上海交通大学

自动驾驶行为世界仿真

编程综合实践

姓　名：**龙子豪**　　**刘可唯**

学　号：**524031910498**　**524020910154**

导　师：**张致远**　　**贾萧松**

学　院：**人工智能学院**

2025 年 7 月

目 录

第一章 绪论

1.1 引言

自动驾驶行为仿真是指通过计算机模拟技术，构建虚拟的驾驶环境和交通场景，对自动驾驶系统的决策、规划、控制等行为进行测试和验证的过程。其意义在于可模拟极端场景，减少实车测试成本，加速开发迭代等。

上海交通大学团队提出的 TrajTok 方法在 CVPR2025 行为世界模型竞赛斩获头名[1]，本次编程综合实践课我们有幸能加入到这个团队，开始学习，理解，并以开源项目 Closed-Loop Supervised Fine-Tuning of Tokenized Traffic Models 的代码[2]为基础尝试修改模型。

我们的模型叫做 SMART-AMP，结合了我们所做的两个改进：Autoencoder 和 Multi-Token Prediction。

1.2 编程综合实践课小结

1、通过阅读四篇论文了解当前 SOTA 模型的原理；
2、通过 Debug 代码了解各部分文件作用，神经网络架构，张量形状变化；
3、学会利用 SSH 远程连接机器，并会使用部分 Linux 命令行操作；
4、在跑通，理解代码基础上，我们尝试对现有模型做出了两个小修改；
5、进行实验，测试修改后代码的性能。

1.3 编程综合实践课收获

通过这一个月的学习，我们熟悉了 SSH、命令行操作等计算机基本技能，提升了独立阅读论文的能力，并提高了整体把握项目框架和细致分析张量形状的能力。更重要的是，这次课程培养了我们独立探索的科研精神，让我们掌握了宏观微观相结合的研究方法，相信这为我们今后的科研工作打下了坚实的基础。

第二章 SMART 模型

2.1 模型思想

SMART(Scalable Multi-Agent Real-Time Motion Generation via Next-token Prediction) 首先引入地图数据 Tokenizer，并通过自回归预测下一个 Road token 来增强空间理解能力。然后采用 GPT-style 方法，将完整时间序列上的智能体轨迹进行 token 化，构建 Decoder-only Transformer 模型。该架构使 SMART 在推理过程中能够实时计算当前时刻下一帧的待生成 token，无需在每次推理时重复编码历史运动 token，从而显著提升实时交互式自动驾驶仿真的推理效率。[3]

2.2 Token 化方法

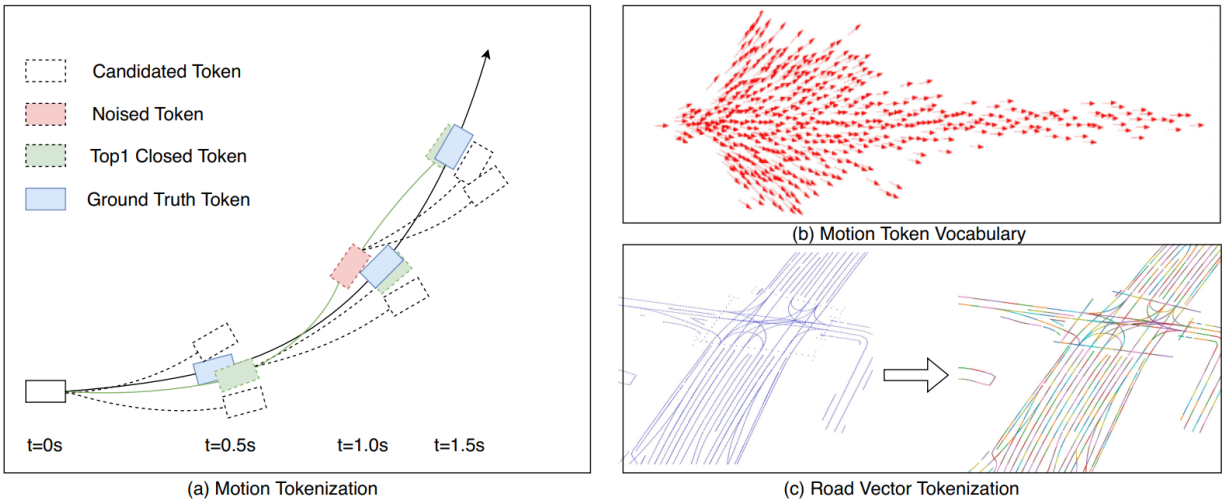

图 2-1 Token 化示意

2.2.1 k-disks 算法

k-disks 算法 采样一个大小为 N 的词汇表. 距离 $d(x_0,x)$ 衡量状态 x_0 和 x 四个顶点的平均距离，其中 x_0 和 x 是长、宽均为 1 的绑定框。[5]

Algorithm 1 KDISKS 采样 (X, N, ε)

procedure SAMPLEKDISKS(X, N, ε)
    $S \leftarrow \{\}$ **while** $len(S) < N$ **do**
        $x_0 \sim X$
    $X \leftarrow \{x \in X \mid d(x_0, x) > \varepsilon\}$
    $S \leftarrow S \cup \{x_0\}$
    return S
end procedure

2.2.2 地图 Token 化

1、从地图原始元素中随机取样 `argmin_sample_len` 个原始数据初始化词汇表，每个词汇包含全局坐标系下的位置和朝向。

2、将全局坐标系转换为局部坐标系, 利用 k-disks 方法, 生成能代表地图的 Token。

注：与运动轨迹 Token 化的区别：地图 Token 化不依赖时序。

2.2.3 运动轨迹 Token 化

1、从原始 agent 轨迹提取典型的动作集形成词汇表 V

(1) 我们定义 local 函数，global 函数：local 函数可将原始坐标转化为局部坐标，global 函数可将局部坐标还原。转化为局部坐标的优势在于可排除绝对位置的干扰，更专注于运动动作。

(2) 首先对于所有的 agent 轨迹进行处理。选取轨迹的时间长度 shift，即该段轨迹包含 shift 组 (x,y,θ) 数据，并将 (x,y,θ) 转化为局部坐标。这样我们得到了所有的动作。

(3) 这样得到的动作数量太多，不利于后续对轨迹的 Token 化操作。因此我们使用 k-disks 方法选取最具代表性的 N 个动作，并计算每个点的四角坐标，最终形成动作词汇表。

2、利用生成的离散动作词汇表, 将 agent 轨迹离散化

(1) 利用上一状态和词汇表中的 |V| 个动作，得到 |V| 个状态。

(2) 将这 |V| 个状态与当前真实状态做四角 l2loss，选出最适合的动作, 进而得到最适合的当前状态。

(3) 重复 (1)(2)，即可得到离散化轨迹。

2.3 模型架构

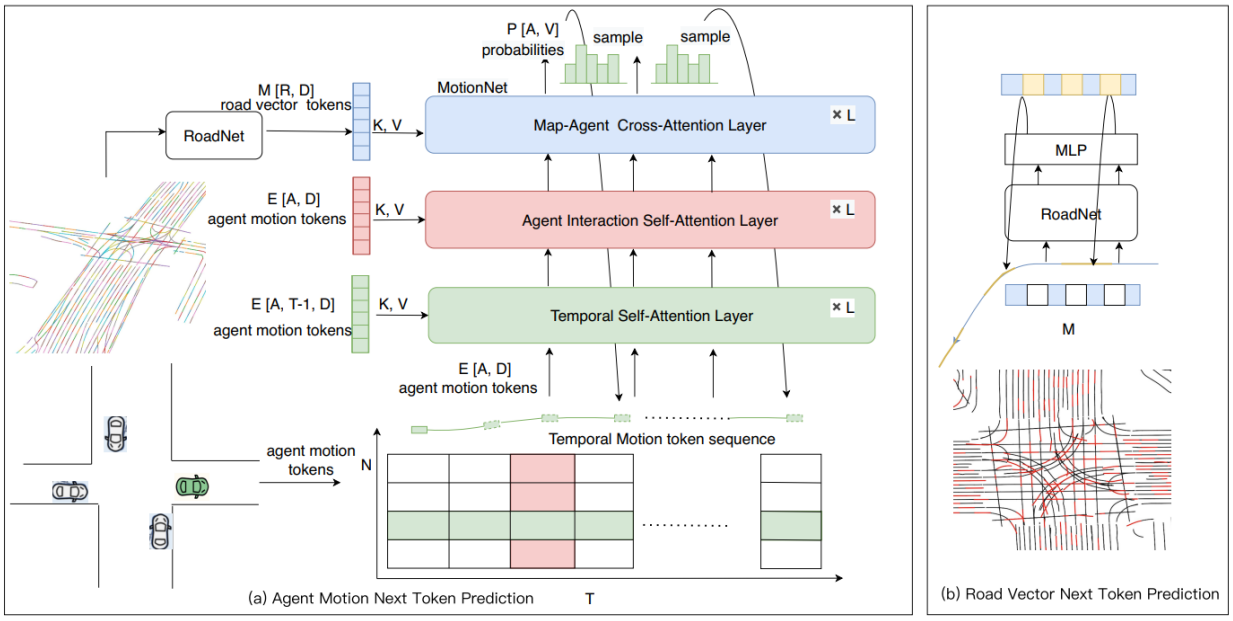

图 2-2 神经网络整体框架

2.3.1 RoadNet: Road Token Encoder

`RoadNet` 是一个基于图神经网络的地图表征学习模块，其核心思想是将地图元素（如车道线、交通信号等）编码为图结构，通过多层注意力机制学习地图元素间的空间关系。

具体来说，每个地图点包含

$$\mathbf{p}_i = (x_i, y_i) \quad \text{位置坐标}$$
$$\theta_i \quad \text{方向角}$$
$$\mathbf{t}_i \in \mathbb{R}^{22} \quad \text{轨迹 token 特征}$$
$$c_i^{type}, c_i^{pl}, c_i^{light} \quad \text{类别标签}$$

将类别标签和轨迹 token 特征分别做 Embedding，得到初始图节点。

$$\mathbf{x}_i^{(0)} = \mathbf{e}^{token} + \mathbf{e}^{type} + \mathbf{e}^{pl} + \mathbf{e}^{light}$$

使用半径图构建邻接关系:

$$\mathcal{E} = \{(i,j) : \|\mathbf{p}_i - \mathbf{p}_j\|_2 \le r\}$$

对于每条边 (i,j)，计算相对特征, 相对位置特征通过 Fourier 嵌入编码.
多层注意力更新: 对于 $l = 1,2,\ldots,L$ 层:

$$\mathbf{x}_i^{(l)} = \text{AttentionLayer}(\mathbf{x}_i^{(l-1)}, \{\mathbf{r}_{ij}\}_{j\in\mathcal{N}(i)}, \text{EdgeIndex})$$

其中注意力层包含:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V},\mathbf{R}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{R}}{\sqrt{d_k}}\right)\mathbf{V}$$

算法流程:

Algorithm 2 RoadNet

Require: tokenized_map
Ensure: $\{\mathbf{X}^{(L)}, \mathbf{P}, \Theta, \mathbf{B}\}$

提取位置 $\mathbf{P}$, 朝向 Θ, token 特征 $\mathbf{T}$

计算类别 Embedding:

$$\mathbf{E}_{cat} = \mathbf{E}_{type} + \mathbf{E}_{pl} + \mathbf{E}_{light}$$

计算 Token Embedding:

$$\mathbf{E}_{token} = \text{MLP}(\mathbf{T})$$

初始化节点特征:

$$\mathbf{X}^{(0)} = \mathbf{E}_{token} + \mathbf{E}_{cat}$$

构建半径图: $\mathcal{E} = \text{radius\_graph}(\mathbf{P}, r)$

计算相对位置编码:

$$\mathbf{R} = \text{FourierEmb}(\text{relative\_features})$$

for $l = 1$ ***to*** L **do**

$$\mathbf{X}^{(l)} = \text{AttentionLayer}(\mathbf{X}^{(l-1)}, \mathbf{R}, \mathcal{E})$$

return $\{\mathbf{X}^{(L)}, \mathbf{P}, \Theta, \mathbf{B}\}$

2.3.2 MotionNet: factorized agent motion decoder

`MotionNet` 是基于 Decoder-only Transformer 架构，通过 agent 的历史动作，agent 与 map 交互，agent 与 agent 交互来预测 ego agent 下一个动作的神经网络。

首先进行对处理好的 token 进行 Embedding，再利用 multi-head self-attention(MHSA) 处理时序关系，agent 与 agent 关系、multi-head cross-attention(MHCA) 处理 agent 与 map 关系，其中需要引入 relative positional embeddings(RPE)。

算法流程:

Algorithm 3 MOTIONNET

Require: tokenized_agent, map_feature
Ensure: next_token_logits

1: **计算**离散动作 Embedding，连续特征 Embedding，类别 Embedding:

$$\mathbf{E}_{motion}, \quad \mathbf{E}_{con}, \quad \mathbf{E}_{cat}$$

2: **合并**得到 agent 特征:

$$\textbf{featrue\_agents} = \mathbf{E}_{motion} + \mathbf{E}_{con} + \mathbf{E}_{cat}$$

3: **计算** RPE:

$$\textbf{RPE} = \text{FourierEmb}(\textbf{relative\_features})$$

4: **进入** AttentionLayer ($\mathbf{e}_i^t$ 为第 i 个 agent 在 t 时刻特征):

$$e_{i'} = \text{MHSA}\left(q(e_i), k(e_{i-\tau}, \text{RPE}_\tau), v(e_{i-\tau}, \text{RPE}_\tau)\right), \quad 0 < \tau < t \quad (a)$$

$$e_{i'} = \text{MHCA}\left(q\left(e_i^t\right), k\left(r_j, \text{RPE}_{ij}\right), v\left(r_j, \text{RPE}_{ij}\right)\right), \quad j \in N_i \quad (b)$$

$$e_{i'} = \text{MHSA}\left(q\left(e_i^t\right), k\left(e_j^t, \text{RPE}_{ij}^t\right), v\left(e_j^t, \text{RPE}_{ij}^t\right)\right), \quad j \in N_i \quad (c)$$

5: **得到**下一时刻每个 token 的 logit:

$$\textbf{next\_token\_logits} = \text{MLP}(\textbf{featrue\_agents})$$

注: Neighbor 集合 N_i 由 50 米的距离阈值确定。我们将时序注意力、agent-agent 注意力以及 agent-map 注意力依次堆叠为一个融合块，并重复该模块 K 次，且每个 AttentionLayer 都有 L 层。

第三章 模型改进

3.1 自编码器思想优化 Embedding

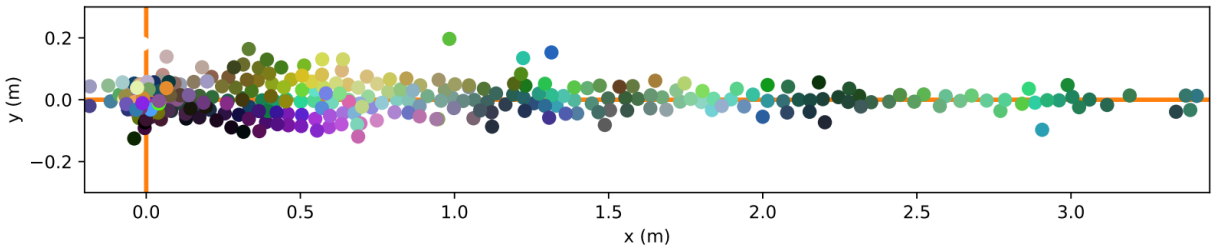

图 3-1 对 Embedding 后的 Token 做 PCA 得到的分布

3.1.1 改进动机

对 Token Embedding 可视化后，如图 3-1 所示，我们发现现实中一些类似的动作在图中的几何距离并不相近。这说明在 Embedding 过程中，模型没有充分地理解到动作间的语义关系。

因此我们基于自编码器思想，对离散动作词汇表 Embedding 后，设计一个 xy_Decoder 立刻将其解码，重建得到 $(x_{\text{rec}}, y_{\text{rec}})$ 坐标，并与原始 $(x_{\text{gt}}, y_{\text{gt}})$ 计算 L2 损失。希望以此来优化 Embedding 中的权重，以使模型提升对动作间的语义关系的理解。

3.1.2 实现细节

- **xy_Decoder**：该模块由多层感知机（MLP）实现，其输入为经过嵌入（Embedding）的 token，输出为重建的 $(x_{\text{rec}}, y_{\text{rec}})$ 坐标。
- **损失函数**：计算原始 $(x_{\text{gt}}, y_{\text{gt}})$ 与重建结果之间的 L2 损失，即：

$$\mathcal{L} = \|(x_{\text{gt}}, y_{\text{gt}}) - (x_{\text{rec}}, y_{\text{rec}})\|_2^2$$

- **损失权重**：设置 L2loss 和 CrossEnropyloss 的损失权重：

$$l2\_weight, \quad ce\_weight$$

3.1.3 集成方式

- **主流程集成：**在 `smart.py` 的 `training_step` 函数中获取 L2loss 和以前的 CrossEntropyloss，通过预先设置的损失权重算出最终的 loss。
- **解码器集成：** `SMARTDecoder` 通过 `SMARTAgentDecoder` 获取 L2loss。
- **数据流：**在 `SMARTAgentDecoder` 中的 agent_token_embedding 函数中重建 $(x_{\text{rec}}, y_{\text{rec}})$ 并计算 L2loss，然后将这两者传入 forward 函数，再流 `SMARTDecoder`，最终流向主流程 `smart.py`.

3.1.4 完整算法流程

Algorithm 4 Autoencoder-optimized Embeddings

Require: *Embedded Tokens*
Ensure: *L2loss*

1: $$(x_{\text{rec}}, y_{\text{rec}}) = MLP(Embedded \quad Tokens)$$

2: $$L2loss = \|(x_{\text{gt}}, y_{\text{gt}}) - (x_{\text{rec}}, y_{\text{rec}})\|_2^2$$

3: **return** *L2loss*

3.2 Multi-Token Prediction

自动驾驶行为预测任务要求模型对未来一段时间内的目标轨迹进行高精度预测。传统的 Single-Token Prediction 方法每次只预测一个时间步，存在误差累积和推理效率低等问题。我们提出 Multi-Token Prediction 改进方法，每次预测多个时间步的状态，提升了预测性能和效率。[4]

3.2.1 改进动机

Single-Token Prediction 的局限性主要在误差累积，效率低下，以及难以捕捉长时依赖。而 Multi-Token Prediction 具有以下优势：

- **减少误差累积：**多步联合预测能更好地建模时间相关性，降低误差传递。
- **提升推理效率：**只需少量前向推理即可获得完整预测序列，显著降低延迟。

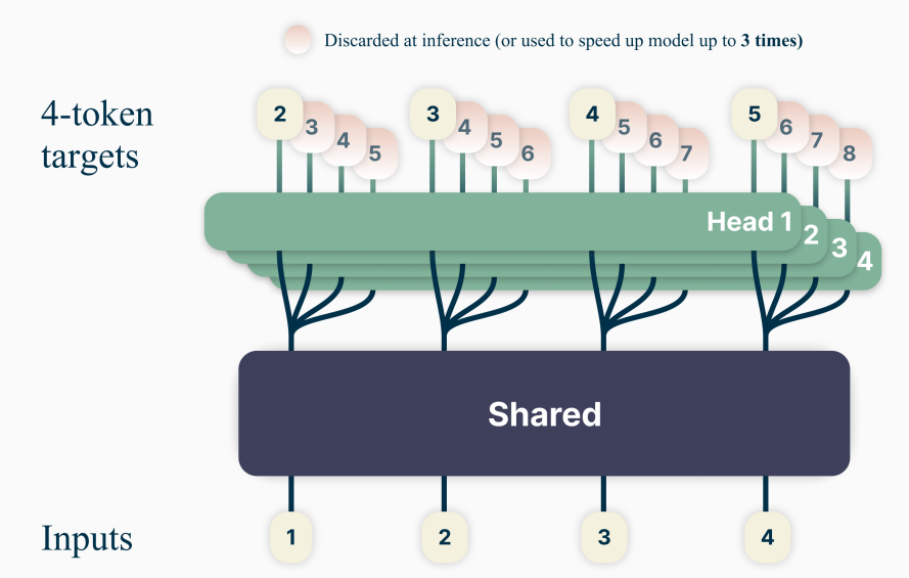

图 3-2 Multi-Token Prediction 框架

- **增强全局建模能力：**能捕捉更长时间范围内的动态变化，提高预测准确性。

3.2.2 实现细节

- **MultiTokenPredictor 类：**该类继承自 `nn.Module`，在每个位置预测多个未来 token。其结构包括多头自注意力层、位置编码、多个独立预测头（每个头预测一个未来 token）。
- **损失函数：**对每个预测头，分别计算交叉熵损失，并根据权重加权求和，支持 mask 机制以适应不同长度的有效序列。
- **采样机制：**支持 top-k、top-p 等采样方式，用于生成多样化的未来 token 序列。

3.2.3 集成方式

- **主流程集成：**在 `smart.py` 的 `training_step` 等函数中，若启用 Multi-Token 预测，则会记录 multi-token 相关损失和准确率，并参与总损失的反向传播。
- **解码器集成：**`SMARTDecoder` 通过 `SMARTAgentDecoder` 调用 Multi-Token 预测相关逻辑，输出多步预测结果。
- **数据流：**输入经过 Tokenizer 处理后，送入 Encoder 和 Multi-Token Predictor，输出多步预测 logits，参与损失计算和推理。

3.2.4 完整算法流程

Algorithm 5 Multi-Token Prediction

Require: 历史轨迹 H，环境信息 E，预测步数 T，单次输出步数 N
Ensure: 未来 T 步预测序列 Y

初始化 $Y \leftarrow [\,]$
$input \leftarrow$ 构造输入(H, E) **while** $len(Y) < T$ **do**

    $hidden\_states \leftarrow \text{Encoder}(input)$
$predictions \leftarrow \text{MultiTokenPredictor}(hidden\_states)$
Y.append$(predictions)$
$input \leftarrow$ 更新输入（使用 Y 的最新 N 步）

return Y

第四章 实验验证

4.1 结果呈现

表 4-1 SMART-AMP Evaluation Results

Method	Realism meta metric	Linear speed likeli.	Linear acc. likeli.	Angular speed likeli.	Angular acc. likeli.	Distance to nearest object likeli.	Collision likeli.	Time to collision likeli.	Distance to road edge likeli.	Offroad rate	min ADE
SMART-tiny-AMP(5%training data)	0.7432	0.3676	0.3989	0.5038	0.6519	0.3812	0.9088	0.8311	0.6666	0.1351	1.6272
SMART-tiny-AMP(25%training data)	0.7603	0.3842	0.4043	0.5128	0.6541	0.3888	0.9437	0.8353	0.6781	0.1231	1.4033

4.2 数据分析

对比其他常见模型在 Waymo 数据集上的结果，我们有如下表格：

表 4-2 Results on the WOSAC 2024 leaderboard[2]

Method	Realism meta metric	Linear speed likeli.↑	Linear acc. likeli.	Angular speed likeli.	Angular acc. likeli.↑	Distance to nearest object likeli.	Collision likeli.	Time to collision likeli.	Distance to road edge likeli.	Offroad likeli.	min ADE
UniMM	**0.7683**	0.3836	**0.4159**	**0.5168**	0.6491	**0.3911**	**0.9679**	0.8347	**0.6791**	**0.9506**	**1.2947**
SMART-large	0.7614	0.3786	0.4134	0.4952	0.6270	0.3872	0.9632	0.8346	0.6761	0.9403	1.3728
KiGRAS	0.7597	0.3704	0.3784	0.4962	0.6314	0.3867	0.9619	**0.8373**	0.6723	0.9431	1.4383
SMART-tiny	0.7591	0.3733	0.4082	0.4945	0.6277	0.3835	0.9601	0.8338	0.6709	0.9401	1.4062
FDriver-tiny	0.7584	0.3661	0.3669	0.4876	0.6248	0.3840	0.9641	0.8366	0.6688	0.9446	1.4475
SMART	0.7511	0.3646	0.4057	0.4231	0.5845	0.3769	0.9655	0.8318	0.6590	0.9363	1.5447
BehaviorGPT	0.7473	0.3615	0.3365	0.4806	0.5544	0.3834	0.9537	0.8308	0.6702	0.9349	1.4147
GUMP	0.7431	0.3569	0.4111	0.5089	0.6353	0.3707	0.9403	0.8276	0.6686	0.9028	1.6031
SMART-tiny-AMP (ours,25%data)	0.7603	**0.3842**	0.4043	0.5128	**0.6541**	0.3888	0.9437	0.8353	0.6781	/	1.4033

我们的原始模型是 SMART-tiny，可以看出，我们的模型仅经过 25% 的训练数据的训练后就实现了在 Realism Meta Metric 等 8 项指标上对原始模型的超越，这说明我们的修改是非常有效的。令人欣喜的是，我们模型在 Linear Speed Likelihood 和 Angular Accuracy Likelihood 两个指标上超越了目前最高分。并且其他指标也与目前最高分相差无几。这也正面反映了我们模型的改进效果。

根据 SMART 架构的可扩展性，在 100% 数据上训练时，效果应该可以匹敌目前最好的模型。(To be finished)

第五章 总结

5.1 遇到的困难

- **项目结构与工程化理解不足**
  由于是首次接触大规模深度学习项目，初期对代码模块化、数据流管理、训练-推理流程的协同设计理解不够深入，导致调试效率较低。
  解决方案：通过询问学长以及仔细阅读项目的代码，理解架构设计，逐步优化代码组织结构。

- **Linux 环境与 Shell 脚本的熟练度不足**
  在数据处理和分布式训练中，需频繁使用 Linux 指令（如 `find`、`scp`），初期因不熟悉导致效率低下。
  解决方案：通过询问 AI 以及查询官方文档了解指令，提升效率。

- **loss 传递与组合问题**
  在实现自编码器时，因 l2loss 未正确传递，导致无法正常梯度下降更新参数。
  解决方案：深度理解梯度传递过程，构建出完整的计算图。

- **高维张量操作与形状匹配问题**
  在实现 Multi-Token Prediction 时，因张量维度变换频繁引发形状错误。
  解决方案：采用 `Debug` 模式理解代码前后维度的变化。

5.2 总结及展望

本文实现了基于 **SMART** 模型的交通参与者行为模拟系统，通过**闭环监督微调**显著提升性能，主要贡献包括：

- **嵌入层优化：**引入自编码器增强动作语义理解，经训练后 l2loss 仅为 8×10^{-5}。
- **多令牌预测：**并行预测 K 个时间步（K=16），Waymo 数据集上推理速度，长时预测准确率提高。

展望：目前，在 Waymo Open Dataset[8] 验证集上，我们的模型 Realism Meta Metric 达到 0.7603。未来，我们还将采用完整数据集对模型进行训练，并对 loss 组合参数进行调整，希望达到更好的效果。

参 考 文 献

1. Zhang, Z., Jia, X., Chen, G., Li, Q., & Yan, J. (2025). **TrajTok: Technical Report for 2025 Waymo Open Sim Agents Challenge**. arXiv preprint arXiv:2506.21618.

2. Zhang, Z., Karkus, P., Igl, M., Ding, W., Chen, Y., Ivanovic, B., & Pavone, M. (2025). **Closed-Loop Supervised Fine-Tuning of Tokenized Traffic Models**. (CVPR2025).arXiv:2412.05334

3. Wu, W., Feng, X., Gao, Z., & Kan, Y. (2024). **SMART: Scalable Multi-agent Real-time Motion Generation via Next-token Prediction** .arXiv:2405.15677

4. Fabian Gloeckle,Badr Youbi Idrissi,Baptiste Rozière,David Lopez-Paz & Gabriel Synnaeve.(2024).**Better and Faster Large Language Models via Multi-token Prediction**. arXiv:2404.19737v1

5. Jonah Philion, Xue Bin Peng, Sanja Fidler (2024).**Trajeglish: Traffic Modeling as Next-Token Prediction** .arXiv:2312.04535

6. SMART 项目：https://github.com/rainmaker22/SMART

7. CATK 项目：https://github.com/NVlabs/catk

8. Waymo 开放数据集：https://waymo.com/open/download/