

# 基于 PatchTST 的时间序列预测算法的改进

黄茂欣

## 摘要

随着深度学习的快速发展，Transformer 技术在自然语言处理（NLP）、计算机视觉（CV）和语音识别等多个领域中取得了突破性进展。近期，Transformer 也开始在时间序列预测领域展现出其潜力。本文主要复现了基于 Transformer 的时间序列预测模型——Patchtst 模型的设计与应用。与传统模型相比，Patchtst 模型在处理长时间序列预测时更加高效，特别是在提取局部语义信息和降低计算复杂度方面具有显著优势。通过实验验证，Patchtst 在多个时间序列预测数据集上优于现有基准模型。本文的创新点在于模型在归一化过程中结合了批量归一化和实例归一化两种方法。这种结合的归一化方法不仅提升了模型的性能，还增强了模型对时间序列的全局动态和局部变化的学习能力。实验结果表明，相比原有模型，改进后的 PatchTST 模型在多个时间序列预测任务上取得了略微的提升，证明了其在时间序列分析及预测领域的应用潜力。

**关键词：**时间序列；PatchTST；Transformer；patching

## 1 引言

在深度学习领域，Transformer 技术在自然语言处理（NLP）、计算机视觉（CV）和语音识别等多个应用中取得显著成就，并近期在处理时间序列数据方面也显示出强大的能力。相关领域出现了多种基于 Transformer 的时间序列模型。其中，基于 Transformer 的 patch 模型主要用于图像处理，而本文的 patchtst 模型则专注于时间序列预测。这种 patch 模型的出现，为研究人员提供了一个新视角：将 patch 作为输入用于不同的研究领域。

在长时间序列预测的研究中，大多数模型旨在设计新型机制以降低原有注意力机制的复杂度，从而提高长期预测的性能。这些模型的主要不足在于，它们通常采用点对点的注意力机制，忽视了 patch 的重要性。此外，这些模型在 patch 内部使用伪时间戳来降低复杂度，却没有充分利用 patch 作为输入单元的潜力，也未能揭示其背后的语义重要性。此外，尽管时间序列表示学习具有巨大潜力，但目前尚未得到充分的开发和应用。

## 2 相关工作

基于 Transformer 的模型：“分块”技术显示出了其在处理各类数据模态时的重要性。特别在局部语义信息至关重要的场景下，分块成为了核心技术。例如，在自然语言处理（NLP）中，BERT (Devlin 等人, 2018 年 [1]) 采用子词级别的标记化策略。在计算机视觉（CV）领域，Vision Transformer (ViT) (Dosovitskiy 等人, 2021 年 [2]) 通过将图像分割为 16x16 像

素的块作为输入，开创了新的处理方式。相似地，BEiT (Bao 等人，2022 年 [3]) 和基于掩码的自动编码器 (He 等人，2021 年 [4]) 也采用了分块策略。在语音识别中，Baeckers 等人 [5] (2020 年) 和 Hsu 等人 [6] (2021 年) 的研究同样利用分块来提高信息提取的效率。

对于长时间序列的预测，Transformer 模型已经被广泛探索。LogTrans (Li 等人，2019 年 [7]) 通过卷积自注意力层与 LogSparse 技术减少计算量并提高性能。FEDformer (Zhou 等人，2022 年 [8]) 采用傅立叶增强结构实现了线性复杂度，而 Pyraformer (Liu 等人，2022 年 [9]) 则通过金字塔式注意力模块降低了复杂性。

这些模型的设计大都旨在简化原始注意力机制的复杂度，以提高长距离预测的性能。LogTrans 避免了键与查询间的逐点乘积，但仍依赖单个时间步。而 Autofomer (Wu 等人，2021 年 [10]) 和 Triformer (Cirstea 等人，2022 年 [11]) 分别通过自动化查询和补丁注意力减少了计算负担，后者甚至使用伪时间戳简化了补丁内查询的复杂性。

在时间序列的表示学习方面，自监督学习的重要性日益凸显，因为它能够学习到时间序列中有用的表示形式。Franceschi 等人 [12] (2019 年)、Tonekaboni 等人 [13] (2021 年)、Yang 和 Hong [14] (2022 年)、以及 Yue 等人 [15] (2022 年) 的研究均表明，Transformer 适合作为学习基础模型和表示的理想工具。尽管如此，像 Zerveas 等人 [16] (2021 年) 和 Eldele 等人 [17] (2021 年) 的工作已经证明了其在时间序列模型转移上的应用前景，但其完整潜力仍有待挖掘。

### 3 本文方法

#### 3.1 模型设计

PatchTST 包含两个关键设计：

Patching：在时间序列预测中的应用是理解各时间步骤间数据关联的关键。由于单个时间步骤本身并不具备类似于自然语言中单词的明确语义，因此局部语义信息的提取对于解析时间序列数据的相关性尤为重要。尽管传统方法多依赖于逐点输入或手动特征提取，PatchTST 模型通过将时间步骤聚合为更粗粒度的子序列“补丁”，能够捕获在单点分析中无法揭示的全面语义信息。

通道独立性：在处理多变量时间序列（多通道信号）时，每个 Transformer 模型的输入标记可以由单一通道或多通道数据构成。基于输入标记的设计，已发展出多种变体的 Transformer 架构。其中，“通道混合”策略涉及将所有时间序列特征融合在输入标记中，并在嵌入空间中混合信息。相对地，“通道独立性”策略则保持每个输入标记仅包含单一通道的信息，这一策略在 CNN (Zheng 等，2014 [18]) 和线性模型 (Zeng 等，2022 [19]) 中已证明其有效性，但在基于 Transformer 的模型中，这一领域仍待进一步探索。

#### 3.2 模型结构

模型的整体结构如图 1 所示。

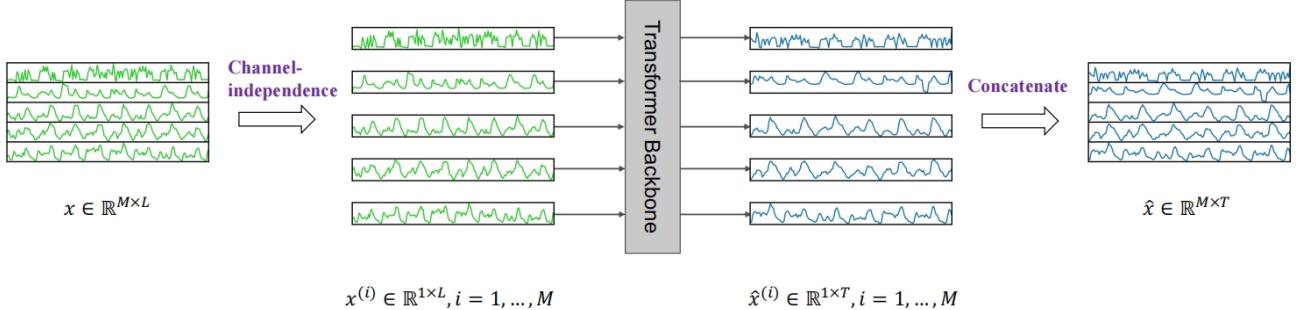


图 1. 模型结构概览

前向过程：模型表示第  $x^i$  个单变量时间序列的长度为  $L$ ，从时间索引 1 开始，即  $x^{(i)} = (x_1^{(i)}, \dots, x_L^{(i)})$ ，其中  $i = 1, \dots, M$ 。输入  $(x_1, \dots, x_L)$  被分割为  $M$  个单变量时间序列  $x^{(i)} \in \mathbb{R}^{1 \times L}$ ，每个都独立地输入到 Transformer 主干网络中根据模型的通道独立设置。然后 Transformer 主干网络将相应地提供预测结果  $\hat{a}^{(i)} = (\hat{x}_{L+1}^{(i)}, \dots, \hat{x}_{L+T}^{(i)}) \in \mathbb{R}^{1 \times T}$

Patching：每个输入的单变量时间序列  $x^{(i)}$  首先被分割成可以重叠或不重叠的补丁。补丁长度表示为  $P$ ，步幅为两个连续补丁之间非重叠区域的距离表示为  $S$ ，那么打补丁的过程将生成一系列的补丁  $x_p^{(i)} \in \mathbb{R}^{P \times N}$ ，其中  $N$  是补丁的数量， $N = (\frac{L-P}{S}) + 2$ 。这里模型在打补丁前向原始序列的末尾重复添加了最后的值  $x_L^{(i)} \in \mathbb{R}$ 。

通过使用补丁，输入令牌的数量可以从  $L$  减少到大约  $L/S$ 。这意味着注意力图的存储器使用和计算复杂度都以  $S$  的系数减少了。因此，在训练内存和 GPU 内存的限制下，补丁设计可以让模型看到更长的历史序列，这可以显著提高预测性能。

Transformer 编码器：模型使用一个普通的 Transformer 编码器，它将观察到的信号映射到潜在的表示空间中。补丁通过一个可训练的线性投影  $W_p \in \mathbb{R}^{D \times P}$  映射到 Transformer 的潜在维度  $D$ ，并且通过一个可学习的加性位置编码  $W_{pos} \in \mathbb{R}^{D \times N}$  来监控补丁的时间顺序： $x_p^{(i)} = W_p x_p^{(i)} + W_{pos}$ ，其中  $x_p^{(i)} \in \mathbb{R}^{D \times N}$  表示将要被输入到 Transformer 编码器的序列。对于每个头  $h = 1, \dots, H$ ，它会将它们转换成查询矩阵  $Q_h^{(i)} = (x_d^{(i)})^T W_Q^h$ ，键矩阵  $K_h^{(i)} = (x_d^{(i)})^T W_K^h$ ， $V_h^{(i)} = (x_d^{(i)})^T W_V^h$ ，其中  $W_Q^h, W_K^h, W_V^h \in \mathbb{R}^{D \times d}$ 。之后，使用了一个缩放乘积来获取注意力输出  $O_h^{(i)} \in \mathbb{R}^{D \times N}$ 。

$$(O_h^{(i)})^T = \text{Attention}(Q_h^{(i)}, K_h^{(i)}, V_h^{(i)}) = \text{Softmax}\left(\frac{Q_h^{(i)}(K_h^{(i)})^T}{\sqrt{d_k}}\right) V_h^{(i)} \quad (1)$$

多头注意力模块还包括批量归一化层和一个带有残差连接的前馈网络，如图 2 所示。之后，它生成一个表示为  $z^{(i)} \in \mathbb{R}^{D \times N}$ 。最终，一个扁平层与线性头用于获取预测结果  $\hat{a}^{(i)} = (\hat{x}_{L+1}^{(i)}, \dots, \hat{x}_{L+T}^{(i)}) \in \mathbb{R}^{1 \times T}$ 。

上面的步骤总体结构如图 2 所示

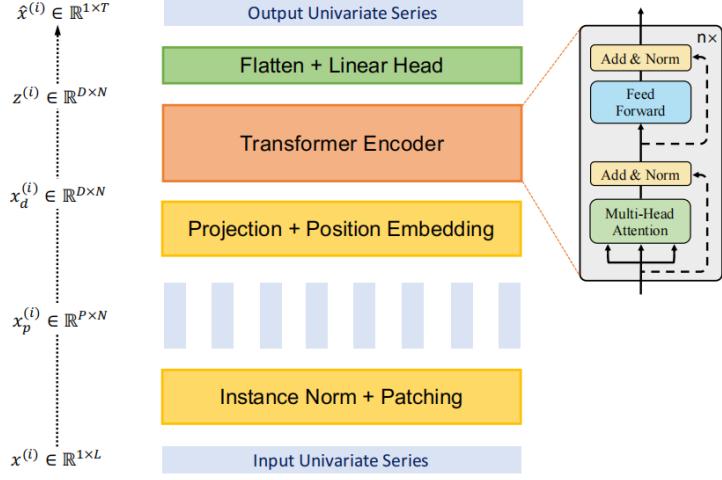


图 2. 模型结构图 2

损失函数：模型选择使用 MSE 损失来衡量预测和实际情况之间的差异。每个通道中的损失被收集并在  $M$  个时间序列上取平均，以得到总的目标损失：

$$\mathcal{L} = \frac{1}{M} \sum_{i=1}^M \|x_{L+1:L+T}^{(i)} - \hat{a}_{L+1:L+T}^{(i)}\|^2 \quad (2)$$

实例归一化：这种技术最近被提出来帮助减轻训练和测试数据之间的分布偏移效应（Ulyanov 等人，2016 [20]；Kim 等人，2022 [21]）。但它简化了每个时间序列实例  $x^{(i)}$  与零均值和单位标准偏差。本质上，模型在 patching 前将每个  $x^{(i)}$  归一化，并在预测输出之前将均值和偏差加回到输出预测中。

### 3.3 自监督表示学习

自监督表示学习已经成为一种流行的方法，用于从未标记的数据中提取高级抽象表示。在本节中，模型应用 PatchTST 来获取有用的多变量时间序列表示。模型将展示所学习的表示可以成功地转移到预测任务中。通过预训练，掩码自编码器已经成功应用于 NLP (Devlin 等人，2019 [1]) 和 CV (He 等人.2021 [4]) 领域。这项技术在概念上很简单：输入序列的一部分被故意隐去，以迫使模型去学习表示剩余内容。

掩码编码器：最近在时间序列分析中被采用，并在分类和回归任务上取得了显著性能 (Zerveas 等人.2021 [16])。模型提出应用多变量时间序列掩码，其中每个输入令牌是一个向量  $x_i$ ，包含第  $i$  个时间点的变量值。然而，这种设置有两个潜在问题：首先，掩码被应用在单个时间步长上。掩码被用来遮住当前时间序列的前一个或后一个时间值，而没有高级理解，这偏离了模型学习重要抽象表示的目标。Zerveas 等人提出了一种混合策略来解决这个问题，在这个策略中，随机掩盖不同大小的时间序列组。

其次，设计用于预测任务的输出层可能会有问题。考虑到表示向量  $z_t \in \mathbb{R}^D$  对应于所有时间步长的映射，将这些向量映射到输出层包含  $M$  个变量的预测水平  $T$  可能需要一个参数矩阵  $W$  大小为  $(L \cdot D) \times (M \cdot T)$ 。如果其中一个或全部这些维度很大，这个矩阵可能特别庞大。当下游训练样本稀缺时，这可能会导致过拟合。

模型提出的 PatchTST 可以克服上述问题。如图 3 所示，模型使用与监督设置相同的 Transformer 编码器。预测头被移除，模型将输入序列分割为规则的非重叠补丁，以确保观察到的补丁不包含掩码补丁的信息。然后模型选择一部分有标记的补丁作为训练和掩码补丁的条件，这些掩码补丁的子集被初始化为零值。模型通过 MSE 损失来训练，以重建被掩码的补丁。

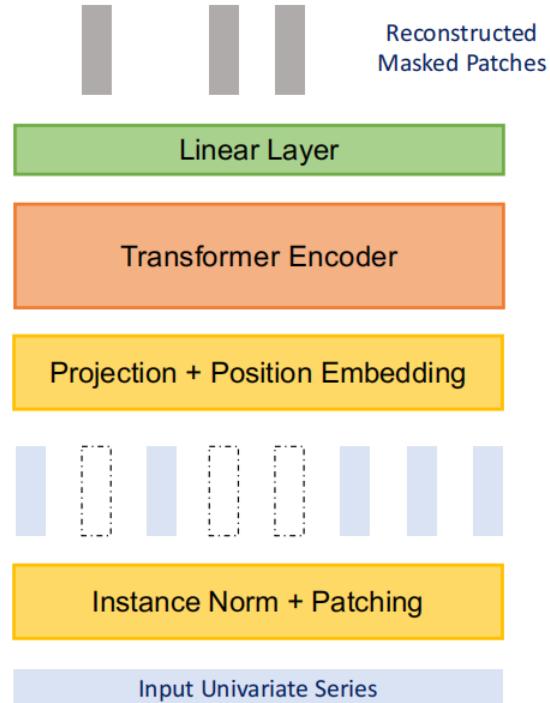


图 3. 模型结构图 3

每种机制都将有其自己的潜在表示，这些表示是通过与下游任务不同的数据进行训练而学习的。这意味着，设计可以允许预训练数据包含更多的时间序列比下游数据，这可能不是由其他方法所能达到的。

## 4 复现细节

### 4.1 与已有开源代码对比

复现主要基于开源的部分代码的基础上，对其归一化方法进行修改，探索其改进的可能性。

### 4.2 实验环境搭建

实验基于 pytorch 实现，运行在单个 NVIDIA GPU 上，核心包的环境如下：

einops==0.4.0

matplotlib==3.7.0

numpy==1.23.5

```
pandas==1.5.3  
patoole==1.12  
reformer-pytorch==1.4.4  
scikit-learn==1.2.2  
scipy==1.10.1  
sktime==0.16.1  
sympy==1.11.1  
tqdm==4.64.1
```

### 4.3 数据集

论文的模型在 8 个流行的数据集上选择了 ETTh1、ETTh2、ETTm1 评估提出的 PatchTST 的性能，这些数据集已广泛用于基准测试，并可公开获取 [10]。在复现时作者选用了 ETTh1、ETTh2 和 ETTm1 数据集进行测试。

### 4.4 基准和实验设置

原论文中模型选择了当前最先进的基于 Transformer 的模型，包括 FEDformer (Zhou 等, 2022 [8])、Autoformer (Wu 等, 2021 [10])、Informer (Zhou 等, 2021 [22])、Pyraformer (Liu 等, 2022 [9])、LogTrans(Li 等, 2019 [7]) 以及最近的非 Transformer-based 模型 DLinear (Zeng 等, 2022 [19]) 作为模型的基线。在本文中，由于时间和计算机性能有限，仅选取了 Transformer 模型作为基准，所有模型都遵循相同的实验设置。

### 4.5 界面分析与使用说明

- checkpoints/: 检查点文件
- data/: 存放数据集
- data-loader.py: 加载数据、数据预处理
- exp/: 训练、测试循环代码
- model/: 模型库，包含了模型的详细结构实现
- results/: 存放训练结果
- utils/: 通用工具，包含了模型的评估指标、时间轴的时间特征处理、指数缩减学习率、提前停止训练策略、数据标准化策略等功能
- run.py: 项目的启动入口，包含程序入口、变量初始化设置等（开始训练和测试）

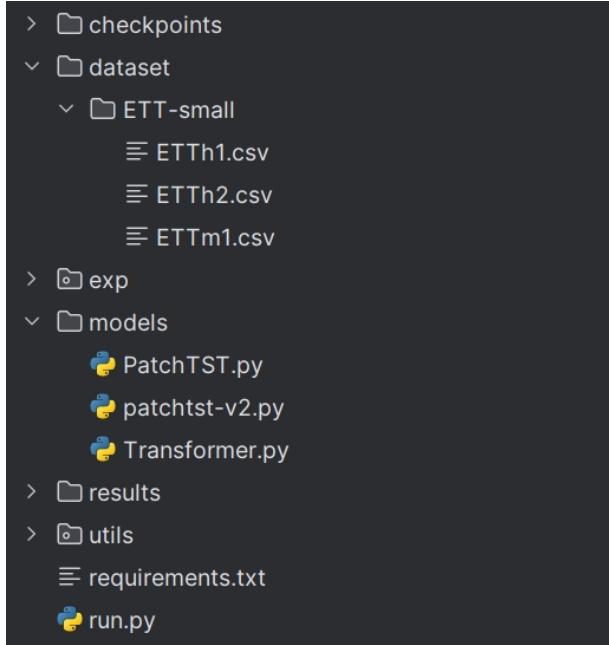


图 4. 界面分析与使用说明

## 4.6 创新点

在归一化过程中加入批量归一化方法：

第一个原因：在训练 ETTh1 数据的样本过程中，作者发现，训练过程较久，于是作者认为可以加入批量归一化方法，可以稳定训练过程，加速收敛。同时，批量归一化方法还可以通过规范化特征的分布，它有助于解决内部协变量偏移问题，使得每个小批次的数据分布更加稳定。第二个原因：批量归一化在批量的维度上操作，适用于捕捉全局统计特征；而实例归一化在单个样本的维度上操作，适用于捕捉局部细节。对于时间序列数据，这可能意味着模型能够同时学习到跨时间点的全局动态以及每个时间点的细微变化。综上，在某些情况下，批量归一化可以提高模型的性能，而实例归一化可以使模型更加健壮。同时使用这两种方法可能会结合两者的优势，达到比单独使用任一方法都要好的效果。

## 5 实验结果分析

先展示实验复现的运行结果，再展示第四节中介绍的改进后的结果。

### 5.1 实验复现结果

将 PatchTST 模型与 Transformer 模型分别在 ETTh1, ETTh2, ETTm1 上的数据集运行，其他实验条件保持不变，结果如下图所示：

图 5. Transformer 在 ETTh1 数据上运行结果

```

ent('--ts_training', type=int, required=True, default=1, help='status')
ent('--model_id', type=str, required=False, default='ETTh2_00_00', help='model id')
ent('--model', type=str, required=False, default='Transformer', help='model name, options: [Autoformer, transformer, timesnet]')
ent('--data', type=str, required=False, default='ETTh2', help='dataset')
ent('--root_path', type=str, default='./dataset/ETT-small', help='root path')
ent('--data_path', type=str, default='ETTh2.csv', help='data file')
ent('--features', type=str, default='M',
     help='forecasting task, options:[M, S, MS]; M:multivariate predict')
ent('--target', type=str, default='OT', help='target feature in S or MS')
ent('--freq', type=str, default='h',
     help='freq for time features encoding, options:[s:secondly, t:minut')
ent('--checkpoints', type=str, default='./checkpoints/', help='location')

sk
ent('--seq_len', type=int, default=96, help='input sequence Length')
ent('--label_len', type=int, default=48, help='start token Length')
ent('--pred_len', type=int, default=96, help='prediction sequence length')
ent('--seasonal_patterns', type=str, default='Monthly', help='subset fc')
ent('--inverse', action='store_true', help='inverse output data', defau

EarlyStopping counter: 2 out of 3
Updating learning rate to 2.5e-05
    iter: 100, epoch: 4 | loss: 0.1934341
    speed: 1.2765s/iter; left time: 2232.5285s
    iter: 200, epoch: 4 | loss: 0.1600685
    speed: 0.0614s/iter; left time: 101.1967s
Epoch: 4 cost time: 50.43700098991394
Epoch: 4, Steps: 264 | Train Loss: 0.1501748 Vali Loss: 1
Test Loss: 2.4604549
EarlyStopping counter: 3 out of 3
Early stopping
>>>>testing :
long_term_forecast_ETTh2_96_96_Transformer_ETTh2_ftM_s19e
6_dm512_nh8_e12_d11_df2048_fc3_ebtimedF_dtTrue_test_0<<<
<<<<<<<<<<<<<<<
test 2785
test shape: (2785, 1, 96, 7) (2785, 1, 96, 7)
test shape: (</>, </>, /) (</>, </>, /)
mse:2.596538305285928, mae:1.2916324138641357
运行时间: 615.05000029102598 秒

```

图 6. Transformer 在 ETTh2 数据上运行结果

图 7. Transformer 在 ETTm1 数据上运行结果

图 8. PATCHTST 在 ETTh1 数据上运行结果

```

(''--task_name'', type=str, required=False, default='Long_term_forecas
    help='task name, options:[long_term_forecast, short_term_forecast,
(''--is_training'', type=int, required=False, default=1, help='status'
(''--model_id'', type=str, required=False, default='ETTh2_96_96', help
(''--model'', type=str, required=False, default='PatchTST',
    help='model name, options: [Autoformer, Transformer, TimesNet]')

(''--data'', type=str, required=False, default='ETTh2', help='dataset
(''--root_path'', type=str, default='./dataset/ETT-small/', help='root
(''--data_path'', type=str, default='ETTh2.csv', help='data file')
(''--features'', type=str, default='M',
    help='forecasting task, options:[M, S, MS]; M:multivariate predict
(''--target'', type=str, default='OT', help='target feature in S or MS
(''--freq'', type=str, default='h',
    help='freq for time features encoding, options:[s:secondly, t:minut
(''--checkpoints'', type=str, default='./checkpoints/', help='location


```

图 9. PATCHTST 在 ETTh1 数据上运行结果

```

(''--task_name'', type=str, required=False, default='Long_term_forecas
    help='task name, options:[long_term_forecast, short_term_forecast,
(''--is_training'', type=int, required=False, default=1, help='status'
(''--model_id'', type=str, required=False, default='ETTm1_96_96', help
(''--model'', type=str, required=False, default='PatchTST',
    help='model name, options: [Autoformer, Transformer, TimesNet]')

(''--data'', type=str, required=False, default='ETTm1', help='dataset
(''--root_path'', type=str, default='./dataset/ETT-small/', help='root
(''--data_path'', type=str, default='ETTm1.csv', help='data file')
(''--features'', type=str, default='M',
    help='forecasting task, options:[M, S, MS]; M:multivariate predict
(''--target'', type=str, default='OT', help='target feature in S or MS
(''--freq'', type=str, default='h',
    help='freq for time features encoding, options:[s:secondly, t:minut
(''--checkpoints'', type=str, default='./checkpoints/', help='location

task
(''--seq_len'', type=int, default=96, help='input sequence Length')
(''--label_len'', type=int, default=48, help='start token Length')
(''--pred_len'', type=int, default=96, help='prediction sequence Length
(''--seasonal_patterns'', type=str, default='Monthly', help='subset fc
(''--inverse'' action='store_true', help='inverse output data' defau


```

图 10. PATCHTST 在 ETTm1 数据上运行结果

汇总模型结果如表 1, 所示, PatchTST 模型胜过了基准模型 Tranformer。定量地说, 与 Transformer 的模型能够提供的最佳结果相比, PatchTST 在 MSE 上实现了总体约 60% 的降低, MAE 上降低了 50%。

表 1. 模型复现结果

模型	Transformer		PatchTST	
指标	MSE	MAE	MSE	MAE
ETTh1	0.778	0.687	0.378	0.397
ETTh2	2.597	1.292	0.291	0.341
ETTm1	0.735	0.631	0.333	0.368

## 5.2 改进后的实验结果

在 PatchTST 模型的归一化方法中加入批量归一化代码, 与原有的归一化方法相结合, , 同样在 ETTh1, ETTh2, ETTm1 数据集上运行, 得到的结果如下图所示:

图 11. PATCHTST 优化后在 ETTm1 数据上运行结果

图 12. PATCHTST 优化后在 ETTh2 数据上运行结果

图 13. PATCHTST 优化后在 ETTm1 数据上运行结果

将优化后的结果与 Transformer、优化前的 PatchTST 的模型结果汇总到表 2 中，可以看到，加入了批量归一化方法的 PatchTST 在 Etth1 大型数据集上有略微的提升。

表 2. 模型复现结果

模型	Transformer		PatchTST		PatchTST-V2	
指标	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	0.778	0.687	0.378	0.398	0.376	0.397
ETTh2	2.597	1.292	0.291	0.341	0.291	0.342
ETTm1	0.735	0.631	0.333	0.368	0.333	0.367

### 5.3 模型优势

时间和空间复杂度的降低: 原始 Transformer 在时间和空间上的复杂度为  $O(N^2)$ , 其中  $N$  是输入标记的数量。通过应用补丁, 模型可以通过步长因子减少:  $N \approx L/S$ , 从而以二次方的方式减少复杂度。这在大型数据集上显著降低了训练时间。

从更长的回溯窗口中学习的能力: 通过增加回溯窗口  $L$ , 可以从 0.518 降低 MSE 到 0.397. 但单纯延长  $L$  会增加更大的内存和计算使用成本。时间序列通常包含大量时间冗余信息, 因此以前的一些工作试图通过降采样或精心设计的稀疏注意力连接来忽略部分数据点。PatchTST 通过分组可能包含相似值的局部时间步骤, 在减少输入标记长度以获得计算优势的同时, 保持了长回溯窗口。

表示学习的能力: 随着强大的自监督学习技术的出现, 需要具有多个非线性抽象层的复杂模型来捕获数据的抽象表示。PatchTST 不仅证实了 Transformer 实际上对时间序列预测有效, 而且还展示了可以进一步提高预测性能的表示能力。

## 6 总结与展望

本文复现了一种新颖的 Transformer 基础模型, 该模型旨在提升时间序列预测任务的性能, 引入了“Patching”和“通道独立结构”两个关键组件。与既往研究相比, PatchTST 更有效地捕捉局部语义信息, 并能从扩展的历史窗口中提取益处。在监督学习任务中, 该模型的表现超越了其他基线模型, 并展现了其在自监督表示学习和迁移学习领域的应用潜力。PatchTST 模型不仅有望成为未来基于 Transformer 的预测模型和时间序列分析的基础构件, 其“Patching”操作, 作为一种简单而有效的方法, 也已被证实可以轻易地应用到其他模型中。在此基础上, 通道独立性的设计有助于探索不同通道间的相互关系。未来, 对这些跨通道依赖关系的精确建模将成为进一步研究的一个关键方向。

## 参考文献

- [1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [3] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.
- [4] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners, 2021.

- [5] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.
- [6] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.
- [7] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhui Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.
- [8] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pages 27268–27286. PMLR, 2022.
- [9] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*, 2021.
- [10] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.
- [11] Razvan-Gabriel Cirstea, Chenjuan Guo, Bin Yang, Tung Kieu, Xuanyi Dong, and Shirui Pan. Triformer: Triangular, variable-specific attentions for long sequence multivariate time series forecasting—full version. *arXiv preprint arXiv:2204.13767*, 2022.
- [12] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.
- [13] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. *arXiv preprint arXiv:2106.00750*, 2021.
- [14] Ling Yang and Shenda Hong. Unsupervised time-series representation learning with iterative bilinear temporal-spectral fusion. In *International Conference on Machine Learning*, pages 25038–25054. PMLR, 2022.
- [15] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8980–8987, 2022.

- [16] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 2114–2124, 2021.
- [17] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-series representation learning via temporal and contextual contrasting. *arXiv preprint arXiv:2106.14112*, 2021.
- [18] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International conference on web-age information management*, pages 298–310. Springer, 2014.
- [19] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023.
- [20] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.
- [21] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.
- [22] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.