Medusa 批量测试&词表缩减

开发批量推理脚本；在给定测试集下，测试：

medusa_head=3

chain_structures = [[0],[0,0],[0,0,0]]

代码块

```
1   {"type": "summary_report", "total_samples_processed": 10, "average_metrics":
    {"wall_time_init": 0.914, "wall_time_medusa": 0.036, "wall_time_tree": 1.591,
    "wall_time_posterior": 0.01, "wall_time_update": 0.024, "compression_ratio":
    3.887, "tokens_per_second": 48.972}}
```

medusa_head=4：

1. chain_structure = [[0],[0,0],[0,0,0],[0,0,0,0]]

代码块

```
1   {"type": "summary_report", "total_samples_processed": 1000, "average_metrics":
    {"wall_time_init": 1.121, "wall_time_medusa": 0.062, "wall_time_tree": 1.135,
    "wall_time_posterior": 0.007, "wall_time_update": 0.017, "compression_ratio":
    4.705, "tokens_per_second": 53.779}}
```

```
==================================================
FINAL AVERAGES (Appending to file...)
wall_time_init           : 1.121
wall_time_medusa         : 0.062
wall_time_tree           : 1.135
wall_time_posterior      : 0.007
wall_time_update         : 0.017
compression_ratio        : 4.705
tokens_per_second        : 53.779
==================================================
```

观察到在我们的应用场景下，chain_structure已经具备加速比4.7+的性能。原61node是过剩的。

2. 8_node = [[0],[0, 0], [1],[0, 1],[0,0,0],[1,0],[1,0,0],[0,0,0,0]]

代码块

```
{"type": "summary_report", "total_samples_processed": 1000, "average_metrics": {"wall_time_init": 1.06, "wall_time_medusa": 0.043, "wall_time_tree": 1.124, "wall_time_posterior": 0.008, "wall_time_update": 0.017, "compression_ratio": 4.752, "tokens_per_second": 54.515}}
```


```
==================================================
FINAL AVERAGES (Appending to file...)
wall_time_init           : 1.06
wall_time_medusa         : 0.043
wall_time_tree           : 1.124
wall_time_posterior      : 0.008
wall_time_update         : 0.017
compression_ratio        : 4.752
tokens_per_second        : 54.515
==================================================
```

3. 8_node_2 = [[0],[0, 0],[0,0,0],[0,0,0,0],[0,0,0,1],[0,0,0,2],[0,0,0,3],[0,0,0,4]]

代码块


```
{"type": "summary_report", "total_samples_processed": 1000, "average_metrics": {"wall_time_init": 0.99, "wall_time_medusa": 0.051, "wall_time_tree": 1.156, "wall_time_posterior": 0.007, "wall_time_update": 0.018, "compression_ratio": 4.747, "tokens_per_second": 56.368}}
```


```
==================================================
FINAL AVERAGES (Appending to file...)
wall_time_init           : 0.99
wall_time_medusa         : 0.051
wall_time_tree           : 1.156
wall_time_posterior      : 0.007
wall_time_update         : 0.018
compression_ratio        : 4.747
tokens_per_second        : 56.368
==================================================
```

4. 16_node = [[0],[0, 0],[0,0,0],[0,0,0,0],[0,0,0,1],[0,0,0,2],[0,0,0,3],[0,0,0,4],[0,0,0,5],[0,0,0,6],[0,0,0,7],[0,0,0,8],[0,0,0,9],[0,0,0,10],[0,0,0,11],[0,0,0,12]]

代码块


```
{"type": "summary_report", "total_samples_processed": 925, "average_metrics": {"wall_time_init": 1.014, "wall_time_medusa": 0.056, "wall_time_tree": 1.1, "wall_time_posterior": 0.008, "wall_time_update": 0.017, "compression_ratio": 4.746, "tokens_per_second": 55.687}}
```

```
==================================================
FINAL AVERAGES (Appending to file...)
wall_time_init           : 1.014
wall_time_medusa         : 0.056
wall_time_tree           : 1.1
wall_time_posterior      : 0.008
wall_time_update         : 0.017
compression_ratio        : 4.746
tokens_per_second        : 55.687
==================================================
```

分析：与8_node_2 相比，并没有明显提升。

初步结论：

在 Compression Ratio高达 4.7+的前提下，继续单纯扩展 Medusa Head 4 的宽度（从 Top-5 增至 Top-13）已无法带来任何性能提升，反而因计算开销增加导致 TPS 轻微下降。主要原因如下：

1. **概率长尾**：正确答案若未命中 Top-5，极大概率也不会出现在 Top-6 到 Top-13 之间（长尾概率几乎为 0），因此额外的候选节点几乎从未被选中。
2. **前缀依赖**：CR 未满 5.0 说明剩余的预测失败部分发生在中间层（如 Head 2 或 3）。
3. **负优化**：16-node 引入了额外的 Logits 排序、Top-K 筛选和显存读写Overhead，在CR无增长的情况下拖累了推理速度。

当前实验中，相对最优的树结构是8_node = [[0],[0, 0], [1],[0, 1],[0,0,0],[1,0],[1,0,0],[0,0,0,0]]。建议使用。

采用共享的LM head，训练Medusa，测试：

medusa_head=4

1. 8_node = [[0],[0, 0], [1],[0, 1],[0,0,0],[1,0],[1,0,0],[0,0,0,0]]

与每个medusa_head各一个LM head的方案相比，推理速度快近一倍，同时compress_ratio维持4.7+

```
==================================================
FINAL AVERAGES (Appending to file...)
wall_time_init          : 0.91
wall_time_medusa        : 0.072
wall_time_tree          : 1.146
wall_time_posterior     : 0.008
wall_time_update        : 0.018
compression_ratio       : 4.726
tokens_per_second       : 54.793
==================================================
```

1. 批量推理+测试，启动脚本：

   /mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/multest_WO_linear/medusa_vlm_inference_controller.py

2. 训练启动脚本：

   /mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/medusa/train/train_qwenvl_WO_share_lm.sh

3. 训练代码：

   /mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/medusa/train/train_legacy_WO_share_lm.py

4. Medusa权重：

   /mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/WO_share_lm_head_4_medusa_mlp_qwen3_nio_medusa_4_lr_0.001_layers_1

5. 实验结果：
   /mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/metrics/WO_lm/WO_lm_metrics_node_8.jsonl

词表缩减

~~idea1: 照搬EAGLE3词表缩减逻辑~~

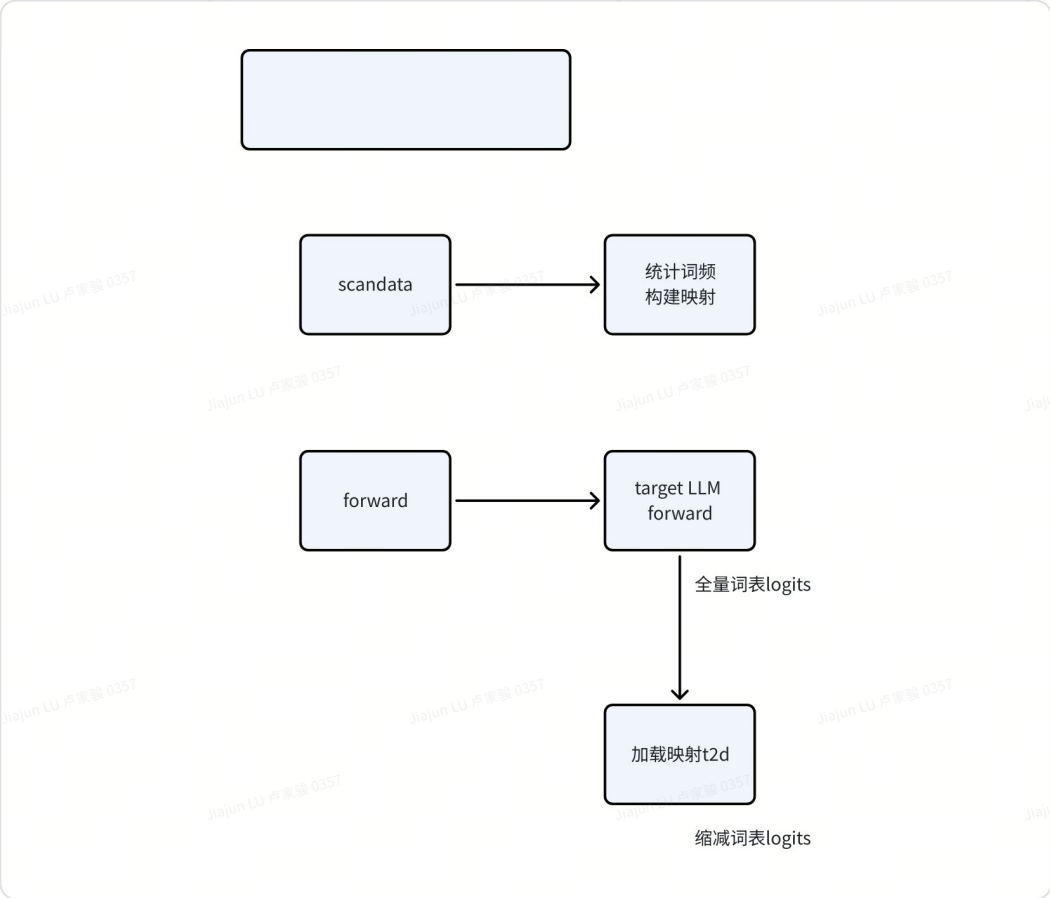
可行性：否。EAGLE3单独实现了草稿模型的LM head，其词表缩减也是针对draft LM head实现的。这与我们共用base LM head的前提矛盾。

~~idea2：借用EAGLE3词表缩减。维持base LLM完全不动；forward时对LM head做切片，仅取缩减后的词表对应的行~~

可行性：理论可行且代码改动小，但不符合业务要求。要求必须缩减base model的词表，以突破带宽限制。

实现t2d映射 （for training）

EAGLE3的实现方式：



迁移到Medusa需要的改动：
1. loss不同，导致需要的映射文件不同。

EAGLE3：拟合target llm和draft model 的logits分布。用t2d bool mask实现。

Medusa采用hard CE：loss_i = loss_fct(medusa_logits, medusa_labels) 。这里medusa_labels来源于base model，而logits是被缩减的。放弃EAGLE3的t2d逻辑，

转用 `kept_indices` (保留索引表，forward时做LM_head切片)和 `orig2draft` (原始转草稿映射表，训练时映射medusa_labels)

medusa.notebooks/create_medusa_indices.py

功能：读取训练集；统计词频；建立 `kept_indices` .pt和 `orig2draft` 文件。

解释：t2d映射文件是一长为[vocab_size]的bool数组，出现在训练集中的id为true，否则为false：

idea3: 直接对base model做词表缩减

必须这么做！现在端侧芯片是带宽受限而非算力受限；当前业务场景下，每次加载15w的词表是冗余的；且能保证训练集token全覆盖。

做法：词频统计得到映射表；生成+手动改新词表（几个.json）【151396->432】；利用added_tokens来做，可以避免分词问题。根据映射表裁剪base model：取embedding和LM head对应行，其余保留；用新base model训练Medusa；推理测试。

🏖️ 路径：

环境：/mnt/beegfs/groups/cv/yuboyang/anaconda3/envs/medusa

Base model词表缩减：

【方式一：pipeline】脚本：./notebooks/vocab_decrease/vocab_decrease_pipeline.py

【方式二：】

1. 统计词频，生成词表映射文件t2d.pt（记录训练集token对应的原词表id）：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/vocab_decrease/create_medusa_indices.py

2. 生成新tokenizer：

2.1
/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/vocab_decrease/create_tokenizer.py

2.2
/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/vocab_decrease/add_tokenizer.py

3. 生成新的base model：裁剪原base model的embedding和LM head，保留其余结构。并进行必要的文件复制，组成完整可用的新base model。

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/vocab_decrease/slice_model_v2.py

——

4. 批量推理+测试，启动脚本：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/vocab_decrease/medusa_vlm_inference_controller.py

5. 训练启动脚本：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/medusa/train/vocab_train_qwenvl_WO_share_lm.sh

6. 训练代码：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/medusa/train/train_legacy_WO_share_lm.py

7. 新词表：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/vocab_decrease/medusa_tiny_tokenizer

8. 缩减词表base model：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/vocab_decrease/medusa_tiny_model

9. Medusa权重：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/vocab_WO_share_lm_head_4_medusa_mlp_medusa_tiny_model_medusa_4_lr_0.001_layers_1

10. 实验结果：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/metrics/vocab_WO_lm_metrics_node_8.jsonl

测试结果：

epoch	Tree_structure	wall_time_init	wall_time_medusa	wall_time_tree	wall_time_posterior	wall_time_update	compression_ratio (↑)	tokens_per_second (↑)
1	8_node（LM_head共享）	0.911	0.036	1.085	0.008	0.018	**4.725**	**57.802**

Base model大小：

Token压缩前：9.1G；Token压缩后：7.6G

实验结果汇总 Medusa_head = 4

1. Base model = /mnt/beegfs/groups/cv/yuboyang/data/qwen3_nio

Tree_structure	wall_time_init	wall_time_medusa	wall_time_tree	wall_time_posterior	wall_time_update	compression_ratio (↑)	tokens_per_second (↑)
chain_structure	1.121	0.062	1.135	0.007	0.017	4.705	53.779
8_node	1.06	0.043	1.124	0.008	0.017	4.752	54.515
8_node_2	0.99	0.051	1.156	0.007	0.018	4.747	56.368

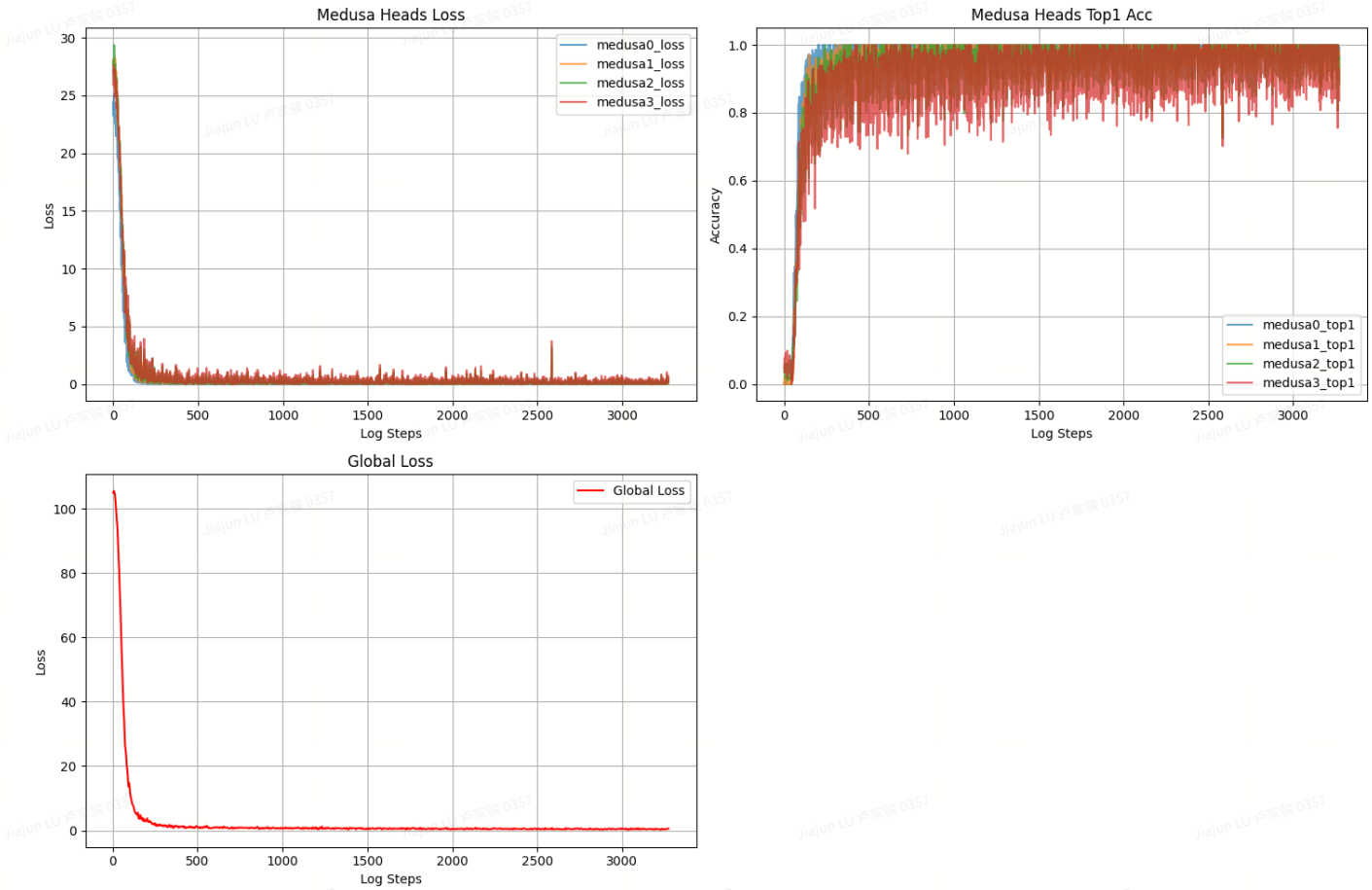
16_node	1.014	0.056	1.1	0.008	0.017	4.746	55.687
8_node（LM_head共享）	0.91	0.072	1.146	0.008	0.018	4.726	54.793
8_node(LM_head共享+词表缩减)	**0.911**	**0.036**	**1.085**	**0.008**	**0.018**	**4.725**	**57.802**

2. Base model = /mnt/beegfs/groups/cv/yuboyang/data/qwen3_nio_2b

Tree_structure	**wall_time_init**	**wall_time_medusa**	**wall_time_tree**	**wall_time_posterior**	**wall_time_update**	**compression_ratio (↑)**	**tokens_per_second (↑)**
8_node（LM_head共享）	0.865	0.051	1.556	0.013	0.022	1.373	24.385

现象：采用新的base model，指标大幅下降。加速比4.7x -> 1.3x

分析1：训练问题？



分析2：输出似乎不对齐？

代码块

```
1   testset第一条: "content": "```json\n{\n    \"person_num\": 1,\n
    \"pet_exist\": false,\n    \"persons\": [\n        {\n
    \"seat_pos\": \"主驾\",\n            \"mask\": \"没戴口罩\",\n
    \"smoke\": \"否\",\n            \"call\": \"否\"\n        }\n    ]\n}\n```"}]}
2   输出第一条:    "output": "person_num: 1\npet_exist: false\npersons[1]
    {seat_pos,mask,smoke,call}:\n  主驾,否,否,否<|im_end|>\n<|im_end|>"
```

定位问题：数据格式不匹配。base model训练数据是Token-Oriented Object Notation (TOON) 格式的，而此处Medusa训练仍用普通JSON格式。

解决问题：用正确的数据集重新训练Medusa

1. 训练数据：

/mnt/beegfs/groups/cv/yuboyang/data/eagle_data/structured_data_shuf_train_toon.jsonl

2. 测试数据：

/mnt/beegfs/groups/cv/yuboyang/data/eagle_data/structured_data_shuf_eval_toon.jsonl

3. 训练脚本：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/medusa/train/train_qwenvl_WO_share_lm_noon.sh

4. 训练代码：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/medusa/train/train_legacy_WO_share_lm_noon.py

5. Medusa权重：

Epoch=1：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/WO_share_noon_medusa_mlp_qwen3_nio_2b_medusa_4_lr_0.001_layers_1

Epoch=2：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/WO_share_noon_ep2_medusa_mlp_qwen3_nio_2b_medusa_4_lr_0.001_layers_1

6. 批量推理+测试 脚本：

/mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/notebooks/multest_WO_linear/medusa_vlm_inference_controller_noon.py

7. 实验结果：
   /mnt/beegfs/groups/cv/jiajun.lu/mllm_medusa/work_dirs/metrics/qwen3_nio_2b_noon_WO_lm_metrics_node_8.jsonl

epoch	Tree_structure	wall_time_init	wall_time_medusa	wall_time_tree	wall_time_posterior	wall_time_update	compression_ratio (↑)	tokens_per_second (↑)
1	8_node（LM_head共享）	0.786	0.056	0.549	0.02	0.008	**3.944**	**43.427**
2	8_node（LM_head共享）	0.818	0.064	0.546	0.013	0.008	**3.996**	**43.362**