字节HLLM:在推荐系统落地User-Item分层的LLM方案

原创 州懂学习笔记 州懂学习笔记 2024年10月27日 23:55 广东


州懂学习笔记
分享大模型推荐系统相关知识和学习笔记
39篇原创内容

公众号

字节HLLM:在推荐系统落地User-Item分层的LLM方案

> 标题: HLLM: Enhancing Sequential Recommendations via Hierarchical Large Language Models for Item and User Modeling
> 地址: https://arxiv.org/pdf/2409.12740
> 公司: 字节
> 代码: https://github. com/bytedance/HLLM

1. 前言

基于ID的推荐系统一般都是Embedding参数量很大而模型参数量较小,这种ID-based的模型主要存在两个问题:

- **冷启乏力:** 严重依赖ID特征导致在冷启动场景中表现不佳
- **能力欠佳:** 相对浅层的网络模型难以建模复杂且多样的用户兴趣

随着近年来LLM的突破性进展, 业界也在不断探索LLM在推荐系统中的应用, 这里大概可以分成三类:

- **信息增强:** 利用LLM为推荐系统提供一些精细化的信息, 例如做Item的特征增强
- **对话式处理:** 将推荐系统转换为与LLM兼容的对话驱动形式
- **直接输入ID:** 修改LLM不再仅处理文本输入/输出, 比如直接输入ID特征给LLM

当前, LLM4Rec还面临着一些挑战, 比如在处理相同时间跨度的用户行为序列时, 使用文本作为输入的LLM要比ID-based方法需要处理更长的序列长度, 且计算复杂度更高。此外, 基于LLM的方法远没有其它领域那么显著的提升。

作者提出, 关于LLM4Rec还有三个关键问题还需要再探索的:

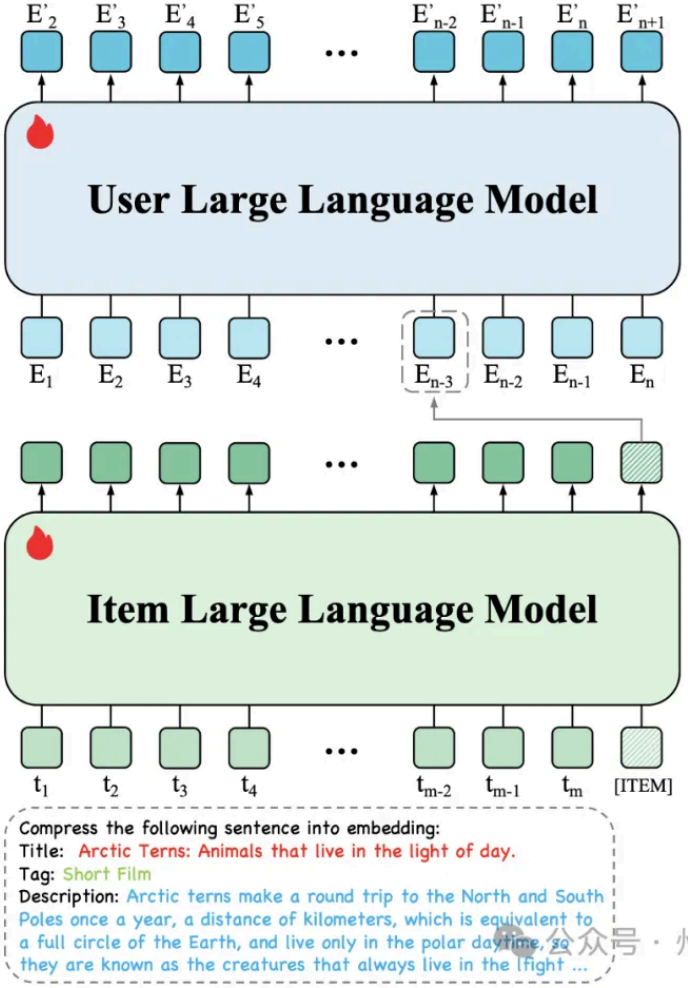
1. **预训练LLM权重的真正价值:** LLM大规模预训练本质上是在做一个世界知识的压缩, 其权重蕴含着世界知识, 但使用大规模推荐数据训练时, 这些权重的真正价值还尚未挖掘。

2. **有必要对推荐任务进行微调吗?** 还不清楚进一步使用推荐任务微调是否有收益。
3. **LLM在推荐场景中能否呈现Scaling Law?** 在更大规模的参数下, LLM4Rec是否也能像其它场景一样具有Scaling Law呢, 还未有结论。

基于此, 作者提出了Hierarchical Large Language Model(HLLM)方法，下面进行详细介绍。

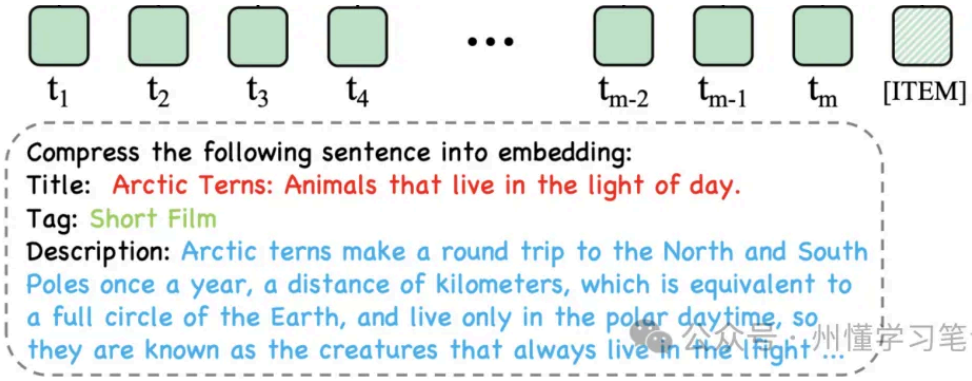
2. 方法

HLLM的整体框架如下图所示, 分为Item LLM和User LLM。

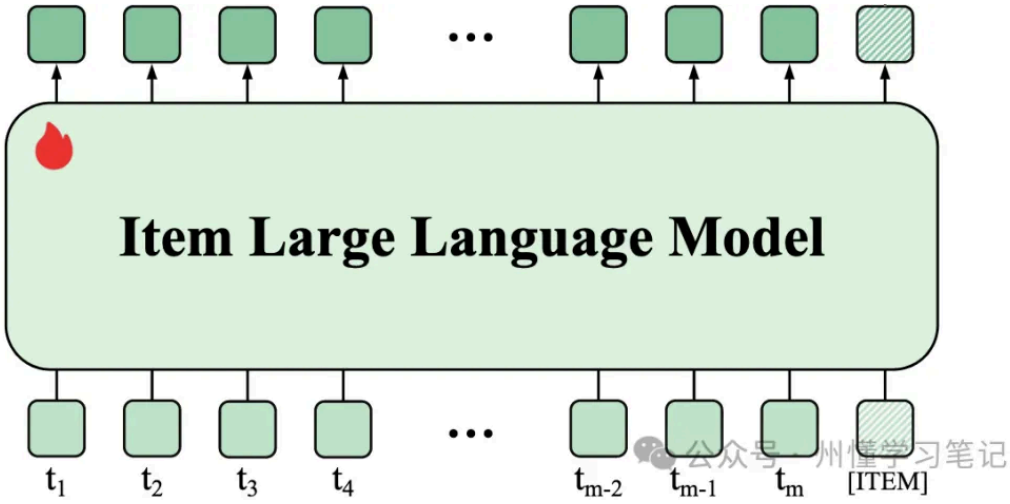


2.1 Item LLM

Item LLM的作用是作为特征提取器, 如下图所示, 使用Item的标题、标签、描述的文本信息作为输入, 并在最后位置额外增加了一个特殊Token［ITEM］。



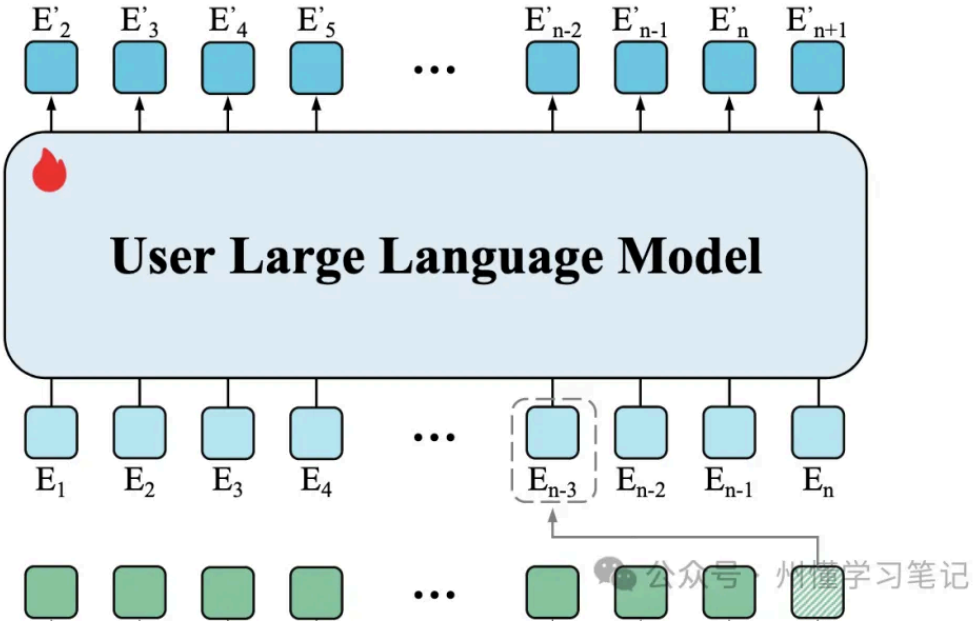
最终的输入为$\{t_1, t_2, \ldots, t_m, [\mathrm{ITEM}]\}$, 其中$m$为文本token的长度, 而特殊Token对应的输出则作为该Item的Embedding。



该段理解有误, 论文AB有提细节。忽略:论文没有提及Item LLM具体的预训练方式, 但笔者觉得这个才是最核心的地方。论文后面的实验部分有提及说Item LLM要使用推荐目标做微调会更好, 但没说在这一阶段推荐数据如何构造, 推荐目标具体如何微调。如果只是next token prediction的训练微调, 那纯粹只是学习Item的语义信息, 笔者感觉效果可能不会很好。参考业界小红书NoteLLM使用共现笔记做GCL实现微调, 相比而言, 笔者更倾向于相信作者也做了类似的事情, 引入了推荐用户行为信号去做微调。这里有其它看法的小伙伴也欢迎留言讨论。

2.2 User LLM

User LLM的输入是用户历史交互序列$U = \{I_1, I_2, \ldots, I_n\}$, 前面的Item LLM就起到传统深度模型的Embedding Lookup Table的作用, 这样就将用户行为序列转化成$\{E_1, E_2, \ldots, E_n\}$。



然后, 再将这些Embedding作为输入, 进行Next Item Prediction, 预测下一位置的Item Embedding。这里User LLM对于Next Item Prediction的训练目标, 按照训练方式的差异, 可以分成生成式推荐与判别式推荐, 而HLLM同时使用了这两种。

2.2.1 生成式推荐训练方式

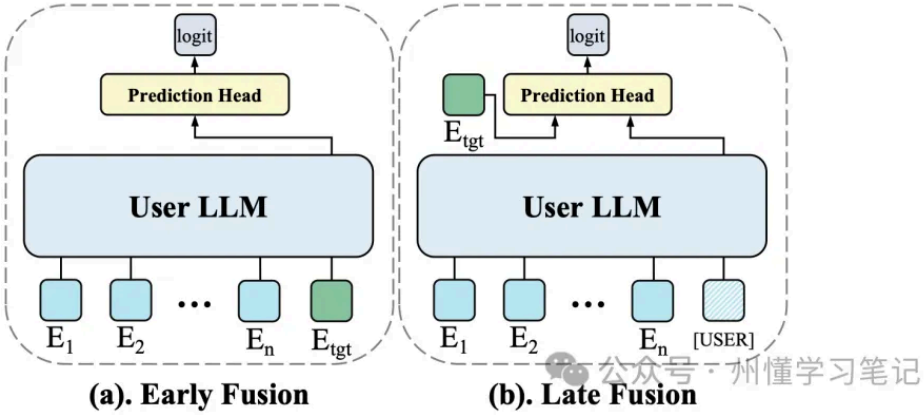
使用大模型自回归的训练方式, 使用InfoNCE来作为生成损失, 对于模型第$i-1$位置输出的Embedding E'_i, 将第i个位置输入的Embedding E_i, 构成pair对(E'_i, E_i)当作正样本:

$$\mathcal{L}_{gen} = -\sum_{j=1}^{b}\sum_{i=2}^{n}\log\frac{e^{s(E'_{j,i},E_{j,i})}}{e^{s(E'_{j,i},E_{j,i})}+\sum_{k}^{N}e^{s(E'_{j,i},E_{j,i,k})}}$$

其中, s是一个相似度函数, $E_{j,i}$表示第j个用户用户序列第i个位置Item所对应的Embedding, $E'_{j,i}$表示由用户LLM为第j个用户预测的第i个位置的Embedding, N为负采样的数量, $E_{j,i,k}$表示第k个负样本的Embedding, b表示Batch中的用户总数, n表示用户历史行为交互的长度

2.2.2 判别式推荐训练方式

判别式推荐的训练方式主要有两种: Early Fusion和Late Fusion, 作者在实际在落地使用的是Late Fusion。



1) Early Fusion

Early Fusion是将Target Item的Embedding E_{tgt}拼接在用户行为序列的最后, 再输入给User LLM, 然后再将对应位置的输出做分类预测。Early Fusion方式的优点是, Target Item可与用户行为序列在User LLM中进行充分的特征交叉, 它的效果一般会更好, 但效率较低。

2) Late Fusion

Late Fusion首先在用户行为序列的最后拼接一个特殊Token [USER], 类似Item LLM的方式, 再使用User LLM编码用户行为序列, 提取得到用户的Embedding(即[USER]位置对应的输出), 再将其与E_{tgt}拼接起来去预测分类。Late Fusion在后期再实现特征交叉, 效果一般会差一些, 但在推理时效率更高。比如当Prediction Head是使用内积去计算User Embedding与Target Item的Logit时, 那Late Fusion的User LLM就是一个双塔模型, 就可以直接使用向量索引做召回。

对于预测部分, 它是个二分类问题，训练损失函数如下:

$$\mathcal{L}_{cls} = -(y \cdot \log(x) + (1-y) \cdot \log(1-x))$$

其中，y表示训练样本的Label, x表示预测的logit。

2.2.3 整体训练Loss

这里, 作者同时使用生成式和判别式的损失, 将它们做加权融合:

$$\mathcal{L}_{dis} = \lambda \mathcal{L}_{gen} + \mathcal{L}_{cls}$$

其中, λ是生成式辅助Loss的权重系数。

此外, 可能有读者会有疑惑, User LLM和Item LLM是联合训练的吗。笔者认为它们是独立训练的, 这里主要原因是不具备联合训练的条件, Item LLM的输入为Item的文本描述, 而User LLM的输入为各Item的Embedding, End2End训练的话, 成本太高, 可能并不现实。

3. 实验部分

3.1 LLM预训练及推荐目标微调的作用

LLM有无预训练对推荐效果的影响 -> 结论:无论是Item LLM还是User LLM，基于预训练微调更好

Item LLM	User LLM	R@5	R@10	N@5	N@10
Scratch	Scratch	3.330	5.063	2.199	2.755
Scratch	**Pre-trained**	3.556	5.416	2.371	2.969
Pre-trained	Scratch	3.521	5.331	2.358	2.940
Pre-trained	**Pre-trained**	**3.755**	**5.581**	**2.513**	**3.100**

Table 2: Ablation studies of pre-training on Pixel200K with HLLM-1B.

预训练权重的质量与推荐效果的影响 -> 结论: 预训练使用的Token越多, 推荐效果越好, 此外, 增加对话场景的SFT对推荐场景并无收益。

#Tokens	R@5	R@10	N@5	N@10	CSR ↑
0T	3.330	5.047	2.199	2.755	-
0.1T	3.539	5.142	2.399	2.915	46.11
1T	3.613	5.409	2.414	2.993	50.22
1T+chat	3.610	5.387	2.411	2.984	51.36
2T	3.650	5.510	2.466	3.063	51.64
3T	**3.755**	**5.581**	**2.513**	**3.100**	**52.99**

Table 3: The impact of different pre-training token counts on Pixel200K with HLLM-1B. "+chat" means SFT on conversation data. The CSR metric is the average performance on the common sense reasoning tasks.

使用推荐目标做微调的影响 -> 结论: 无论是Item LLM还是User LLM,都非常有必要使用推荐目标做微调。

Item LLM	User LLM	R@5	R@10	N@5	N@10
Frozen	**Learnable**	0.588	0.945	0.372	0.486
Learnable	Frozen	1.619	2.470	1.070	1.343
Learnable	**Learnable**	**3.755**	**5.581**	**2.513**	**3.100**
SASRec-1B		1.973	2.868	1.352	1.640

Table 4: Ablation studies of fine-tuning on Pixel200K with HLLM-1B.

3.2 HLLM是否具有Scaling Law

坦白说, 这部分实验做的没啥说服力, 看看就好。

Item LLM部分

Item Model	#Params	R@5	R@10	N@5	N@10
BERT-Base	110M	2.576	4.020	1.694	2.158
BERT-Large	340M	3.032	4.635	1.993	2.508
TinyLlama	1.1B	**3.484**	**5.239**	**2.319**	**2.883**

Table 5: Experiments with different sizes of the item model on Pixel200K. SASRec is used as the user model for all.

User LLM部分

User Model	#Params	R@5	R@10	N@5	N@10
SASRec	4M	3.484	5.239	2.319	2.883
Llama-2L	0.1B	3.494	5.233	2.338	2.898
TinyLlama	1.1B	**3.521**	**5.331**	**2.358**	**2.940**

Table 6: Experiments with different sizes of the user model on Pixel200K. Llama-2L maintains the same architecture as Llama but uses only 2 decoder layers. TinyLlama-1.1B is used as the item model for all. All user models are trained from scratch.

3.3 对比基线

整体效果

Dataset	Method	R@10	R@50	R@200	N@10	N@50	N@200	Impv. (avg)
Pixel8M	$SASRec_{vit}$ (2024)	3.589	-	-	1.941	-	-	-27.72%
	HSTU* (2024)	4.848	10.315	18.327	2.752	3.939	5.135	+0.0%
	SASRec*	5.083	10.667	18.754	2.911	4.123	5.331	+3.82%
	HSTU-1B*	5.120	11.010	19.393	2.879	4.159	5.411	+5.37%
	SASRec-1B*	5.142	10.899	19.044	2.915	4.166	5.383	+4.83%
	HLLM-1B (Ours)	**6.129**	**12.475**	**21.179**	**3.539**	**4.919**	**6.221**	**+22.93%**
Amazon Books	SASRec (2018)	3.06	7.54	14.31	1.64	2.60	3.62	+0.0%
	LEARN (2024)	4.07	9.79	18.74	2.24	3.71	4.83	+34.42%
	HSTU-large (2024)	4.78	10.82	19.08	2.62	3.93	5.17	+47.80%
	SASRec*	5.35	11.91	21.02	2.98	4.40	5.76	+64.96%
	SASRec-1B*	5.09	11.11	19.45	2.86	4.17	5.42	+55.68%
	HSTU-large*	5.00	11.29	20.13	2.78	4.14	5.47	+55.61%
	HSTU-1B*	5.25	12.03	21.60	2.89	4.36	5.80	+64.37%
	HLLM-1B (Ours)	**6.97**	**14.61**	**24.78**	**3.98**	**5.64**	**7.16**	**+108.68%**
	HSTU-large†*	6.49	12.22	19.81	3.99	5.24	6.38	+88.94%
	HLLM-1B-Scratch† (Ours)	6.85	13.95	23.19	4.02	5.56	6.95	+103.65%
	HLLM-1B† (Ours)	9.28	17.34	27.22	5.65	7.41	8.89	+166.42%
	HLLM-7B† (Ours)	**9.39**	**17.65**	**27.59**	**5.69**	**7.50**	**8.99**	**+169.58%**

3.4 训练和 Serving 效率

与HSTU的对比, 结论: 仅需要1/6~1/4的数据就可达到HSTU的同等水平

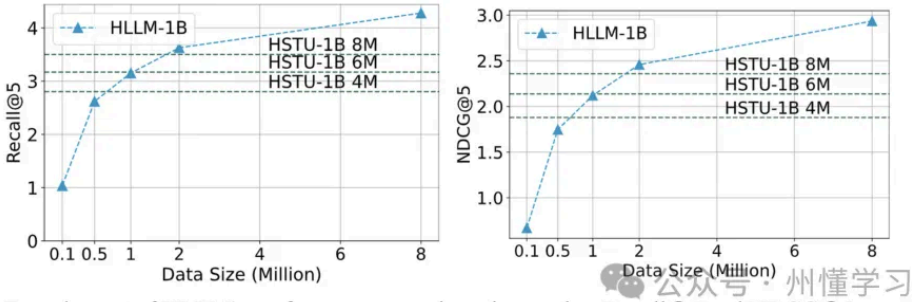


Figure 3: Experiments of HLLM's performance at various data scales. Recall@5 and NDCG@5 are reported.

推理时, 可使用Item Cache方法提效, 虽然效果会有一定下降, 但仍好于HSTU

Method	R@5	R@10	N@5	N@10
HSTU-1B	3.501	5.120	2.358	2.879
$HLLM\text{-}1B_{cache}$	3.585	5.218	2.432	2.958
HLLM-1B	**4.278**	**6.106**	**2.935**	**3.524**

3.5 线上AB

关键指标提升了0.705%.

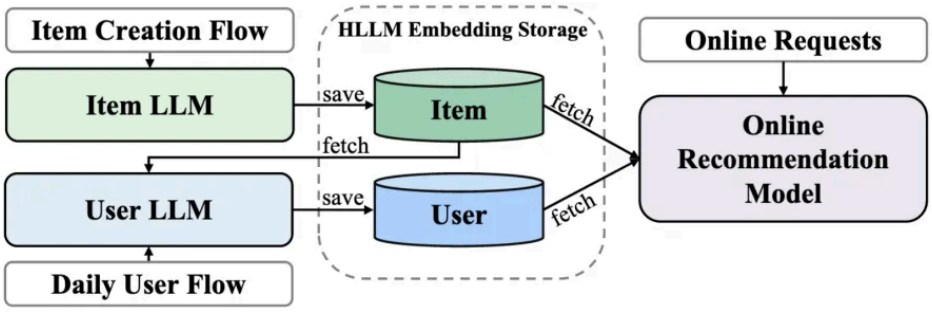


Figure 4: An overview of the online system.