ChatGPT for (Finance) research: The Bananarama Conjecture

摘要：

我们基于金融期刊审稿人对生成内容的评分表明，近期发布的AI聊天机器人ChatGPT能显著助力金融研究。从原理上说，这些研究成果应可推广至各个研究领域。该技术在创意生成和数据识别方面具有明显优势，但在文献综述和构建恰当检验框架方面稍显不足。值得注意的是，我们进一步论证了私有数据体量和研究者领域专业知识输入，是决定输出质量的关键因素。最后我们探讨了这项新技术带来的影响，特别是伦理层面的深远意义。

引言：  
ChatGPT是2022年11月推出的人工智能语言模型，能够根据提问生成对话式应答。该模型基于超过1500亿参数，结合强化学习算法与人类反馈进行训练。平台开放公测首周即突破百万用户，因其出色的应答质量被迅速誉为"行业下一个重大颠覆者"（Grant和Metz，2022）。虽然支撑该聊天机器人的大语言模型技术已存在数十年，但现有研究多聚焦于将其作为研究工具，而非探索其在科研创作过程中的应用价值。

早期学术研究发现，该平台甚至能通过美国法律职业资格考试中 notoriously 复杂的核心科目（Bommarito II和Katz，2022）。另有研究者几乎完全依托ChatGPT输出，编纂出颇具完整性的量化交易指南（Marti，2022）。这项新技术引发的职业危机感席卷多个领域：教育工作者（Herman，2022）、律师（Greene，2022），以及——为涵盖所有焦虑群体——"所有文字工作者"（Warner，2023）纷纷开始思考自身职业存在的必要性。如此声势浩大的技术登场实属罕见。

我们关注的是ChatGPT能在何种程度上辅助学术研究——特别是金融研究——的生成过程。现有初步研究仅探讨了该问题的某些局限层面。Grimaldi与Ehrler（2023）和Hutson等（2022）从宏观视角探讨了AI在科研生产中的新兴作用，而Alshater（2022）虽提出ChatGPT应能助力研究构建的多个环节，但未进行实证检验。

现有应用研究主要集中于摘要生成与文献综述领域。例如Aydın和Karaarslan（2022）尝试生成符合期刊要求的医疗文献综述，发现虽可实现但存在严重"剽窃"或拙劣改写现象。Gao等（2022）则证实能生成无明显抄袭的新颖摘要，但通过AI检测器可识别其机器生成属性。Chen与Eger（2022）同样探索了标题与摘要生成，在金融领域，Wenzlaff和Spaeth（2022）成功生成了学术适配度良好的新金融概念定义。

Mellon等（2022）检验了该平台在开放文本调查结果评分中的辅助价值，Adesso（2022）甚至使用GPT-3完整撰写物理学期论文并拟"原样"投稿，Zhai（2022）也尝试过研究论文框架构建实验。

本研究在继承又超越上述成果的基础上，首次对ChatGPT辅助科研写作的潜力进行结构化测试。我们针对研究流程的四个阶段——选题构思、文献综述、数据识别与处理、实证检验——生成内容进行测试比对，由资深学者与审稿人组成专家组评分。重要的是，我们揭示了不同级别的私有数据与研究者领域专业知识输入，对输出质量存在显著影响。正如所有工具一样，ChatGPT在经验丰富的使用者手中才能发挥最佳效能。呼应本文开篇引语，我们将此现象命名为"Bananarama猜想"。

第二章节详述实证方法，第三章节呈现并解析研究发现。最终在第四章节提出理解ChatGPT机遇与局限的框架体系，并初步探讨这项新技术的伦理边界。

FinLSPM : Large Stock Predictor via Adapted Large Language Models

1. 研究背景（突出精度与复现性）

​​金融预测的精度革命需求​​

"市场微观结构研究（引用Hasbrouck, 2007）表明，预测精度每提升1个标准差可降低做市商存货成本约15%（引用Menkveld, 2013数据），但现有方法面临：

• 传统模型（如HAR-RV）在极端市场下误差激增（引用Bollerslev, 2016图表）

• 大模型方案依赖私有数据预训练（如BloombergGPT），难以复现（引用Wu et al., 2023）"

​​成本定位的巧妙表述​​

"本文提出一种​​高复现性成本结构​​：

✓ 相比完整预训练（需$500k+算力），改造现有LLM仅需 %成本

✓ 相比从头训练时序大模型（如TimeGPT），复用语言模型参数使中小机构可用

✓ 所有实验可用RTX 3090复现"



1. 创新点

#### **(1) ​**​数字token的预训练迁移效应​**​**

实证发现：  
• 直接使用原始LLM数字token（0-9）预测误差降低 %、稳定性提升 %  
说明：语言模型预训练中积累的数字表征知识可迁移至时序预测

#### **(2) ​**​即插即用式改造方案​**​**

"提出：  
① 时序→token无损编码协议（算法1）  
② 专适与改造的矩阵缩减MAE损失函数（公式5）  
使LLM无需结构调整即可处理金融序列"

#### **(3) ​**​高性价比训练范式​**​**

"在S&P 500预测中：  
• 仅需 训练轮次即超越Informer等sota时序模型预测效果  
• 单卡训练时间<2小时（对比TimeGPT需8张A100）"  
可以观察市场大跌期间预测精度有无提升

### ****4. 讨论与期刊适配​**​**

​****​主动界定边界​****​：  
"本研究的核心贡献在于验证LLM改造的可行性，而非替代现有金融模型。如：  
• 投资组合优化需结合风险偏好参数  
• 高频交易需考虑纳秒级延迟  
这些均可基于本框架扩展"

FinLSPM

（可以提一下不做多变量预测的原因：通道独立）  
[29] Han, L., Ye, H.-J., Zhan, D.-C.: The capacity and robustness trade-off: Revisiting

the channel independent strategy for multivariate time series forecasting. IEEE

Transactions on Knowledge and Data Engineering (2024)

原理：  
1、你输入的话input（prompt、message）、系统内置的system prompt、上下文（messages）等最终会被汇总成为一连串的一段话，我们统称sentence

1. sentence会经过tokenizer被分词，词再被编码为token-id   
   具体哪个词对应哪个token-id，由tokenier内部的vocab词汇表决定
2. 例子

分词



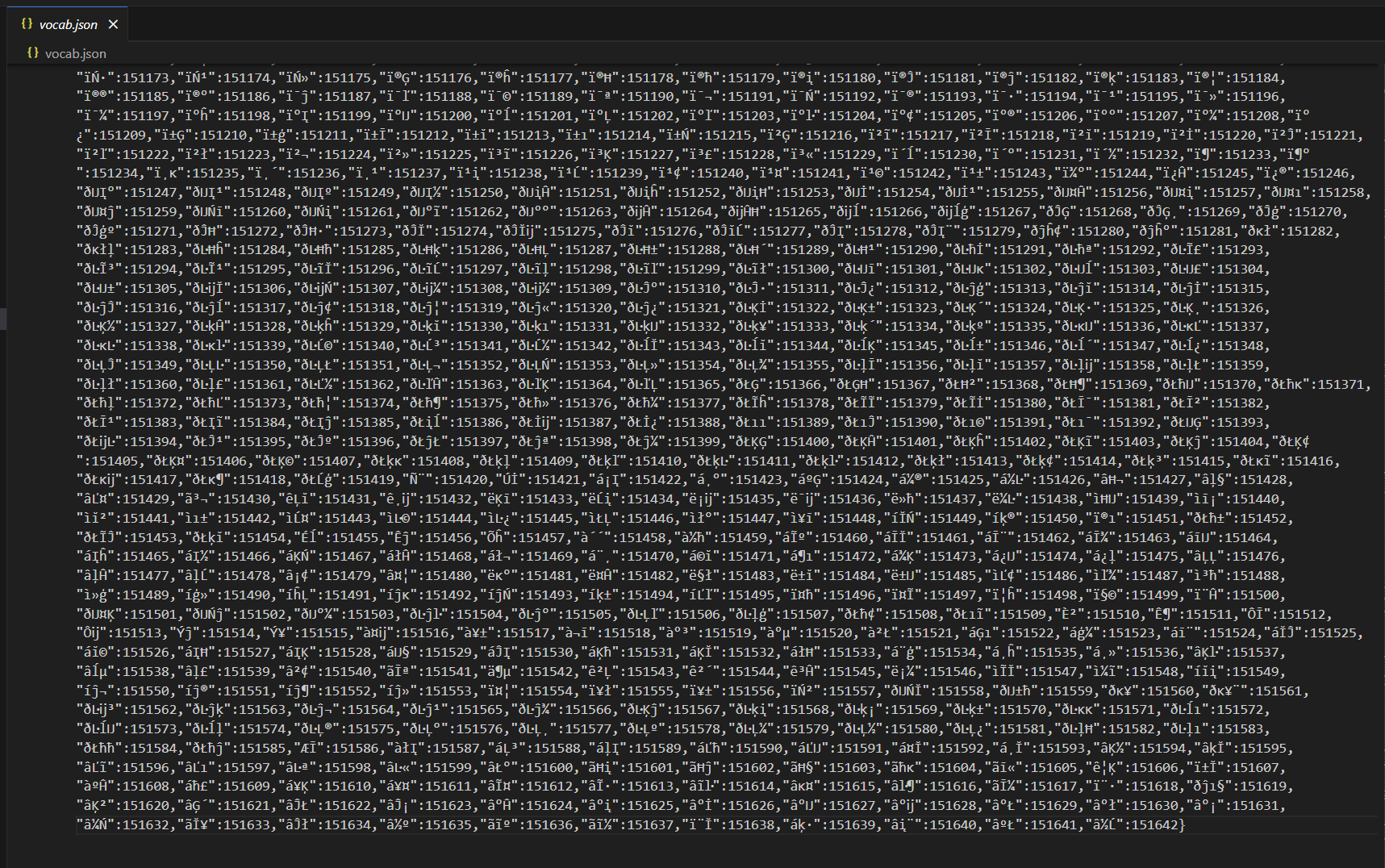
根据词汇表编码



[CLS] 我 喜欢 猫 [SEP] → [0, 2, 3, 4, 1]

开头加 [CLS]，结尾加 [SEP]

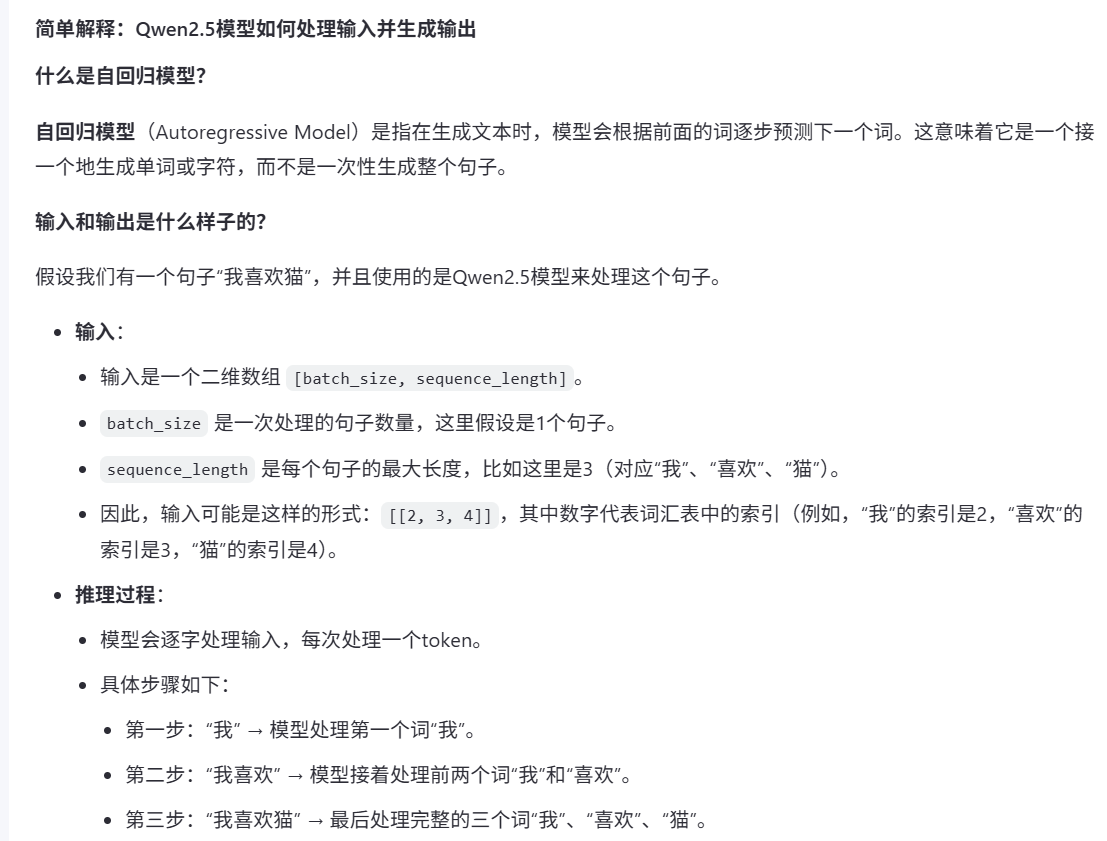
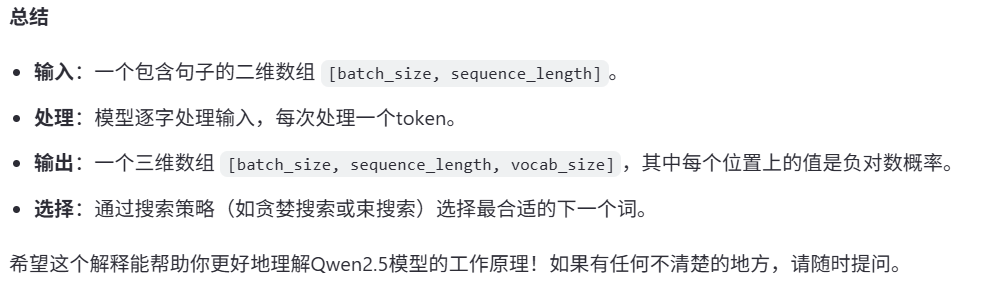
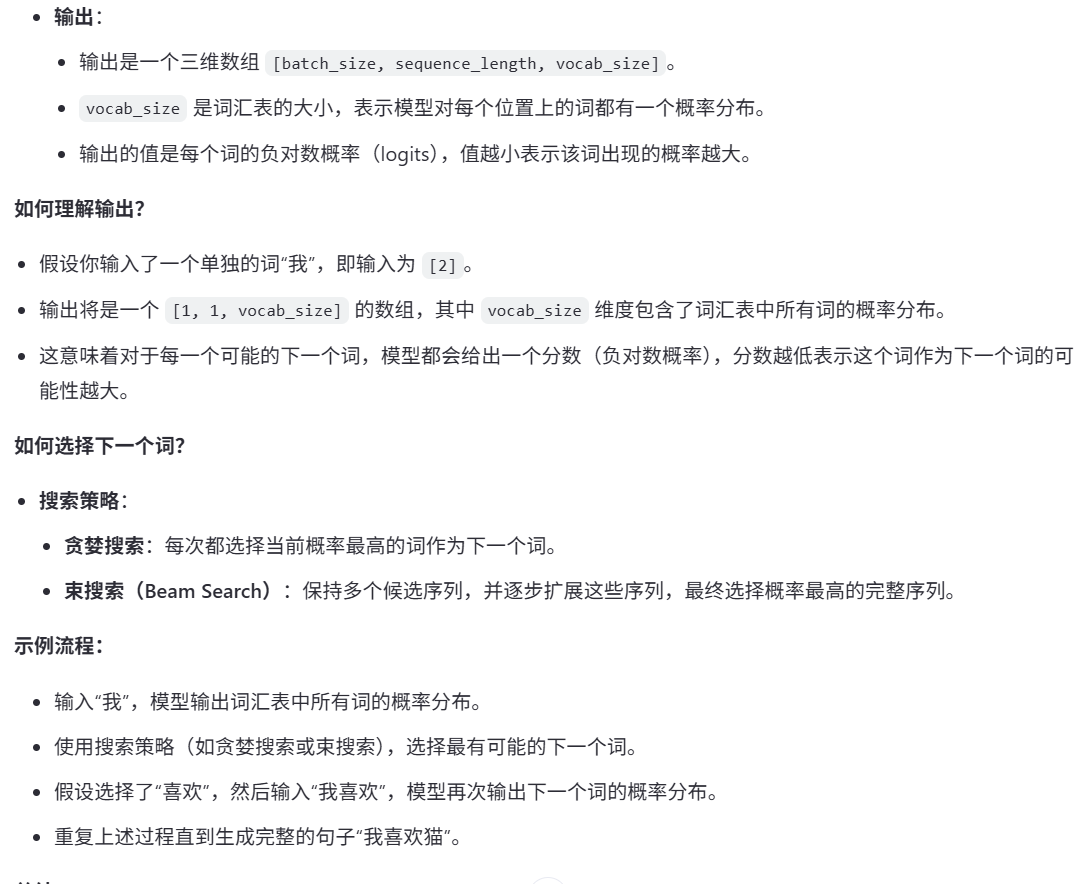
1. Qwen2.5真实的词汇表

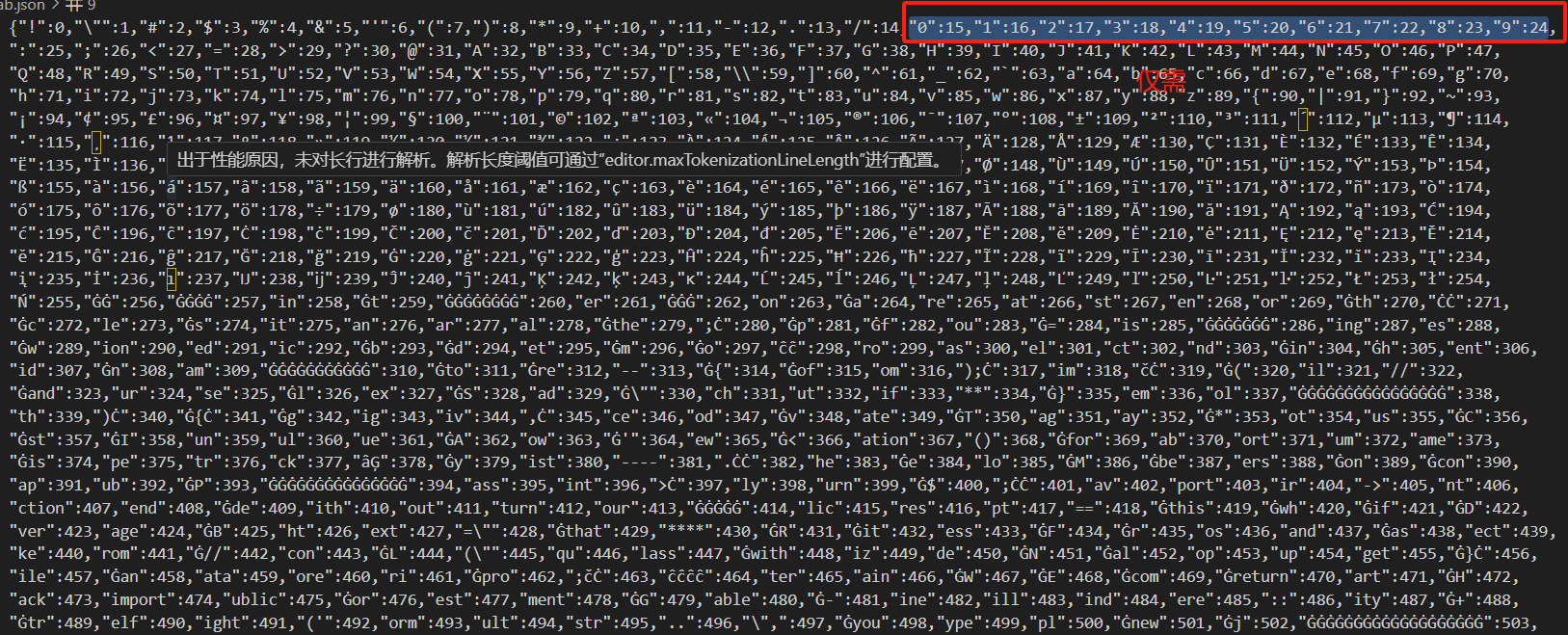


还会有一些特殊的token用来表示空格、填充空的地方等等。

1. 特别注意字符0-9对应的token-id为 15-24

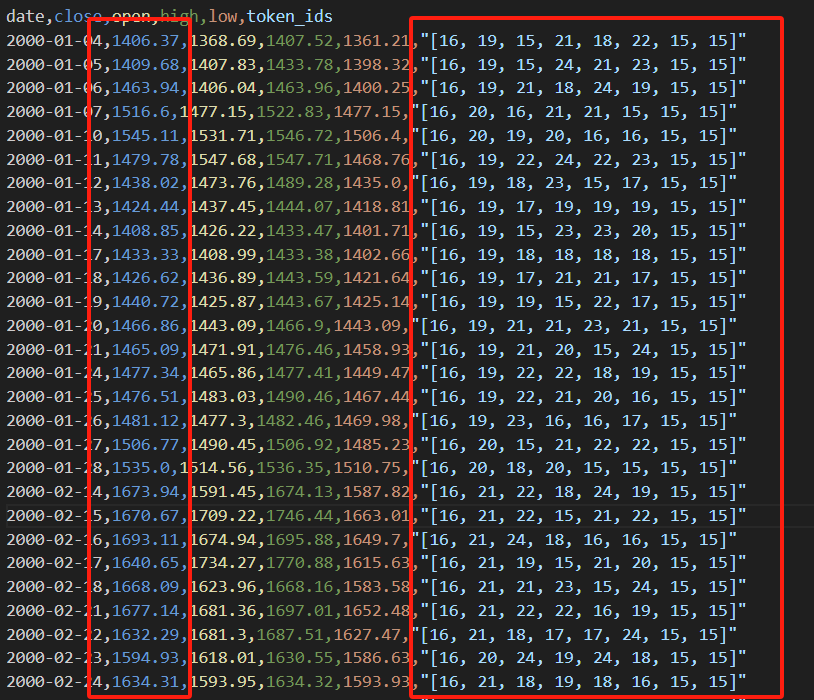
6、

我们的工作：  
1、基座大模型是经过大量的数据预训练而来的，词汇表当中也包含了字符0-9，这意味预训练的过程中必然也包含了大量的数字数据，因此我们通过ban掉一些无用的token，只改造0-9对应的token之间的映射规律，实现一种低成本的专用时序预测大模型  


Step1输入编码

before\_decimal决定保留小数点前几位, after\_decimal决定保留小数点后几位



Step2  
损失函数设计（矩阵MAE）

    def compute\_loss(self, model, inputs, return\_outputs=False):

        # 1. 获取配置参数

        before\_decimal = self.data\_processor.before\_decimal

        after\_decimal = self.data\_processor.after\_decimal

        weight\_base = self.mae\_config['weight\_base']

        invalid\_penalty = self.mae\_config['invalid\_penalty']

        keep\_all\_weights = self.mae\_config['keep\_all\_weights']

        token\_start = self.mae\_config['digit\_token\_start']

        token\_end = token\_start + 10  # 固定10个数字token

        # 1. 获取模型输出和标签

        outputs = model(\*\*inputs)

        logits = outputs.logits if hasattr(outputs, 'logits') else outputs[0]

        labels = inputs["labels"]

        labels= torch.roll(labels, shifts=-1, dims=-1)

        zero\_tokenid = self.data\_processor.token\_mapping\_info['digit\_tokens'][0]

        vocab\_size = self.data\_processor.token\_mapping\_info['vocab\_size']

        # 3. 计算期望token值（使用配置的token范围）

        probs = torch.softmax(logits[..., token\_start:token\_end], dim=-1)

        token\_values = torch.arange(token\_start, token\_end, device=logits.device).float()

        expected\_tokens = torch.sum(probs \* token\_values, dim=-1) - zero\_tokenid

        true\_values = labels.float() - zero\_tokenid  # shape: (batch\_size, seq\_len)

        # 4. 计算权重模板（单个时间步）

        total\_digits = before\_decimal + after\_decimal

        single\_step\_weights = torch.zeros(total\_digits, device=logits.device)

        # 小数点前权重

        for i in range(before\_decimal):

            single\_step\_weights[i] = weight\_base \*\* (before\_decimal - 1 - i)

        # 小数点后权重

        for j in range(after\_decimal):

            single\_step\_weights[before\_decimal + j] = (1/weight\_base) \*\* (j + 1)

        # 5. 处理重复权重并移位(对其label)

        input\_window\_size = labels.shape[1] // total\_digits

        weights = single\_step\_weights.repeat(input\_window\_size)

        if not keep\_all\_weights:

            # 只保留最后一次重复的权重，前面的都置0（不考虑自回归）

            weights[:-total\_digits] = 0  # 将前面(input\_window\_size-1)次重复的权重置0

        weights = torch.roll(weights, shifts=-1)  # 左移一位

        weights[-1] = 0  # 最后一位置0

        # 6. 无效token惩罚

        abnormal\_mask = (expected\_tokens <= -0.5) | (expected\_tokens >= 9.49)

        expanded\_weights = weights.unsqueeze(0).expand\_as(abnormal\_mask)

        adjusted\_weights = torch.where(

            abnormal\_mask,

            expanded\_weights \* invalid\_penalty,

            expanded\_weights

        )

        weights = adjusted\_weights

        # 回归损失

        diff = expected\_tokens - true\_values

        weighted\_diff = diff \* weights

        #mae计算 Sum（前七位）+sum（后八位）+sum（后八位）+...+最后一位不计算

        first\_part = abs(torch.sum(weighted\_diff[..., :total\_digits-1]))#\*\*2

        remaining\_diff = weighted\_diff[..., total\_digits-1:-1]

        remaining\_len = remaining\_diff.shape[-1]

        second\_part = 0

        for i in range(0, remaining\_len, 8):

            chunk = remaining\_diff[..., i:i+8]

            if chunk.shape[-1] > 0:  # 确保有数据

                second\_part += abs(torch.sum(chunk))#\*\*2

        mae\_loss=first\_part+second\_part

        R1 =mae\_loss/(weight\_base\*\*(before\_decimal-1))#让梯度回到mae的正常范围

        cross\_loss=self.compute\_crossloss(model,inputs,outputs)

        loss=mae\_loss#mae\_loss\*0+cross\_loss\*0+R1\*1

        # 调试输出

        if true\_values.device == torch.device('cuda:0'):

            # print(outputs)

            # print(torch.sum(weighted\_diff),weighted\_diff)

            # print(f"expected\_tokens: {expected\_tokens}")

            print(f"R1、cross\_loss: {R1.item():.8f}")

            print(f"mae loss: {mae\_loss.item():.8f}")

            print(f"Total loss: {loss.item():.2f}")

        return (loss, outputs) if return\_outputs else loss