DeepSeek-TS+: MLA-Mamba及GRPO用于多序列预测统一框架

QuantML QuantML 2025年02月09日 21:16 上海

我们在之前的文章中介绍过如何使用DeepSeek的核心结构MoE以及MLA用于选股：

QuantML-Qlib开发版 | MoE混合专家系统用于提升Transformer表现【附代码】

QuantML-Qlib重磅更新：DeepSeek核心模型结构用于选股

除了这两个结构之外，DeepSeek还采用了GRPO的强化学习方法用于更新策略。

本文介绍利用状态空间增强的多头潜在注意力（MLA）和组相对策略优化（GRPO）进行自适应预测

引言

DeepSeek的技术给我留下了深刻的印象——其高效的多头潜在注意力（MLA）和组相对策略优化（GRPO）技术启发了我，将它们应用于多产品时间序列预测。在我们的方法中，我们将MLA扩展为MLA-Mamba，允许潜在特征通过具有非线性激活的状态空间模型随时间动态演变。这赋予了我们的模型一种自适应记忆能力，能够像销售团队在市场激增期间调整策略一样适应趋势。与此同时，GRPO引入了一种智能决策过程，通过将预测与基线进行比较，持续优化预测结果，类似于管理者实时调整预测。这种动态调整有助于我们的模型有效应对销售模式的突然变化。

我们将这种方法与经典的ARMA模型和标准的基于GRU的网络进行了比较。虽然ARMA能够处理线性趋势，GRU能够捕捉时间依赖性，但我们的DeepSeek-TS框架旨在建模复杂的产品间关系并适应非线性动态，从而实现更准确、更稳健的预测。

在接下来的部分中，我们将详细剖析扩展的MLA（MLA-Mamba）和GRPO框架的技术细节，并展示它们的协同作用如何增强多产品时间序列预测。

多头潜在注意力（MLA）

DeepSeek中MLA的核心思想是将键（Key）和值（Value）压缩到低维潜在空间。这使得模型在推理过程中能够存储更小的KV缓存。该过程可以分解为几个关键的数学步骤：

键和值的潜在压缩

考虑一个输入标记，用向量 h_t 表示，其维度为 d。在标准的Transformer中，该向量通过学习矩阵被投影到查询（Query）、键（Key）和值（Value）空间。然而，在DeepSeek的MLA中，我们首先将 h_t 压缩为一个专门用于键和值的低维潜在向量。这一过程可以通过以下公式表示：

$$c_{KV,t} = W_{DKV} h_t$$

其中：

- $c_{KV,t}$ 是压缩后的潜在向量；
- W_{DKV} 是降维投影矩阵；
- d_c 是压缩维度，且 $d_c < d$。

这种低秩近似类似于推荐系统中使用的矩阵分解技术，其中大矩阵通过两个较小矩阵的乘积来近似，以捕捉最重要的特征。在我们的案例中，W_{DKV} 学习捕捉 h_t 中对计算注意力至关重要的方面。

从潜在代码重构键和值

一旦我们得到了 $c_{KV,t}$，就需要通过上投影来重构用于注意力机制的键和值向量。这一过程可以通过以下公式实现：

$$k_{C,t} = W_{UK}c_{KV,t}v_{C,t} = W_{UV}c_{KV,t}$$

其中：

- $k_{C,t}$ 和 $v_{C,t}$ 分别是重构后的键和值向量；
- W_{UK} 和 W_{UV} 是上投影矩阵，其维度为 $\mathbb{R}^{d_h n_h \times d_c}$（其中 d_h 是每个头的维度，n_h 是头的数量）。

关键在于，在推理过程中，我们无需为每个标记缓存完整的键和值向量（这将需要存储每个标记的 $d_h n_h$ 个元素），而只需缓存压缩后的潜在向量 $c_{KV,t}$，其中每个标记仅包含 d_c 个元素。当 d_c 远小于 $d_h n_h$ 时，这种减少是显著的。

查询压缩（可选，用于提高训练效率）

除了键和值之外，DeepSeek还对查询进行了类似的低秩压缩，以减少训练过程中的激活内存。该过程与上述类似：

$$c_{Q,t} = W_{DQ}h_t q_{C,t} = W_{UQ}c_{Q,t}$$

其中：

- $c_{Q,t}$ 是压缩后的查询向量；
- W_{DQ} 和 W_{UQ} 是查询的降维和上投影矩阵；
- d_c 是查询压缩维度。

虽然压缩查询并不会减少KV缓存（因为查询是即时计算的），但它有助于在训练过程中减少整体激活内存。

效率提升：一个示例

假设你有一个Transformer层，其中每个标记的原始键值维度为1024（即 $d_h n_h = 1024$）。使用MLA时，如果你选择压缩维度 $d_c = 128$，那么每个标记缓存的数据量将从1024个元素减少到128个元素——减少了8倍。当处理长序列或大规模部署模型时，这一减少是显著的。

此外，在推理过程中，如果上投影矩阵 W_{UK} 和 W_{UV} 可以吸收进其他权重矩阵（例如 W_Q 或 W_O），那么你可能根本无需显式计算或存储键和值。这将带来更大的效率提升。

组相对策略优化（GRPO）

虽然MLA专注于高效的注意力机制，但DeepSeek还引入了一种新颖的优化方法，即组相对策略优化（GRPO），用于更新模型的决策策略。这种方法基于强化学习原理，旨在平衡探索与利用，同时确保策略更新的稳定性。

策略优化基础

在强化学习中，策略 $\pi(a|s;\theta)$ 定义了在状态 s 下采取行动 a 的概率，参数为 θ。目标是最大化预期累积奖励：

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right]$$

其中：

- R_t 是时间 t 的奖励；
- $\gamma \in [0,1]$ 是折扣因子，用于确定未来奖励的重要性。

使用组进行相对策略优化

GRPO引入了一种新思想，即通过将新策略的输出与旧策略（固定版本）的输出组进行比较来评估新策略。关键在于，通过将新策略的输出与旧策略的输出进行比较，可以更稳健地衡量某些行动的优势。

设：

- π_{old} 是用于生成相同查询 q 的输出组的旧策略；
- π_{new} 是当前可更新的策略。

比率：

$$r_t = \frac{\pi_{\text{new}}(a_t|q)}{\pi_{\text{old}}(a_t|q)}$$

衡量了新策略相对于旧策略的偏离程度。为了稳定训练，通常会将该比率限制在 $1-\epsilon$ 和 $1+\epsilon$ 之间，其中 ϵ 是一个小的超参数。这种限制确保策略在单次更新中不会发生过大的变化。

优势估计

优势函数 A_t 衡量了行动 a_t 相对于平均表现的好坏程度：

$$A_t = R_t - V(s_t)$$

其中 $V(s_t)$ 是一个基线价值函数，表示从状态 s_t 开始的预期奖励。在GRPO中，优势用于加权更新，确保那些导致高于平均水平奖励的行动得到强化。

策略梯度更新公式为：

$$\nabla_\theta J(\theta) \approx \mathbb{E}[\nabla_\theta \log \pi(a_t|s_t;\theta) A_t]$$

该更新规则表明，我们应该调整 θ 的方向，以增加具有正优势的行动的概率，同时减少具有负优势的行动的概率。

GRPO的新颖贡献

GRPO在先前的方法（如近端策略优化（PPO）和直接策略优化（DPO））的基础上进行了扩展，但引入了一种新机制，通过比较旧策略的输出组与新策略的输出来进行更新。这种“组相对”比较允许更稳定和可靠的更新。策略更新不仅考虑了新策略的输出，还考虑了它们相对于旧策略提供的稳定基线的表现。

关键公式（概念性）：

$$L(\theta) = \min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t)$$

其中：

$$r_t = \frac{\pi_{\text{new}}(a_t|s_t;\theta)}{\pi_{\text{old}}(a_t|s_t)}$$

clip函数限制了比率 r_t，目标是最大化 $L(\theta)$。该公式确保，如果新策略的偏离过大，更新将受到限制，从而防止可能导致训练不稳定的过度变化。

GRPO的实际应用示例

以我论文中的一个例子为例。假设我们的强化学习代理的任务是从给定的查询中选择最佳模型或模型组合。该查询来自我们的MLA过程生成的潜在表示 z_t。假设旧策略 π_{old} 为某个特定查询分配了一个行动概率分布（例如选择XGBoost、LightGBM、DNN或混合模型），并为“选择DNN”这一行动赋予了0.25的概率。与此同时，新策略 π_{new} 可能为“选择DNN”这一行动赋予了0.35的概率。我们随后计算这些概率的比率。如果 ϵ 设为0.2，我们将比率 r_t 限制在区间 [0.8, 1.2] 内，以确保学习过程中的稳定性。

$$r_t = \frac{0.35}{0.25} = 1.4$$

如果 ϵ 设为0.2，那么我们将 r_t 限制在区间 [0.8, 1.2] 内。假设计算出的“选择DNN”这一行动的优势 A_t 为+0.5（表明在这种情况下选择DNN是有益的）。

策略梯度更新将使用限制后的比率：

$$\text{Update term} = \min(1.4 \times 0.5, 1.2 \times 0.5) = \min(0.7, 0.6) = 0.6$$

这种受控的更新有助于确保策略仅逐渐向新的行动概率倾斜，基于该行动相对于组的表现有多好。

将MLA和GRPO整合到统一框架中

让我们总结一下DeepSeek的突破性成果——MLA和GRPO——并将它们整合到一个自适应模型中。目标是构建一个系统，该系统不仅使用高效的低秩注意力来处理长序列，还利用强化学习智能地选择或混合模型。

1. **输入编码和潜在压缩**：每个输入标记 h_t 首先由编码器处理。编码器使用以下公式将 h_t 压缩为潜在表示：

$$c_{KV,t} = W_{DKV} h_t$$

将原始维度 d 降低到较小的潜在空间维度 d_c。

2. **键和值的重构**：通过以下公式重构用于注意力的键和值：

$$k_{C,t} = W_{UK} c_{KV,t} v_{C,t} = W_{UV} c_{KV,t}$$

这种重构确保在保持KV缓存较小的同时保留了必要的上下文信息。

3. **查询压缩（可选）**：

$$c_{Q,t} = W_{DQ} h_t q_{C,t} = W_{UQ} c_{Q,t}$$

4. **注意力计算**：使用压缩后的查询、键和值计算多头注意力。每个头应用通常的注意力公式，但降低的维度使得该过程更加高效。

5. **通过GRPO进行策略决策**：模型随后使用强化学习模块来选择最佳行动——无论是选择一个模型还是混合多个模型。强化学习策略 $\pi(a|s;\theta)$ 接收状态（包括来自MLA模块的潜在特征和额外的统计摘要），并输出一个行动。GRPO通过比较新策略的输出与早期固定策略的输出组来更新策略。计算优势，并对更新进行限制以确保稳定性。

$$L(\theta) = \min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t)$$

其中：

$$r_t = \frac{\pi_{\text{new}}(a_t|s_t;\theta)}{\pi_{\text{old}}(a_t|s_t)}$$

目标是最大化 $L(\theta)$。该公式确保，如果新策略的偏离过大，更新将受到限制，从而防止可能导致训练不稳定的过度变化。

将MLA和GRPO应用于多产品时间序列预测

在前一节中，我们讨论了MLA和GRPO在DeepSeek中的有效协同工作，它们构成了DeepSeek的核心技术。这促使我探索这些技术是否可以应用于LLM之外的其他机器学习领域。经过彻底的理论分析，我确认它们可以扩展到新的领域。因此，我提出了一个统一框架，将MLA和GRPO结合起来用于多产品时间序列预测。具体来说，使用一个包含“日期”、“产品1销售额”、“产品2销售额”……“产品5销售额”等列的DataFrame，我们的目标是预测每种产品在未来10天内的平均销售额。我们的方法将“Mamba风格”的状态空间建模与潜在注意力相结合，并利用基于强化学习的策略优化（GRPO）动态调整预测。

以下是数学基础、算法细节以及一个实际示例。

问题设置和数据描述

假设销售数据是一个多变量时间序列 (x_t)，其中每个 x_t 表示时间 t 时 p 种产品（此处 $p = 5$）的销售额。目标是预测每种产品在未来10天内的平均销售额。挑战在于时间序列可能存在内部相关性和滞后效应。例如，

产品1在第 t 天的销售额可能不仅取决于其自身的过去销售额，还可能受到产品2或产品3过去几天销售额的影响。

将MLA扩展到状态空间（Mamba）框架

为了捕捉时间序列预测中固有的时间动态，我建议将潜在压缩步骤与通过非线性激活增强的状态空间更新相结合。在这个框架中，我假设压缩后的潜在向量随时间演变如下：

$$c_{KV,t+1} = \text{ReLU}(Mc_{KV,t} + n(x_t))$$

其中：

- $M \in \mathbb{R}^{d_c \times d_c}$ 是一个转移矩阵，用于建模潜在状态动态；
- $n(x_t)$ 是一个函数，将当前输入 x_t（例如时间 t 的销售数据）映射到潜在空间中的一个修正项；
- 整个更新应用了ReLU激活，引入非线性并确保更新后的潜在状态是非负的。

类似地，可以对查询压缩应用类似的更新：

$$c_{Q,t+1} = M'c_{Q,t} + n'(c_t)$$

其中 $M' \in \mathbb{R}^{d'_c \times d_c}$ 的定义类似。

解释：这种状态空间更新“记住”了过去的潜在状态 $c_{KV,t}$，并使用新信息 x_t 进行调整。通过对整个和 $Mc_{KV,t} + n(x_t)$ 应用ReLU激活，模型捕捉了历史状态和新输入之间复杂的非线性相互作用。非线性有助于建模复杂的时序模式，同时确保潜在表示保持非负。这种方法类似于RNN或LSTM更新其隐藏状态的方式。

引入多头注意力

在获得动态潜在状态 $c_{KV,t}$ 和 $c_{Q,t}$ 后，我将它们投影到多头注意力的键、值和查询中。假设将潜在空间划分为 h 个头。对于头 i：

$$Q_i = c_{Q,t}W_{Q,i}K_i = c_{KV,t}W_{K,i}V_i = c_{KV,t}W_{V,i}$$

其中 $W_{Q,i}$、$W_{K,i}$、$W_{V,i}$ 是权重矩阵，d_h 是每个头的维度。

每个头的注意力计算如下：

$$\text{Attention}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_h}}\right)V_i$$

所有头的输出随后被拼接并使用输出矩阵 W_O 进行投影：

$$\text{MLA}(h_t) = \text{Concat}(\text{Attention}_1, \ldots, \text{Attention}_h)W_O$$

这种多头机制允许模型捕捉销售数据中不同方面的时序关系。例如，一个头可能学习趋势成分，另一个头可能专注于季节性等。

扩展GRPO以适应多时间序列预测

在这个框架中，我首先定义了一个给定输入窗口长度 T 和预测范围 H 天的预测问题。对于每个产品时间序列，目标 y 被计算为未来 H 天内原始销售额的平均值，即 $T+H$。

$$y = \frac{1}{H}\sum_{t=T+1}^{T+H} x_t$$

其中 x_t 表示第 t 天的销售额。

GRPO模型使用两层GRU从归一化的输入序列 $X \in \mathbb{R}^{T\times D}$（其中 D 是产品数量）中提取时间特征。设最终时间步的隐藏状态为 $h_T \in \mathbb{R}^{d_h}$。然后 h_T 通过两个独立的线性投影进行映射：

- 预测分支使用权重矩阵 W_f 计算预测值：

$$g = W_f h_T$$

其中 $g \in \mathbb{R}^D$ 是每个产品的预测平均销售额向量。

- 策略分支通过另一个线性映射 W_p 计算标量值 p（策略值）：

$$p = W_p h_T$$

其中 $p \in \mathbb{R}$。

GRPO启发的损失的核心思想是根据一个“优势”信号调整预测，该信号衡量预测误差相对于一个常数基线 b（此处选择 $b = 0.5$）。

具体来说，优势定义为：

$$A = \text{mean}(y - g)$$

其中均值是在产品维度上取的。策略值与基线之间的比率 r 计算为：

$$r = \frac{p}{b}$$

为了确保训练过程中的稳定性并避免过大的策略更新，使用了裁剪机制。设：

$$r_{\text{clipped}} = \text{clip}(r, 1-\epsilon, 1+\epsilon)$$

其中 ϵ 是一个小值（例如，0.1）。GRPO启发的策略损失随后被公式化为：

$$L_{\text{policy}} = -\mathbb{E}[\min(r \cdot A, r_{\text{clipped}} \cdot A)]$$

如果新策略（即预测）相对于基线没有足够改进，该损失将惩罚模型。总损失函数是预测损失和策略损失的组合：

$$L = L_{\text{forecast}} + \lambda L_{\text{policy}}$$

其中：

$$L_{\text{forecast}} = 0.5\text{MSE}(y, g) + 0.5\text{MAE}(y, g)$$

λ 是一个控制策略损失权重的超参数。

这种方法嵌入了一个时间外验证方案中：时间序列数据按时间顺序划分，确保仅使用过去的数据进行训练，未来数据用于验证——从而避免数据泄露。在验证过程中，归一化的预测 y 使用存储的均值和标准差还原回原始规模，并计算平均绝对百分比误差（MAPE）：

$$\text{MAPE} = \frac{100}{N} \sum_{i=1}^{N} \frac{|y_i - g_i|}{y_i + \epsilon}$$

其中 ϵ 是一个小常数，用于避免除以零。

在这里，GRPO方法用于多时间序列预测，使用GRU编码器提取时间依赖特征，并生成预测和策略值。预测使用组合的均方误差（MSE）/平均绝对误差（MAE）进行评估，而策略分支使用裁剪的优势机制提供额外的梯度信号，最终导致对预测范围内平均销售额的更稳健预测。

代码实验：结果概述

在这个实验中，我们的目标是预测每种产品在未来5天内的平均销售额。使用AR(1)过程结合产品特定的噪声和偏移生成合成销售数据，生成一个现实的600天数据集。目标被定义为预测范围内的平均销售额。

数据被归一化，然后按时间顺序划分（前80%用于训练，剩余20%用于验证），以确保时间外评估且没有任何泄露。

我比较了四种预测方法。首先，我使用了一个受GRPO启发的模型，该模型结合了扩展的MLA模块、GRU编码器和额外的策略分支。其次，我实现了一个受GRPO启发的预测模型，该模型采用了扩展的MLA（Mamba风格）机制，通过ReLU激活的非线性状态空间方法更新潜在状态。第三，应用了一个简单的基于GRU的预测模型（没有GRPO修改）。最后，使用ARIMA(1,0,1)模型以滚动预测的方式在原始数据上进行经典ARMA方法的预测。

这个代码实验为后续的性能评估奠定了基础，我们将比较这四种方法的平均绝对百分比误差（MAPE）。

代码如下：

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.arima.model import ARIMA

# Load data
df = pd.read_csv("sales_data.csv")

# Optional: visualize the generated sales data
df.plot(x="date", y=["product1_sales", "product2_sales", "product3_sales", "product4_s
```

```
plt.title("Simulated Sales Data for 5 Products")
plt.show()

# ------------------------------
# Data Preparation for Time Series Forecasting with Normalization
# ------------------------------
class SalesDataset(Dataset):
    def __init__(self, df, input_window=30, forecast_horizon=5):
        """
        df: DataFrame with columns: date, product1_sales, ..., product5_sales
        input_window: number of days used as input
        forecast_horizon: number of days to forecast; target = avg sales over these da
        """
        self.input_window = input_window
        self.forecast_horizon = forecast_horizon

        df['date'] = pd.to_datetime(df['date'])
        df.sort_values('date', inplace=True)
        self.df = df.reset_index(drop=True)
        # Use only the sales columns (all except 'date')
        data = df.drop(columns=['date']).values.astype(np.float32)

        # Compute normalization parameters on the entire dataset
        self.mean = data.mean(axis=0)
        self.std = data.std(axis=0) + 1e-6  # avoid division by zero

        # Normalize the data
        self.data = (data - self.mean) / self.std
        self.n_samples = len(self.data) - input_window - forecast_horizon + 1

    def __len__(self):
        return self.n_samples

    def __getitem__(self, idx):
        # Input: sales for input_window days (normalized)
        x = self.data[idx: idx + self.input_window]
        # Target: average sales over the next forecast_horizon days (normalized)
        y = np.mean(self.data[idx + self.input_window: idx + self.input_window + self.
        return x, y

def prepare_dataloaders(df, input_window=30, forecast_horizon=5, batch_size=32, train_
    dataset = SalesDataset(df, input_window, forecast_horizon)
    n_total = len(dataset)
    n_train = int(n_total * train_ratio)
    # Chronological (out-of-time) split: first n_train samples for training, remaining
    train_dataset = torch.utils.data.Subset(dataset, list(range(n_train)))
    val_dataset = torch.utils.data.Subset(dataset, list(range(n_train, n_total)))
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)
```

```
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
    return train_loader, val_loader

train_loader, val_loader = prepare_dataloaders(df, input_window=30, forecast_horizon=5

# ------------------------------
# Model Components: Simple GRU-based Forecasting Model with GRPO-inspired Framework
# ------------------------------
class ForecastingGRPOModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_products, forecast_horizon, dropout=
        """
        This model forecasts the average sales of the next 'forecast_horizon' days for
        using a GRU encoder. It includes a policy branch to compute a GRPO-inspired lo
        """
        super(ForecastingGRPOModel, self).__init__()
        self.gru = nn.GRU(input_dim, hidden_dim, num_layers=2, batch_first=True, dropo
        self.fc_forecast = nn.Linear(hidden_dim, num_products)
        self.policy_net = nn.Linear(hidden_dim, 1)
        self.lambda_policy = lambda_policy

    def forward(self, x):
        # x: (batch, seq_len, input_dim)
        gru_out, _ = self.gru(x)
        last_hidden = gru_out[:, -1, :]  # (batch, hidden_dim)
        forecast = self.fc_forecast(last_hidden)  # (batch, num_products)
        policy_value = self.policy_net(last_hidden)  # (batch, 1)
        return forecast, policy_value

# ------------------------------
# Training and Validation Functions (Using MAPE as Error Metric)
# ------------------------------
def train_model_full(model, dataloader, optimizer, device, grad_clip=1.0):
    model.train()
    total_loss = 0.0
    for x, y in dataloader:
        x = x.to(device)  # (batch, seq_len, input_dim)
        y = y.to(device)  # (batch, num_products)
        optimizer.zero_grad()
        forecast, policy_value = model(x)
        # Forecast loss: use a combination of MSE and MAE
        loss_mse = F.mse_loss(forecast, y)
        loss_mae = F.l1_loss(forecast, y)
        loss_forecast = 0.5 * loss_mse + 0.5 * loss_mae

        # GRPO-inspired policy loss:
        # Compute advantage as the mean error over products for each sample.
        advantage = (y - forecast).mean(dim=1, keepdim=True)
        baseline = 0.5  # chosen constant baseline
```

```
            r_t = policy_value / baseline
            epsilon = 0.1
            r_t_clipped = torch.clamp(r_t, 1 - epsilon, 1 + epsilon)
            policy_loss = -torch.min(r_t * advantage, r_t_clipped * advantage).mean()

            loss = loss_forecast + model.lambda_policy * policy_loss
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), grad_clip)
            optimizer.step()
            total_loss += loss.item() * x.size(0)
        return total_loss / len(dataloader.dataset)

def validate_model(model, dataloader, device, dataset_obj, debug=False):
    model.eval()
    all_preds = []
    all_targets = []
    with torch.no_grad():
        for x, y in dataloader:
            x = x.to(device)
            y = y.to(device)
            forecast, _ = model(x)
            all_preds.append(forecast.cpu().numpy())
            all_targets.append(y.cpu().numpy())
    all_preds = np.concatenate(all_preds, axis=0)
    all_targets = np.concatenate(all_targets, axis=0)
    # Invert normalization: forecast_orig = forecast * std + mean
    mean = dataset_obj.mean
    std = dataset_obj.std
    all_preds_orig = all_preds * std + mean
    all_targets_orig = all_targets * std + mean
    if debug:
        print("Prediction range:", np.min(all_preds_orig), np.max(all_preds_orig))
        print("Target range:", np.min(all_targets_orig), np.max(all_targets_orig))
    mape = np.mean(np.abs((all_targets_orig - all_preds_orig) / (all_targets_orig + 1e
    return mape

# ------------------------------
# ARMA Forecasting for Comparison
# ------------------------------
def arma_forecast(series, forecast_horizon):
    """
    Fits an ARIMA(1,0,1) model on the provided series and forecasts forecast_horizon s
    Returns the average forecast.
    """
    try:
        arma_model = ARIMA(series, order=(1, 0, 1))
        arma_fit = arma_model.fit(disp=0)
        forecast = arma_fit.forecast(steps=forecast_horizon)
        return np.mean(forecast)
```

```
        except Exception as e:
            return series[-1]

def evaluate_arma(df, input_window=30, forecast_horizon=5, train_ratio=0.8):
    """
    For each product (column), use a rolling ARMA forecast over the validation period
    Returns a dictionary of MAPE values per product and the overall average MAPE.
    """
    n = len(df)
    train_end = int(n * train_ratio)
    products = [col for col in df.columns if col != "date"]
    mape_dict = {}
    all_mapes = []

    # Rolling forecast: start from i = (train_end - input_window) to (n - input_window
    for prod in products:
        preds = []
        actuals = []
        for i in range(train_end - input_window, n - input_window - forecast_horizon +
            series = df[prod].values[:i + input_window]
            pred = arma_forecast(series, forecast_horizon)
            preds.append(pred)
            actual = np.mean(df[prod].values[i + input_window: i + input_window + fore
            actuals.append(actual)
        preds = np.array(preds)
        actuals = np.array(actuals)
        prod_mape = np.mean(np.abs((actuals - preds) / (actuals + 1e-6))) * 100
        mape_dict[prod] = prod_mape
        all_mapes.append(prod_mape)
    overall_mape = np.mean(all_mapes)
    return mape_dict, overall_mape

# ------------------------------
# Main Training
# ------------------------------
def main():

    df = pd.read_csv("sales_data.csv")

    # Prepare out-of-time (chronological) dataloaders (first 80% for training, remaini
    train_loader, val_loader = prepare_dataloaders(df, input_window=30, forecast_horiz

    # For validation inversion, we need access to the dataset normalization parameters
    dataset_obj = SalesDataset(df, input_window=30, forecast_horizon=5)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Define model parameters
```

```
input_dim = train_loader.dataset[0][0].shape[-1]  # e.g., 5 products
hidden_dim = 256      # Hidden dimension for GRU
num_products = input_dim  # Predict average sales for each product
forecast_horizon = 10  # Note: This parameter is used in the models, even though t
lambda_policy = 0.06   # Weight for GRPO-inspired policy loss

# ------------------------------
# GRPO-inspired Forecasting Model (Existing)
# ------------------------------
model = ForecastingGRPOModel(input_dim, hidden_dim, num_products, forecast_horizon
model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.9)
n_epochs = 22

print("Training GRPO-inspired Forecasting Model...")
for epoch in range(n_epochs):
    train_loss = train_model_full(model, train_loader, optimizer, device, grad_cli
    mape = validate_model(model, val_loader, device, dataset_obj, debug=True)
    scheduler.step()
    print(f"Epoch {epoch+1}/{n_epochs} - GRPO Model Train Loss: {train_loss:.4f},

# ------------------------------
# ARMA Evaluation for Comparison
# ------------------------------
print("\nEvaluating ARMA Forecasting on raw data...")
arma_mapes, overall_arma_mape = evaluate_arma(df, input_window=30, forecast_horizo
print("ARMA MAPE per product:", arma_mapes)
print("Overall ARMA MAPE:", overall_arma_mape, "%")

# ------------------------------
# Simple GRU Forecasting Model for Comparison
# ------------------------------
class SimpleGRUForecastingModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_products, forecast_horizon, drop
        super(SimpleGRUForecastingModel, self).__init__()
        self.gru = nn.GRU(input_dim, hidden_dim, num_layers=2, batch_first=True, d
        self.fc_forecast = nn.Linear(hidden_dim, num_products)

    def forward(self, x):
        gru_out, _ = self.gru(x)
        last_hidden = gru_out[:, -1, :]
        forecast = self.fc_forecast(last_hidden)
        return forecast

def train_simple_model(model, dataloader, optimizer, device, grad_clip=1.0):
    model.train()
```

```
    total_loss = 0.0
    for x, y in dataloader:
        x = x.to(device)
        y = y.to(device)
        optimizer.zero_grad()
        forecast = model(x)
        loss_mse = F.mse_loss(forecast, y)
        loss_mae = F.l1_loss(forecast, y)
        loss = 0.5 * loss_mse + 0.5 * loss_mae
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), grad_clip)
        optimizer.step()
        total_loss += loss.item() * x.size(0)
    return total_loss / len(dataloader.dataset)

def validate_simple_model(model, dataloader, device, dataset_obj, debug=False):
    model.eval()
    all_preds = []
    all_targets = []
    with torch.no_grad():
        for x, y in dataloader:
            x = x.to(device)
            y = y.to(device)
            forecast = model(x)
            all_preds.append(forecast.cpu().numpy())
            all_targets.append(y.cpu().numpy())
    all_preds = np.concatenate(all_preds, axis=0)
    all_targets = np.concatenate(all_targets, axis=0)
    mean = dataset_obj.mean
    std = dataset_obj.std
    all_preds_orig = all_preds * std + mean
    all_targets_orig = all_targets * std + mean
    if debug:
        print("Simple GRU Prediction range:", np.min(all_preds_orig), np.max(all_p
        print("Simple GRU Target range:", np.min(all_targets_orig), np.max(all_tar
    mape = np.mean(np.abs((all_targets_orig - all_preds_orig) / (all_targets_orig
    return mape

print("\nTraining Simple GRU Forecasting Model for Comparison...")
simple_model = SimpleGRUForecastingModel(input_dim, hidden_dim, num_products, fore
simple_model.to(device)
optimizer_simple = torch.optim.Adam(simple_model.parameters(), lr=0.0003)
scheduler_simple = torch.optim.lr_scheduler.StepLR(optimizer_simple, step_size=10,
n_epochs_simple = 22
for epoch in range(n_epochs_simple):
    train_loss_simple = train_simple_model(simple_model, train_loader, optimizer_s
    simple_mape = validate_simple_model(simple_model, val_loader, device, dataset_
    scheduler_simple.step()
    print(f"Epoch {epoch+1}/{n_epochs_simple} - Simple GRU Train Loss: {train_loss
```

```
# ------------------------------
# New Method: GRPO-inspired Forecasting with Extended MLA (Mamba-style) Mechanism
# ------------------------------
class ForecastingGRPOMLAModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_products, forecast_horizon, drop
        """
        This model extends the GRPO-inspired forecasting approach by incorporating
        extended MLA (Mamba-style) mechanism. The latent state is updated in a sta
        manner with a nonlinear activation applied to the entire update.
        """
        super(ForecastingGRPOMLAModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.lambda_policy = lambda_policy
        self.dropout = nn.Dropout(dropout)
        # Map the input to the latent space.
        self.input_transform = nn.Linear(input_dim, hidden_dim)
        # State-space transition matrix (M)
        self.M = nn.Linear(hidden_dim, hidden_dim, bias=False)
        # Nonlinear activation function for the complete state update.
        self.activation = nn.ReLU()
        self.fc_forecast = nn.Linear(hidden_dim, num_products)
        self.policy_net = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        # x: (batch, seq_len, input_dim)
        batch_size, seq_len, _ = x.size()
        # Initialize latent state as zeros.
        h = torch.zeros(batch_size, self.hidden_dim, device=x.device)
        # Iteratively update latent state with a state-space update.
        for t in range(seq_len):
            x_t = x[:, t, :]  # (batch, input_dim)
            # Compute the correction without activation first.
            correction = self.input_transform(x_t)
            # Update latent state using the ReLU activation applied to the entire
            h = self.activation(self.M(h) + correction)
            h = self.dropout(h)
        forecast = self.fc_forecast(h)
        policy_value = self.policy_net(h)
        return forecast, policy_value

print("\nTraining GRPO-inspired Forecasting Model with Extended MLA (Mamba-style)
model_extended = ForecastingGRPOMLAModel(input_dim, hidden_dim, num_products, fore
model_extended.to(device)
optimizer_extended = torch.optim.Adam(model_extended.parameters(), lr=0.0003)
scheduler_extended = torch.optim.lr_scheduler.StepLR(optimizer_extended, step_size
n_epochs_extended = 22
for epoch in range(n_epochs_extended):
```

```
        train_loss_extended = train_model_full(model_extended, train_loader, optimizer
        mape_extended = validate_model(model_extended, val_loader, device, dataset_obj
        scheduler_extended.step()
        print(f"Epoch {epoch+1}/{n_epochs_extended} - Extended MLA Model Train Loss: {

if __name__ == "__main__":
    main()
```

评估GRPO启发的预测模型

GRPO启发的模型整合了扩展的MLA模块与GRU，并新增了一个策略分支，在经过22个训练周期后，其平均绝对百分比误差（MAPE）稳步下降并最终稳定在大约21.6%。该模型的预测范围始终与目标范围高度一致，这表明其自适应机制能够有效捕捉底层的销售模式。

与简单GRU模型的对比

相比之下，缺乏GRPO特定改进的简单GRU模型，其MAPE平均值略高，约为22.3%。尽管简单GRU的预测范围也与目标范围相似，但在GRPO模型中观察到的微小改进表明，额外的策略损失和扩展的潜在更新为降低预测误差做出了适度但有意义的贡献。

扩展的MLA（Mamba风格）模型的见解

进一步采用扩展的MLA（Mamba风格）机制的GRPO启发模型，通过在完整的状态更新中应用ReLU激活的非线性状态空间更新，实现了低至20.8%至21.3%的MAPE。这一改进凸显了利用更丰富的潜在表示来捕捉时间序列动态的优势。

经典ARMA方法

最后，ARMA方法显示出显著更高的误差。每个产品的MAPE大致在12.6%到43%之间，整体MAPE约为26.3%。与深度学习方法相比，ARMA在处理复杂的多维销售数据方面效果较差。

总结

在本研究中，我们探索了一种创新的多时间序列预测方法，通过结合GRPO和采用Mamba风格状态空间更新的扩展MLA模块来实现。实验结果表明，这种GRPO启发的模型能够实现比简单GRU模型和经典ARMA方法更低的MAPE。由策略分支和状态更新中的非线性激活驱动的增强潜在表示，似乎能够更有效地捕捉销售数据的复杂动态。

展望未来，将GRPO和扩展MLA框架应用于其他领域具有巨大的潜力。例如，这种方法可以适应于金融时间序列预测，其中捕捉市场趋势的微妙变化至关重要。它也可能有助于医疗保健诊断，通过从多个时间依赖信号预测患者结果，从而实现早期干预。

未来的研究可以专注于通过尝试不同的基线值或裁剪阈值来进一步完善GRPO机制，以及探索扩展的MLA模块如何与其他深度学习架构相结合。此外，引入元学习技术可能使模型能够更好地泛化到不同的领域。总体而言，本研究表明，将强化学习与先进的注意力机制相结合是构建更智能、更具适应性的预测系统的一个有希望的方向。

作者：Shenggang Li

完整代码及数据下载见星球，加入QuantML星球，与700+专业人士一起交流学习：

往期回顾

QuantML-Qlib开发版：

- QuantML-Qlib重磅更新：DeepSeek核心模型结构用于选股
- QuantML-Qlib Factor | 融合TA-Lib100+技术指标，自定义构建AlphaZoo
- QuantML-Qlib Model | 还在使用MSE？试试这些更加适合金融预测的损失函数
- QuantML-Qlib Model | 如何运行日内中高频模型
- QuantML-Qlib Model | 超越GRU，液态神经网络LNN用于股票预测
- QuantML-Qlib Model | 华泰SAM：提升AI量化模型的泛化性能 研报复现
- QuantML-Qlib Model | 华泰AlphaNet模型复现
- QuantML-Qlib Model | 清华大学&华泰证券 在高胜率时交易

- QuantML-Qlib Factor | 高效优雅的因子构建方法：以开源金工切割动量因子为例
- QuantML-Qlib Model | 滚动模型训练
- QuantML-QlibModel | KAN + GRU 时序模型用于股票预测
- QuantML-Qlib开发版 | 蚂蚁&清华 TimeMixer：可分解多尺度融合的时间序列模型用于金融市场预测
- QuantML-Qlib Model | Kansformer：KAN+Transformer时序模型用于股票收益率预测
- QuantML-QlibModel | 使用OPTUNA优化模型超参
- QuantML-QlibDB | Clickhouse 行情存储与读取方案
- QuantML-Qlib LLM | GPT-4o复现因子计算代码
- QuantML-Qlib开发版 | 最新xLSTM用于股票市场预测
- QuantML-Qlib开发版 | 强化学习因子挖掘
- QuantML-Qlib开发版 | 清华大学时序SOTA模型iTransformer用于股票市场预测QuantML-Qlib开发版 | 最新神经网络结构KAN用于因子挖掘
- QuantML-Qlib开发版 | 直接读取pg/mysql/mongodb数据库
- QuantML-Qlib开发版 | MoE混合专家系统用于提升Transformer表现
- QuantML-Qlib开发版 | 一键数据更新
- QuantML-Qlib开发版 | AAAI最佳论文Informer用于金融市场预测
- QuantML-Qlib开发版 | 取代Transformer的下一代神经网络结构Mamba用于金融市场预测
- QuantML-Qlib开发版 | 时序SOTA模型PatchTST用于金融市场预测
- QuantML-Qlib开发版 | 一行代码运行DLinear模型用于股票预测

研报复现：

- 重磅更新！80+量化策略复现（持续更新中）
- BARRA CNE6模型复现
- 研报复现 | QRS择时信号及改进
- 研报复现 | 跳跃因子系列-下
- 研报复现 | 跳跃因子系列-上
- 研报复现 | 锚定反转因子
- 研报复现 | 另类ETF交易策略：日内动量
- 研报复现 | 国盛金工：如何将隔夜涨跌变为有效的选股因子？——基于对知情交易者信息优势的刻画
- 研报复现 | 招商证券：基于鳄鱼线的指数择时及轮动策略
- 研报复现 | 华西金工-股票网络与网络中心度因子研究
- 研报复现 | 基于筹码分布的选股策略
- 研报复现 | 开源金工-高频追涨杀跌因子复现
- 研报复现 | 开源证券 ：形态识别，均线的
- 券商研报因子复现及表现研究

前沿论文代码：

- 端到端基于LLM的增强型交易系统
- 基于分层强化学习的日内风险因子挖掘
- DeepScalper：深度强化学习捕捉日内交易的短暂机会
- TradingAgents：基于多智能体LLM的金融交易框架
- Kaggle - Optiver trading at the close第一名解决方案及部分代码

- 量化交易全攻略：从入门到精通的终极指南
- 普林斯顿&牛津大学 | 大模型在金融领域的应用、前景和挑战
- Style Miner：基于强化学习算法的风格因子构造
- AQR创始人Cliff Asness：市场效率下降假说
- 增强动量策略：动量Transformer模型
- XGBoost 2.0 ： 提升时间序列预测能力
- NIPS 24 | FinCon: 基于LLM的多智能体交易及组合管理框架
- NIPS 24 | CausalStock：基于端到端因果发现的新闻驱动股价预测模型
- JFE | 高效估计买卖价差的模型、实证与应用
- 超越传统网格交易：新型网格交易系统
- JFE | ETF日内套利研究
- NIPS 24 | 超越CVXPY,新型端到端优化器
- 揭秘Jane Street低延迟系统的优化技巧——减少系统抖动
- 南京大学LAMDA-强化学习DRL挖掘逻辑公式型Alpha因子
- 3万个因子，数据挖掘能超越同行审议的因子吗?
- KDD 24 | 基于增强记忆的上下文感知强化学习的高频交易框架
- FinRobot：用于金融领域的大模型AI平台
- KDD 23 | DoubleAdapt: 显著提升各类模型表现的元学习模型
- 市场微观结构教程：深度订单簿预测
- 基于高频和日频因子的端到端直接排序组合构建模型
- BOA 312页报告：Everything you wanted to know about quant
- 深度学习模型DeepLOB用于订单簿价格预测
- What KAN I say? KAN代码全解析
- 取代MLP? MIT全新神经网络结构KAN,3天1.4k star
- WWW'24 | FinReport: 结合新闻语义信息的多因子模型显著提升预测准确性
- WWW'24 | UniTime: 融合文本信息的时间序列预测模型
- WWW'24 | EarnMore: 如何利用强化学习来处理可定制股票池中的投资组合管理问题
- KDD'23 | AlphaMix: 高效专家混合框架（MoE）显著提高上证50选股表现
- IJCAI'23 | StockFormer: RL+Self-Attention优化摆动交易提高股票预测精度
- AAAI-24 | EarnHFT:针对高频交易的分层强化学习（RL）框架
- AAAI-24 | MASTER 结合市场信息的自动特征选择的股票预测模型，25%年化收益
- COLING 2024 | AlphaFin: 结合深度学习及大模型用于股票预测和金融问答，击败现有预测模型