中国研究生创新实践系列大赛
“华为杯”第二十一届中国研究生数学建模竞赛

学　　校	西安交通大学
参赛队号	**No.24106980214**
队员姓名	**1.** 陈泽冰
	2. 张宇舜
	3. 石旭贸

中国研究生创新实践系列大赛
“华为杯”第二十一届中国研究生数学建模竞赛

题　　目　WLAN 组网中网络吞吐量建模

摘　　要:

WLAN 优化的核心问题在于吞吐量预测。为了在复杂的实际场景中实现精准且高效的吞吐量预测，本文采用动态图神经网络（DGNN）作为求解策略。DGNN 能够有效处理图结构和时序关系的特征，深入分析影响 WLAN 中接入点（AP）及整个网络吞吐量的关键因素，捕捉网络节点间的复杂交互关系。我们将 WLAN 的部署过程抽象为动态图，并将吞吐量预测问题建模为图节点特征的预测问题。通过构建和训练模型，成功应对了 WLAN 吞吐量预测中的挑战。

对于**问题一**，要求建模实测 WLAN 网络中各参数对 AP 发送机会的影响，并通过训练模型预测每个 AP 的发送机会。求解过程中，考虑了 AP 节点判断信道空闲的机制，包括接收信号强度 RSSI、CCA 门限和 NAV 门限的判断过程，以及 AP 节点之间的同步与异步传输关系。通过构建 WLAN 网络的动态图结构，训练神经网络模型，预测了测试集 test_set_1_2ap 和 test_set_1_3ap 中的 AP 发送数据帧序列的总时长（seq_time）。同时，通过对各参数的掩盖推理，量化了各参数对 AP 发送机会的影响顺序：网络拓扑 > 门限 > 节点间 RSSI > 业务流量。

对于**问题二**，要求基于实测的 WLAN 网络测试信息，尤其是节点间 RSSI 和门限信息，结合问题一中对 AP 发送机会的分析，训练模型预测每个 AP 最常使用的调制编码方案（MCS，NSS）。在求解过程中，考虑了 AP 节点的（MCS，NSS）与 STA 节点信噪比（SINR）之间的关系，以及 AP 节点传输方式（同步、异步和混合传输）的影响。通过构建 WLAN 网络的动态图结构，训练神经网络模型，预测了测试集 test_set_2_2ap 和 test_set_2_3ap 中的（MCS，NSS）。

对于**问题三**，综合问题一和问题二的分析，要求预测实际 WLAN 网络中的吞吐量。求解过程结合了 WLAN 网络的大尺度信息（如节点间 RSSI 等）与小尺度信息（如 MCS、NSS 等），通过构建包含静态图结构与动态时序信息的动态图，训练神经网络模型，成功预测了测试集 test_set_1_2ap 和 test_set_1_3ap 中的 AP 网络吞吐量。

关键字： **动态图神经网络**　LSTM　WLAN 优化　吞吐量预测　网络拓扑

目录

1. 问题重述

1.1 问题背景

无线局域网（Wireless Local Area Network，WLAN），是一种允许设备通过无线信号连接到互联网的技术。它以其低成本、高吞吐量和便利性，成为现代社会中不可或缺的通信方式。WLAN 技术使得用户无需物理连接即可在家庭、学校、办公室和酒店等环境中轻松访问网络，极大地提升了工作效率和生活质量。随着智能手机、平板电脑和智能家居设备的普及，WLAN 已成为连接这些设备的主要方式之一。此外，WLAN 技术还在推动物联网的发展，为智能家居、智慧医疗和工业制造等新兴领域提供了强大的支持，展现出广阔的应用前景和重要的社会价值。[1]

基本服务集（Basic Service Set, BSS）是构成无线局域网络的基础单元，它由一个无线接入点（Access point, AP）和一组位于其特定覆盖区域内的站点（Station, STA）组成（如图1-1）。AP 在 BSS 中扮演核心角色，负责管理与它关联的所有 STA，这个过程被称为 STA 的关联。AP 可以是家庭或办公室中的无线路由器，也可以是公共场所提供的 WiFi 热点等设备。STA 则涵盖了各种无线终端设备，如智能手机、笔记本电脑、物联网设备等。在 BSS 的通信过程中，AP 向 STA 发送数据被称为下行传输，而 STA 向 AP 发送数据则被称为上行传输。为了确保通信的有序进行，每个节点的发送和接收操作是互斥的，即在任何给定时刻，节点要么在发送数据，要么在接收数据，但不能同时进行。这种机制有助于避免数据传输中的冲突和干扰，确保了无线网络的稳定性和效率。BSS 的设计使得无线网络能够在有限的频谱资源下，有效地支持多个用户的数据通信需求。

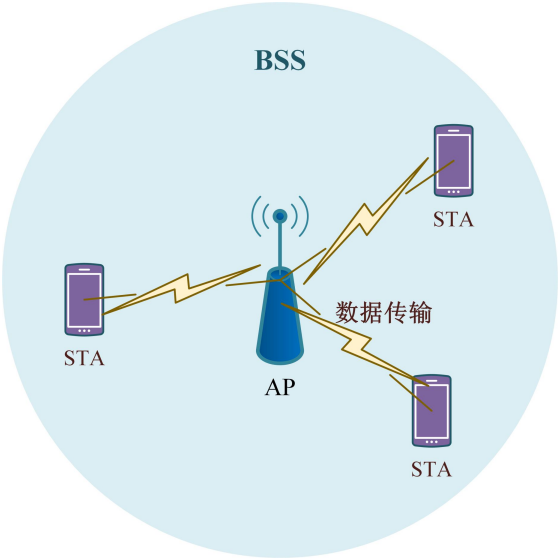

图 **1-1**　**BSS** 单元示意图

WLAN 优化问题的基础问题是吞吐量预测，它是指节点在单位时间内成功传输的比特数。为了提高 WLAN 系统的吞吐量，WLAN 采用了载波侦听多址接入/退避（CSMA/CA）机制来避免冲突；通过设置包检测（Packet Detection, PD）门限和能量检测（Energy Detection, ED）门限判断信道的忙闲状态；支持携带网络分配矢量（Network Allocator Vector, NAV）字段以保障低延时等高优先级业务的传输；采用了经典的自适应调制解调（Adaptive Modulation and Coding, AMC）算法自适应寻找最优的物理层速率（PHY Rate）发送数据。这些技术的综合应用

有效提升了 WLAN 系统的整体吞吐量。

1.2 任务重述

本任务提供的输入数据如下：

- **13 个 WLAN 网络的训练集：**包括架构、通信强度等 WLAN 网络各方面信息；
- **4 个 WLAN 网络的测试集：**仅提供部分 WLAN 网络信息

1.2.1 分析 WLAN 参数对 AP 发送机会的影响

WLAN 网络中包含很多信息，如信道、带宽、流量、发送功率、信道接入机制等基本信息，各个节点之间的接收信号强度、信噪比、传输时间等架构信息，以及节点动态位置、动态干扰等临时信息。考虑网络拓扑、业务流量、门限、节点间 RSSI 等基本信息，分析其中各参数对 AP 发送机会的影响，并给出影响性强弱的顺序。通过训练的模型，预测每个 AP 的发送机会，即发送数据帧序列的总时长（seq_time），并在两个测试集上预测 AP 发送数据帧序列的总时长。

本任务要求的输出如下：

- **参数影响性排序：**一个偏序关系，其元素是在训练集中提供的各个参数，形式如下所示：

$$Para_1 \succ Para_2 \succ Para_3 \succ ...$$

- **帧序列总时长：**补全两个 excel 文件的 seq_time 列属性。

1.2.2 完成对（MCS, NSS）的预测

调制编码模式（MCS）和空间流数量（NSS）是影响无线通信速率的关键参数。任务要求利用实测训练集中的节点间 RSSI 信息和门限信息，结合 AP 发送机会的分析，建立模型预测在测试中 AP 发送数据时选用最多次数的（MCS, NSS）组合。事实上，这一预测反映了通信速率的信噪比（SINR）水平，是无线通信自适应调制编码算法的结果。此外，任务需在两个测试集 test_set_2_2ap 和 test_set_2_3ap（仅提供模型输入信息）上预测（MCS, NSS），以验证模型的有效性。

本任务要求的输出如下：

- **预测的（MCS，NSS）组合：**补全两个 excel 文件的 MCS 和 NSS 列属性。

1.2.3 预测 WLAN 吞吐量

在完成前两个任务的基础上，需要进一步建立模型来预测整个 WLAN 系统的吞吐量。吞吐量是衡量网络性能的关键指标，它受到 AP 发送机会、（MCS, NSS）选择等多种因素的影响。任务要求在两个测试集 test_set_1_2ap 和 test_set_1_3ap 上使用建立的模型预测网络吞吐量。考虑到无线信道的动态变化特性，实测中的 RSSI 信息可能不足以完全反映信道的真实变化，因此允许在建模时使用实测中统计的数据帧真实（MCS, NSS）作为模型输入变量，以提高预测精度。

本任务要求的输出如下：

- **预测的吞吐量：**补全两个 excel 文件的 throughput 列属性。

2. 模型假设与符号说明

2.1 模型假设

(1) 假设一：环境噪声为 $-100dBm$；
(2) 假设二：一次测试中，从天线中读取的数据在时间上大致均匀分布，只有小

幅度随机扰动；

(3) 假设三：当 AP 间需要传输信息时，其传输方式由传输的 $RSSI$ 值决定，具体方式为：

$$type(edge_{AP-AP}) = \begin{cases} \text{同步传输}, & RSSI > ED_{default} \\ \text{混合传输}, & RSSI \in [PD_{default}, ED_{default}] \\ \text{异步传输}, & RSSI < PD_{default} \end{cases}$$

2.2 符号说明

本文采用的变量符号如下表所示：

表 **1** 变量符号表

变量名	定义
$ap2$	由两个 AP 组成的网络构成的全体数据集
$ap3$	由三个 AP 组成的网络构成的全体数据集
$ap2 + ap3$	由 ap2 和 ap3 构成的数据集
$d_{e_{ij}}$	连接点 i 和 j 的边 e 的特征向量长度
d_v	节点 v 的特征向量长度
$edge$	图网络的边
$ED_{default}$	能量检测门限典型值，固定为-62dBm
G_t	网络 G 在 t 时刻的快照
$PD_{default}$	包检测门限典型值，固定为-82dBm
$RSSI_{edge}$	图网络的边的通信强度
$RSSI_{env}$	环境噪声，固定为-100dBm
$RSSI_{vec}$	每条边上的 RSSI 信息向量，维度为 10*1
seq_time	发送数据帧序列的总时长
SG_t	t 时刻 WLAN 组网快照
$test\_dur$	一次测试的时间，在数据集中固定为 60s
Δt	一次测试中 AP 持续侦听的间隔，在数据集中固定为 0.5s

本文采用的名词缩写如下表所示：

表 **2**　名词缩写表

简称	全称	含义
AP	Access Point	无线接入点
BSS	Basic Service Set	基本服务集
DGNN	Dynamic Graph Neural Network	动态图神经网络
ED	Energy Detection	能量检测
MCS	Modulation and Coding Scheme	调制编码方案
NAV	Network Allocator Vector	网络分配矢量
NSS	Number of Spatial Stream	空间流数
PD	Packet Detection	包检测
RMSE	Root Mean Square Error	均方误差
RSSI	Received Signal Strength Indication	接收信号能量强度
SINR	Signal To Interference And Noise Ratio	信噪比
STA	Station	站点
TCP	Transmission Control Protocol	传输控制协议
UDP	User Datagram Protocol	用户数据报协议
WLAN	Wireless Local Area Network	无线局域网

3. 模型框架

本任务旨在深入分析 WLAN 中影响 AP 及整个网络吞吐量的关键因素。我们将考虑包括硬件配置、网络架构、以及动态场景信息在内的多种网络信息。

为了全面描述和解决文档中提出的三个问题，我们采用图神经网络（GNN）作为求解策略。GNN 能够高效处理图结构数据，自然适合捕捉 WLAN 网络中节点（如 AP 和 STA）之间的复杂交互关系。

在接下来的章节中，我们将详细介绍 GNN 模型的构建过程，包括网络的图表示、特征选择、模型训练和优化策略。此外，我们将在后续的章节中，针对每个问题分别说明模型在输入、输出和预测策略上的具体差异和调整。

3.1 WLAN 网络的图表示

根据图1-1和背景知识知，每个 AP 和 STA 拥有各自的属性和状态信息，例如它们的位置、发送和接收信号的强度（RSSI）、信道接入机制、以及它们在网络中的角色（AP 或 STA），因此可以被视为网络中的一个独立实体。这些实体的这些特征使得它们可以自然地被抽象为图中的顶点。同时，AP 和 STA 之间的通信关系具有明确的连接性，即它们之间可以建立直接的无线连接来传输数据。这种连接性可以通过无线信号传播来实现，且每个连接都有其特定的属性，如信号强度、传输速率、丢包率等。这些连接关系具有明确的方向性，因此它们可以被抽象为图中的边。

因此，使用一张图结构 $G = (V, E)$ 来表示 WLAN 网络是恰当的，因为它能够准确地反映网络中设备（顶点）及其相互连接（边）的特性。这种方式有效地将独立、零散的设备与设备间关系聚合到同一个图内，为后续分析和优化 WLAN

网络建立了数据基础。

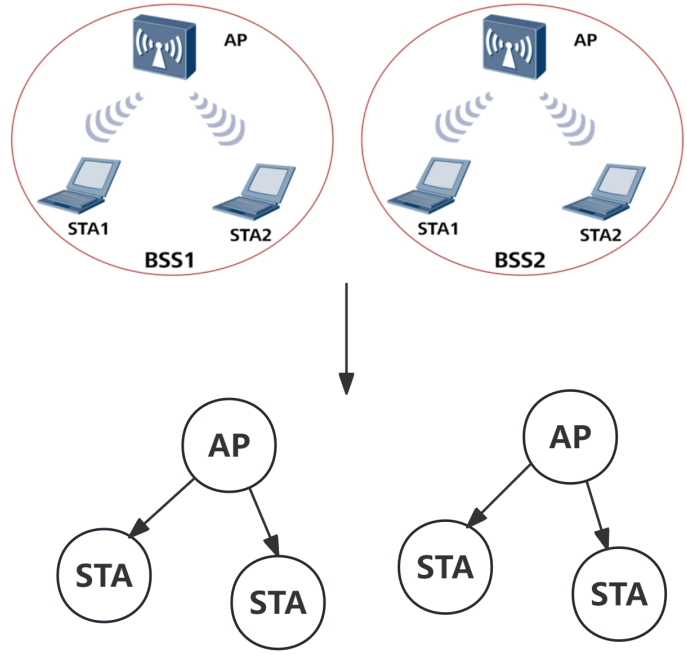

图 **3-1**　将 **WLAN** 网络转换为图表示

在 WLAN 组网架构中，一个基本服务集（BSS）由处于某一区域的站点（STA）与一个无线接入点 (AP) 组成，每个时刻 STA 只能关联到一个 AP。假设当 AP 向 STA 发送下行消息时，我们认为此时 AP 和 STA 之间存在一条实边。由于信道中传输的数据可以被一定区域内的任何其他节点接收，当数据的目标地址不是当前节点时，这些数据对当前节点来说是干扰。故我们认为对于某个节点而言，外部干扰源除环境底噪外，是离当前节点较近的（AP，STA）对之间的信号传递。假设组网中的所有节点每时每刻存在着信号的相互影响，即所有节点之间都存在一条虚边。因此，约定如下定义：

- 图表示：令 $G = (V, E)$ 表示 WLAN 网络的图，其中 V 是节点集合，包括 AP 和 STA；E 是边集合，表示节点间的连接；
- 节点特征：每个节点 $v_i \in V$ 有一组特征 $h_v = (node\_type, protocol, ...)$，包括但不限于 RSSI、SINR、是否为 AP/STA 等；
- 边特征：每条边 $e_{ij} \in E$ 连接节点 v_i 和 v_j，有一组特征 $h_{e_{ij}} = (edge\_type, ...)$，表达了节点间通信的质量等特征。

3.2 图神经网络模型

图神经网络（Graph Neural Network，简称 GNN）是一种专门处理图结构数据的深度学习模型。[7] 它们能够对节点、边以及整个图的拓扑结构进行编码，通过聚合邻居节点的信息来学习节点的表示。GNN 的核心思想是通过迭代的消息传递过程，每个节点会收集其邻居节点（以及邻居的邻居，取决于网络的深度）的特征信息，以此来更新自己的特征表示。这种信息的聚合和转换过程使得 GNN 能够捕捉到节点间的复杂关系，并且生成能够表征节点、边甚至整个图的低维向量。GNN 在节点分类、链接预测、图分类以及各种网络分析任务中都取得了显著的成功。

GNN 的架构通常由多个图卷积层组成，每一层都会更新节点的特征表示。在每一层中，节点会根据自己的特征以及邻居节点的特征来计算新的特征表示。这个过程可以通过不同的聚合函数来实现，例如求和、平均或者更复杂的注意力机制。随着深度学习的发展，GNN 也引入了各种改进，如残差连接、批量归一化和不同种类的激活函数，以提高模型的性能和泛化能力。

如图是一个 GNN 的示意图：

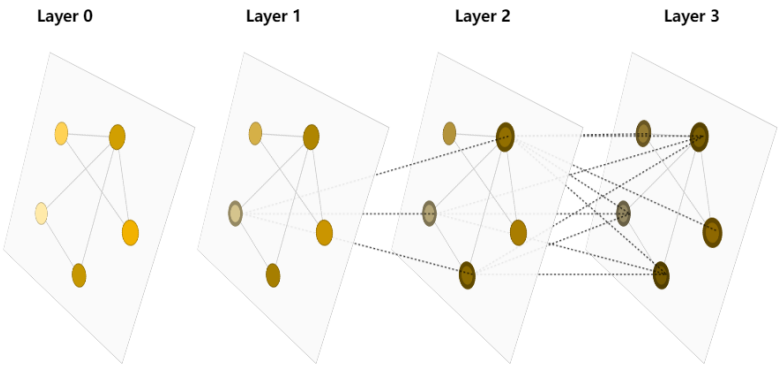

图 **3-2**　**GNN** 示意图

此外，GNN 可以扩展到动态图上，即图中的节点和边可以随时间变化，称之为动态图神经网络（DGNN）[8]。在这种情况下，GNN 可以结合序列模型如循环神经网络（RNN）或长短期记忆网络（LSTM），来捕捉时间序列上的依赖关系。DGNN 在处理时间序列图数据，如社交网络动态、交通网络流量预测等方面，展现出了强大的能力。为了将 DGNN 应用于 WLAN 网络，需要对其做一定的改造。受到 Zhou 等人工作 [2] 的启发，我们使用了如下的网络模型 [2]。如图3-3所示的是一个 2 层的神经网络，包含两个图卷积层和一个 LSTM 层。

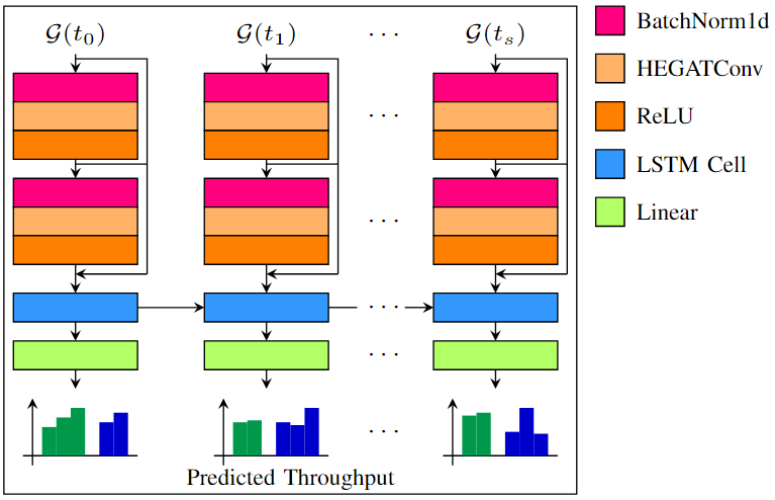

图 **3-3**　使用 **GNN** 预测吞吐量等参数的模型框架，来自 **[2]**

(1) **单层网络**

对于单层网络，其时间属性 t 是唯一的。对于任一时刻 t 的 WLAN 组网快照 G_t，假设有 n 个 AP 和 m 个 STA。由于 AP 和 AP 之间存在干扰，故有 $n(n-1)$ 条有向边来表征 AP 之间的相互影响。因为每个时刻一个 STA 只能关联一个 AP，故用 $2m$ 条有向边来表征 STA 与 AP 之间的数据传输。同时，

当某个 STA 与某个 AP 进行数据传输时，其他 AP 的行为会对当前数据传输造成干扰，故用 $2m(n-1)$ 条有向边来表征其他 AP 对当前传输的影响。因此，在静态图中，有以下三种类型的边：

$$AP \to AP, AP \to STA, STA \to AP$$

假设节点初始的特征矩阵为 $\{h_v^{(0)}\} = H_v^{(0)} \in R^{n+m,d_v}$, 边初始的特征矩阵为 $\{h_{e_{ij}}^{(0)}\} = H_e^{2n(n-1)+2mn,d_e}$, 其中 d_v 和 d_e 分别是节点和边的特征向量长度。
对于每条边 $r \in \{AP \to AP, STA \to AP, AP \to STA\}$，它的边特征矩阵通过以下方式更新 [4]：

$$h_v^{(k)}(r) = ReLU\left(\sum_{u \in N_r(v)} a_{uv} W_r^{(k)} h_u^{(k-1)} + b_r^{(k)}\right)$$

其中，$N_r(v)$ 是结点 v 具有关系 r 的邻居集合，$W_r^{(k)}$ 是关系 r 的可学习权重矩阵，$b_r^{(k)}$ 是关系 r 的可学习偏置矩阵。边特征感知注意力分数 a_{uv} 通过以下方式计算：

$$a_{uv} = w_a^{(k)} h_{uv}^{(k)}$$

$$h_{uv}^{(k)} = LeakyReLU\left(W_a^{(k)}\left[h_u^{(k-1)} \parallel h_{uv}^{(k-1)} \parallel h_v^{(k-1)}\right]\right)$$

其中，$w_a^{(k)}$ 是可学习的注意力权值向量，$W_a^{(k)}$ 是可学习的边权值矩阵，$\parallel$ 表示连接操作。在这里，注意力权值向量 $w_a^{(k)}$ 和边权值矩阵 $W_a^{(k)}$ 在所有关系中共享，并且在每个关系的消息传递过程之前，所有边类型的隐藏边特征 $h_{uv}^{(k)}$ 都会被更新。
对于连接函数，由于关系的数量仅为三个，因此直接执行顺序连接是可行的。经过连接得到的第 k 层节点特征为：

$$h_v^{(k)} = \|_{r \in R}\, h_v^{(k)}(r)$$

对于只有一条通信 AP→STA 边的 STA 节点，在 $r \in \{AP \to AP, STA \to AP\}$ 上缺失的 $h_v^{(k)}(r)$ 将替换为零向量。

(2) **过平滑问题**

在深度 GNN 中，过平滑问题通常发生在所有结点的嵌入向量收敛到相同的向量，从而失去区分各个结点的能力。尽管我们的网络仅需要 2 或 3 层，但 WLAN 部署的特殊图结构加剧了过平滑问题。在同一接入点（AP）下的任意两个站点（STA）共享相同的 1 跳和 2 跳邻居集合。不同 AP 下的任意两个 STA 甚至共享所有其他 AP 的 2 跳邻居集合。为了缓解过平滑问题，我们采用了 JK-Net[3] 的设置，在每个单层网络之后添加一个残差连接。K 层单层网络的输出结点嵌入为

$$h_v = \phi\left(\|_{0 \le k \le K}\, h_v^{(k)}\right)$$

其中，单射函数 $\phi(\cdot)$ 通过一个单层的感知机实现。

(3) **完整网络设计**

在每个单层网络对每个 WLAN 快照进行消息传递并获得 $t = t_0, t_1, ..., t_s$ 时刻的 h_v^t 后，我们应用 LSTM 序列模型来捕捉时间信息 [6]。这个框架计算最终的动态结点嵌入 $\tilde{h}_v^t$ 的方法如下：

$$\begin{cases} f_v^t = \sigma\left(W_f h_v^t + U_f \tilde{h}_v^{t-1} + b_f\right) \\ i_v^t = \sigma\left(W_i h_v^t + U_i \tilde{h}_v^{t-1} + b_i\right) \\ o_v^t = \sigma\left(W_o h_v^t + U_o \tilde{h}_v^{t-1} + b_o\right) \\ c_v^t = f_v^t \circ_v^{t-1} + i_v^t \circ \sigma\left(W_c h_v^t + U_c \tilde{h}_v^{t-1} + b_c\right) \\ \tilde{h}_v^t = o_v^t \circ \sigma\left(c_v^t\right) \end{cases}$$

其中，$\sigma(\cdot)$ 是 Sigmoid 函数，即 $\sigma(x) = \frac{1}{1+e^{-x}}$，$\circ$ 是逐点乘法运算符，W 和 U 是可学习的权重矩阵，b 是可学习的偏置，$t-1$ 表示前一个时间戳，c_v^t 初始化为 0。我们应用一个线性层来生成预测的吞吐量 $\hat{y}_v^t$，并添加一个 Softplus 函数以确保预测的吞吐量严格为正，公式为：

$$\hat{y}_v^t = log\left(1 + exp(W_y \tilde{h}_v^t)\right)$$

在每个单层网络后，我们应用可学习仿射参数进行批量归一化。图神经网络中的所有可学习参数都是通过训练集中的真实值吞吐量 y_v^t 进行端到端监督训练的。损失函数旨在最小化所有目标 STAs 在所有时间上的均方根误差（RMSE），即：

$$L = \sqrt{\frac{\sum_v \sum_t (y_v^t - \hat{y}_v^t)^2}{|\{v\}||\{t\}|}}$$

3.3 求解思路

本论文的整体解题思路和建模流程如图3-4所示。

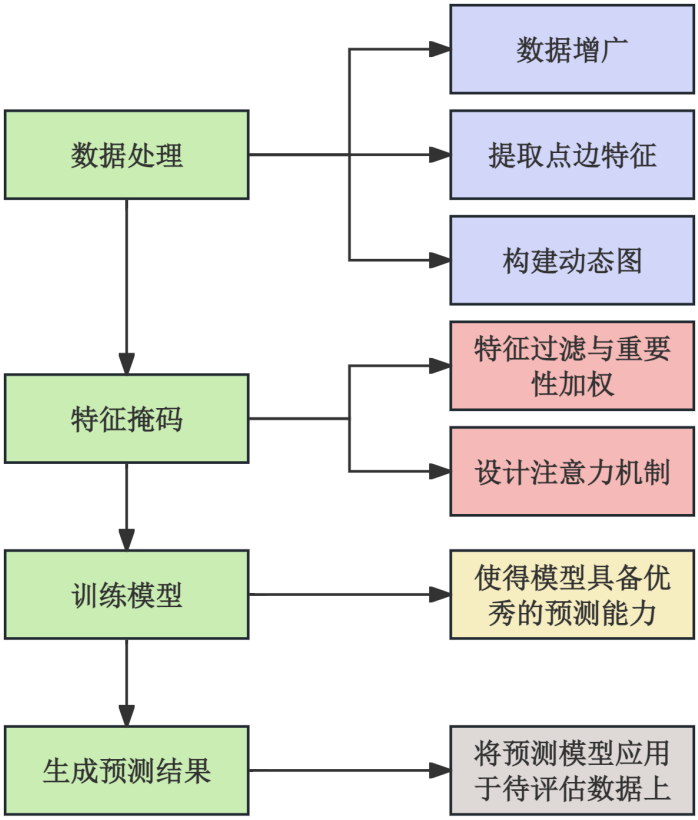

图 **3-4**　三个任务的整体思路流程

3.3.1 数据处理

(1) **数据增广：**数据集中包含 $RSSI_vec$（10*1 的列向量）这个关键指标，表示从 AP 天线接收到的来自特定 AP（标识为 AP_x）的信号强度（RSSI）信息。如果在测量 RSSI 时，AP_x 并未发送数据，则该值代表的是环境噪声。为了从测量数据中排除这种噪声，我们需要对 RSSI 值进行筛选，这可能导致数据集中的 RSSI 数据长度不统一。

由于每个数据点的 RSSI 数据长度可能不同，我们需要将这些数据转换为具有一致长度且数量更多的图数据。在实际的模型中，转化后的一致长度为 120，即采集长度 $test\_dur$（数据集中均为 60 秒）内每隔 Δt（数据集中均为 0.5 秒）采集一次得到的样本数量。为此，我们采用以下设计策略：

(a) **时间轴扩展：**设待转换的 RSSI 数据长度为 n，那么其采样时间就是在 $[0, n)$ 上均匀分布的点列。为了实现时间轴的扩展，我们首先将每个数据点的时间标签按照倍增比例为 $\frac{test\_dur}{n \cdot \Delta t}$ 倍的比例进行调整。具体来说，如果原始数据点的时间标签为 $t_i \in [0, n)$，则新的时间标签为 $\frac{test\_dur}{n \cdot \Delta t} \times t_i$。这样，数据点就在 $[0, 60)$ 秒的时间轴上均匀分布；

(b) **插值拟合：**在时间轴扩展之后，我们得到了 n 个数据点在 $[0, test\_dur)$ 秒时间轴上的新位置。接下来，我们需要在这 n 个数据点之间进行插值拟合，以生成一条平滑的曲线。通过插值，我们可以得到一个连续的函数，该函数在 $[0, test\_dur)$ 秒的时间范围内定义了数据点之间的关系；

(c) **均匀采样：**有了插值后的连续函数，我们可以在 $[0, test\_dur)$ 的时间轴上均匀地采样数据点。具体操作是，从 0 秒开始，每隔 Δt 在插值函数上取

一个数据点的值，直到 $test\_dur$ 为止。这样，我们就得到了 $\frac{test\_dur}{\Delta t}$ 个在 $[0, test\_dur)$ 时间轴上均匀分布的数据点。

(d) **可视化验证：**使用 Matplotlib 提供的图形化工具绘制原始数据点和增广后的数据点，其增广结果与原数据符合较好。

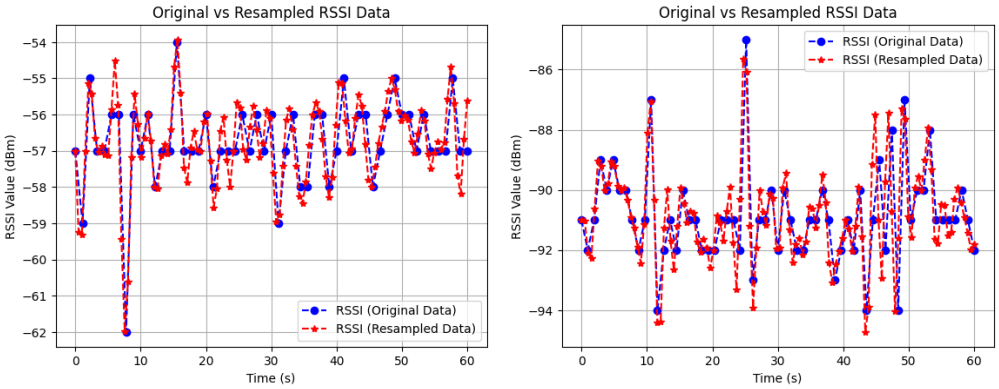

图 **3-5**　同步传输（左）和异步传输（右）下数据增广前后对比

(2) 提取点边特征

- **分配数据集属性：**该步骤的主要目标是从 excel 格式的文件中读取 WLAN 网络信息。查阅数据集可知，该数据集具有如表3所示的参数，将其分为三个类型（图特征、点特征、边特征），图特征将用于划分训练数据点，而点特征和边特征将分别绑定在图网络的点和边上：

表 **3**　数据集属性分配结果

参数名称	含义	特征类型
test_id	测试编号	图特征
test_dur	一次测试的时间	图特征
loc	AP 和 STA 部署在某一指定位置的编号	点特征
bss_id	同一拓扑里的 BSS 编号	点特征
ap_name	AP 的名字	预处理后舍去
ap_id	AP 编号	边特征
sta_mac	STA 的 MAC 地址	预处理后舍去
sta_id	STA 编号	边特征
protocol	AP 发送的流量类型	点特征
pkt_len	聚合前的数据长度	预处理后舍去
pd	包检测门限	点特征
ed	能量检测门限	点特征
nav	NAV 门限	点特征
erip	AP 的发送功率	点特征
sinr	信噪比	点特征
$RSSI_{vec}$	RSSI 信息	边特征

- **计算 SINR 属性：**节点的 PHY 层对信号进行解调时，要求一定的 SINR，其 SINR 越高，可支持成功解调的 MCS 和 NSS 越高。SINR 属性的精确值计算由传输方式决定，其类型已在2.1节给出。SINR 的计算方式如下：

$$SINR = \begin{cases} W_{\text{有效}}/RSSI_{env}, & type(edge) = \text{同步传输} \\ W_{\text{有效}}/W_{\text{干扰}}, & type(edge) = \text{异步传输} \\ \delta \cdot SINR_{\text{同步}} + (1-\delta) \cdot SINR_{\text{异步}}, & type(edge) = \text{混合传输} \end{cases}$$

根据2.1节做出的假设，有 $RSSI_{env} = -100dBm$，

$$\delta = \frac{RSSI_{AP-AP} - ED_{default}}{PD_{default} - RSSI_{AP-AP}}$$

表示同步传输与混合传输之比；而将有效信号的 RSSI 从 dBm 转换为 mW 和将结果从 mW 转换为 dBm 的两个函数分别如下 [5]：

$$P_{mW} = 10^{P_{dBm}/10}$$

$$P_{dBm} = 10 \cdot log_{10}(P_{mW})$$

- **处理错误数据：**除了分配属性，我们还需要处理属性中缺失或不正确的数据。本任务提供的数据集存在一定错误，如 training_set_2ap_loc2_nav82.csv

数据集的测试点 80 和 81（第 80-81 行）。如表4所示，其属性值与上文期望的整数列表完全不符，且无法通过近似填充的方式回填为正确数字，因此选择直接删除整个数据点。

表 **4** 数据集错误数据示例

属性	测试点 78 （有效）	测试点 80 （无效）
eirp	28	29
ap_from_ap_0_sum_ant_rssi	[-66, -67, -67, -67, -85,...]	471f
ap_from_ap_0_max_ant_rssi	[-72, -74, -72, -74, -90,...]	sta_1
sta_mac	471f	0.003477128
sta_id	sta_1	0
sta_to_ap_0_sum_ant_rssi	[-74, -69, -87, -87, -87,...]	33.22
sta_to_ap_0_max_ant_rssi	[-94, -94, -94, -93, -77,...]	104
sta_to_ap_0_mean_ant_rssi	[-83, -78, -96, -96, -96,...]	空值
sta_to_ap_1_sum_ant_rssi	[-68, -67, -68, -68, -68,...]	空值

(3) **构建动态图：**经过点边特征提取后，excel 文件已转换为标准的静态图结构，为了将其转化为适合我们的图神经网络处理的动态图数据格式，需要进行二次构建。

(a) **插入虚边：**首先，为简化操作，我们将这些未记录边的数值统一设定为 $RSSI_{env}$，即 $-100dBm$；

(b) **构建 BIN 文件：**BIN 文件是我们的图神经网络模型的输入格式，每个 BIN 文件包含一个动态图。在完成“数据增广”后，每个测试标识（test_id）对应 120 个静态图。假设 CSV 数据集中包含 len(csv) 个不同的测试标识。我们将 len(csv) 个静态图合并成一个 BIN 文件，这样每个 BIN 文件中的图都具有相同的网络拓扑结构（因为它们来自同一个 CSV 文件），并且它们在时间上是同步的，这使得网络特征更加集中和一致。对于每个时间点 t，我们聚合一个 BIN 文件，因此每个 CSV 文件最终被聚合成 120 个 BIN 文件。每个 BIN 文件包含 len(csv) 个在拓扑结构上等效的静态图，它们共同构成一个动态图文件。

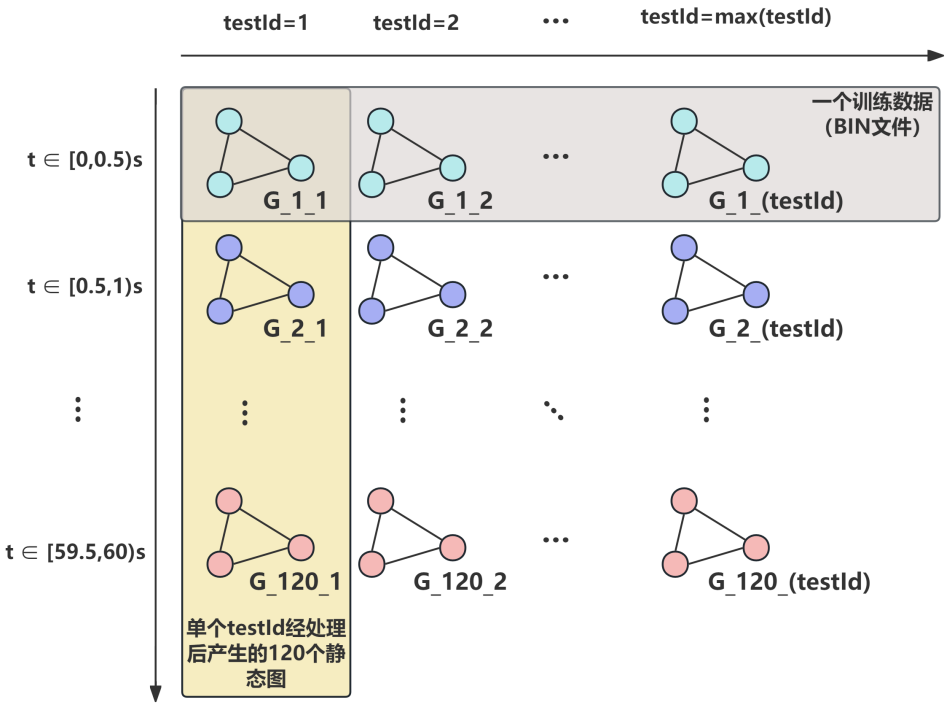

图 **3-6** **BIN** 文件生成示意图

3.3.2 特征掩码

掩码通常是一个与原始特征大小相同的二值或布尔数组，其中选定的区域标记为 1（或 True），而其余区域标记为 0（或 False）。在解决网络拓扑、业务流量、门限和节点间 RSSI 对模型测试效果影响的强弱时，我们通过分别掩蔽相关特征，并比较模型在测试集上的表现，从而判断这些特征的影响力。由于我们使用统一的模型来处理这三个问题，模型在训练和推理阶段所用的统计信息不同。在计算损失函数和进行预测时，我们也使用掩码来选择所需的统计特征。

3.3.3 训练模型

通过以上两步,我们完成了全部的训练数据构建。例如,对于 training_set_2ap_loc0_nav82 数据集，它将被转化成如下所示的训练数据格式：

```
training_set_2ap_loc0_nav82
└edge_ap_ap
| └test1
| | └ap_ap_000.csv # 每个csv文件代表一个t时刻的静态图G
| | └......
| | └ap_ap_120.csv
| └test2
| └......
| └test41 #该文件共有41个test_id
└edge_ap_sta #其它类型的点、边数据格式同上
└edge_sta_ap
└node_ap
└node_sta
```

设计具有如表5所示参数的 GNN 网络，并在上述的数据集上进行充分的训练，得到具有不同预测性能的神经网络模型。

表 **5** **GNN** 模型结构与参数统计

层名称	类型（深度-编号）	参数量
ModuleDict	1-1	——
BatchNorm 1d	2-1	24
BatchNorm 1d	2-2	10
EHeteroGraphConv	2-3	——
ModuleDict	3-1	16,896
BatchNorm 1d	2-4	256
BatchNorm 1d	2-5	256
EHeteroGraphConv	2-6	——
ModuleDict	3-2	197,760
Perceptron	2-7	35,968
Dropout	3-3	——
LSTM	2-8	264,192
Perceptron	2-9	129
Dropout	3-4	——
Linear	2-10	7,744
Dropout	2-11	——
LeakyReLU	2-12	——
Linear	2-13	2,080
Linear	2-14	33
Softplus	2-15	——

我们训练模型的服务器配置如表6所示：

表 **6** **GNN** 网络训练所使用服务器数量

参数名称	取值
系统环境	Ubuntu 20.04.6 LTS
显卡配置	NVIDIA GeForce RTX 4090
显卡数量	8

3.3.4 生成预测结果

在本节中，我们将运用“训练模型”阶段得到的具有优秀预测能力的模型，在待评估的测试数据集上进行预测，以生成最终的预测结果。由于不同子问题对

输出内容的要求不同，具体的预测结果将在每一个子问题的“问题求解”模块具体展开，此处不再赘述。

4. 问题一的分析与求解

4.1 问题分析

问题一旨在分析网络拓扑、业务流量、门限和节点间 RSSI 测试信息对接入点（AP）发送机会的影响，并确定这些参数的影响强度顺序。同时，需要训练模型来预测每个 AP 之间的发送机会。

AP 节点对信道空闲的判断与当前 AP 的接收信号强度以及 CCA 门限值、NAV 门限值密切相关。在 WLAN 中，同一信道的各节点共享信道，因此 AP 在向终端设备（STA）发送消息之前，需要通过载波侦听多址接入/退避机制避免冲突，信道忙闲综合 CCA 门限判断与 NAV 静默期决定。当 AP 节点有数据发送时，首先在分布式协调帧间距时段内进行载波侦听，若接收到的信号强度 RSSI 低于 CCA 门限，初步判断信道为空闲，否则判断信道为繁忙。同时，当 AP 节点的 PHY 层接收到相邻的 BSS 的数据帧时，若 RSSI 低于 NAV 门限则停止接收，直接丢弃；若 RSSI 高于 NAV 门限，则完成接收，并将其上送到 MAC 层。

AP 节点发送数据的发送频率与两两 AP 节点之间的传输方式也有显著关系。在同步传输的情况下，两个 AP 的数据传输交替进行，只在偶然同时回退到 0 时同时进行传输传输，发送频率较低；在异步传输的情况下，两个 AP 的数据传输互不影响，发送频率较高。两两 AP 节点之间的传输方式由它们之间的 RSSI 以及门限信息决定，当两个 AP 节点的 RSSI>ED 时，进行同步传输；当两个节点的 RSSI<PD 时，进行异步传输；当 PD<RSSI<ED 时，异步传输与同步传输混合。

AP 节点的发送机会与网络拓扑以及业务流量有关。因此要考虑处于不同时刻，不同地域，不同基本服务集的 AP 节点，使用不同的业务流量协议时，对 AP 节点发送机会的影响。问题一的分析综合了上述因素。

我们分别进行了统一建模和按照 AP 数量建模，分别适用于复杂网络情况与已知 AP 数量的网络情况。模型综合考虑了网络拓扑结构、业务流量、门限和节点间 RSSI 测试信息对接入点 AP 发送机会的影响，通过由数据表生成的图数据集，以 7：1.5：1.5 的比例将其划分为训练集、验证集、测试集，并按比例随机采样，进行 50 轮的训练，学习率设置为 0.001，预测图的 seq_time 结果。

针对影响强弱性顺序，我们通过在测试数据集上屏蔽相关特征来进行消融实验，比较预测结果的精确度。

4.2 问题求解

原数据集内的变量较多，为了比较网络拓扑、业务流量、门限、节点间 RSSI 四个参数对 seq_time 的影响，首先给出如下变量到四个参数的映射定义：

- 属于网络拓扑的特征：bss_id，loc_id，ap_id 和 sta_id；
- 属于业务流量的特征：protocol；
- 属于门限的特征：pd，ed，nav；
- 属于节点间 RSSI 的特征：$RSSI\_vec$。

接下来，在测试数据集上，通过掩盖掉相关特征，比较预测结果的精确度的波动情况。实验环境使用均方误差（RMSE）描述精确度。均方误差越大，说明预测精确度越低。比较精确度波动情况的流程如下图所示：

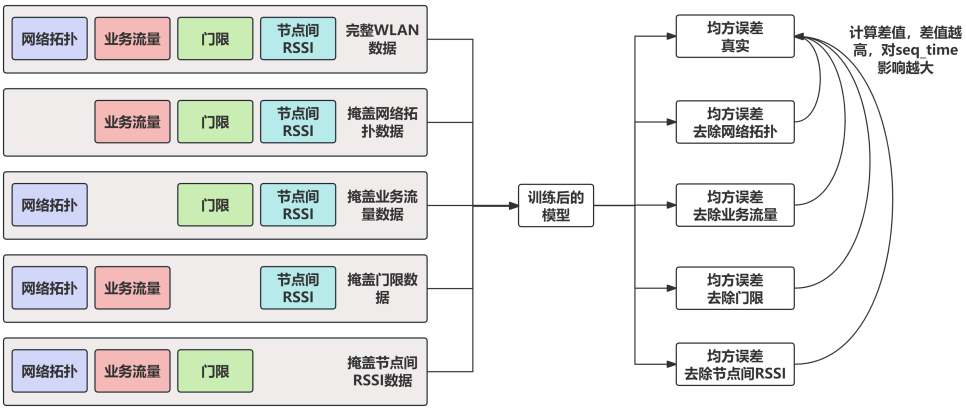

图 **4-1**　问题一求解流程

4.2.1 统一建模与独立建模

在统一建模中，将 AP 个数为 2 和 3 的情况合为一个数据集，记为 ap2+ap3；在独立建模中，将 AP 个数为 2 和 3 的情况分别建立两个数据集，分别记为 ap2 和 ap3。在这三个数据集上分别训练得到 seq_time 预测模型。训练过程的 loss 函数随训练轮数变化如下：

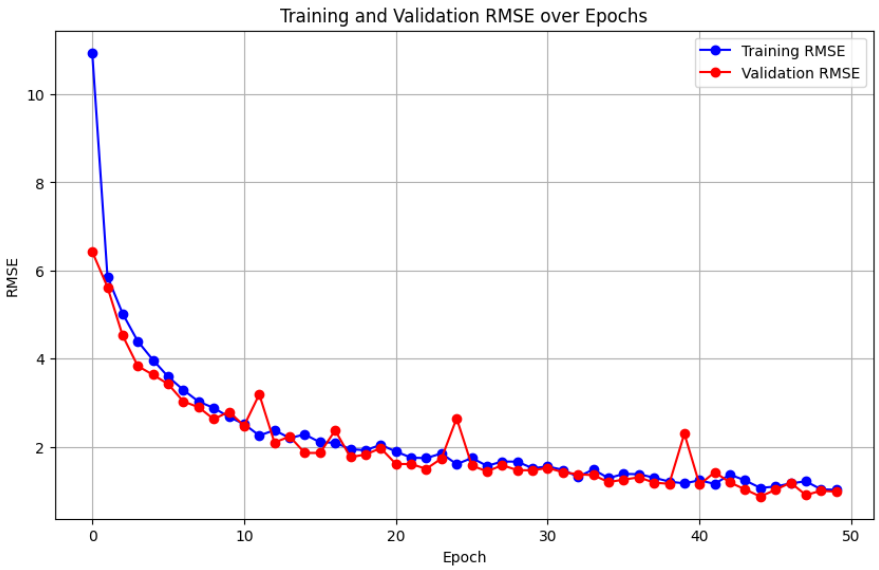

图 **4-2**　问题一 **ap2+ap3** 统一预测模型 **loss** 函数随训练轮数变化

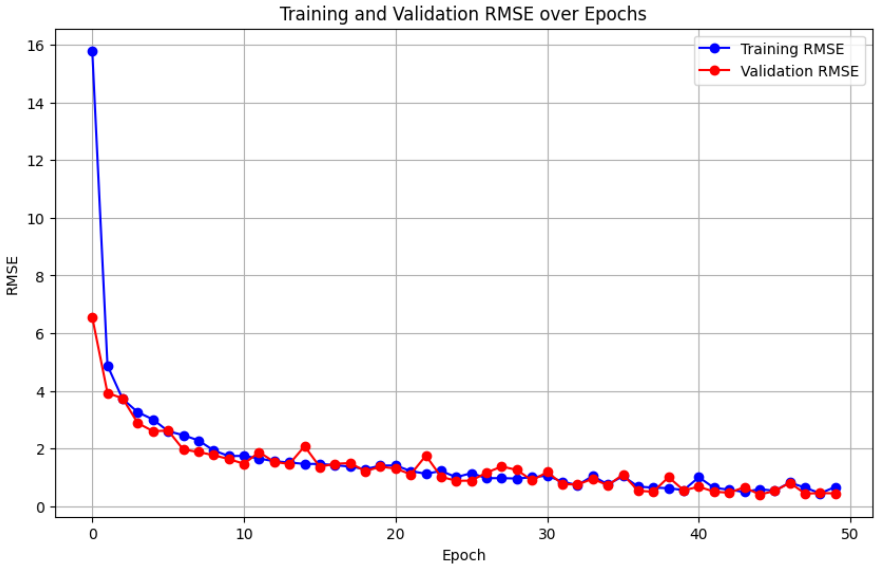

图 **4-3**　问题一 **ap2** 独立预测模型 **loss** 函数随训练轮数变化

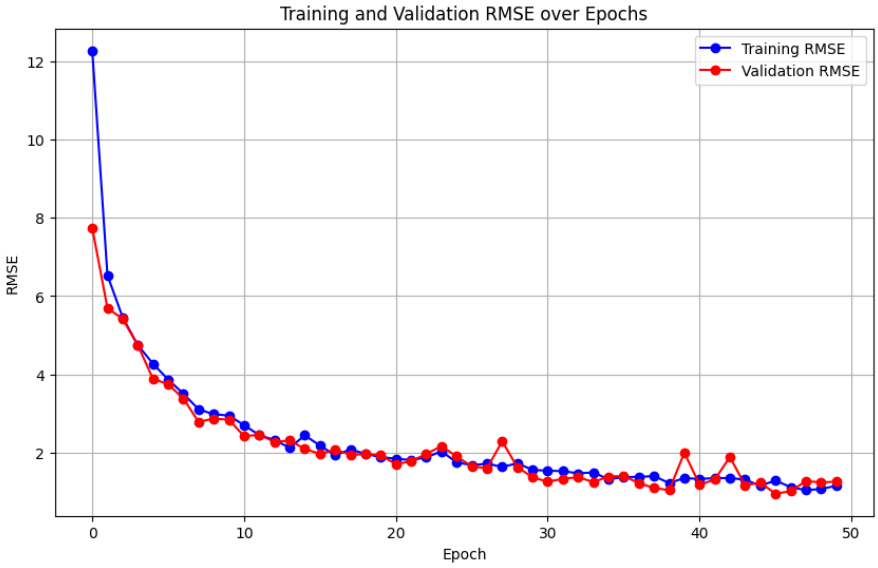

图 **4-4**　问题一 **ap3** 独立预测模型 **loss** 函数随训练轮数变化

现在，依次掩盖每一个参数，得到的新预测性能如下图：

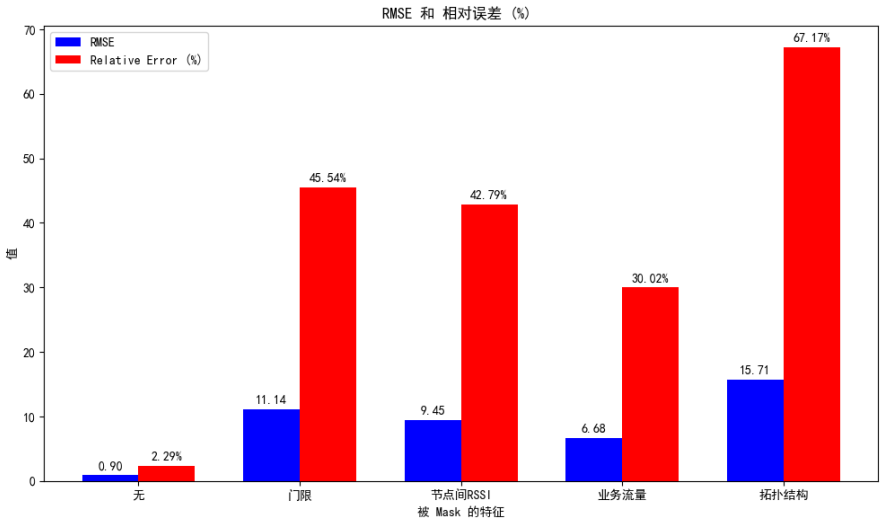

图 **4-5**　问题一 **ap2+ap3** 统一预测模型接受被掩盖后的数据的预测能力

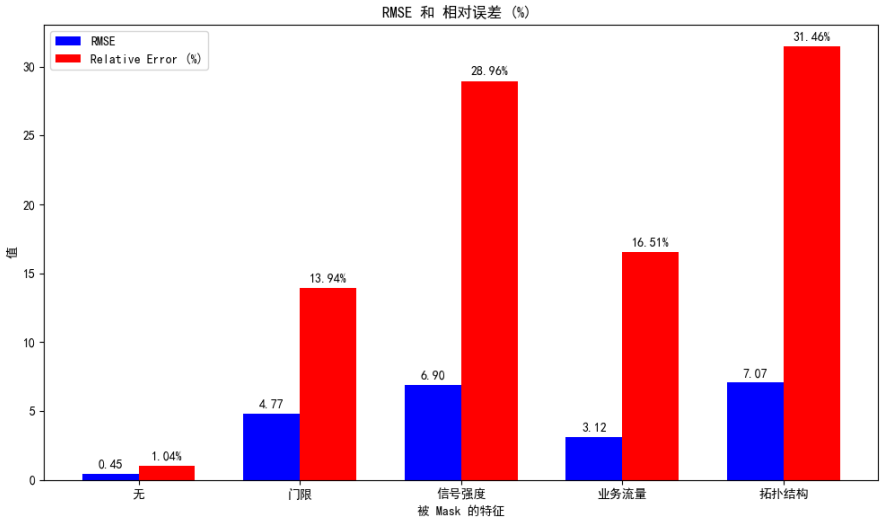

图 **4-6**　问题一 **ap2** 独立预测模型接受被掩盖后的数据的预测能力

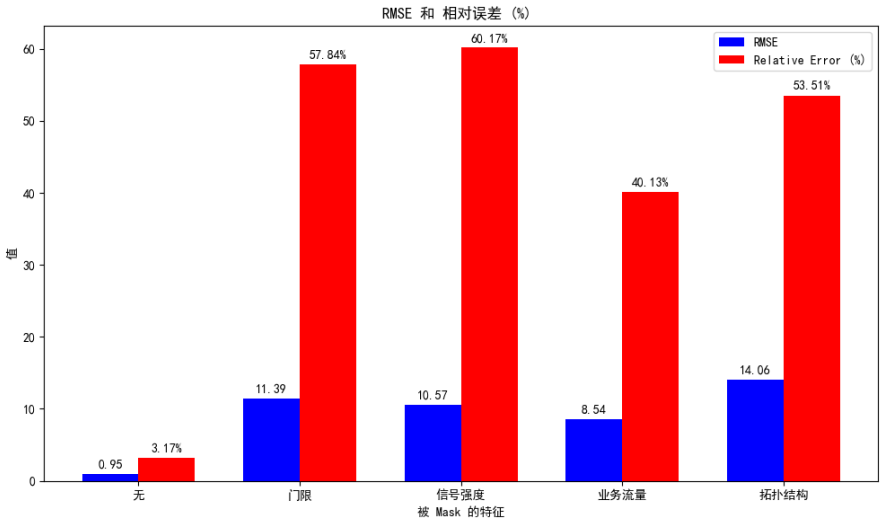

图 **4-7**　问题一 **ap3** 独立预测模型接受被掩盖后的数据的预测能力

4.2.2 建模结果

计算 RMSE 提高程度，得到下表：

表 **7**　问题一 **ap2+ap3** 统一预测模型的 **RMSE** 相较于未删减参数时的提高程度

掩盖的参数	ap2+ap3 提高程度
网络拓扑	14.8144
业务流量	5.7852
门限	10.2472
节点间 RSSI	8.5558

表 **8**　问题一 **ap2** 独立预测模型的 **RMSE** 相较于未删减参数时的提高程度

掩盖的参数	ap2 提高程度
网络拓扑	6.6178
业务流量	2.6752
门限	4.3219
节点间 RSSI	6.4540

表 **9**　问题一 **ap3** 独立预测模型的 **RMSE** 相较于未删减参数时的提高程度

掩盖的参数	ap3 提高程度
网络拓扑	13.1051
业务流量	7.5878
门限	10.4413
节点间 RSSI	9.6153

根据 RMSE 的定义，不难得出：RMSE 相较于未删减参数时的提高程度越大，说明删去该参数后模型对 seq_time 的预测能力越差，即该参数影响 seq_time 的能力越强。在三个模型性能下，分别按照该数值排序，有：

- **ap2+ap3 统一模型：**

$$14.8144 > 10.2472 > 8.5558 > 5.7852$$

- **ap2 独立预测模型：**

$$6.6178 > 6.4540 > 4.3219 > 2.6752$$

- **ap3 独立预测模型：**

$$13.1051 > 10.4413 > 9.6153 > 7.5878$$

因此，三个模型分别确定四个参数对 seq_time 的影响顺序是：

- **ap2+ap3 统一模型：**

$$\text{网络拓扑} \succ \text{门限} \succ \text{节点间 RSSI} \succ \text{业务流量}$$

- **ap2 独立预测模型：**

$$\text{网络拓扑} \succ \text{节点间 RSSI} \succ \text{门限} \succ \text{业务流量}$$

- **ap3 独立预测模型：**

$$\text{网络拓扑} \succ \text{门限} \succ \text{节点间 RSSI} \succ \text{业务流量}$$

最后，在我们提出的三种模型中，由于 ap2+ap3 统一模型的 RMSE 波动 14.8144 在所有模型中绝对值最大，因此采用统一模型作为最终的预测结果。因此，得到最终四个参数对 seq_time 的影响顺序是：

$$\text{网络拓扑} \succ \text{门限} \succ \text{节点间 RSSI} \succ \text{业务流量}$$

进一步对 ap2+ap3 统一预测模型的预测 seq_time 进行比较，发现其预测值与真实值相差非常小：

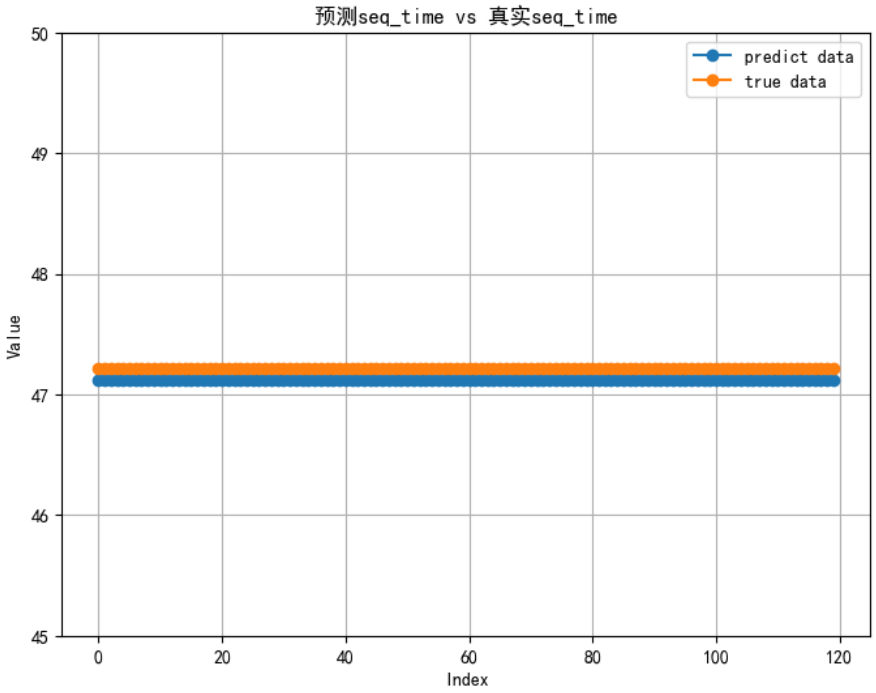

图 **4-8**　问题一 **ap2+ap3** 统一预测模型预测 **seq_time** 与实际 **seq_time** 比较

针对补全两个 excel 文件的 seq_time 列属性需求，我们将统一模型应用于测试集提供的删减后数据，并将得到的结果写入 excel，其结果具体见附件文件。

5. 问题二的分析与求解

5.1 问题分析

AP 节点发送数据的调制编码方案 MCS 和空间流数 NSS 的组合 (MCS,NSS)，共同表征了 AP 节点向下行 STA 节点的发送数据速率，（MCS，NSS）越高，表示数据发送时携带的有效比特数越多。WLAN 网络使用 AMC 算法自适应寻找最优的 (MCS，NSS) 来发送数据，使得 WLAN 网络的吞吐量最大。

AP 节点的 AMC 选用的（MCS，NSS）与下行 STA 节点的信噪比 SINR 相关。下行 STA 节点对来自所属基本服务集的 AP 节点的信号进行解调时，受到相邻 AP-STA 对之间的数据传输的影响以及 AP 间的相互干扰，会发生丢包。STA 节点受到的干扰越少，其 SINR 越高，接收到的有效比特数越多，丢包率越低；STA 节点受到的信号干扰越多，其 SINR 越低，接收到的有效比特越少，丢包率越高。AP 节点的 AMC 为了提高网络吞吐量，若下行 STA 的丢包率 PER 高于一定阈值，WLAN 网络会降低（MCS，NSS）以减少发送的有效比特数，否则提高（MCS，NSS）以增加发送的有效比特数。

AP 节点的 AMC 选用的（MCS，NSS）与 AP 之间的传输方式相关。以两个 AP 传输一段时间为例，有三种情形：仅有同步传输时，两个 AP 对下行 STA 的数据传输互不影响，下行 STA 接收数据的干扰主要来自环境底噪，SINR 较高，AP 选用的（MCS，NSS）较高；仅有异步传输时，两个 AP 对下行 STA 的数据传输始终互相干扰，环境底噪可忽略，下行 STA 接收数据的干扰主要来自邻区 AP，SINR 较低，AP 选用的（MCS，NSS）较低；混合传输时，下行 STA 接收数据的干扰同时来自环境底噪与邻区 AP。

5.2 问题求解

5.2.1 统一建模与独立建模

与问题一类似，我们在 ap2+ap3、ap2、ap3 三个数据集上分别训练预测 MCS 和 NSS 图网络模型。训练过程的 loss 函数随训练轮数变化如下：

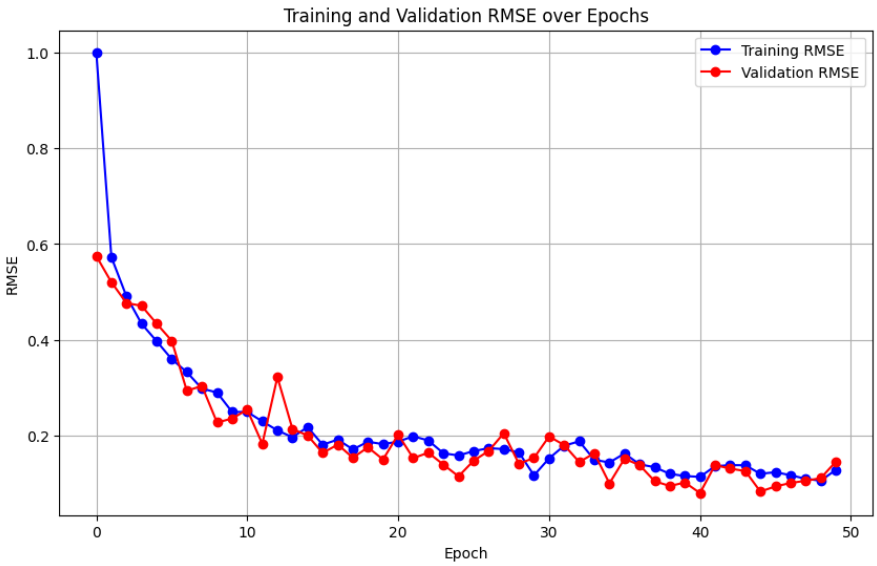

图 **5-1**　问题二 **ap2+ap3** 统一预测模型 **loss** 函数随训练轮数变化

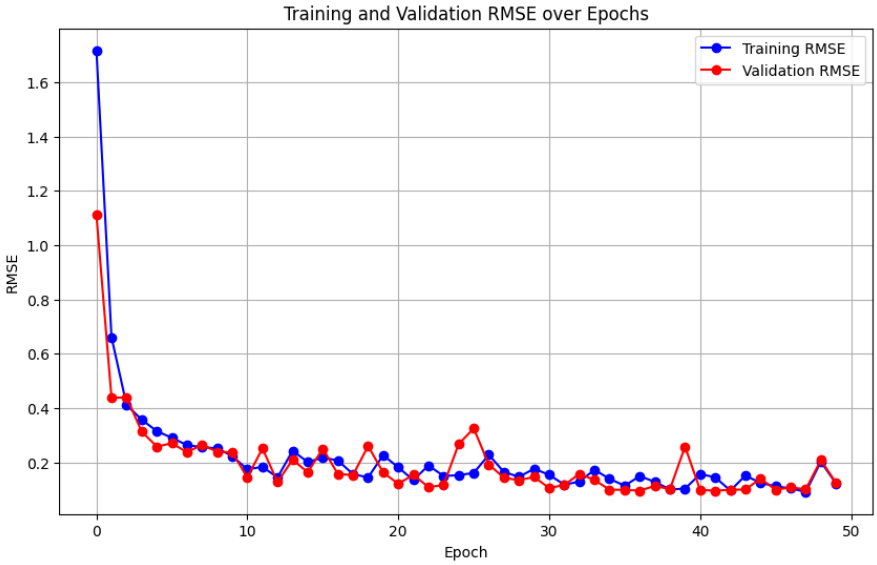

图 **5-2**　问题二 **ap2** 独立预测模型 **loss** 函数随训练轮数变化

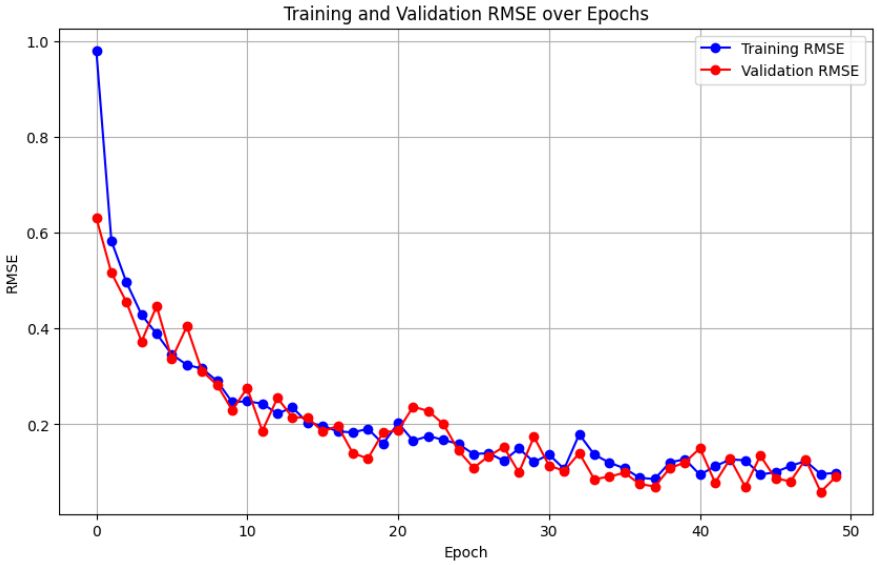

图 **5-3**　问题二 **ap3** 独立预测模型 **loss** 函数随训练轮数变化

5.2.2 建模结果

在测试集上进行交叉验证，评估该预测模型的性能，得到的 RMSE 和相对误差如下：

表 **10**　问题二 **ap2+ap3** 统一预测模型在所有测试集上评估的性能结果

评估参数	ap2+ap3	ap2	ap3
RMSE	0.0800	0.0492	0.0875
相对误差	1.27%	1.02%	1.40%

表 **11**　问题二 **ap2** 独立预测模型在所有测试集上评估的性能结果

评估参数	ap2+ap3	ap2	ap3
RMSE	1.5085	0.0953	2.1562
相对误差	10.41%	0.92%	14.70%

表 **12**　问题二 **ap3** 独立预测模型在所有测试集上评估的性能结果

评估参数	ap2+ap3	ap2	ap3
RMSE	0.6885	2.0717	0.0626
相对误差	5.39%	15.72%	0.72%

根据表格显示，针对（MCS，NSS）预测问题，统一模型再次展现出了全数据集上的掌控。同时，对于每个细分数据集，独立模型也未能展现出跨数量级的优异表现。因此，我们选择使用统一模型预测（MCS，NSS）属性。

进一步对 ap2+ap3 统一预测模型的预测 seq_time 进行比较，发现其预测值与真实值相差非常小：

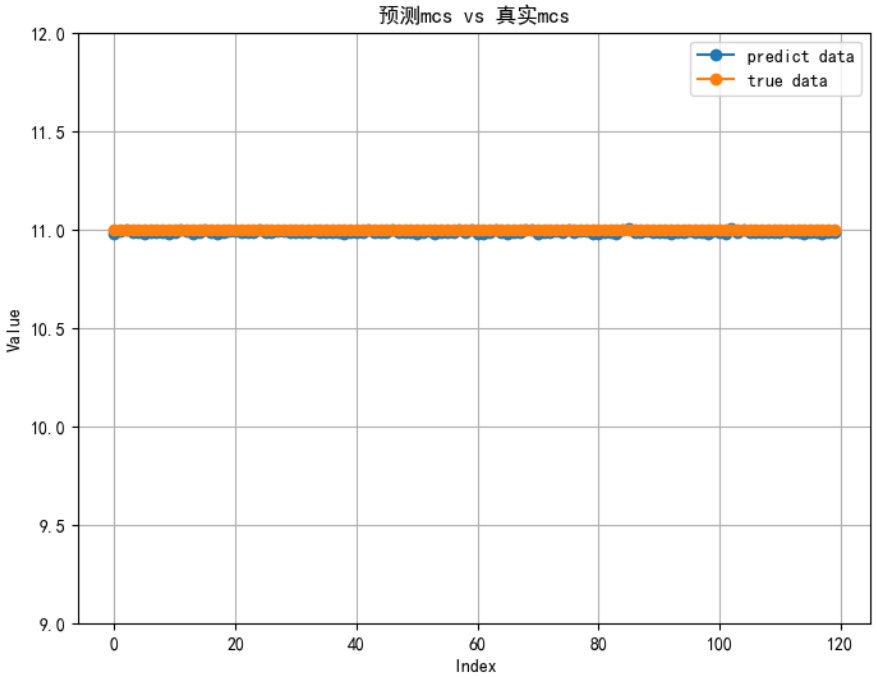

图 **5-4**　问题二 **ap2+ap3** 统一预测模型预测 **mcs** 与实际 **mcs** 比较

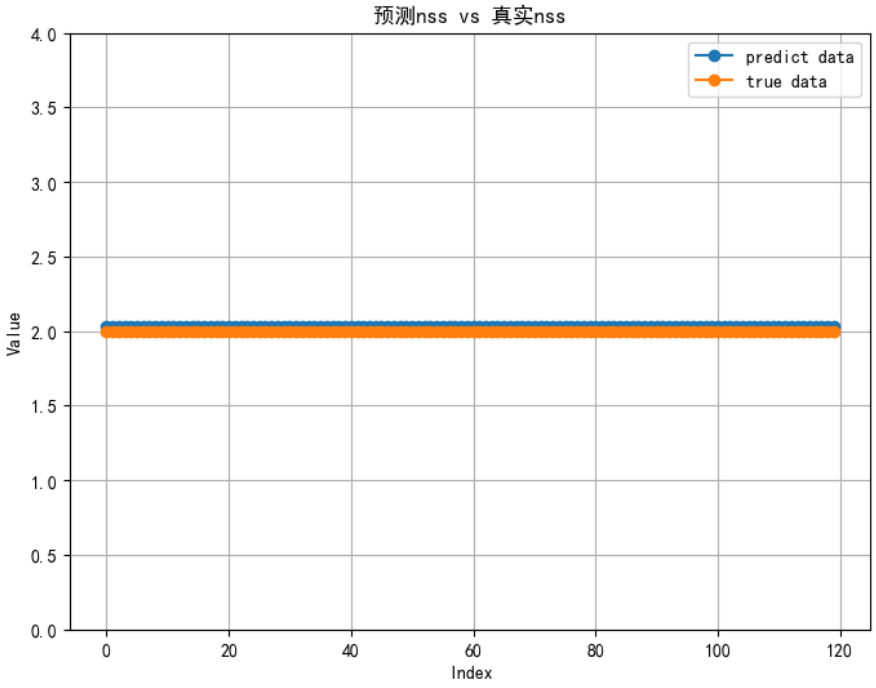

图 **5-5**　问题二 **ap2+ap3** 统一预测模型预测 **nss** 与实际 **nss** 比较

针对补全两个 excel 文件的（MCS，NSS）列属性需求，我们将统一模型应用于测试集提供的删减后数据，并将得到的结果写入 excel，其结果具体见附件文件。

6. 问题三的分析与求解

6.1 问题分析

为了准确预测 AP 节点的吞吐量，我们结合了对网络静态图结构和动态时序信息，分析影响 AP 节点吞吐量的多种关键因素。

AP 节点的吞吐量与 WLAN 网络的静态图结构密切相关。网络的静态图结构指的是网络中的固定特性和配置参数，这些参数在一定时间内相对稳定。AP 节点与 STA 节点之间的带宽、流量、发送功率、信道接入机制、节点位置、信噪比、接入门限值等信息都会对吞吐量产生直接影响。（MCS，NSS）的值确定了 AP 在发送下行数据时的 PHY Rate 以及数据帧的比特数，决定了能够传输的数据速率；AP 的发送机会，即发送帧序列的总时长，决定 AP 在一次测试中能够发送的时间。信道接入机制影响了多个节点如何共享信道资源，信噪比反映了信号质量与干扰的比例。这些信息反映通信质量，影响 AP 节点能够处理的数据量，是影响 AP 节点吞吐量的重要因素。

AP 节点的吞吐量与 WLAN 网络的动态时序信息密切相关。节点间随时间变动的干扰是影响 AP 节点吞吐量的关键动态因素，不仅影响 AP 节点和 STA 节点之间的通信质量，还导致信道资源的竞争加剧，从而降低吞吐量。另一个动态因素是网络中节点的负载变化。这些时序上的波动都会对 AP 节点的吞吐量产生显著影响。

6.2 问题求解

6.2.1 统一建模与独立建模

与问题一类似，我们在 ap2+ap3、ap2、ap3 三个数据集上分别训练预测吞吐量图网络模型。训练过程的 loss 函数随训练轮数变化如下：

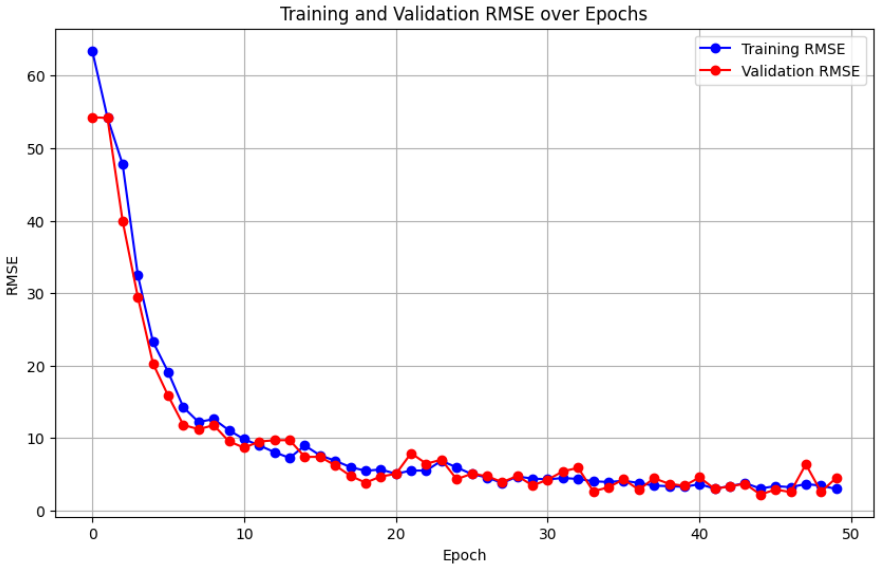

图 **6-1**　问题三 **ap2+ap3** 统一预测模型 **loss** 函数随训练轮数变化

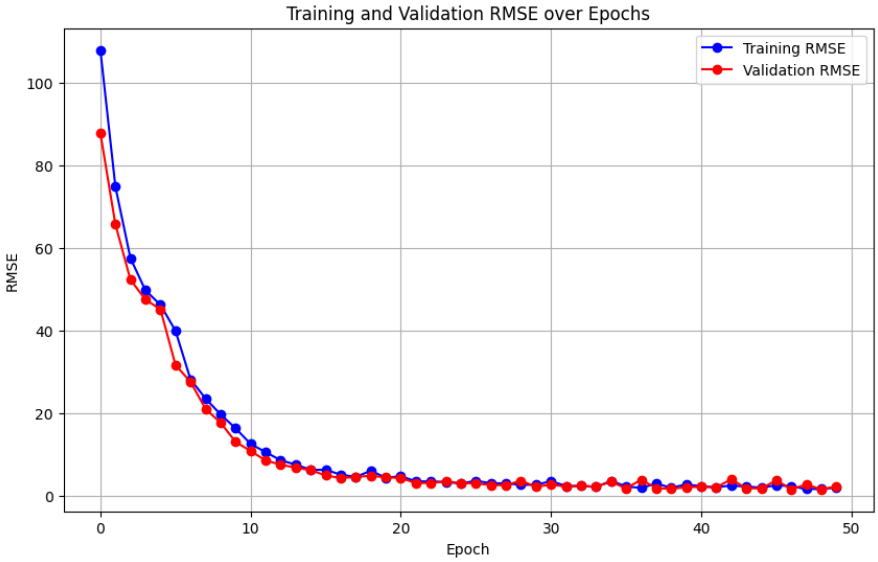

图 **6-2**　问题三 **ap2** 独立预测模型 **loss** 函数随训练轮数变化

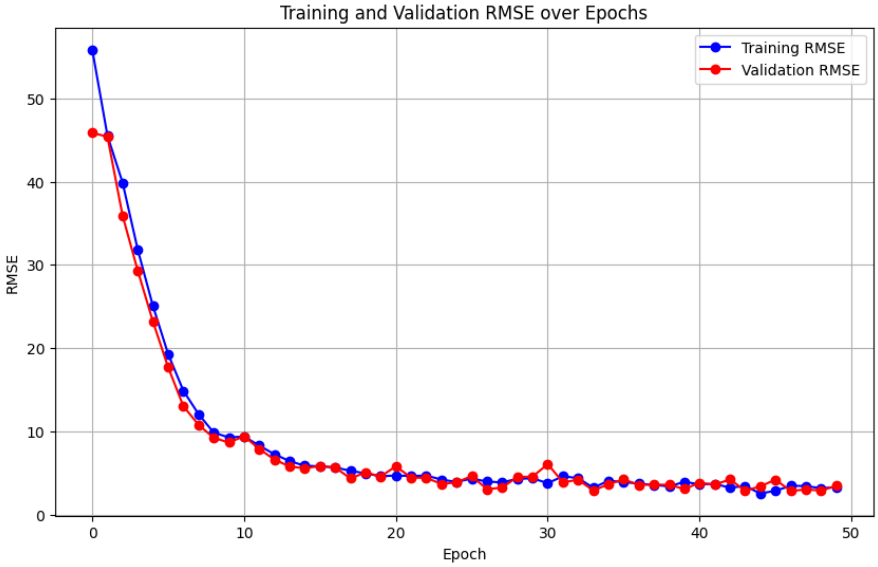

图 **6-3**　问题三 **ap3** 独立预测模型 **loss** 函数随训练轮数变化

6.2.2 建模结果

在测试集上进行交叉验证，评估该预测模型的性能，得到的 RMSE 和相对误差如下：

表 **13**　问题三 **ap2+ap3** 统一预测模型在所有测试集上评估的性能结果

评估参数	ap2+ap3	ap2	ap3
RMSE	2.2549	2.0955	2.2158
相对误差	2.41%	1.41%	2.82%

表 **14**　问题三 **ap2** 独立预测模型在所有测试集上评估的性能结果

评估参数	ap2+ap3	ap2	ap3
RMSE	39.0559	1.5765	56.2437
相对误差	73.72%	1.11%	107.01%

表 **15**　问题三 **ap3** 独立预测模型在所有测试集上评估的性能结果

评估参数	ap2+ap3	ap2	ap3
RMSE	29.4258	87.5359	2.7971
相对误差	2.41%	56.21%	3.16%

根据表格显示，针对吞吐量预测问题，独立模型的最大相对误差均高于 50%，这种误差是不可接受的；而统一模型在三个数据集上的最大相对误差均小于 3%，具有极高的精度。因此，在吞吐量预测问题上，我们选择统一模型作为预测模型。针对补全两个 excel 文件的（MCS，NSS）列属性需求，我们将对应的独立模型应用于测试集提供的删减后数据，并将得到的结果写入 excel，其结果具体见附件文件。

进一步对 ap2+ap3 统一预测模型的预测吞吐量进行分析，发现其预测值与真实值相差较小：

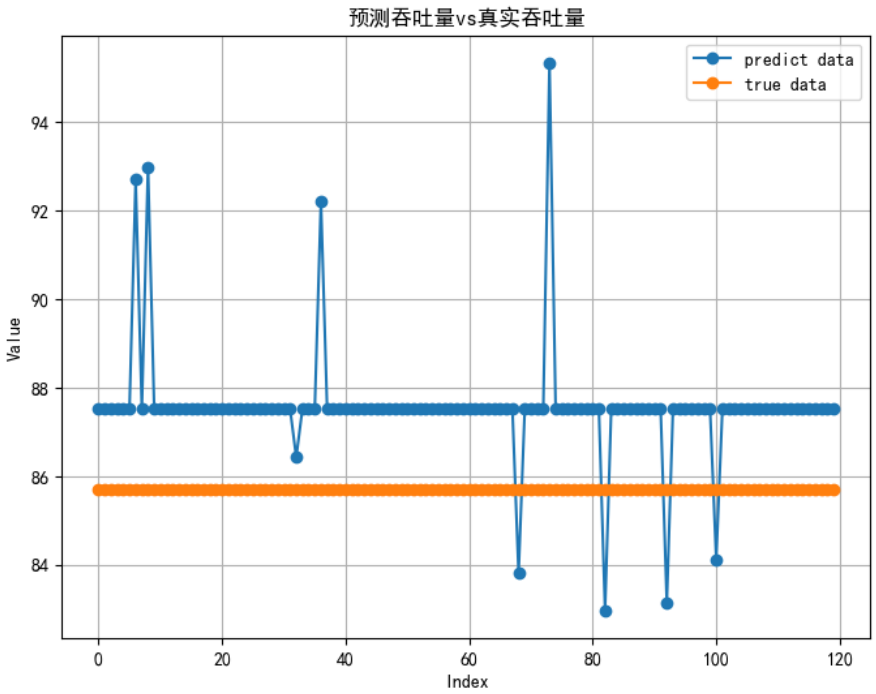

图 **6-4**　问题三 **ap2+ap3** 统一预测模型预测吞吐量与实际吞吐量比较

针对补全两个 excel 文件的吞吐量列属性需求，我们将模型应用于测试集提供的删减后数据，并将得到的结果写入 excel，其结果具体见附件文件。

7. 模型的评价

7.1 创新点

相对于传统的图神经网络，我们进行了如下的创新：

(1) 异构性处理：由于 WLAN 部署图是异构的，包含不同类型的节点（如 AP 和 STA）和边（如 AP-STA 和 AP-AP），在处理这些异构关系时采用了额外的聚合函数和结合函数，以确保模型能够区分不同类型的邻居并充分利用它们的特征；
(2) 注意力机制：与 GAT 中的注意力机制类似，我们在神经网络训练的消息传递层中引入了这一机制，以在计算邻居节点信息时给予不同邻居不同的权重；
(3) 时序建模：为了捕捉 WLAN 部署中的时序信息，我们在多个卷积层之后添加了 LSTM（长短期记忆网络）层。这使得神经网络能够处理序列数据，并对 WLAN 部署中的动态变化进行建模；
(4) 过平滑问题处理：为了避免深层 GNN 中常见的过平滑问题，我们采用了 JK-Net 的残差连接策略，在每个卷积层之后添加残差连接，以帮助模型保留更多的节点特征信息。

7.2 模型的优点

(1) 模型基于节点间的 RSSI 信号强度及门限信息，将不同时刻的 WLAN 组网抽象为不同的异构图，以充分利用图神经网络挖掘组网信息的能力。同时，结合长短期记忆网络（LSTM）在时间序列数据中捕捉和保存长期信息的优势，实现了吞吐量的精准预测；
(2) 模型在图神经网络的基础上，基于 AP 间 RSSI 信号强度和门限信息，考虑了 AP 间 RSSI 强度对传输方式的影响，以及 AP 间传输方式对 STA 节点接收数据时 SINR 的影响，使得模型对不同时刻 WLAN 组网状态有更细致的了解。

7.3 模型的缺点

(1) 模型消息传递机制较为复杂，可能会导致更长的推理时间。虽然现有的模型框架的推理时间能满足各种下游应用的需求，但复杂的消息传递机制仍然增加了计算负担；
(2) 当信道配置动态分配时，需要结合结构信息和时间信息来生成准确的吞吐量预测。这表明我们的模型在处理动态配置时表现更好，但在静态配置下可能不如其他方法。

8. 参考文献

[1] Farhad A, Pyun J Y. Resource management for massive internet of things in IEEE 802.11 ah wlan: Potentials, current solutions, and open challenges[J]. Sensors, 2022, 22(23): 9509.
[2] Zhou H, Kannan R, Swami A, et al. HTNet: Dynamic WLAN Performance Prediction using Heterogenous Temporal GNN[C]//IEEE INFOCOM 2023-IEEE Conference on Computer Communications. IEEE, 2023: 1-10.
[3] Xu K, Li C, Tian Y, et al. Representation learning on graphs with jumping knowledge networks[C]//International conference on machine learning. PMLR, 2018: 5453-5462.
[4] Kamiński, Kamil, et al. ”Rossmann-toolbox: a deep learning-based protocol for the prediction and design of cofactor specificity in Rossmann fold proteins.”

Briefings in bioinformatics 23.1 (2022): bbab371.
[5] 华为. (n.d.). 功率和信号强度 - WLAN 网络规划指导. Retrieved September 25, 2024, from https://support.huawei.com/enterprise/zh/doc/EDOC1000113314/c3242b10
[6] Shi X, Chen Z, Wang H, et al. Convolutional LSTM network: A machine learning approach for precipitation nowcasting[J]. Advances in neural information processing systems, 2015, 28.
[7] Xu, Keyulu, et al. "How powerful are graph neural networks?." arXiv preprint arXiv:1810.00826 (2018).
[8] Dai, Jiuqian, et al. "DGNN: Denoising graph neural network for session-based recommendation." 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA). IEEE, 2022.

附录 A 训练模型代码 train.py

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--gpu', type=int, default=0, help='which gpu to
    use')
parser.add_argument('--data', type=str, help='which dataset to use')
parser.add_argument('--batch_size', type=int, default=1, help='batch
    size')
parser.add_argument('--num_snapshot', type=int, default=10,
    help='number of snapshot')
parser.add_argument('--graph', action='store_true', help='whether to
    use graph information')
parser.add_argument('--hetero', action='store_true', help='whether to
    treat as heterogeneous graph')
parser.add_argument('--dynamic', action='store_true', help='whether to
    use dynamic information')
parser.add_argument('--layer', type=int, default=2, help='number of
    layers')
parser.add_argument('--dim', type=int, default=128, help='hidden
    dimension')
parser.add_argument('--lr', type=float, default=0.001, help='learning
    rate')
parser.add_argument('--epoch', type=int, default=50, help='number of
    epochs')
parser.add_argument('--output', type=str, default='', help='save
    predictions to files')
parser.add_argument('--path', type=str, default='', help='save
    predictions to files')
parser.add_argument('--save_name', type=str, default='', help='save
    predictions to files')
args = parser.parse_args()



import os
os.environ['CUDA_VISIBLE_DEVICES'] = '3'

import torch
import dgl
import pickle
import xgboost
import time
import numpy as np
from minibatch import get_dataloader
from model import NetModel
from datetime import datetime
from sklearn import linear_model

t_inf = 0


train_dataloader, valid_dataloader, test_dataloader =
```

```
    get_dataloader(args.path, args.batch_size, all_cuda=True)

model = NetModel(args.layer, args.dim, args.graph, args.hetero,
    args.dynamic, args.num_snapshot).cuda()
optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)

def eval(model, dataloader, output=''):
    global t_inf
    if output != '':
        output_fn = 'output/{}.pkl'.format(output)
        if not os.path.exists(output_fn):
            os.makedirs(os.path.dirname(output_fn))
        outs = dict()
    model.eval()
    rmse = list()
    rmse_tot = 0
    with torch.no_grad():
        for g, idx in dataloader:
            t_s = time.time()
            g = g.to('cuda')
            mask = g.nodes['ap'].data['mask'] > 0
            mask = mask.view(-1)
            pred = model(g)[mask]
            # true =
                g.nodes['ap'].data['predict'][mask][:,1:3].view(-1,2) #
                MCS,NSS
            # true = g.nodes['ap'].data['predict'][mask][:,7].view(-1,1)
                # seq_time
            true = g.nodes['ap'].data['predict'][mask][:,8].view(-1,1) #
                吞吐量
            loss = torch.sqrt(torch.nn.functional.mse_loss(pred, true) +
                1e-8)
            rmse.append(float(loss) * pred.shape[0])
            rmse_tot += pred.shape[0]
            t_inf += time.time() - t_s
            if output != '':
                out = {'pred':pred.cpu().detach().numpy(),
                    'true':true.cpu().detach().numpy()}
                outs[int(idx)] = out
    if output != '':
        with open(output_fn, 'wb') as f:
            pickle.dump(outs, f)
    return np.sum(np.array(rmse)) / rmse_tot

# torch.autograd.set_detect_anomaly(True)
best_e = 0
best_valid_rmse = float('inf')
model_fn = 'models/{}.pkl'.format(args.save_name)
if not os.path.exists('models'):
    os.mkdir('models')
for e in range(args.epoch):
    model.train()
    train_rmse = list()
    rmse_tot = 0
```

```
    for g, _ in train_dataloader:
        g = g.to('cuda')
        optimizer.zero_grad()
        mask = g.nodes['ap'].data['mask'] > 0
        mask = mask.view(-1)
        # mask = mask.unsqueeze(1)
        pred = model(g)[mask]
        # true = g.nodes['ap'].data['predict'][mask][:,1:3].view(-1,2) # MCS,NSS
        # true = g.nodes['ap'].data['predict'][mask][:,7].view(-1,1) # seq_time
        true = g.nodes['ap'].data['predict'][mask][:,8].view(-1,1) # 吞吐量
        # print(pred.shape)
        # print(true.shape)
        loss = torch.sqrt(torch.nn.functional.mse_loss(pred, true) + 1e-8)
        train_rmse.append(float(loss) * pred.shape[0])
        rmse_tot += pred.shape[0]
        # with torch.autograd.detect_anomaly():
        loss.backward()
        optimizer.step()
    train_rmse = np.sum(np.array(train_rmse)) / rmse_tot
    valid_rmse = eval(model, valid_dataloader)
    print('Epoch: {} Training RMSE: {:.4f} Validation RMSE: {:.4f}'.format(e, train_rmse, valid_rmse))
    if valid_rmse < best_valid_rmse:
        best_e = e
        best_valid_rmse = valid_rmse
        torch.save(model.state_dict(), model_fn)

print('Loading model in epoch {}...'.format(best_e))
model.load_state_dict(torch.load(model_fn))
print('Test RMSE: {:.4f}'.format(eval(model, test_dataloader, output=args.output)))
```

附录 B 网络模型代码 model.py

```
import torch
import dgl
import dgl.nn.pytorch as dglnn
from torch import nn
from torch.nn import init

class NetModel(torch.nn.Module):

    def __init__(self, num_layer, dim, is_graph, is_hetero, is_dynamic, num_snapshot):
        super(NetModel, self).__init__()
        self.num_layer = num_layer
        self.is_graph = is_graph
        self.is_hetero = is_hetero
```

```
        self.is_dynamic = is_dynamic
        self.is_jk = True if is_hetero else False
        self.num_snapshot = num_snapshot

        mods = dict()
        dim_node_in = 10
        dim_edge_in = 5
        dim_tot = 17
        for l in range(self.num_layer):
            conv_dict = dict()
            conv_dict['ap-ap'] = EGATConv(dim_node_in, dim_edge_in, dim,
                dim, 1)
            conv_dict['ap-sta'] = EGATConv(dim_node_in, dim_edge_in, dim,
                dim, 1)
            conv_dict['sta-ap'] = EGATConv(dim_node_in, dim_edge_in, dim,
                dim, 1)
            mods['nn' + str(l)] = torch.nn.BatchNorm1d(dim_node_in,
                track_running_stats=False)
            mods['ne' + str(l)] = torch.nn.BatchNorm1d(dim_edge_in,
                track_running_stats=False)
            mods['l' + str(l)] = EHeteroGraphConv(conv_dict)
            dim_node_in = dim
            dim_edge_in = dim
            dim_tot += dim
        mods['comb'] = Perceptron(20 + dim*num_layer, dim)
        mods['rnn'] = torch.nn.LSTM(dim, dim, 2)
        mods['predict'] = Perceptron(dim, 2, act=False)
        mods['softplus'] = torch.nn.Softplus()
        self.mods = torch.nn.ModuleDict(mods)

    def forward(self, g):
        # h = [g.nodes['sta'].data['feat'],
            g.edges['sta-ap'].data['feat']]
        h = [g.nodes['sta'].data['feat'], g.nodes['ap'].data['feat']]
        h_node = {'ap':g.nodes['ap'].data['feat'],
            'sta':g.nodes['sta'].data['feat']}
        h_edge = {'ap-ap':g.edges['ap-ap'].data['feat'],
            'ap-sta':g.edges['ap-sta'].data['feat'],
            'sta-ap':g.edges['sta-ap'].data['feat']}
        for l in range(self.num_layer):
            h_node['ap'] = self.mods['nn' + str(l)](h_node['ap'])
            h_node['sta'] = self.mods['nn' + str(l)](h_node['sta'])
            h_edge['ap-ap'] = self.mods['ne' + str(l)](h_edge['ap-ap'])
            h_edge['ap-sta'] = self.mods['ne' + str(l)](h_edge['ap-sta'])
            h_edge['sta-ap'] = self.mods['ne' + str(l)](h_edge['sta-ap'])
            h_node, h_edge = self.mods['l' + str(l)](g, h_node, h_edge)
            h.append(h_node['sta'])
        h = self.mods['comb'](torch.cat(h, dim=1)) if self.is_jk else
            h[-1]
        if self.is_dynamic:
            h = h.view((self.num_snapshot, -1, h.shape[-1]))
            h = self.mods['rnn'](h)[0].view(-1, h.shape[-1])
        h = self.mods['predict'](h)
        # to ensure positive prediction
```

```
        h = self.mods['softplus'](h)
        return h

class Perceptron(torch.nn.Module):

    def __init__(self, in_dim, out_dim, dropout=0, norm=False,
        act=True):
        super(Perceptron, self).__init__()
        self.weight = torch.nn.Parameter(torch.empty(in_dim, out_dim))
        torch.nn.init.xavier_uniform_(self.weight.data)
        self.bias = torch.nn.Parameter(torch.empty(out_dim))
        torch.nn.init.zeros_(self.bias.data)
        self.norm = norm
        if norm:
            self.norm = torch.nn.BatchNorm1d(out_dim, eps=1e-9,
                track_running_stats=True)
        self.dropout = torch.nn.Dropout(dropout)
        self.act = act

    def forward(self, f_in):
        f_in = self.dropout(f_in)
        f_in = torch.mm(f_in, self.weight) + self.bias
        if self.act:
            f_in = torch.nn.functional.relu(f_in)
        if self.norm:
            f_in = self.norm(f_in)
        return f_in

    def reset_parameters():
        torch.nn.init.xavier_uniform_(self.weight.data)
        torch.nn.init.zeros_(self.bias.data)

class EGATConv(nn.Module):

    def __init__(self,
                 in_node_feats,
                 in_edge_feats,
                 out_node_feats,
                 out_edge_feats,
                 num_heads,
                 bias=True):

        super().__init__()
        self._num_heads = num_heads
        self._out_node_feats = out_node_feats
        self._out_edge_feats = out_edge_feats
        self.fc_node = nn.Linear(in_node_feats,
            out_node_feats*num_heads, bias=True)
        self.fc_ni = nn.Linear(in_node_feats, out_edge_feats*num_heads,
            bias=False)
        self.fc_fij = nn.Linear(in_edge_feats, out_edge_feats*num_heads,
            bias=False)
        self.fc_nj = nn.Linear(in_node_feats, out_edge_feats*num_heads,
            bias=False)
```

```
106         self.attn = nn.Parameter(torch.FloatTensor(size=(1, num_heads,
                out_edge_feats)))
107         if bias:
108             self.bias = nn.Parameter(torch.FloatTensor(size=(num_heads *
                    out_edge_feats,)))
109         else:
110             self.register_buffer('bias', None)
111         self.reset_parameters()
112 
113     def reset_parameters(self):
114         gain = init.calculate_gain('relu')
115         init.xavier_normal_(self.fc_node.weight, gain=gain)
116         init.xavier_normal_(self.fc_ni.weight, gain=gain)
117         init.xavier_normal_(self.fc_fij.weight, gain=gain)
118         init.xavier_normal_(self.fc_nj.weight, gain=gain)
119         init.xavier_normal_(self.attn, gain=gain)
120         init.constant_(self.bias, 0)
121 
122     def forward(self, graph, nfeats, efeats, get_attention=False):
123         with graph.local_scope():
124             f_ni = self.fc_ni(nfeats)
125             f_nj = self.fc_nj(nfeats)
126             f_fij = self.fc_fij(efeats)
127             graph.srcdata.update({'f_ni': f_ni})
128             graph.dstdata.update({'f_nj': f_nj})
129             # add ni, nj factors
130             graph.apply_edges(dgl.function.u_add_v('f_ni', 'f_nj',
                    'f_tmp'))
131             # add fij to node factor
132             f_out = graph.edata.pop('f_tmp') + f_fij
133             if self.bias is not None:
134                 f_out = f_out + self.bias
135             f_out = nn.functional.leaky_relu(f_out)
136             f_out = f_out.view(-1, self._num_heads, self._out_edge_feats)
137             # compute attention factor
138             e = (f_out * self.attn).sum(dim=-1).unsqueeze(-1)
139             graph.edata['a'] = dglnn.edge_softmax(graph, e)
140             graph.ndata['h_out'] = self.fc_node(nfeats).view(-1,
                    self._num_heads,
141                                                              self._out_node_feats)
142             # calc weighted sum
143             graph.update_all(dgl.function.u_mul_e('h_out', 'a', 'm'),
144                              dgl.function.sum('m', 'h_out'))
145 
146             h_out = graph.ndata['h_out'].view(-1, self._num_heads,
                    self._out_node_feats)
147             if get_attention:
148                 return h_out, f_out, graph.edata.pop('a')
149             else:
150                 return h_out.view(-1, self._out_node_feats),
                        f_out.view(-1, self._out_edge_feats)
151 
152 class EHeteroGraphConv(nn.Module):
153 
```

```
    def __init__(self, mods):
        super(EHeteroGraphConv, self).__init__()
        self.mods = nn.ModuleDict(mods)
        for _, v in self.mods.items():
            set_allow_zero_in_degree_fn = getattr(v,
                'set_allow_zero_in_degree', None)
            if callable(set_allow_zero_in_degree_fn):
                set_allow_zero_in_degree_fn(True)

    def forward(self, g, n_inputs, e_inputs):
        n_outputs = {nty : [] for nty in g.dsttypes}
        e_outputs = {}
        for stype, etype, dtype in g.canonical_etypes:
            rel_graph = g[stype, etype, dtype]
            if rel_graph.number_of_edges() == 0:
                continue
            if stype not in n_inputs:
                continue
            if stype != dtype:
                h_n = torch.cat([n_inputs[stype], n_inputs[dtype]], dim=0)
            else:
                h_n = n_inputs[stype]
            dstdata, e_output =
                self.mods[etype](dgl.to_homogeneous(rel_graph), h_n,
                e_inputs[etype])
            if stype != dtype:
                dstdata = dstdata[n_inputs[stype].shape[0]:]
            n_outputs[dtype].append(dstdata)
            e_outputs[etype] = e_output
        n_rsts = {}
        if False:
            print("n_outputs: " + str(len(n_outputs["ap"])) + " " +
                str(len(n_outputs["sta"])))
            print("n_outputs: " + str(n_outputs["ap"][0].shape) + " " +
                str(n_outputs["sta"][0].shape))
            print("e_outputs: " + str(e_outputs["ap-ap"].shape) + " " +
                str(e_outputs["ap-sta"].shape) + " " +
                str(e_outputs["sta-ap"].shape))
        for nty, alist in n_outputs.items():
            if len(alist) != 0:
                n_rsts[nty] = torch.mean(torch.stack(alist), dim=0)
        return n_rsts, e_outputs
```

附录 C 第一问预测 seq_time 代码 predict_seq_time.py

```
import argparse
parser.add_argument('--output', type=str, default='', help='save
    predictions to files')
parser.add_argument('--path', type=str, default='', help='save
    predictions to files')
parser.add_argument('--save_name', type=str, default='', help='save
    predictions to files')
```

```
args = parser.parse_args()



import os
os.environ['CUDA_VISIBLE_DEVICES'] = '4'

import torch
import dgl
import pickle
import xgboost
import time
import numpy as np
from minibatch import get_dataloader
from model import NetModel
from datetime import datetime
from sklearn import linear_model
import pandas as pd
model_fn =
    "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/seq_time_ap2_ap3.pkl"
# model_fn =
    "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/seq_time_ap3.pkl"
# model_fn =
    "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/seq_time_ap2.pkl"
t_inf = 0


train_dataloader, valid_dataloader, test_dataloader =
    get_dataloader(args.path, args.batch_size, all_cuda=True)
model = NetModel(args.layer, args.dim, args.graph, args.hetero,
    args.dynamic, args.num_snapshot).cuda()

def eval(model, dataloader, output=''):
    global t_inf
    if output != '':
        output_fn = 'output/{}'.format(output)
    model.eval()
    rmse = list()
    rmse_tot = 0
    with torch.no_grad():

        cat_pred = torch.zeros(78, 1).to("cuda:0")
        cat_true = torch.zeros(78, 1).to("cuda:0")

        count = 0
        res = torch.zeros(78, 1)
        res = res.to('cuda:0')
        for g, idx in dataloader:
            count += 1
            t_s = time.time()
            g = g.to('cuda')
            mask = g.nodes['ap'].data['mask'] > 0
            mask = mask.view(-1)
            pred = model(g)[mask]
```

```
            res += pred
            true = g.nodes['ap'].data['predict'][mask][:,7].view(-1,1) # seq_time
            # true = g.nodes['ap'].data['predict'][mask][:,8].view(-1,1) # 吞吐量
            loss = torch.sqrt(torch.nn.functional.mse_loss(pred, true) + 1e-8)
            rmse.append(float(loss) * pred.shape[0])
            rmse_tot += pred.shape[0]
            t_inf += time.time() - t_s

            cat_pred = torch.cat((cat_pred, pred), dim=1)
            cat_true = torch.cat((cat_true, true), dim=1)
        # 将预测和真实值写入文件
        res /= count
        with open(output_fn, 'w') as f:
            for p in zip(res.cpu().detach().numpy()):
                f.write(f"{p[0][0]}\n") # 将预测值和真实值写入文件
        print(count)

        cat_pred = cat_pred.cpu().numpy()
        cat_true = cat_true.cpu().numpy()
        # np.savetxt('cat_pred.txt', cat_pred, delimiter='\t')
        # np.savetxt('cat_true.txt', cat_true, delimiter='\t')
        df = pd.DataFrame(cat_pred)
        df.to_csv('seq_pred.txt', sep='\t', index=False, float_format='%.6f')
        df = pd.DataFrame(cat_true)
        df.to_csv('seq_true.txt', sep='\t', index=False, float_format='%.6f')
    return np.sum(np.array(rmse)) / rmse_tot

print('Loading model in epoch...')
model.load_state_dict(torch.load(model_fn))
print('Test RMSE: {:.4f}'.format(eval(model, train_dataloader, output=args.output)))
```

附录 D 第二问预测 mcs 和 nss 代码 predict_mcs_nss.py

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--output', type=str, default='', help='save predictions to files')
parser.add_argument('--path', type=str, default='', help='save predictions to files')
parser.add_argument('--save_name', type=str, default='', help='save predictions to files')
args = parser.parse_args()


import os
os.environ['CUDA_VISIBLE_DEVICES'] = '4'
```

```
11
12  import torch
13  import dgl
14  import pickle
15  import xgboost
16  import time
17  import pandas as pd
18  import numpy as np
19  from minibatch import get_dataloader
20  from model import NetModel
21  from datetime import datetime
22  from sklearn import linear_model
23  model_fn =
        "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/mcs_nss_ap2_ap3.pkl"
24  # model_fn =
        "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/mcs_nss_ap3.pkl"
25  t_inf = 0
26
27
28  train_dataloader, valid_dataloader, test_dataloader =
        get_dataloader(args.path, args.batch_size, all_cuda=True)
29  model = NetModel(args.layer, args.dim, args.graph, args.hetero,
        args.dynamic, args.num_snapshot).cuda()
30
31  def eval(model, dataloader, output=''):
32      global t_inf
33      if output != '':
34          output_fn = 'output/{}'.format(output)
35          # if not os.path.exists(output_fn):
36          #     os.makedirs(os.path.dirname(output_fn))
37          outs = dict()
38      model.eval()
39      rmse = list()
40      rmse_tot = 0
41
42      with torch.no_grad():
43
44          cat_pred = torch.zeros(78, 2).to("cuda:0")
45          cat_true = torch.zeros(78, 2).to("cuda:0")
46
47          count = 0
48          sum_error = 0
49
50          res = torch.zeros(78, 2)
51          res = res.to('cuda:0')
52          for g, idx in dataloader:
53              count += 1
54              t_s = time.time()
55              g = g.to('cuda')
56              mask = g.nodes['ap'].data['mask'] > 0
57              mask = mask.view(-1)
58              pred = model(g)[mask]
59              res += pred
60              true = g.nodes['ap'].data['predict'][mask][:,1:3].view(-1,2)
```

```
            # MCS,NSS
        # true = g.nodes['ap'].data['predict'][mask][:,7].view(-1,1)
            # seq_time
        # true = g.nodes['ap'].data['predict'][mask][:,8].view(-1,1)
            # 吞吐量
        loss = torch.sqrt(torch.nn.functional.mse_loss(pred, true) +
            1e-8)
        # print(pre)
        absolute_error = torch.abs(pred - true)/true * 100
        sum_error += absolute_error.sum(dim=0)/absolute_error.shape[0]
        count += 1

        rmse.append(float(loss) * pred.shape[0])
        rmse_tot += pred.shape[0]
        t_inf += time.time() - t_s

        cat_pred = torch.cat((cat_pred, pred), dim=1)
        cat_true = torch.cat((cat_true, true), dim=1)
    # 将预测和真实值写入文件
    res /= count
    with open(output_fn, 'w') as f:
        for p in zip(res.cpu().detach().numpy()):
            f.write(f"{np.round(p[0])}\n") # 将预测值和真实值写入文件
    print(count)
    print(str(sum_error/count) + "%")
    cat_pred = cat_pred.cpu().numpy()
    cat_true = cat_true.cpu().numpy()
    # np.savetxt('cat_pred.txt', cat_pred, delimiter='\t')
    # np.savetxt('cat_true.txt', cat_true, delimiter='\t')
    df = pd.DataFrame(cat_pred)
    df.to_csv('mcs_nss_pred.txt', sep='\t', index=False,
        float_format='%.6f')
    df = pd.DataFrame(cat_true)
    df.to_csv('mcs_nss_true.txt', sep='\t', index=False,
        float_format='%.6f')
return np.sum(np.array(rmse)) / rmse_tot

print('Loading model in epoch...')
model.load_state_dict(torch.load(model_fn))
print('Test RMSE: {:.4f}'.format(eval(model, train_dataloader,
    output=args.output)))
```

附录 E 第三问预测吞吐量代码 predict_throughput.py

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--output', type=str, default='', help='save
    predictions to files')
parser.add_argument('--path', type=str, default='', help='save
    predictions to files')
parser.add_argument('--save_name', type=str, default='', help='save
    predictions to files')
```

```
args = parser.parse_args()

import os
os.environ['CUDA_VISIBLE_DEVICES'] = '4'

import torch
import dgl
import pickle
import pandas as pd
import xgboost
import time
import numpy as np
from minibatch import get_dataloader
from model import NetModel
from datetime import datetime
from sklearn import linear_model
model_fn =
    "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/throughput_ap2_ap3.pkl"
# model_fn =
    "/home/sxm/HomeWorkSpace/MathModel/ZB_Way/HTNet/models/throughput_ap3.pkl"
t_inf = 0


train_dataloader, valid_dataloader, test_dataloader =
    get_dataloader(args.path, args.batch_size, all_cuda=True)
model = NetModel(args.layer, args.dim, args.graph, args.hetero,
    args.dynamic, args.num_snapshot).cuda()

def eval(model, dataloader, output=''):
    global t_inf
    if output != '':
        output_fn = 'output/{}'.format(output)
    model.eval()
    rmse = list()
    rmse_tot = 0
    with torch.no_grad():
        count = 0
        res = torch.zeros(123, 1)
        cat_pred = torch.zeros(123, 1).to("cuda:0")
        cat_true = torch.zeros(123, 1).to("cuda:0")
        res = res.to('cuda:0')
        for g, idx in dataloader:
            count += 1
            t_s = time.time()
            g = g.to('cuda')
            mask = g.nodes['ap'].data['mask'] > 0
            mask = mask.view(-1)
            pred = model(g)[mask]
            res += pred
            # true = g.nodes['ap'].data['predict'][mask][:,7].view(-1,1)
                # seq_time
            true = g.nodes['ap'].data['predict'][mask][:,8].view(-1,1) #
                吞吐量
            # print(pred)
```

```
            # print(true)
            loss = torch.sqrt(torch.nn.functional.mse_loss(pred, true) +
                1e-8)
            rmse.append(float(loss) * pred.shape[0])
            rmse_tot += pred.shape[0]
            t_inf += time.time() - t_s

            cat_pred = torch.cat((cat_pred, pred), dim=1)
            cat_true = torch.cat((cat_true, true), dim=1)
        # 将预测和真实值写入文件
        res /= count
        with open(output_fn, 'w') as f:
            for p in zip(res.cpu().detach().numpy()):
                f.write(f"{p[0][0]}\n") # 将预测值和真实值写入文件
        print(count)

        cat_pred = cat_pred.cpu().numpy()
        cat_true = cat_true.cpu().numpy()
        df = pd.DataFrame(cat_pred)
        df.to_csv('cat_pred.txt', sep='\t', index=False,
            float_format='%.6f')
        df = pd.DataFrame(cat_true)
        df.to_csv('cat_true.txt', sep='\t', index=False,
            float_format='%.6f')

    return np.sum(np.array(rmse)) / rmse_tot

print('Loading model in epoch...')
model.load_state_dict(torch.load(model_fn))
print('Test RMSE: {:.4f}'.format(eval(model, train_dataloader,
    output=args.output)))
```