有一个decoder层用于预测下一个feature，可以用很少的token训好），同时接收率保持很高（lookahead、medusa等接收率太低）

基于如下两个观察：

- 在feature层面做autoregressive预测比token层面更简单，因为feature序列相比于token序列更具有规律性，因此draft model中仅有一个decoder layer就足够处理了



- 采样过程内禀的不确定性制约了next feature预测的表现。假设类似普通LLM自回归生成token的过程，只根据上一步的feature f_{i-1} 来预测下一个feature f_i，从而进行feature-level的自回归生成，那么由于缺少 $f_{i-1} \rightarrow e_i$ 这个上一步token采样结果的信息，会导致输入的feature f_{i-1} 存在歧义（不知道它上一步到底通过lm head采样出了什么token），会进一步导致 $f_i \rightarrow e_{i+1}$ 采样到的 e_{i+1} 和上一步实际采样到的 e_i 难以保持连贯，这就在token层面破坏了前后连续性。因此，在draft model中会联合使用 (f_{i-1}, e_i) 来推测下一个feature f_i，使得 f_i 中也包括了上一步的采样结果（“拍板”）信息，再通过lm head解码出 e_{i+1}

算法设计：

在模型架构上：draft model只有一个embedding层（和大模型共享）、一个lm head（和大模型共享）、一个可训练的autoregression head（包括一个fc层+一个decoder layer）。

假设只用chain draft（不引入tree）

假设prompt长度为t：$[token_1, \cdots, token_t]$

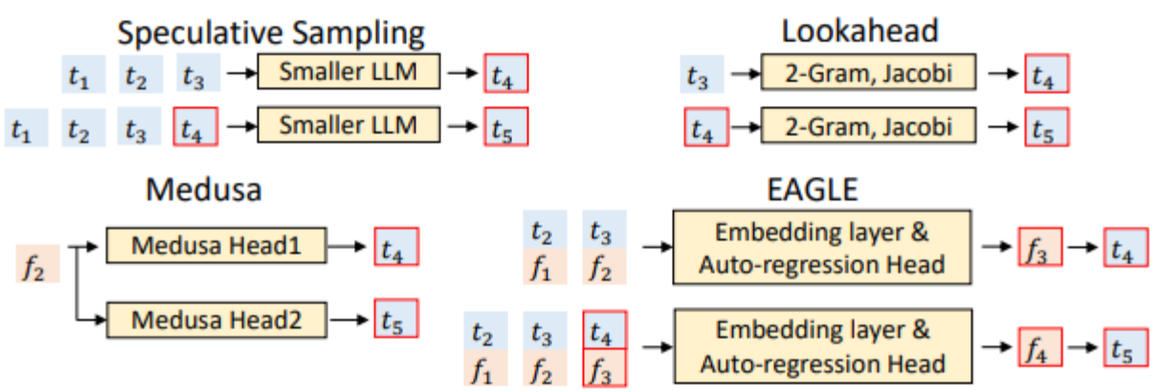
- 首先使用大模型进行一次forward，得到prompt的features：$[f_1, \cdots, f_t]$，并解码出 $token_{t+1}$（本质上，$token_{t+1}$ 是由 f_t 经过lm head变换得到的）
- 然后，将token序列 $[token_2, \cdots, token_{t+1}]$ 输入到draft model的embedding层，得到它们的embedding张量：$[emb_2, \cdots, emb_{t+1}]$，其形状为(bsz, seq_len, hidden_dim)
- 然后，将大模型得到的features $[f_1, \cdots, f_t]$ 和上述embedding张量沿着feature维度拼接成一个张量：$[(f_1, emb_2), \cdots, (f_t, emb_{t+1})]$，其形状为(bsz, seq_len, 2*hidden_dim)

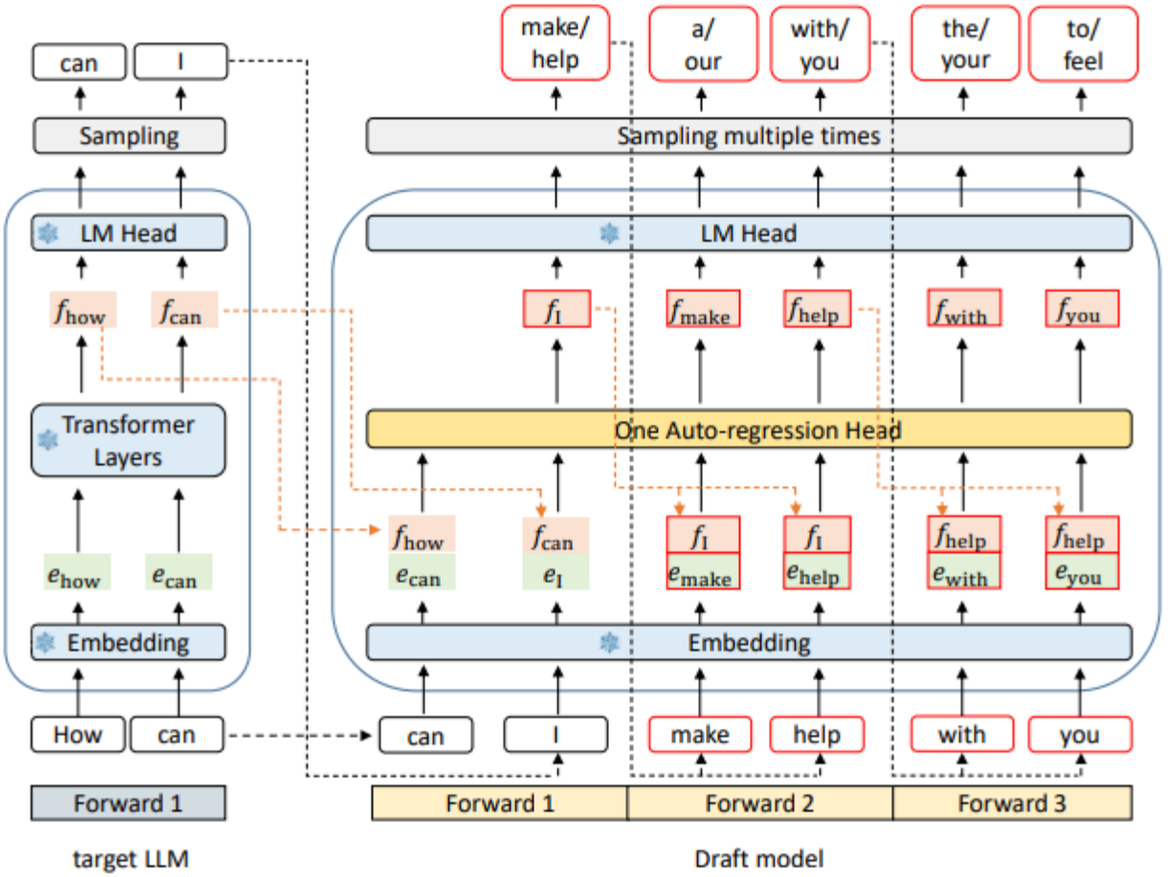
- 然后，使用autoregressive head中的fc层将其维数映射回hidden_dim，也即最终把features和embeddings融合成了一个(bsz, seq_len, hidden_dim)形状的张量，此时每个token的representation中都包含了其自己的embedding信息和生成它的上一个token的feature信息（也即，$token_t$ 现在的representation里含有了 f_{t-1}, emb_t 的信息）
- 然后，将其输入decoder layer（本次会输入整个序列的融合特征，用来生成kv cache）并预测得到下一个位置的feature（也即 $token_{t+1}$ 的feature）： f_{t+1}
- 然后，将 f_{t+1} 输入到lm head，变换得到 $token_{t+2}$ ，这也就得到了1个draft出来的token

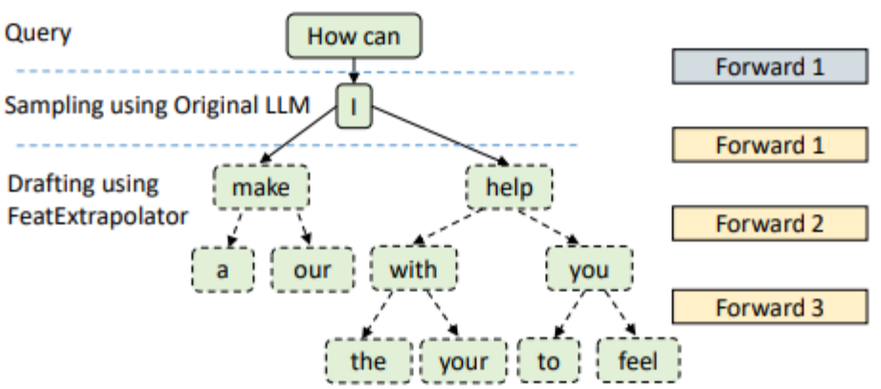
下面开始计算第2个draft token：

- 将 $token_{t+2}$ 输入到draft model的embedding层，得到其embedding： e_{t+2}
- 然后，将上一轮得到的 $token_{t+1}$ 的feature f_{t+1} 和 e_{t+2} 融合，得到表征 (f_{t+1}, e_{t+2}) ，维数为2*hidden_dim
- 然后，使用fc等将其降维成hidden_dim
- 然后，将该representation输入decoder layer，利用其中储存的history的kv cache，得到下一个位置的feature： f_{t+2}
- 然后，将 f_{t+2} 输入到lm head，变换得到 $token_{t+3}$ ，这也就得到了第2个draft出来的token

本质上，相当于draft model内部的autoregressive过程是针对feature进行的，只不过每次forward得到下一个feature后会捎带使用lm head来解码出其对应的draft token







draft model的训练：

- regression loss：对于每个i，希望draft model得到的feature（ $\hat{f}_{i+1}$ ）能够尽可能拟合主模型的feature（ f_{i+1} ）

$$\hat{f}_{i+1} = Draft\_Model(T_{2:i+1}, F_{1:i})$$

$$L_{reg} = \text{Smooth L1}(f_{i+1}, \text{Draft\_Model}(T_{2:i+1}, F_{1:i})).$$

- classification loss：除了直接在数值上拟合feature张量这个中间目标以外，最终目标是希望draft model得到的feature $\hat{f}_{i+1}$ 通过lm head输出的token分布 $\hat{p}_{i+2}$ ，能够尽可能拟合主模型真正的feature f_{i+1} 通过lm head输出的token分布 p_{i+2} ：

$$
\begin{aligned}
p_{i+2} &= \text{Softmax}(\text{LM\_Head}(f_{i+1})), \\
\hat{p}_{i+2} &= \text{Softmax}(\text{LM\_Head}(\hat{f}_{i+1})), \\
L_{cls} &= \text{Cross\_Entropy}(p_{i+2}, \hat{p}_{i+2}).
\end{aligned}
$$

由于classification loss在数量级上大致为regression loss的10倍，因此将二者以 $w_{cls} = 0.1$ 加权后作为draft model的总loss：

$$L = L_{reg} + w_{cls} L_{cls}$$

[Deepseek-V3 MTP]

deepseek中的MTP主要是为了使得训练信号密度更高从而在训练时提高数据效率，还可以让模型预先计划好它的representations来更好预测未来的tokens

普通的MTP（类似medusa）是使用若干个head来并行地decode出未来的若干个token，head_i负责未来的第i个token。而在deepseek中，使用一系列（D个）MTP module来串行计算出未来的若干个（D个）token（有点类似specdec中使用小模型来串行地快速draft出未来若干个tokens）。

MTP module的结构非常类似eagle 1中draft model，其相比于之前的MTP方法增加了causal chain的连接关系。每个MTP module包括：一个embedding层（和大模型共享）、一个output head（和大模型共享）、一个projection层和一个transformer block（可训练，每个MTP module自己拥有一份）。

在训练过程中，对于第i个输入token t_i ：

- 在第k=1个MTP module处：
  - 将其在上一个MTP module（对于第1个MTP module来说也即主模型）的feature h_i^0 和第i+1个token的embedding emb_{i+1} （分别RMSNorm后）沿feature维度进行concat，然后再使用Linear层降维到正常的hidden_dim：

$$h_i'^1 = Linear_k(RMS(h_i^0), RMS(emb_{i+1}))$$

- 然后再将这个融合后的representation输入transformer block，得到第i个token在本MTP block处的feature h_i^1 。

  事实上，这一步会计算出所有 $1:T-1$ 个输入token在该层的隐含状态 $h_{1:T-1}^1$ ，其中T是输入序列长度：

$$h_{1:T-1}^1 = TRM_1(h'^1_{1:T-1})$$

- 最后，将 h_i^1 作为输入给到lm head处，其会计算出第k=1个额外预测token的概率分布，并计算celoss：

$$p_{i+2}^1 = LMHead(h_i^1)$$

> 注：$p_{i+1}^0 = LMHead(h_i^0)$ 由主模型自己算出
>
> 在第1个MTP module处，得到的 h_i^1 用于预测第 $i+2$ 个token： $h_i^0 + emb_{i+1} \rightarrow h_i^1 \rightarrow p_{i+2}^1$

- 在第k=2个MTP module处：
  - 将其在上一个MTP module的feature h_i^1 和第i+2个token的embedding emb_{i+2} （分别RMSNorm后）沿feature维度进行concat，然后再使用Linear层降维到正常的hidden_dim：

$$h_i'^2 = Linear_k(RMS(h_i^1), RMS(emb_{i+2}))$$

  - 然后再将这个融合后的representation输入transformer block，得到第i个token在本MTP block处的feature h_i^2 。

    事实上，这一步会计算出所有 $1:T-2$ 个输入token在该层的隐含状态 $h_{1:T-2}^2$ ，其中T是输入序列长度：

$$h_{1:T-2}^2 = TRM_2(h'^2_{1:T-2})$$

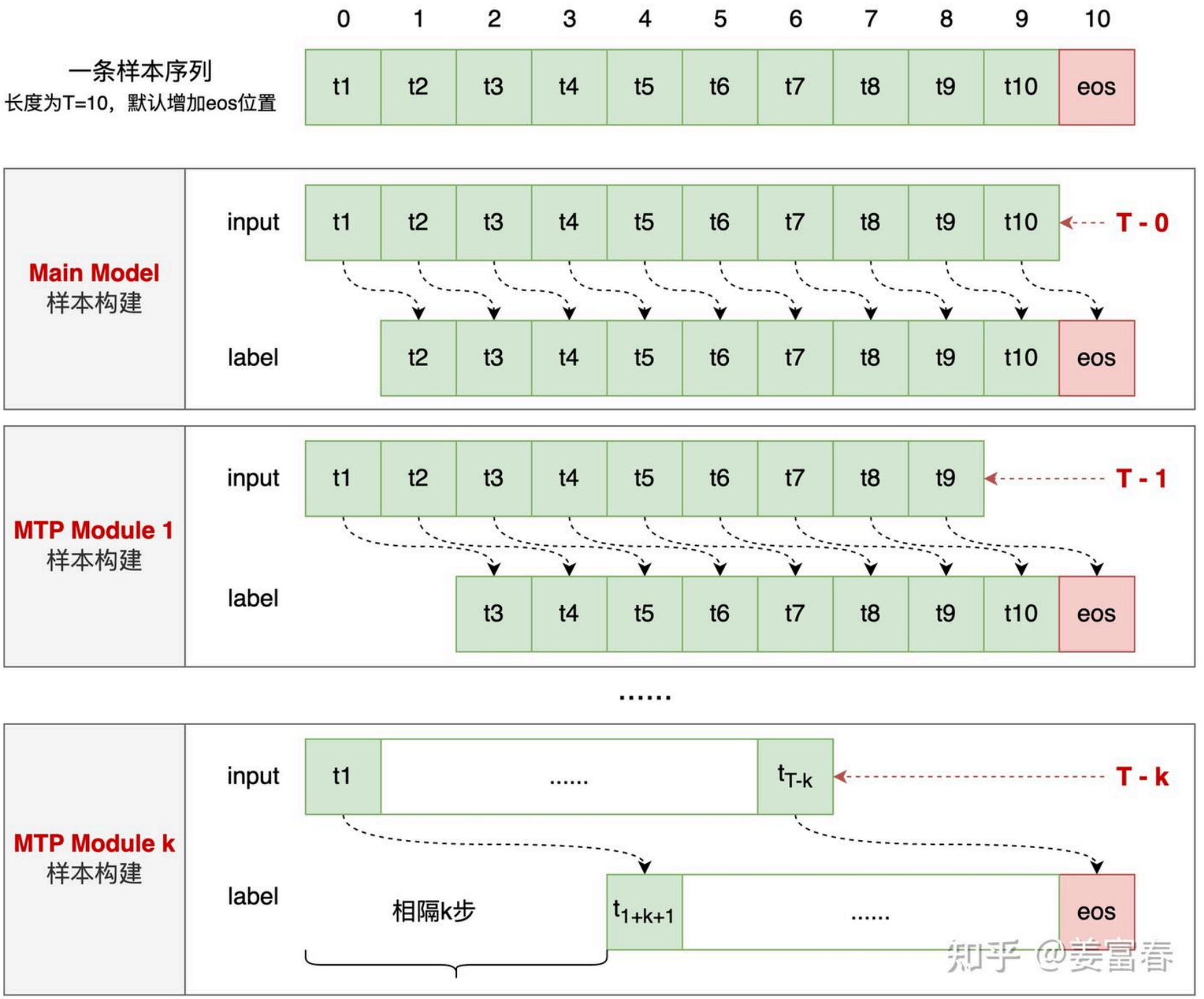
  - 最后，将 h_i^2 作为输入给到lm head处，其会计算出第k=2个额外预测token的概率分布，并计算celoss：

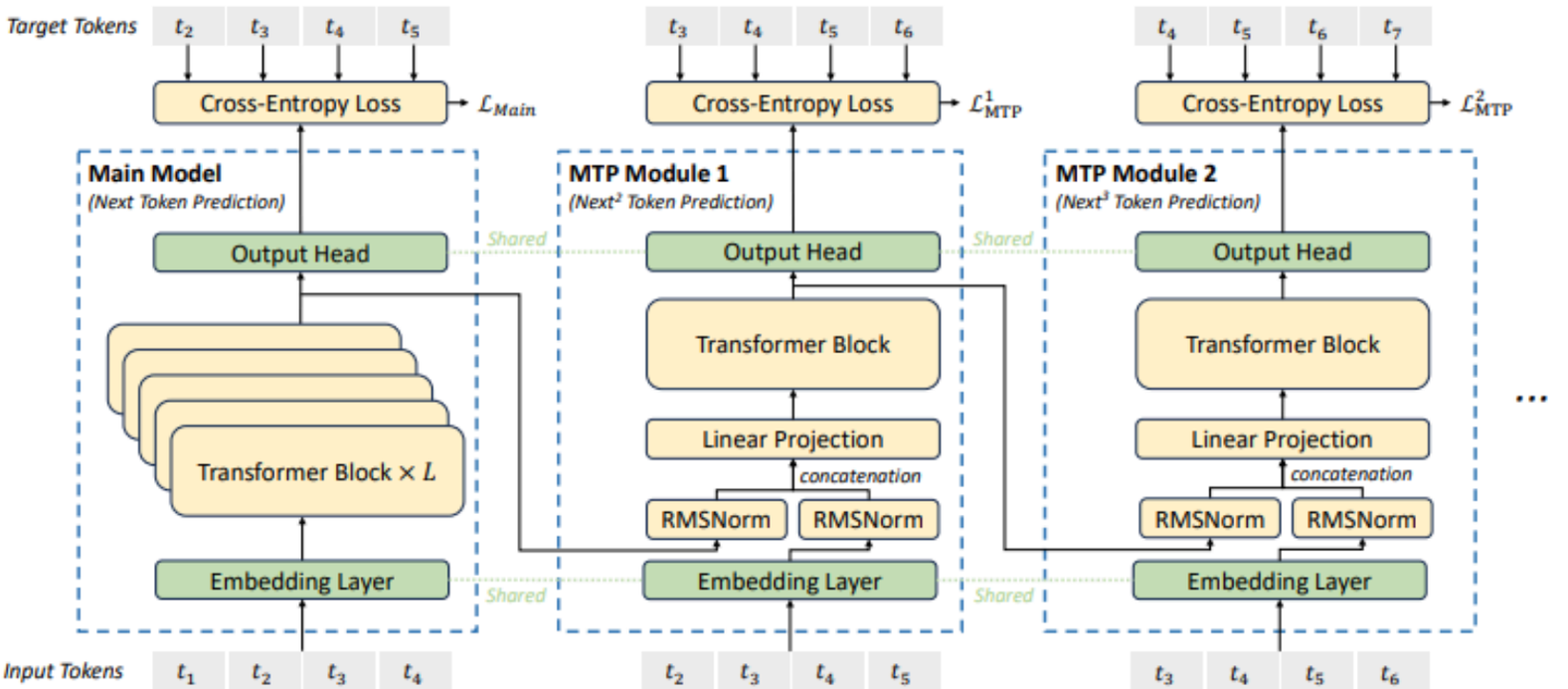
$$p_{i+3}^2 = LMHead(h_i^2)$$

> 在第2个MTP module处，得到的 h_i^2 用于预测第 $i+3$ 个token： $h_i^1 + emb_{i+2} \rightarrow h_i^2 \rightarrow p_{i+3}^2$

总的来看，也即：

第k个MTP module会让前 $T-k$ 个tokens分别预测其之后第 k 个token：主模型预测next token，MTP module 1预测next next token，....。这样对于比较靠前的token来说，能够多次利用它，并建立它和更久远之后的token的联系


0 1 2 3 4 5 6 7 8 9 10
一条样本序列
长度为T=10，默认增加eos位置
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 eos
Main Model
样本构建
input
label
T - 0
MTP Module 1
样本构建
T - 1
.....
MTP Module k
样本构建
T - k
相隔k步
eos



对于第k个MTP module来说，其窗口范围内预测的tokens的celoss：

$$\mathcal{L}_{\text{MTP}}^{k} = \text{CrossEntropy}(P_{2+k:T+1}^{k}, t_{2+k:T+1}) = -\frac{1}{T}\sum_{i=2+k}^{T+1}\log P_i^k[t_i],$$

最终计算所有深度下的MTP celoss的平均值，然后再除以一个scaling系数，即可得到总的MTP loss：

$$\mathcal{L}_{\text{MTP}} = \frac{\lambda}{D}\sum_{k=1}^{D}\mathcal{L}_{\text{MTP}}^{k}.$$

推理过程：从外部来看，和eagle1是一样的。参考上图：

MTP module1： $f_4 + emb_5 \rightarrow f_4' \rightarrow t_6$

MTP module2： $f_4' + emb_6 \rightarrow f_4'' \rightarrow t_7$

MTP module3： $f_4'' + emb_7 \rightarrow f_4''' \rightarrow t_8$

相当于若干步之后的draft token其实还是从输入序列最后一个token的feature $f_4, f_4', f_4'', \cdots$ 通过跨步生成的

注意，每个module处输入的token数量永远保持输入序列长度（上图中为4）。尽管每个新token总是由最后一个输入序列token的feature偏移得到的，但计算过程中还是需要前边token的features用于做attention

如果只保留MTP module1，把它当eagle用：

$$
\begin{aligned}
f_4 + emb_5 &\rightarrow f_5(f_4') \rightarrow t_6 \\
f_5 + emb_6 &\rightarrow f_6 \rightarrow t_7 \\
f_6 + emb_7 &\rightarrow f_7 \rightarrow t_8
\end{aligned}
$$

Learning harmonized representations for speculative sampling（HASS）

Motivation：

1、训练和decoding过程中的context不align。eagle等模型训练时输入draft model的都是主模型的真实features $f_1, \cdots, f_t$，但实际decoding过程中，未经主模型验证过的早期draft tokens用的都是draft model自己生成的近似版的features，这就造成了训练和推理过程中难以对齐。

2、训练和decoding过程的目标也存在差异。decoding阶段，draft model的目标是给出大模型更有可能赋予高概率的tokens。在这种情况下，draft model应该更关注召回需要的tokens，而这些tokens的具体顺序则不那么重要。大部分LLM应用都用top-k sampling等，对于这些decoding策略的目标来说，高概率的tokens对于确定输出具有更大的影响力。因此，draft model的训练策略应该有意识考虑decoding阶段遇到的这些情况。

本文提出HASS，通过harmonized objective distillation来解决训推过程目标差异的问题，通过harmonized context alignment来解决context alignment问题。

1、harmonized objective distillation

HASS通过推荐系统中的ranking distillation方法来优先那些最decoding-desired的tokens。在推荐系统中，希望训练出一个学生模型，其能够给予那些被teacher模型赋予top-rank的items高的rank。设draft model为student model，主模型为teacher model，则：能够给主模型赋予高概率的tokens赋予高概率的draft model，其在decoding阶段自然接收率很高。考虑一个K个tokens的集合 $\hat{\Omega}$，它是主模型赋予最高概率的token集合 $\hat{\Omega} \subset \Omega$，其中 Ω 是整个词表。则HASS考虑如下top-k distillation loss：

$$
L_{\text{Top-K}} = -\sum_{x \in \hat{\Omega}} q(x) \log p(x),
$$

这里 q, p 分别是主模型和draft model的next token probability。如果想和eagle集成，则可以通过主模型的hidden states来获得最高概率集合 $\hat{\Omega}$

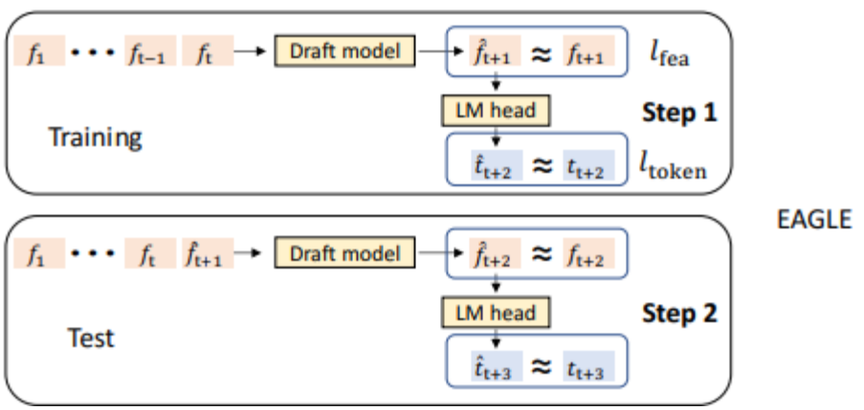
2、harmonized context alignment

zhuanlan.zhihu.com

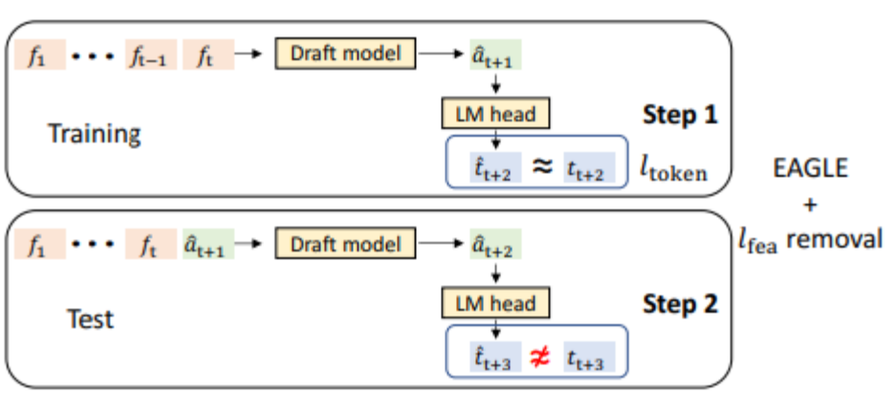
EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test

Motivation：

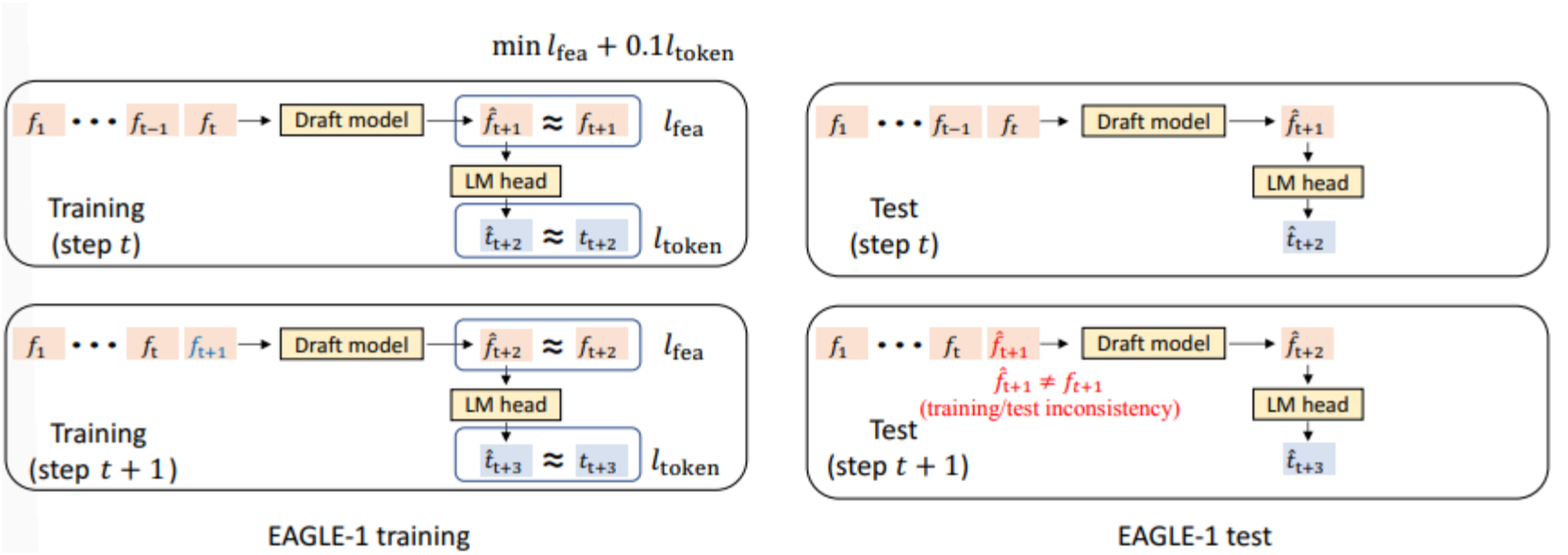
想通过增大训练数据量来让draft model更强，但发现eagle1中增大训练数据量效果不明显，这是因为eagle1的训练目标中包括了feature拟合的loss和token拟合的loss，其中的feature拟合loss成为了一种限制，限制了draft model的表达能力，使得其难以受益于增大的训练数据量。



移除feature拟合项后，，发现第1个draft token（ $[(f_1, emb_2), \cdots, (f_t, emb_{t+1})] \rightarrow \hat{a}_{t+1} \rightarrow token_{t+2}$ ）的接收率显著上升（因为训练过程本质上就是预测下一个draft token）。然而，由于缺少了feature张量层面的拟合，尽管第1个token的接收率很高，但得到它的由draft model产生的feature $\hat{a}_{t+1}$ 却和真正的 f_{t+1} 相差很大，这就导致从此之后的draft model解码过程中，所用的feature序列 $f_1, f_2, \cdots, f_t, \hat{a}_{t+1}$ 和训练时的分布相差很大，导致接下来解码预测的特征 $\hat{a}_{t+2}$ 彻底混乱，进一步导致第2个draft token及以后的tokens的接收率非常低

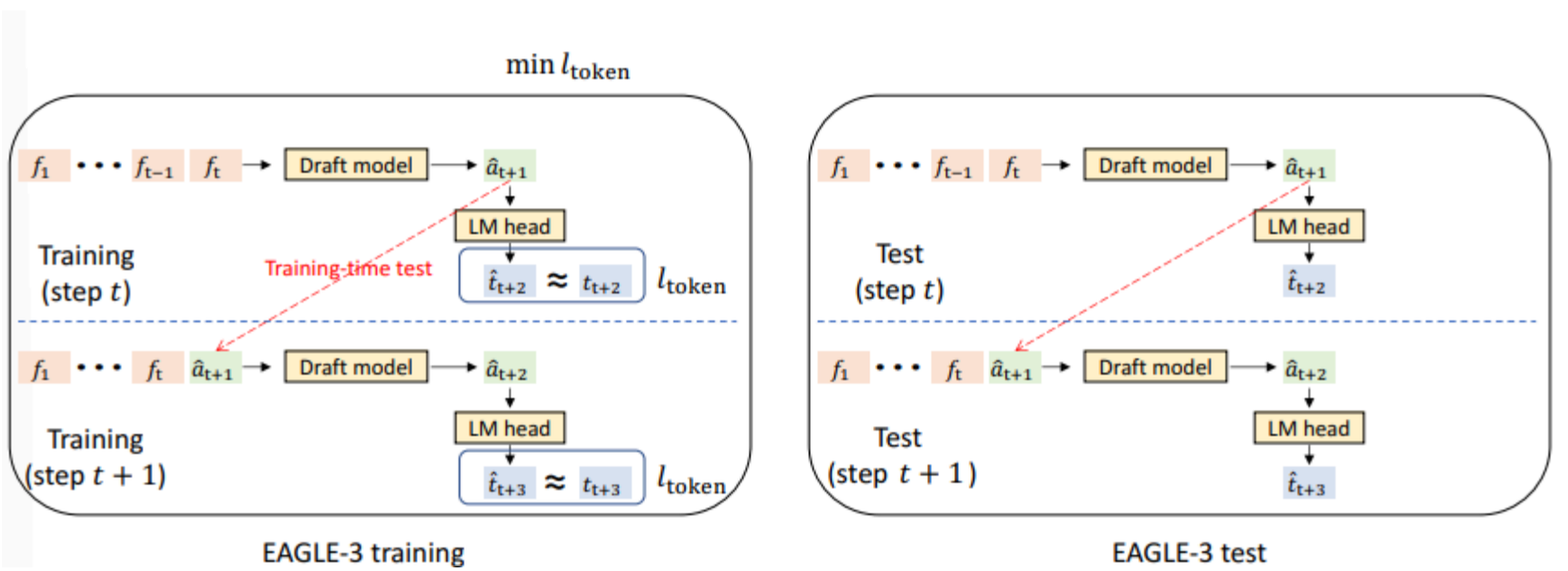


另外，eagle的training和testing过程的目标存在不一致性：训练中，第2个draft token及以后都是由主模型ground truth的feature f_{t+1} 等生成的，然而推理时只能使用draft model自己生成的 $\hat{f}_{t+2}$ 等生成，造成了不一致性。（在训练目标去除feature拟合项后更加严重）



EAGLE-1 training　　EAGLE-1 test

因此，提出training-time test，其核心改变在于在训练中对于新生成的draft token，使用的都是draft model自己生成的feature $\hat{a}_{t+1}$ 等。另外，由于只是用top-layer特征来预测next-next-token存在本质的困难，文章提出进一步结合利用中间层的特征而不是只依赖top层特征

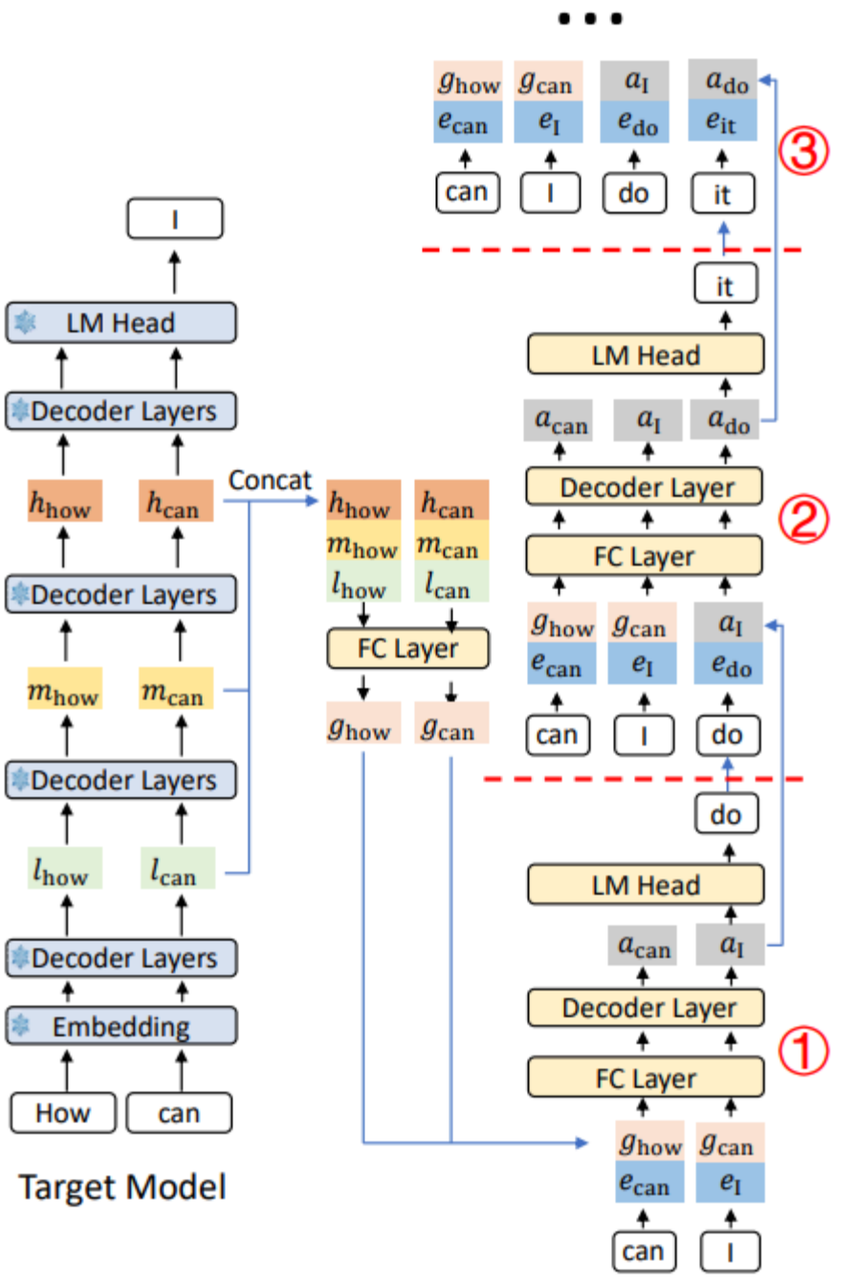


EAGLE-3 training　　EAGLE-3 test

Eagle3推理流程：

在prefill或上一轮verification stage时，和eagle1一样，也是先由主模型生成1个新token $token_{t+1}$，同时记录第1~t个输入token在主模型中的low,middle,high三个level的features（l, m, h），将每个token的这3个特征连接并用一个线性层融合成

一个综合特征： $Linear(l_i, m_i, h_i) \rightarrow g_i$



训练流程：

在eagle中，输入draft model的都是来自target model的top layer features $f_1, \cdots, f_t$ 。而在eagle3中，输入draft model的既有可能是来自于主模型的融合features $g_1, g_2, \cdots, g_t$ ，也可能是draft model自己之前生成的 $a_{t+1}, a_{t+2}, \cdots$ 。因此，需要训练draft model来适应这些不同的输入。在训练过程中，一个输入句子会执行若干test steps来逐步生成 $a_{t+1}, a_{t+2}, \cdots$ ，并将它们输入回draft model进一步用于训练