

2023 集成电路 EDA 设计精英挑战赛

作品设计报告

**赛题六：基于机器学习的 SoC 电源网
络静态压降预测**

参赛队 ID: *****

1 摘要

本研究旨在设计并实现一个基于机器学习的集成电路电源网络静态压降（IR Drop）预测系统。该系统采用了类图像处理技术、非监督学习聚类方法和浅层神经网络技术，整体框架基于 Python 和 PyTorch 实现。重要的算法组件包括 Word2Vec 算法用于实现聚类，将实例名称映射到名称坐标系中；pix2pix 算法用于将 17 个通道的原始数据转化为 2 个通道的预测数据；以及自主开发的浅层神经网络 YogurtPyramid，用于进一步逼近和优化预测结果。我们的模型以五种主流开源芯片（RISCY、zero-riscy、RISCY-FPU、NVDLA、Vortex）的 eff_res 数据、min_path_res 数据和 inst_power 数据为训练基础，成功预测了每个芯片内实例的 IR Drop 数据。在赛题方提供的评价指标上，模型表现出色，平均 wall time 为 69.94 秒，平均 MAE 值为 5.2784，展示了与业界先进水平相符的高精度预测能力。

关键词：机器学习、集成电路、静态压降预测、Word2Vec 聚类、Pix2Pix 预测

2 数据提取与处理部分

2.1 程序设计框架

本节介绍了我们为“基于机器学习的 SoC 电源网络静态压降预测”赛题所开发程序的架构。程序的设计旨在处理和分析大量的集成电路数据，通过机器学习模型准确预测电源网络的静态压降。

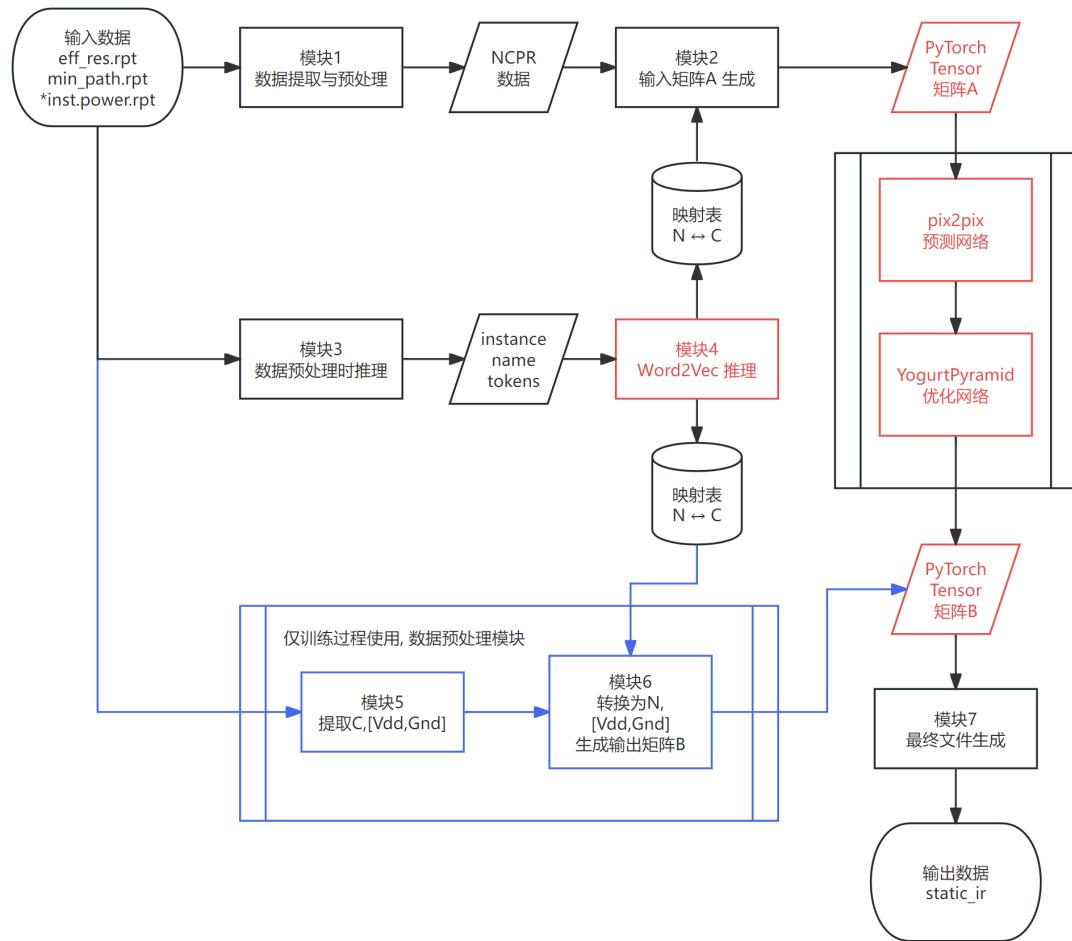


图 1 总程序框架图

2.1.1 数据提取与预处理（模块 1）

程序首先从三个关键报告文件中提取必要数据：min_path_res.rpt, eff_res.rpt, 和 pulpino_top.inst.power.net。这些报告文件分别包含有关每个 SoC 实例的最小和有效电阻值、坐标、命名以及静态和动态功耗的数据。

通过一个精细设计的提取流程，程序解析报告文件，分离出每个实例（instance）的名称（N）、坐标（C）、功耗（P）和电阻（R）信息。这些信息构成了后续机器学习模型训练的基础，也被称为“NCPR”数据。

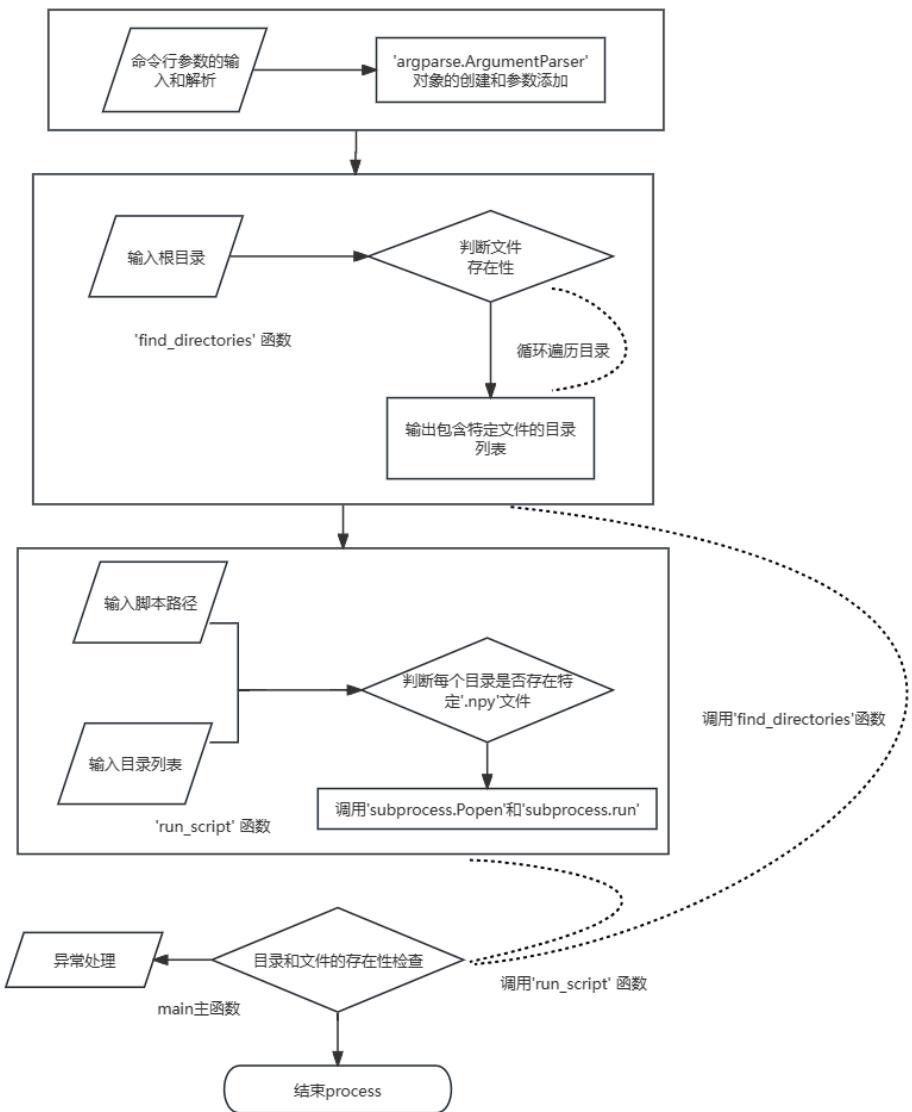


图 2 数据预处理入口流程设计

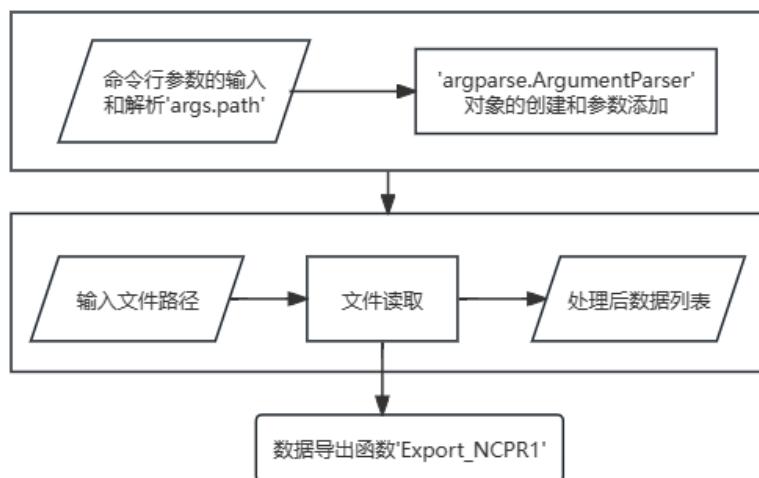


图 3 模块 1 整体程序流程设计

2.1.2 数据预处理时推理（模块 3、4）

由于我们已经发现不同芯片实例的名称的相似性与这些实例物理坐标和电气特性的相似性，因此我们创新性的采用了将实例名称映射至“名称坐标系”的方法，以实现更有规律的聚类处理。此方法将所有的实例名称转化为 256×256 等大小的整数矩阵坐标中，以更大程度避免浮点数的坐标在使用基于物理坐标系的图像处理算法造成的预测精度问题。

我们的程序运用预先训练好的 Word2Vec 算法模型，对不同芯片的每个实例名称进行预处理，推断出其在二维名称坐标系中的位置。这个推理过程产生一个映射表，以.json 格式存储，为后续的数据训练和推理提供关键的空间特征信息。

此过程我们分别使用三个脚本完成：Folder2_Voelib.py、train.py、infer.py。

```
core_region_i/instr_mem/sp_ram_wrap_i/sp_ram_bank_i  
core_region_i/instr_mem/sp_ram_wrap_i/sp_ram_bank_i  
core_region_i/data_mem/sp_ram_bank_i  
core_region_i/data_mem/sp_ram_bank_i  
FE_PHC48_uart_cts  
FE_PHC47_uart_dsr  
FE_PHC46_uart_rx  
FE_PHC37_gpio_in_6           sample:inst_names_list  
FE_PHC23_gpio_in_9  
FE_PHC10_gpio_in_4  
FE_OFC131_FE_OFN3_n1  
FE_OFC97_n1  
FE_OFC96_n1
```

图 4 数据预处理时（模块 3）结束后的效果示意，从所有芯片中提取 instance name

2.1.3 输入矩阵（A）生成（模块 2）

基于映射表和 NCPR 数据，程序构建了用于机器学习模型训练的三维多通道输入矩阵“A”。这个矩阵的前两维基于名称坐标系，而第三维则由 P 和 R 数据组成的特征通道构成。此过程使用 Convent_traindata.py 完成。

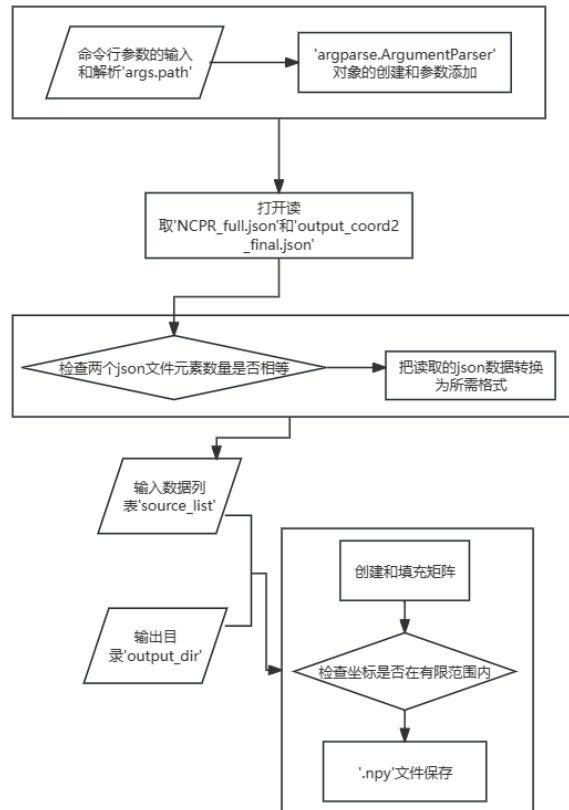


图 5 模块 2 程序流程设计

2.1.4 等待分支机制

对于每一个子芯片而言，由于程序的模块 2 需要等待模块 3 和模块 4 输出的二维坐标转换表的数据，需要预先设定轮询等待机制，如下所示：

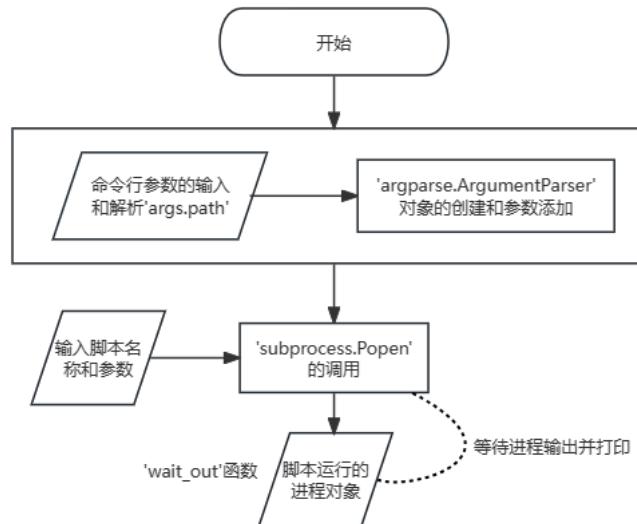


图 6 等待分支机制流程设计

2.1.5 模型训练与结果矩阵（B）输出（模块 7）

在训练阶段，程序从 static_ir 文件中提取每个实例的压降数据，生成与“A”矩阵对齐的“B”矩阵。这个“B”矩阵作为训练过程中的真实值（ground truth）参与模型训练，帮助评估和优化预测性能。

最终，神经网络模型输出一个格式与“B”矩阵相同的预测结果矩阵“B’”。该预测矩阵经过转换和处理后，生成所需的无后缀名文件 static_ir，包含每个实例的坐标、名称、电压降和地弹数据，为静态压降预测提供精确信息。

此过程我们使用一个脚本 utils_postprocess.py 完成。

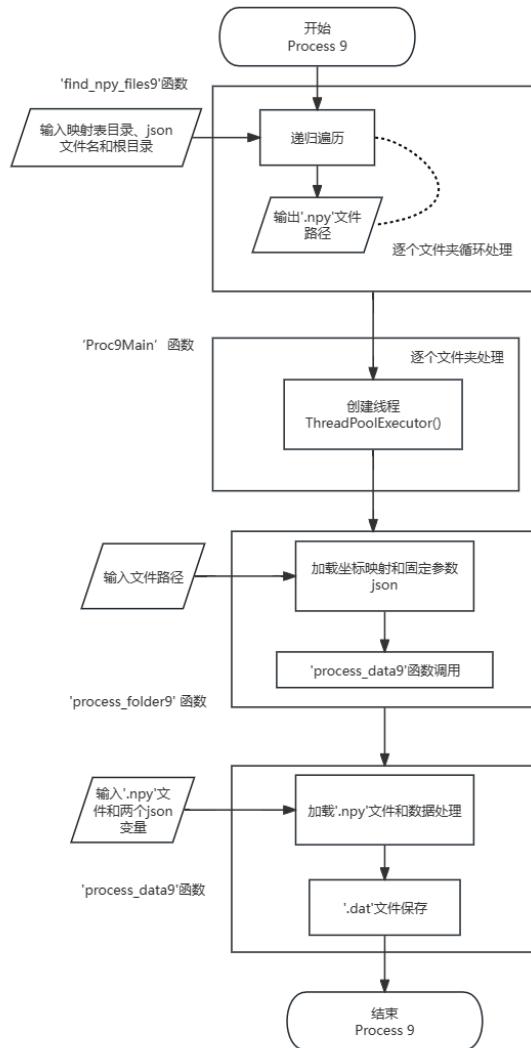


图 7 模块 7 的流程实现

2.1.6 仅训练过程使用的预处理部分（模块 5、6）

模块 5、6 仅在我们的模型训练过程中使用，用于生成数据集和测试集的输出矩阵（B）。在我们的程序代码中分别被命名为 Process6.py 和 Process7.py。具体流程图如下所示：

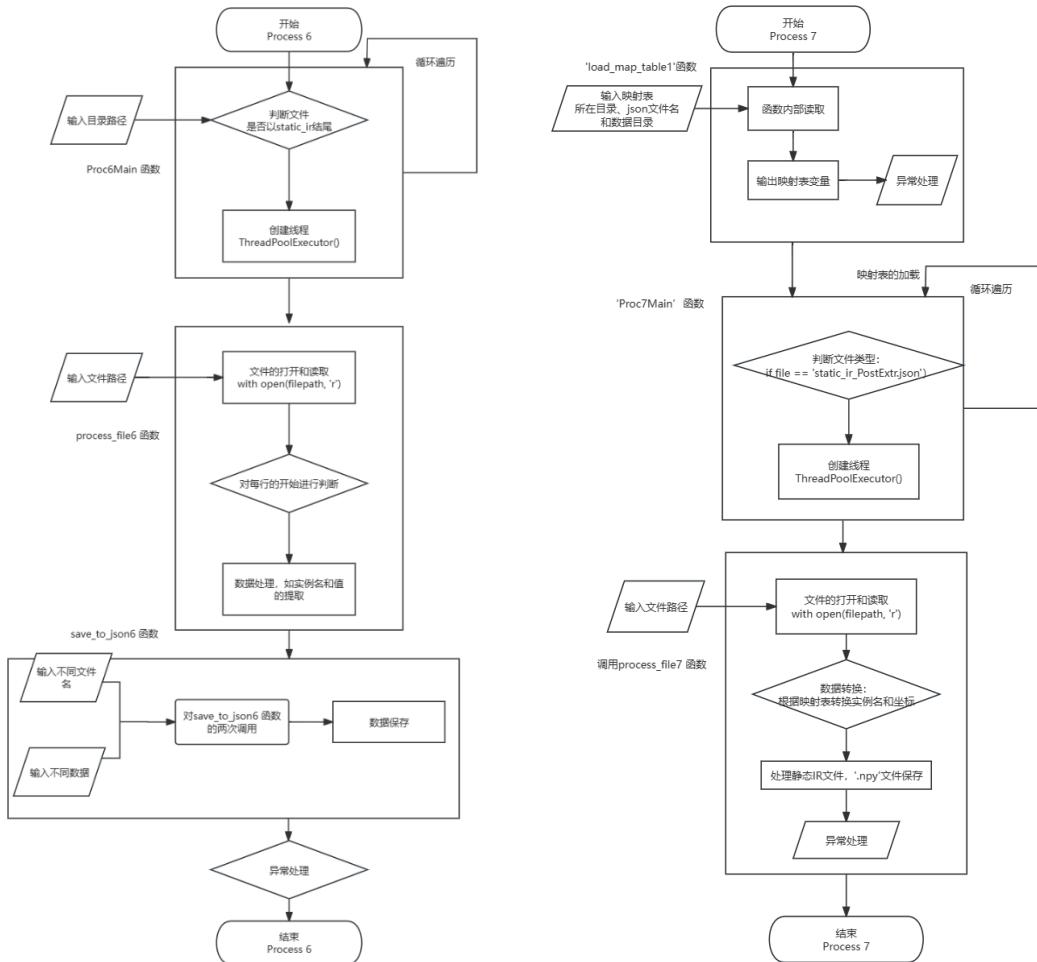


图 8 Process6.py (模块 5) 流程图

图 9 Process7.py (模块 6) 流程图

最终汇总，可以得到如下关于数据预处理部分的流程：

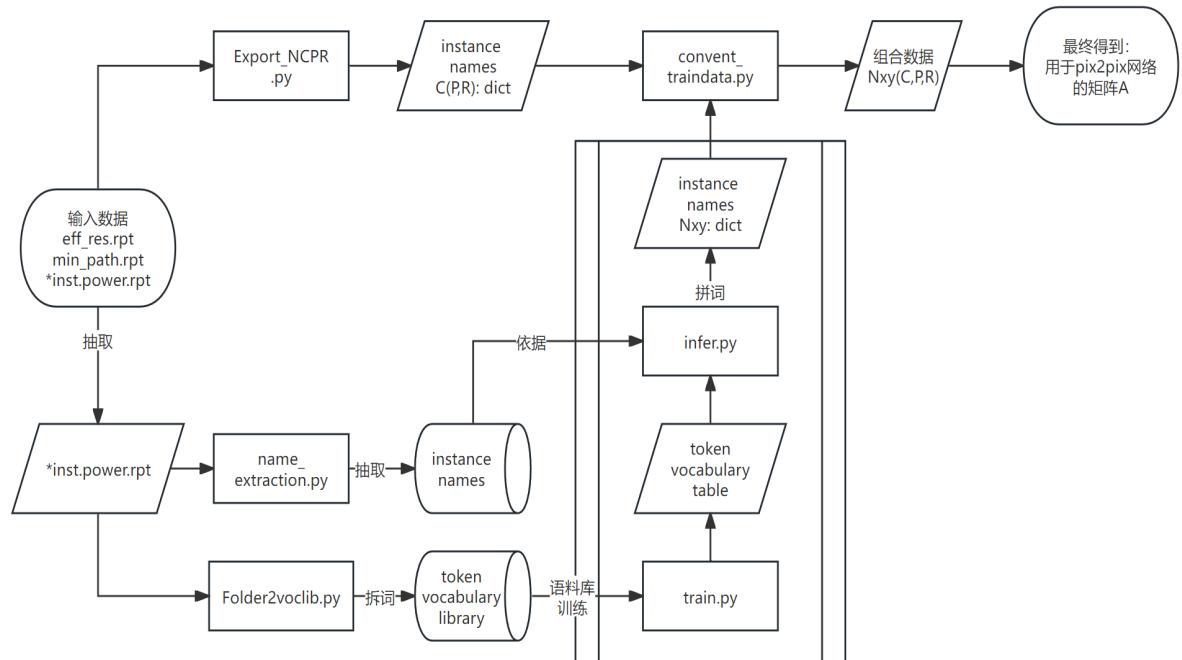


图 10 数据预处理部分总体逻辑流程图

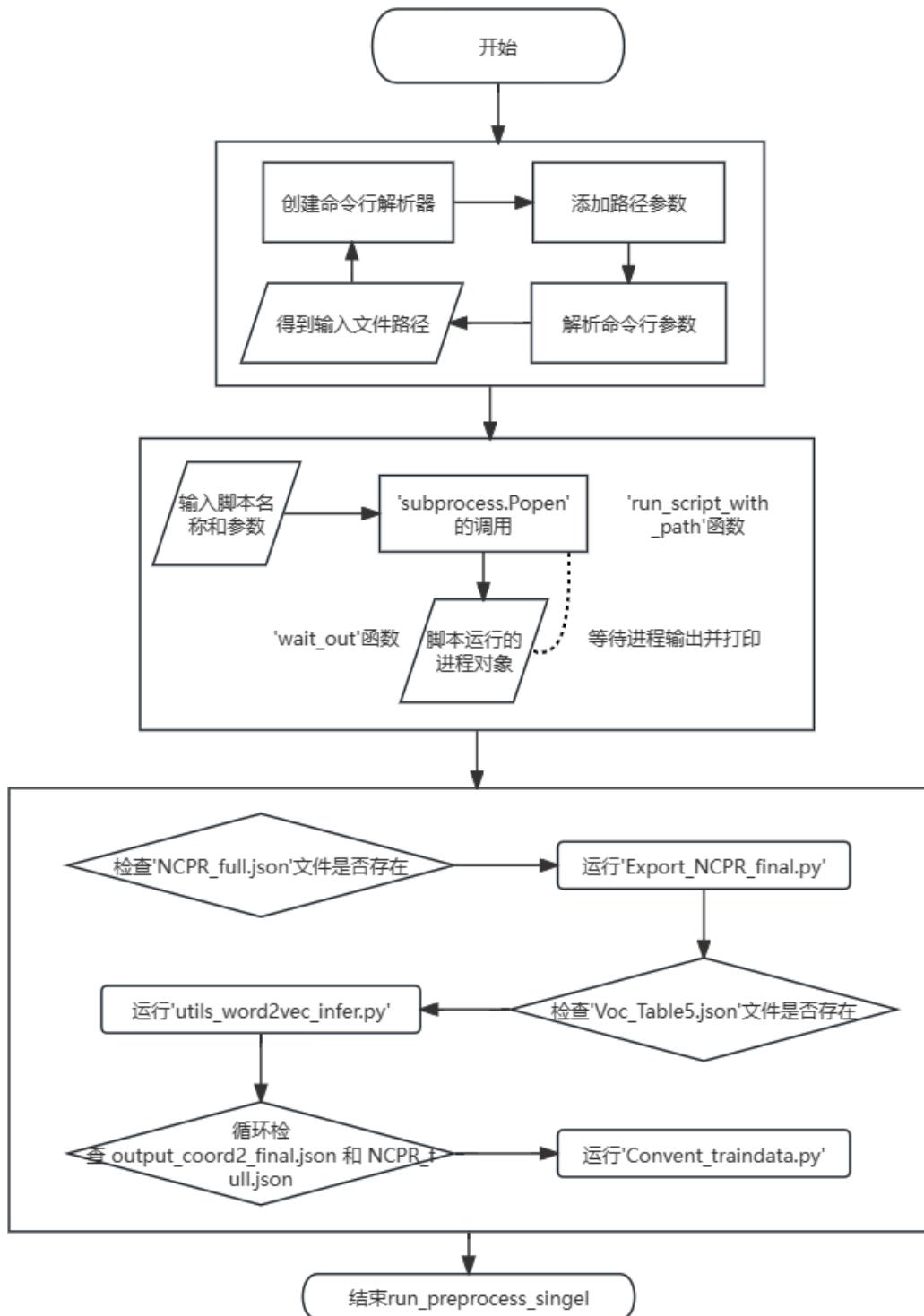


图 11 数据预处理的总体代码实现流程图

2.1.7 数据预处理自动化脚本的实现

类封装：模块化和对象化

我们的最终数据预处理程序版本的呈现，将所有的流程分别封装为三个类：class Export_NCPR、class W2V_infer、class Convent_data，所有的三个类均在utils_preprocess.py 脚本定义并实现的。每个类都承担着特定的数据处理功能，以确保数据准备阶段的高效和准确。

一键式自动化推理脚本：

在我们的最终呈现中，autoprocess.py 脚本被设计来自动化数据预处理流程。它接收一个文件路径作为参数，并在该路径下递归地搜索所有原始数据文件。检索到的文件路径随后传递给 utils_preprocess.py，以便进行进一步的数据处理。此外，为了优化处理效率，autoprocess.py 中引入了自定义的进程数参数（process_num），允许我们控制同时运行的预处理进程的最大数量。

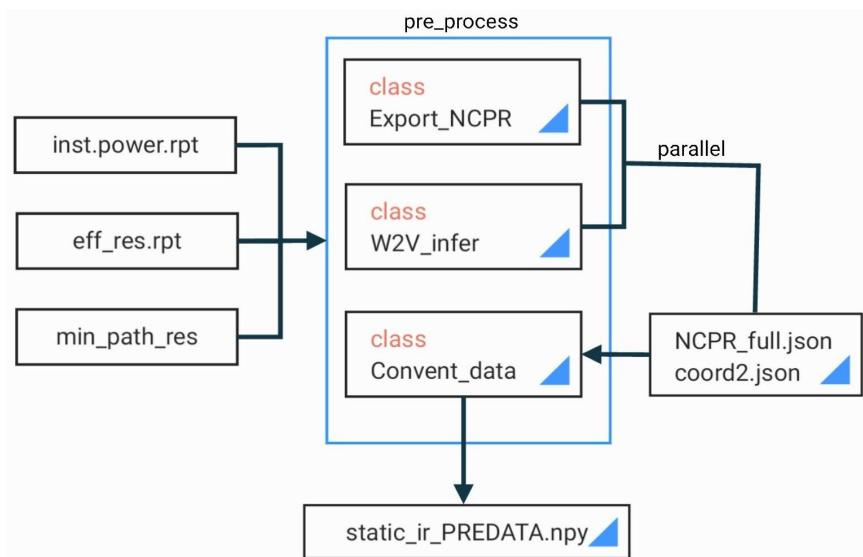


图 12 数据预处理自动化脚本的实现

Class Export_NCPR

原模块 1 的改进由 Class Export_NCPR 和 Class W2V_infer 两个类共同完成。其中，Class Export_NCPR 负责完成“NCPR”数据的导出。这一步骤是数据处理的关键环节，旨在从原始数据中提取和组织必要的信息，以便进行后续分析。特别地，Export_NCPR.Make_Name_list() 函数在这个类中扮演着重要角色。它的主要功能是从每个报告文件中提取名称（N）信息。这些名称信息是生成后续映射表的重要输入，因此该函数对整个数据预处理流程至关重要。

Class W2V_infer

关于映射表的推理过程，我们引入了类 Cass W2V_infer，以及原有程序中的脚本 utils_word2vec_train.py 的功能函数完成从 instance name 到二维向量的推理过程。

Class Convent_data

原模块 2 的改进由 Cass Convent_data 完成。同样地，该类基于映射表和 NCPR 数据，程序构建了用于机器学习模型训练的三维多通道输入矩阵“A”。这个矩阵的前两维基于名称坐标系，而第三维则由 P 和 R 数据组成的特征通道构成。

3 核心算法部分

我们使用了 Word2Vec、pix2pix、自行构建的 YogurtPyramid 三种算法，分别实现 IR drop 回归预测环节中的聚类、预测和优化三个环节。

3.1 Word2Vec

Word2Vec，一种无监督学习的应用于 NLP 自然语言处理的神经网络模型，便于将文字转化为向量坐标，此处用于将 instance name 转换为平面坐标系进行聚类操作。

3.1.1 Word2Vec 模型结构

Word2Vec 模型的预定义位于 gensim.models 库中。该模型是一个浅层网络模型，模型结构如下：

- 1) 输入层 $\times 1$: 大小等于词汇表的大小;
- 2) 隐藏层 $\times 1$: 大小等于所指定的向量维度;
- 3) 输出层 $\times 1$: 大小等于词汇表的大小。

3.1.2 Word2Vec 模型的训练与推理过程

3.1.2.1 训练阶段

我们采用了 Word2Vec 模型来捕捉芯片实例名称中隐含的语义结构，并转化为能够反映实例间关系的向量表征。在数据预处理阶段，我们首先对芯片组内所有实例的命名进行了收集，并对训练过程中可能导致错误的特殊字符进行了替换处理，确保数据质量的同时，提升了训练的稳定性。

芯片实例的名称通常为多级结构，例如“core_region_i/instr_mem/boot_rom_wrap_i/boot_code_i/U429”。我们通过分隔符“/”将每个名称解构为更细粒度的 token，如“core_region_i”和“U429”等。这些 token 形成了 Word2Vec 模型训练的基

础语料库。

利用此语料库，我们训练了 Word2Vec 模型，并将得到的向量表征存储于一个.bin 文件中，为后续推理阶段提供了一个预训练的词嵌入模型。

3.1.2.2 推理阶段

在推理过程中，每个芯片的实例名称经过与训练阶段相同的预处理步骤，被拆分为一系列 token。这些 token 随后被送入预训练的 Word2Vec 模型中，模型输出相应的五维向量表征。为了从实例名称中提取全局特征，我们采用向量和的方法合并同一实例名称下的 token 向量，生成了该实例的聚合向量表征。

为了便于可视化及后续处理，我们使用主成分分析（PCA）算法将五维向量降至二维空间。同时，为将实例映射至标准化的二维坐标系，我们设计了一个以 PCA 降维结果为中心的坐标标准化过程。通过螺旋搜索算法，我们在一个 256x256 的坐标空间内为每个实例寻找到了唯一的整数坐标点。如遇到坐标冲突，算法将持续搜索直至定位到未被占用的空间位置，保证了每个实例在坐标系中的独立性和辨识度。

这一过程不仅增强了模型对实例名称语义的捕捉能力，还为后续的多维度分析与预测提供了一个结构化和高度信息化的数据表示方式。

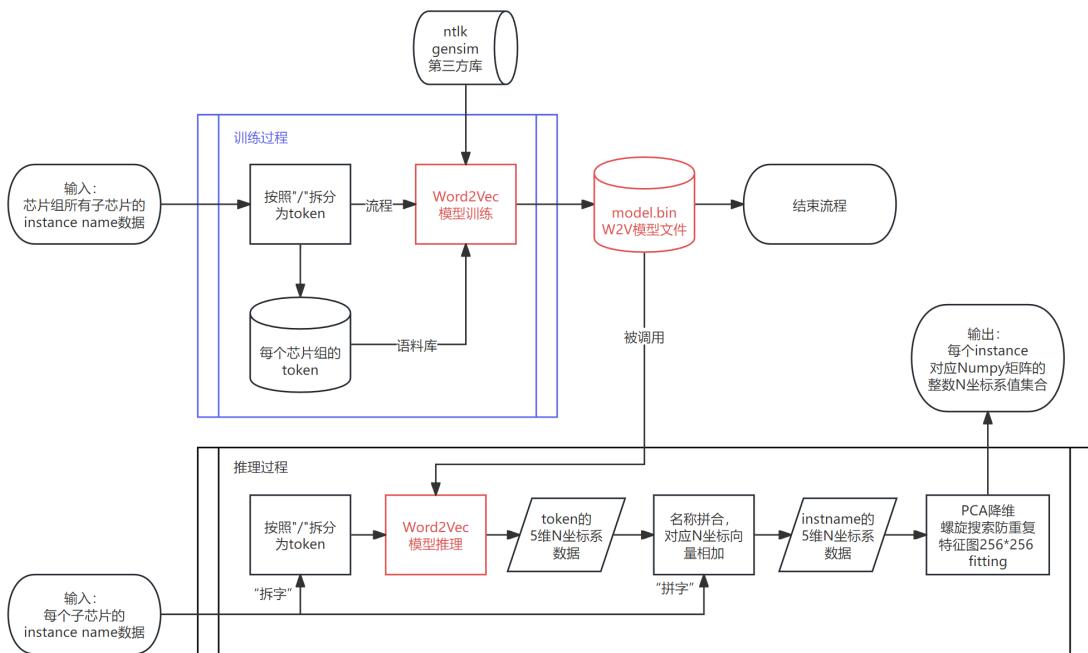


图 13 Word2Vec 算法在本次应用的总流程图

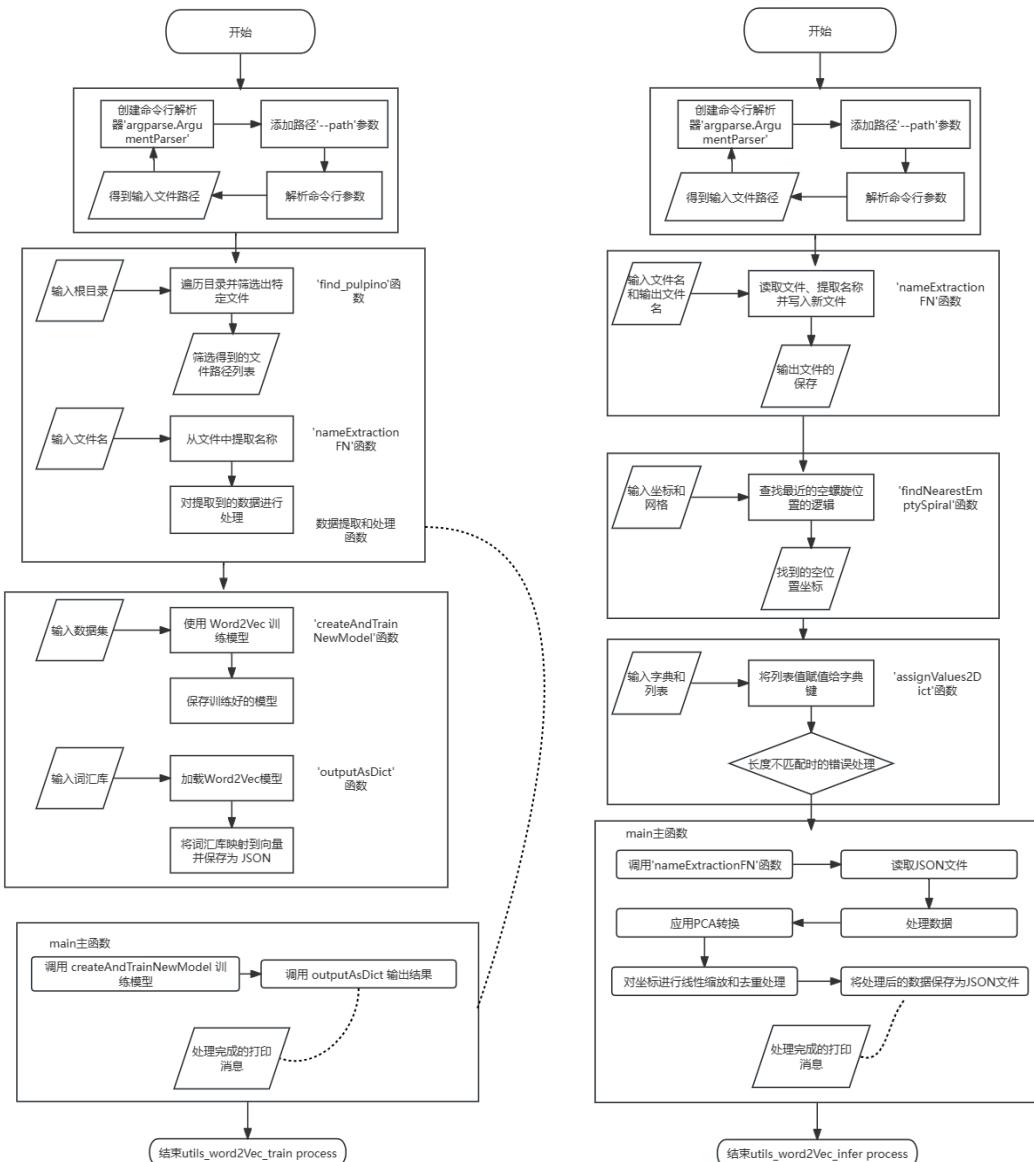


图 14 Word2Vec 推理和训练的代码实现流程

3.1.3 Word2Vec 训练参数

```

def createAndTrainNewModel(dataset, savefilename='model.bin'):
    model1 = gensim.models.Word2Vec(dataset, min_count=1, # data是被处理好的数据
                                    vector_size=5, # 5个维度
                                    window=500,
                                    epochs=350)
    model1.save(savefilename) # 将这个模型保存为.bin文件

```

图 15 Word2Vec 训练与创建函数代码

默认使用 Skip-gram 模型进行训练，它通过最大化目标函数来学习词嵌入，该函数在给定目标词的情况下，试图最大化上下文词的条件概率分布。而在 Word2Vec 的应用

中，由于其为无监督学习模型，传统的损失值概念不适用。此模型的核心在于利用实例名称与物理坐标、电气特性之间的关联性，实现高度相关的实例聚类，以支持后续分析。

3.1.4 效果分析

3.1.4.1 Word2Vec 实例名称分解与语料库构建（“拆字拼字”）

针对各芯片组（例如 zero-riscy、RISCY-FPU）中的实例名称，我们将其按照“/”符号进行分解，得到子 token，如“core_region_i”、“axi_mem_if_SP_i”等，并从中除去重复项，构成 Word2Vec 训练的语料库。

```
11999 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[12]
12000 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[11]
12001 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[10]
12002 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[9]
12003 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[8]
12004 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[7]
12005 core_region_i/instr_mem_axi_if/axi_mem_if_SP_i/READ_CTRL/RDATA_REG_reg[6]
```

图 16 实例名称分解与语料库构建效果

3.1.4.2 五维坐标系构建效果

利用 Word2Vec 模型处理 zero-riscy 芯片组的词汇，对拆分后的子词汇按照“/”重新进行编码，生成五维坐标系。通过向量加和法，我们合成了每个子 token 的五维坐标，并将结果保存至 output_coord5.json 文件中，如实例 zero-riscy_freq_500_mp_4_fpu_50_fpa_1.0_p_3_hi_ap 中所示，各 token 的映射坐标得到了清晰体现。

```
{"core_region_i/instr_mem/sp_ram_wrap_i/sp_ram_bank_i": [10.502414464950 正在处理
-4.683367013931274, -8.739851474761963, -22.869052410125732, -21.2
.331512451171875], "core_region_i/data_mem/sp_ram_bank_i": [7.2
.63730263710022, -3.619531512260437, -6.499229669570923, -17.2
.558006286621094, -15.851162910461426], "FE_PHC44_uart_cts": [2.2
.577075958251953, -1.2844622135162354, -1.9026453495025635, -6.2
.020689964294434, -5.2372636795043945], "FE_PHC43_uart_dsr": [2.2
.625345468521118, -1.1692352294921875, -1.942711148834229, -5.2
.681122779846191, -5.388060092926025], "FE_PHC42_uart_rx": [2.2
.50732684135437, -1.2721190452575684, -1.9924957752227783, -5.2
.672440528869629, -5.430764198303223], "FE_OFIC131_n1": [2.5394949913024902, 2
```

图 17 instance name 映射至五维度坐标系的结果

3.1.4.3 二维坐标系映射效果

将五维坐标通过主成分分析（PCA）方法降维至二维空间，并将其映射至 256*256 等

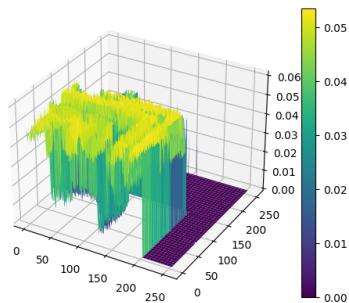
的整数“名称坐标系”空间，生成的坐标空间与预期的名称相似性对应关系一致，验证了我们的坐标映射假设。

```
{
  "core_region_i/instr_mem/sp_ram_wrap_i/sp_ram_bank_i": [3, 54], 
  "core_region_i/data_mem/sp_ram_bank_i": [3, 55], 
  "FE_PHC44_uart_cts": [3, 56], 
  "FE_PHC43_uart_dsr": [4, 55], 
  "FE_PHC42_uart_rx": [3, 53], 
  "FE_OFC131_n1": [4, 54], 
  "FE_OFC130_n1": [4, 56], 
  "FE_OFC125_n1": [5, 55], 
  "FE_OFC124_n1": [4, 53], 
  "FE_OFC123_n1": [2, 53], 
  "FE_OFC122_n1": [5, 54], 
  "FE_OFC120_n1": [2, 54], 
  "FE_OFC119_n1": [4, 52], 
  "FE_OFC110_n1": [2, 55], 
  "FE_OFC106_n1": [2, 56], 
  "FE_OFC105_n1": [2, 57], 
  "FE_OFC103_n1": [5, 53], 
  "FE_OFC102_n1": [5, 56], 
  "FE_OFC99_n1": [5, 52], 
  "FE_OFC97_n1": [3, 52]
}
```

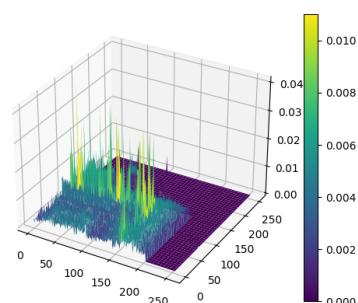
图 18 经过 PCA 和螺旋搜索算法后的二维坐标系结果，所有的坐标均为整数，方便输入至 PyTorch Tensor

3.1.4.4 聚类映射分析

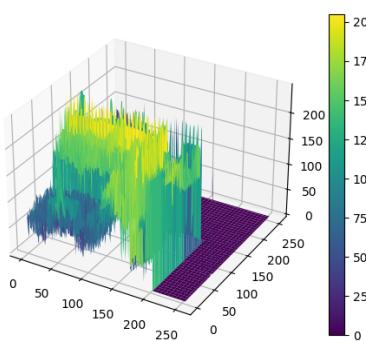
以 zero_riscy_freq_500_mp_4_fpu_50_fpa_1.0_p_3_fi_ap 为例，我们的聚类映射结果准确展示了实例名称相似性背后的电气特性相似性，并显示出明确的几何规律性和层次性。这为预测结果的准确性和后续优化工作提供了有力的数据支持。



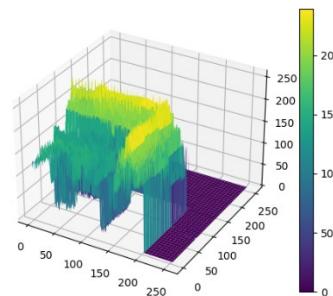
VDD_drop 分布，两个横轴分别为名称坐标系 Cx Cy，垂直方向的纵轴为 VDD_drop 电压随对应 instance 的分布



GND_bounce 分布，两个横轴分别为名称坐标系 Cx Cy，垂直方向的纵轴为 GND_bounce 电压随对应 instance 的分布



物理 x 坐标分布，两个横轴分别为名

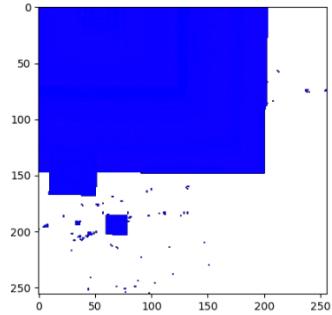


物理 y 坐标分布，两个横轴分别为名称坐标系 Cx Cy，垂直方向的纵轴为

称坐标系 Cx Cy，垂直方向的纵轴为

对应 instance 物理 y 坐标的位置

对应 instance 物理 x 坐标的位置



所有 instance 坐标映射到 256*256 整

数名称坐标系的映射结果，显示聚类

集中程度高且呈现矩形图形，仅有极

少量 instance 呈离散分布

图 19 基于 Word2Vec 的聚类分布结果

3.2 pix2pix 模型

Pix2Pix 模型是一种基于图像的回归卷积 GAN 神经网络，以 UNet 架构为基础，具体到 UNet-256 和 UNet-128 的模型版本，广泛应用于图像风格转换等生成性任务，模型参数总量约为 54.8 百万。

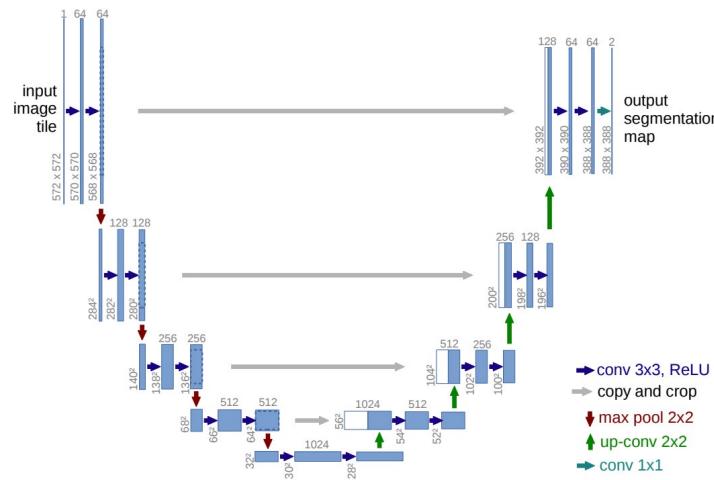


图 20 pix2pix 模型采用的 U-Net 生成器网络架构

在 Dataloader 阶段，我们从原始维度为 $\text{IMG_SIZE} * \text{IMG_SIZE} * 17$ 的矩阵开始，经过一系列操作，得到输入的 A 矩阵和目标的 B 矩阵，分别具有 $\text{IMG_SIZE} * \text{IMG_SIZE}$

* 3 和 $\text{IMG_SIZE} * \text{IMG_SIZE} * 2$ 的维度。

以 zero-riscy 项目为案例，A 矩阵的处理包括通道提取、比例调整、剪裁、填充、滤波和边缘增强，而 B 矩阵则经过边缘增强和比例逆向调整等步骤。这一系列精细化的预处理步骤确保了模型能够在后续的学习过程中更准确地理解和重建图像数据。

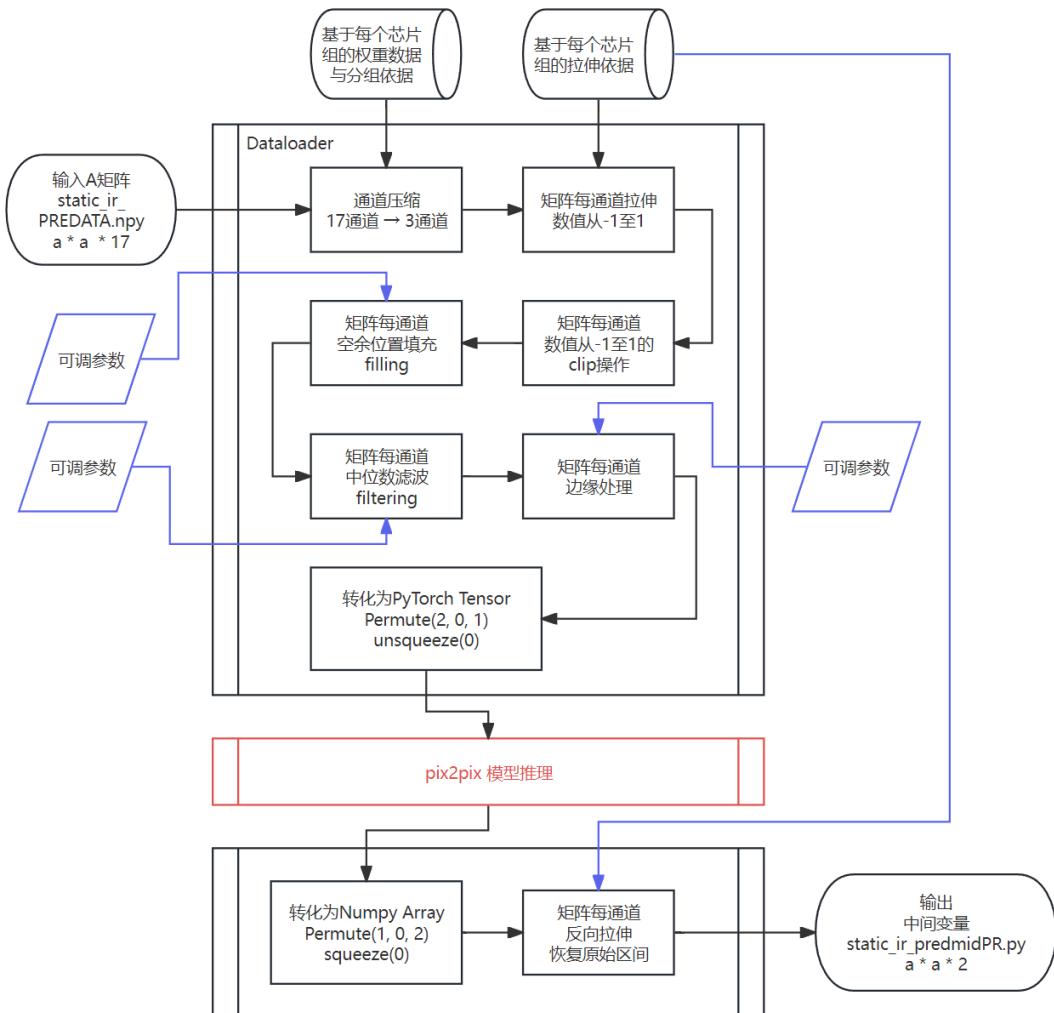


图 21 pix2pix Dataloader 总体流程图

3.2.1 Dataloader 具体流程描述及效果分析

3.2.1.1 通道提取过程

在构建 Pix2Pix 模型的输入矩阵时，我们从三个.rpt 文件中得到的 17 个数据通道开始。首先，通过筛选和提取操作，我们将 17 个通道的数据合并成 3 个主要通道，以用于输入矩阵。这一过程包括基于各通道数据的均值和方差进行聚类分析，以确定哪些通道将被合并。

通道 编号	通道释义	通道类型	通道 编号	通道释义	通道类 型
[0]	pin_location[x]	C 坐标类	[9]	Switching_power	P 功耗类
[1]	pin_location[y]	C 坐标类	[10]	Internal_power	P 功耗类
[2]	Bbox[x] (左上)	C 坐标类	[11]	Total_power	P 功耗类
[3]	Bbox[y] (左上)	C 坐标类	[12]	Loop_r	R 电阻类
[4]	Bbox[x] (右下)	C 坐标类	[13]	Vdd_r	R 电阻类
[5]	Bbox[y] (右下)	C 坐标类	[14]	Gnd_r	R 电阻类
[6]	Freq	P 功耗类	[15]	MRV_VDD	R 电阻类
[7]	Togglerate	P 功耗类	[16]	MRV_VSS	R 电阻类
[8]	Leakage_power	P 功耗类			

表 1 17 个通道的类型

```
[(46.988836090087894, 3404.06258577715), (90.62360961914064, 8658.60252991401),  
(46.78696851741027, 3387.203657690948), (90.47320528242493, 8633.6797529837), (4  
7.19070349920654, 3421.2079696371334), (90.77624323667908, 8683.434547876695), (4  
291831970.21484375, 1.2263973470544445e+17), (0.27553558349609375, 0.09114480897  
551402), (8.517136800231934e-10, 2.340591761421108e-18), (2.1227717716979977e-07  
, 1.861992757139964e-12), (5.98529980734253e-07, 2.2090238872077104e-12), (8.116  
588913040161e-07, 4.400458107719888e-12), (98.98012109375, 12156.683687118362),  
(49.81419999694824, 3232.0826384231514), (49.16592227172851, 3062.155046184876),  
(74.43076992538452, 8310.681180277028), (73.64994722793578, 8050.038387061977)]
```

图 22 17 个通道按序分布的均值和方差值

通过 K-Means 聚类方法，我们分析了 17 个通道的数据，并指定将它们分为 5 个类别（即 0、1、2、3、4）。我们发现，VDDx、VDDy、VSSx、VSSy、mean(BBox_x) 和 mean(BBox_y) 等六个通道（从 0 开始计数）因为与输入实例坐标和输出实例坐标具有高度的计算关联性，被选为模型输入的关键特征。然而，第 6 和第 7 通道（频率和翻转率）并未显示出随实例变化的分布特性，因此未被纳入 Pix2Pix 模型输入矩阵的通道范围。

```
Cluster labels for each channel's centroid: [2 3 2 3 2 3 1 0 0 0 0 0 3 4 4 2 2]
```

图 23 17 个通道使用 K-Means 聚类结果

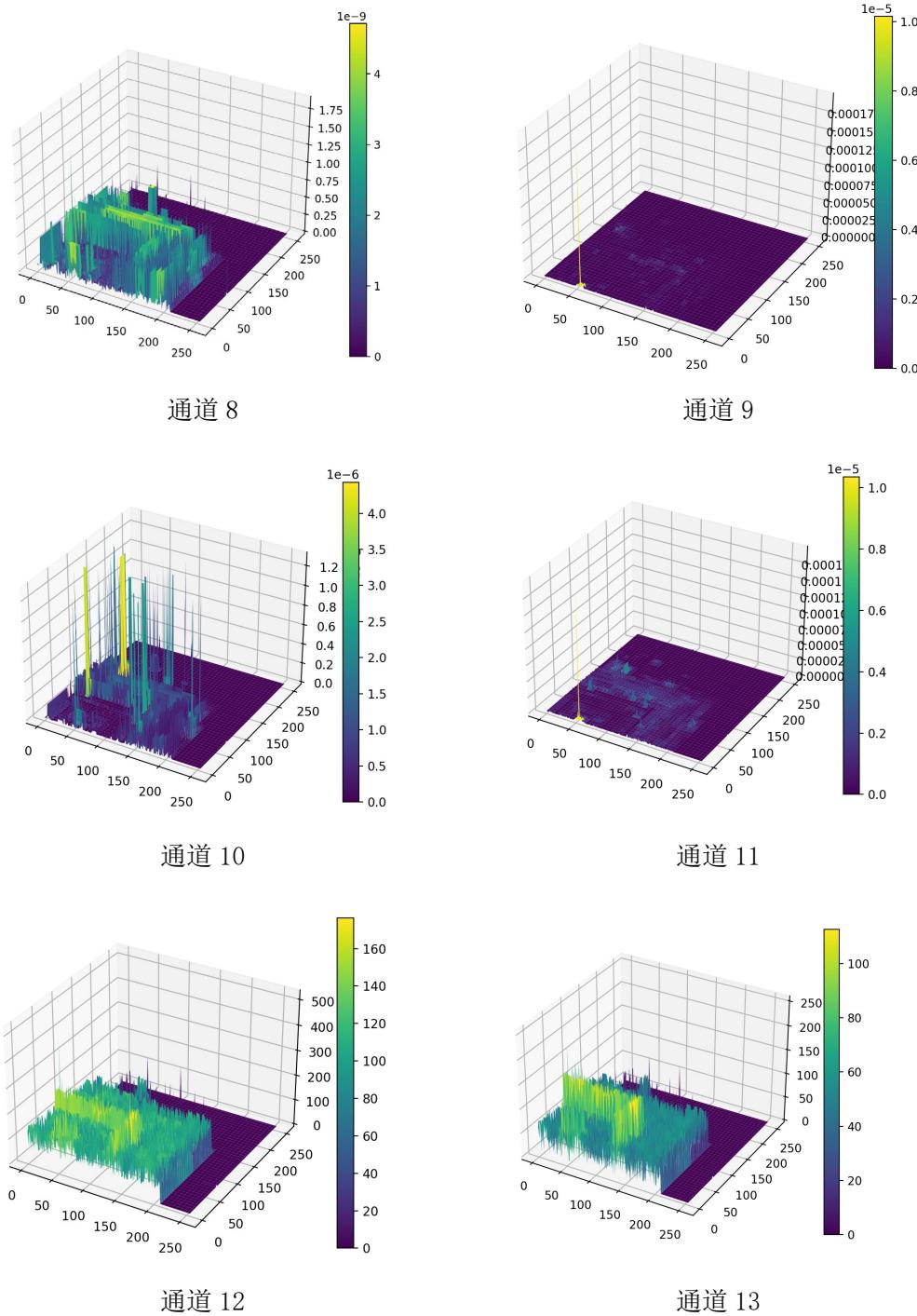
最终萃取的三个通道数据如下所示：

- 新通道 0：结合了原通道 8、9、10、11 的数据（从 0 开始计数），这些均被聚类到类别 0。
- 新通道 1：结合了原通道 12、13、14 的数据（从 0 开始计数），这些主要聚类到类

别 3 和 4，且数值在同一数量级上。

- 新通道 2：结合了原通道 15、16 的数据（从 0 开始计数），这些被聚类到类别 2。

在合并的结果中，以 zero-riscy_freq_500_mp_4_fpu_50_fpa_1.0_p_3_fi_ap 实例为例，其平面坐标轴对应名称坐标系的 x 轴和 y 轴，而垂直方向的坐标轴代表相应通道的数值。这种通道提取和合并的策略确保了输入矩阵中包含了对 Pix2Pix 模型训练至关重要的信息，同时剔除了不具备显著分布特性的冗余数据。



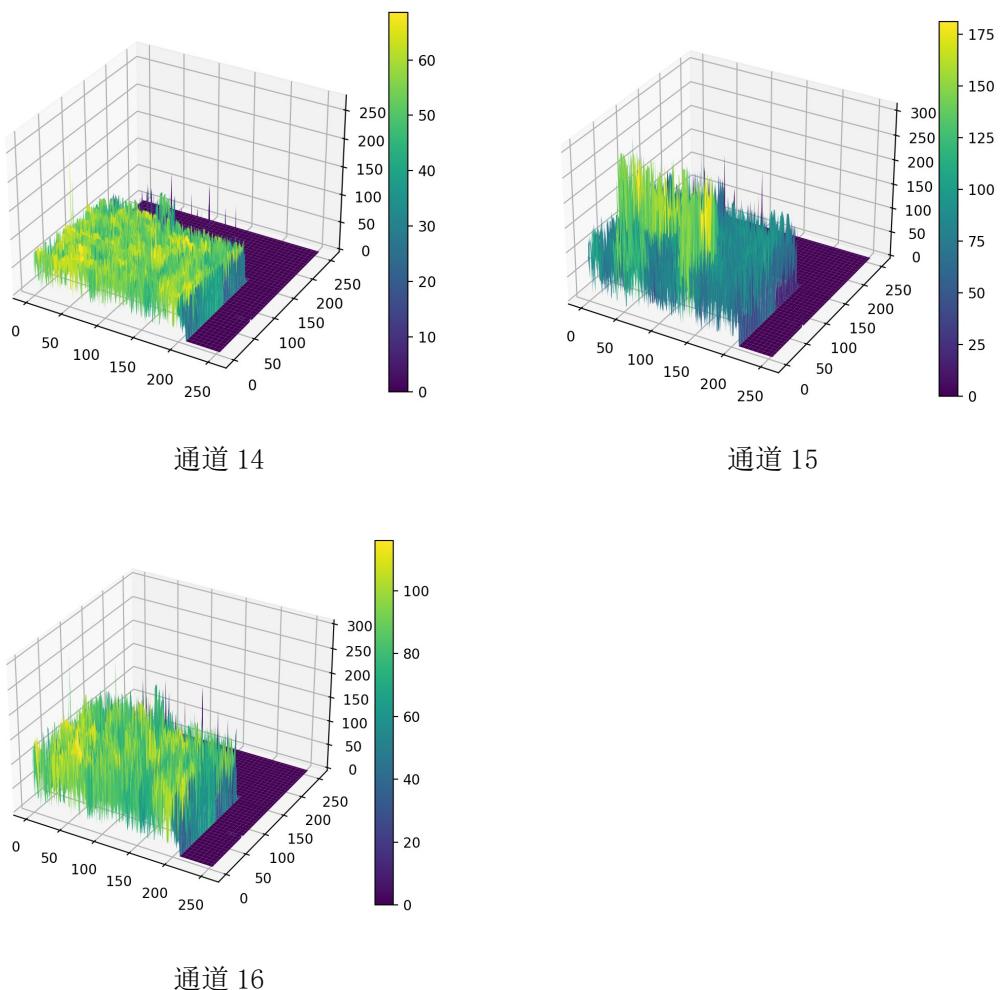
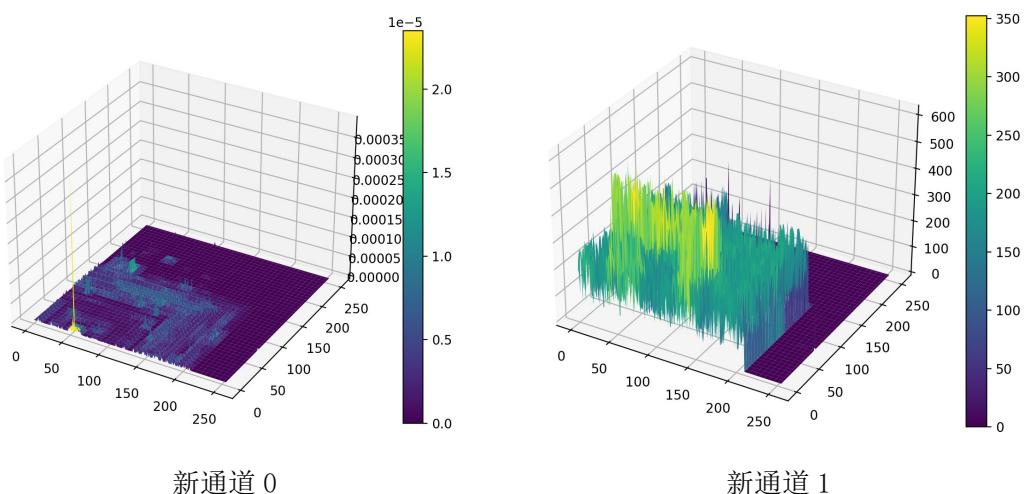


图 24 原始矩阵 A 的可被提取的通道分布

合并后的三个通道如下图所示：



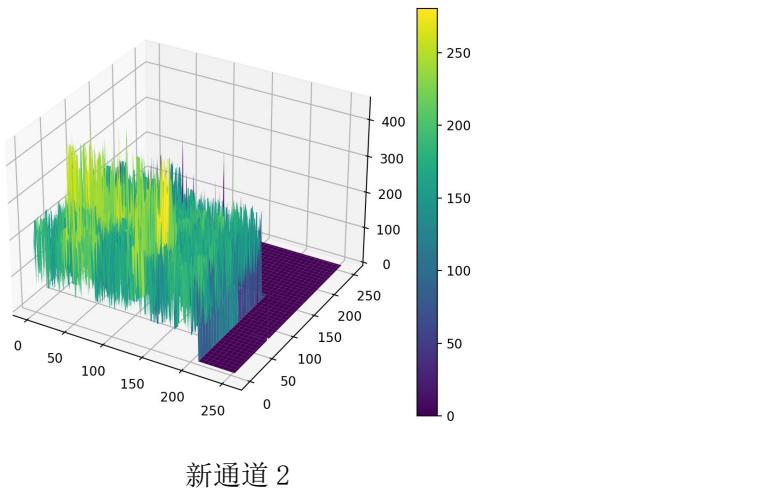


图 25 合并后的三个通道

3.2.1.2 拉伸处理步骤

在构建 Pix2Pix 模型的输入矩阵时，拉伸处理是一个关键步骤，其目的是对数据进行规范化以适应模型的输入要求。这一过程涉及对每个芯片组（如 RISCY、RISCY-FPU、zero-riscy）中所有子芯片的输入矩阵进行分析，具体来说，是计算所有通道中的最大值，并确定这些值的 97.5% 分位点。下图为以 zero-riscy 为例的结果。

```
[0] 280.4215 0.0
[1] 391.056 0.0
[2] 279.97374427500006 0.0
[3] 390.768036 0.0
[4] 289.373993 0.0
[5] 392.496033 0.0
[6] 2000000000.0 0.0
[7] 2.0 0.0
[8] 2.40305e-08 0.0
[9] 0.000293198475 0.0
[10] 1.4514835000000003e-05 0.0
[11] 0.000293198475 0.0
[12] 2052.863 0.0
[13] 1791.705 0.0
[14] 1401.423 0.0
[15] 1792.34225 0.0
[16] 1441.0 0.0
-----
[0] 0.1787 0.0
[1] 0.09474 0.0
[2] 280.4215 0.0
[3] 391.056 0.0
```

图 26 以 zero-riscy 为例的最大值 97.5% 分位点

随后，基于聚类结果和每个类别中数据的数量级差异，调整各通道数据的权重，确保新通道中原有子通道的数据权重大致相同。这一步骤的目的是平衡每个通道的影响力，

避免任何单一通道对最终结果产生过度的影响。

```
A_index = [0, 1, 2]
A_max = [(2.40305e-08 * 1000 + 2.932e-04 + 1.4515e-05 * 10 + 2.932e-04),
          | | | (2052.86 + 1791.71 + 1401.42), (1792.34 + 1441.00)]
A_min = [0, 0, 0]
```

图 27 以 zero-riscy 为例的权重结果

经过拉伸处理，我们将各通道的数据范围调整至-1 到 1 之间，并进行剪裁处理（clip），以保证数据在 Pix2Pix 神经网络的有效计算范围内。这些精细的数据预处理步骤不仅确保了输入矩阵的质量，而且对于提高模型预测精度和稳定性具有重要作用。

3.2.1.3 填充处理策略

针对 Pix2Pix 模型的训练和推理过程中，我们特别关注了不同芯片实例数量差异所导致的大量空数据点问题。为了减少这些空数据点对模型参数拟合能力的负面影响，我们采用了一种精心设计的填充方法。该方法的填充规则如下：

1. **寻找和填充零值区域：**我们在每个通道的二维矩阵中按顺序寻找 10x10 的零值方格进行填充。选择 10x10 而非 1x1 像素点的填充方式，是为了在一定程度上保持图像的局部空间关联性。这种方法不仅有助于体现空间的连续性，还能有效提升模型的泛化能力。
2. **填充数据的选取和随机性：**填充所用的数据来源于 10x10 的非零值方格区域。我们通过固定种子值的随机算法来选择填充数据，以确保在不同输入数据集中填充顺序的一致性。此外，随机选取填充数据还有助于确保填充的内容多样性，从而避免模型仅对特定数据分布具有较强的预测能力，同时提升对其他数据分布的预测能力。

需要指出的是，由于某些离散分布的实例在名称平面坐标系中的稀疏分布，填充效果可能会受到轻微影响。这一点在模型训练和推理过程中需予以特别注意，以确保填充策略的有效性和模型输出的准确性。

3.2.1.4 滤波与边缘处理

为了优化 Pix2Pix 模型的预测精度，我们对输入矩阵 A 和预测输出矩阵 B 实施了两个关键的数据处理步骤：滤波处理和边缘处理。

1. **滤波处理：**分析发现，基于名称坐标系的实例参数分布矩阵具备以下特点：总体分布集中且层次分明，每层聚类呈矩形或条带状，但在细节上存在大量波峰和抖

动。为了平衡这种矛盾，我们采用了 7×7 网格的中位数滤波处理。此举旨在在数据过度平滑化和原始数据噪声影响之间找到平衡，以最大程度提升预测的准确性。过强的滤波会导致数据分布过于平滑，而不足的滤波则会使神经网络的预测结果过于受原始数据毛刺影响。

2. **边缘处理：**在网络训练过程中，边缘区域的预测误差较大。为了解决这一问题，我们在最新的处理方案中，对所有基于名称坐标系的训练数据的输入矩阵 A 和输出矩阵 B 距离边缘 3 个像素的区域进行了归零处理。这样的边缘处理有助于避免边缘区域的预测误差对整体模型的影响，防止因边缘问题导致的梯度剧烈变化。经过归一化、剪裁、数据填充、滤波和边缘处理后，三个通道的数据分布呈现了显著的改善，更加适合 Pix2Pix 模型进行有效的学习和预测。这些处理步骤的综合应用，确保了数据质量，同时为提高模型预测的准确度提供了坚实基础。

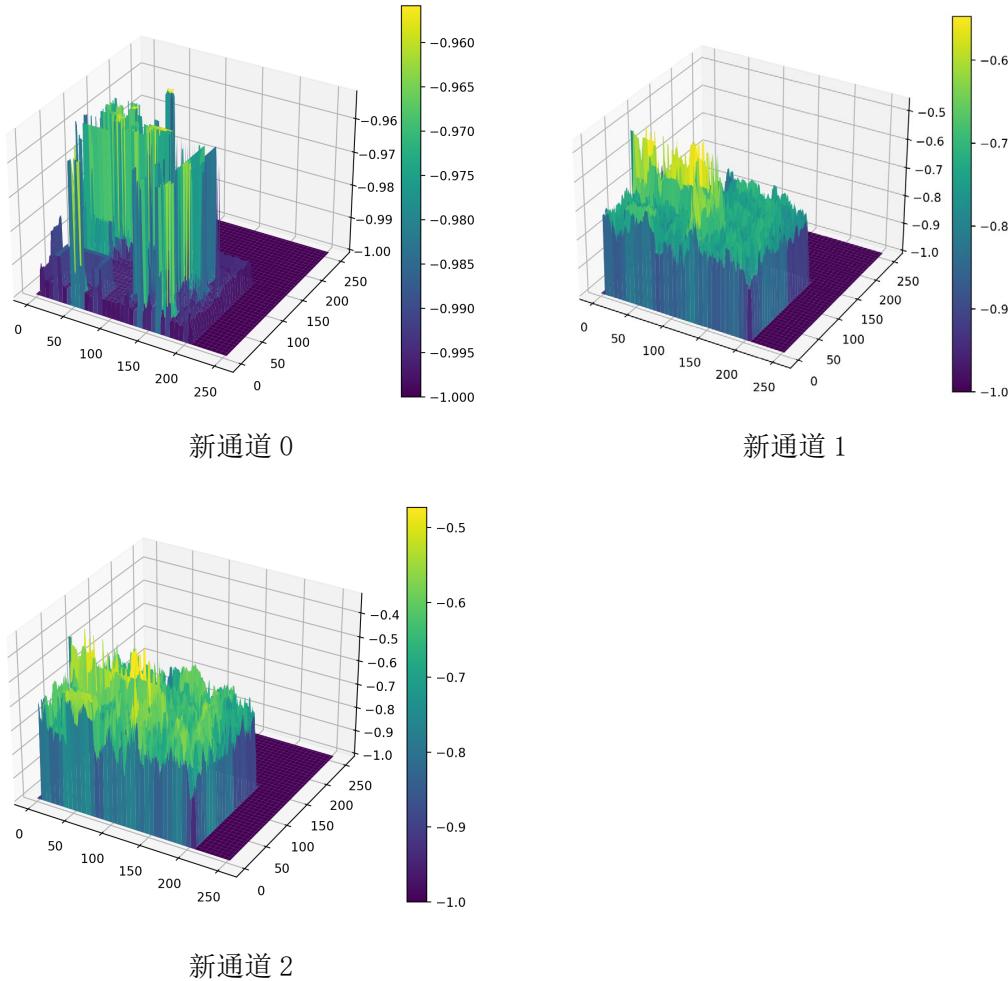


图 28 输入矩阵 A 的新通道处理结果

3.2.2 Pix2Pix 模型训练过程

在 Pix2Pix 模型的训练环节中，每个实例（instance）输入的是 $256 \times 256 \times 3$ 通道的

矩阵 (A)，而模型输出为 256x256x2 通道的矩阵 (B)。我们的模型设计采用了经典的 U-Net256 作为生成器 G，同时选用了针对像素级细节具有高精准度的 pixel 模型作为判别器 D。

整个训练过程分为 400 轮，其中前 200 轮采用自定义的余弦退火 (cosine) 训练策略，以实现学习率的动态调整。余下的 200 轮则在余弦退火策略的基础上增加了线性递减的学习率调整，旨在进一步优化模型的训练效果。这种结合动态和线性调整的学习率策略有助于模型在不同阶段达到更好的性能。

对于生成器 G 和判别器 D，我们采用了默认的学习率参数，即 0.0002。这一设置旨在平衡训练速度和模型性能，确保模型能在合理的时间内达到优化的预测效果。通过这样的训练设置，我们的模型旨在捕捉和预测 SoC 电源网络中的静态压降，以提高芯片设计的准确性和可靠性。

每个 pix2pix 模型按照实际学习状况，取生成器 G_L1 的最低值，并保证 G_GAN、D_real 与 D_fake 的值达到稳定、但不至于过低的水平。

参数名	值	含义
dataset_mode	yogurtnetpr3	使用我们的自定义数据格式，输入 3 个通道，输出 2 个通道，每个通道为 256×256 的矩阵
input_nc	3	输入矩阵的通道数为 3
output_nc	2	输出矩阵的通道数为 2
gpu_ids	0, 1	使用两张显卡进行训练
n_epochs	1500	训练第一阶段轮数为 1500，该阶段学习率大小不变；
n_epochs_decay	1200	该阶段学习率呈线性下降；
Lr	0.0002	生成器 (G) 的初始学习率为 0.0002，同时也是学习率的默认参数
Lr_D	0.0000001	判别器 (D) 的初始学习率为 0.000 000 1

表 2 Word2Vec 训练参数

```

2           batch_size: 1
3               |   beta1: 0.5
4           checkpoints_dir: ./checkpoints
5           continue_train: False
6               |   crop_size: 256
7               |       dataroot: /home/stxianyx/code/eda/data      [default: None]
8           dataset_mode: yogurtnetpr3                      [default: aligned]
9               |   direction: AtoB

10          display_env: main
11          display_freq: 400
12              |   display_id: 1
13          display_ncols: 4
14          display_port: 8097
15          display_server: http://localhost
16          display_winsize: 256

17          epoch: latest
18          epoch_count: 1
19              |   gan_mode: vanilla
20              |       gpu_ids: 0,1
21              |       init_gain: 0.02
22              |       init_type: normal
23              |       input_nc: 3                                [default: 6]
24              |       isTrain: True                            [default: None]
25              |       lambda_L1: 100.0
26              |       load_iter: 0                               [default: 0]
27              |       load_size: 286
28                  |           lr: 0.0002
29                  |           lr_D: 0.0002
30              |       lr_decay_iters: 50
31                  |           lr_policy: cosine                     [default: linear]

32          max_dataset_size: inf
33              |   model: yogurtpix2pix                         [default: cycle_gan]
34              |       n_epochs: 200                             [default: 100]
35          n_epochs_decay: 200                          [default: 100]
36          n_layers_D: 3
37              |       name: yogurtpr3_ryzero_11220852NEW    [default: experiment_name]
38              |       ndf: 64
39              |       netD: pixel                           [default: basic]
40              |       netG: unet_256
41              |       ngf: 64
42          no_dropout: False
43          no_flip: False
44          no_html: False
45              |       norm: batch

46          num_threads: 4
47              |   output_nc: 2
48                  |           phase: eda-riscy                   [default: train]
49                  |           pool_size: 0
50                  |           preprocess: resize_and_crop
51                  |           print_freq: 2                      [default: 100]
52          save_by_iter: False
53          save_epoch_freq: 50                         [default: 5]
54          save_latest_freq: 5000
55          serial_batches: False
56              |       suffix:
57          update_html_freq: 1000
58              |       use_wandb: False
59                  |           verbose: False
60          wandb_project_name: CycleGAN-and-pix2pix

```

图 29 模型参数 log 显示结果

下面以子芯片数较多的三个芯片组 zero-riscy、RISCY、RISCY-FPU 为例，展现模型的

训练效果:

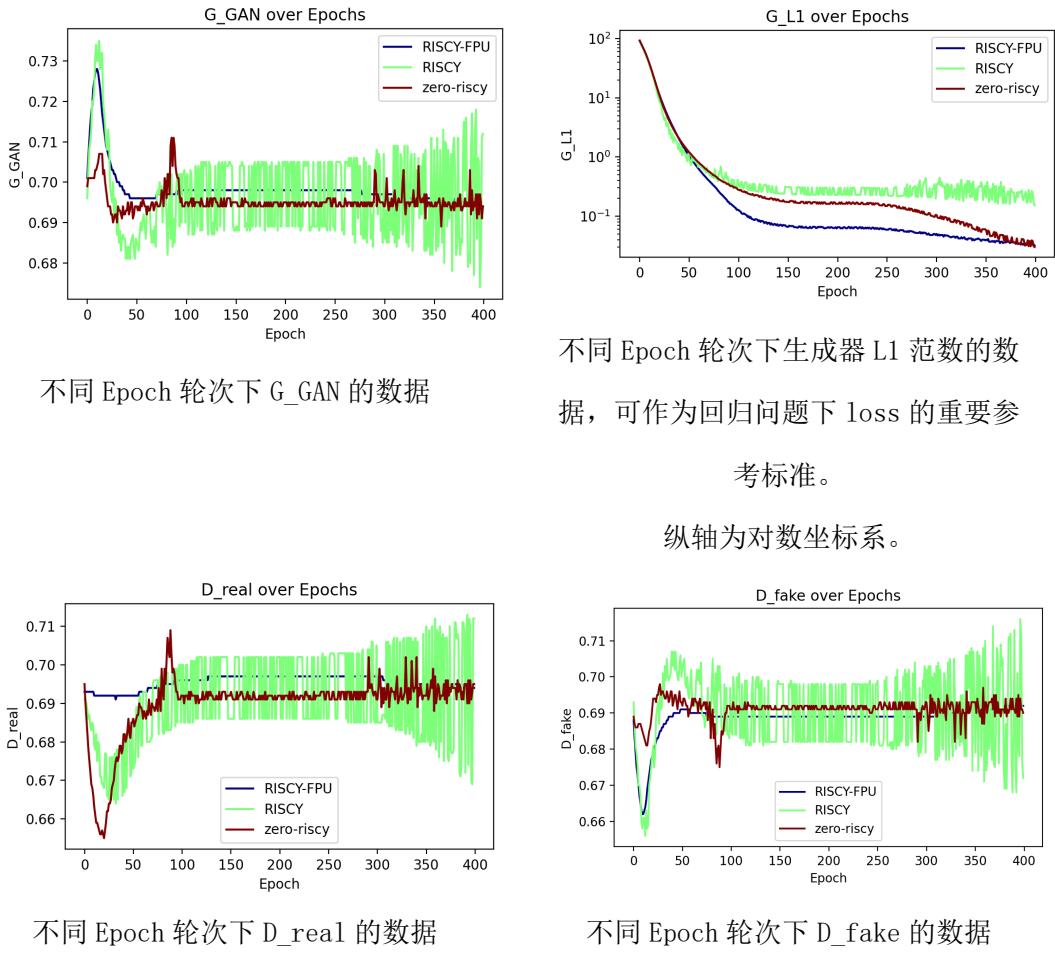


图 30 pix2pix 训练效果

模型效果:

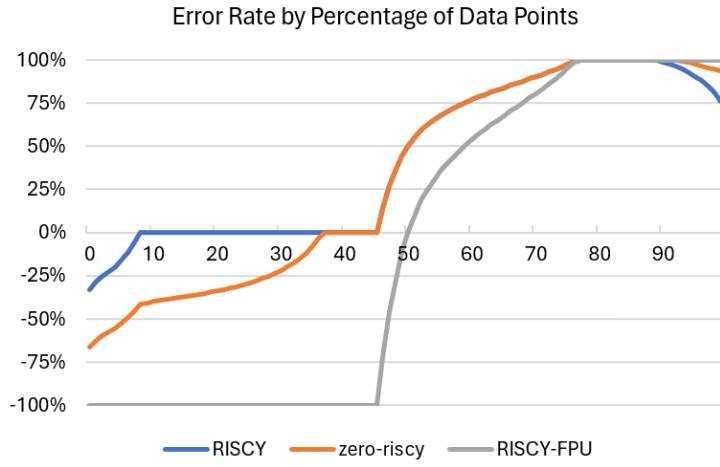


图 31 instance 输出电气特性误差分布图

上图展现名称坐标系当前像素数占总像素数的不同分位点比例下，基于名称坐标系的误差比率。

误差比率计算公式：(预测值 - 真实值) ÷ 真实值 * 100%

VDD_drop 预测误差结果：

芯片组	误差最小值	误差平均值	误差最大值
RISCY	-0.995439	1.920088	54.864323
zero-riscy	-0.999420	-0.155816	19.903573
RISCY-FPU	-1.0	-0.461310	8.974089

表 3 VDD_drop 预测误差结果

GND_bounce 预测误差结果：

芯片组	误差最小值	误差平均值	误差最大值
RISCY	-0.997035	1.891936	126.228388
zero-riscy	-0.999482	0.045353	31.875163
RISCY-FPU	-1.0	-0.531775	7.579361

表 4 GND_bounce 预测误差结果

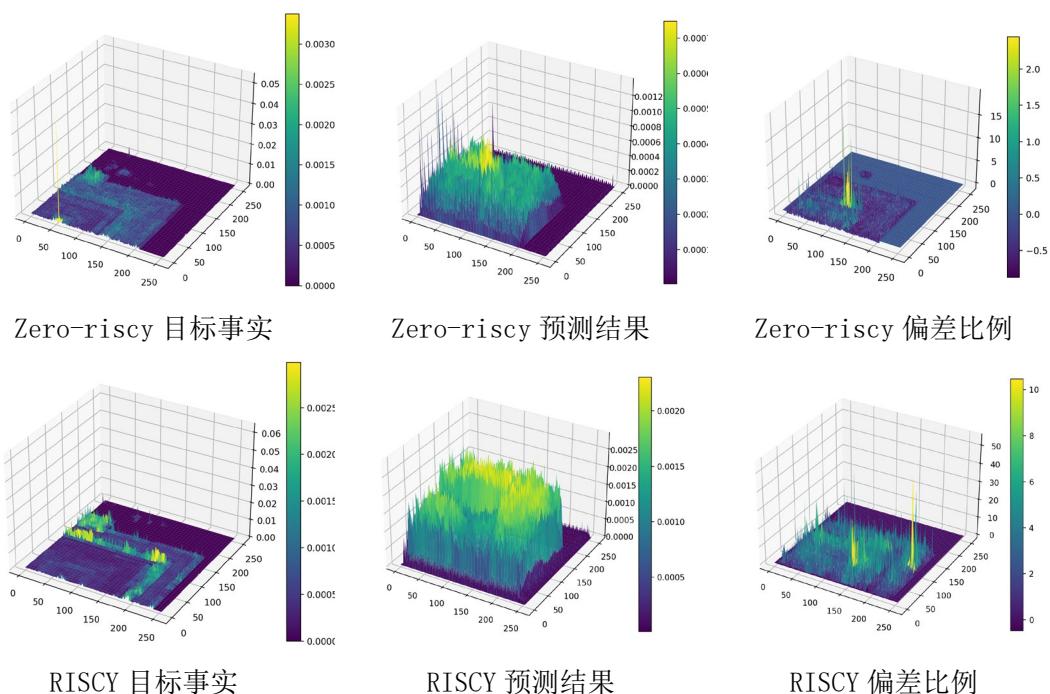
pix2pix 模型预测效果图（以 VDD_drop 为例）

抽取样本：

zero-riscy 选用 zero-riscy_freq_200_mp_1_fpu_60_fpa_1.5_p_2_hi_ap_;

RISCY 选用 RISCY_freq_50_mp_1_fpu_50_fpa_1.5_p_4_hi_ap_;

RISCY-FPU 选用 RISCY-FPU_freq_50_mp_1_fpu_50_fpa_1.5_p_0_hi_ar_。



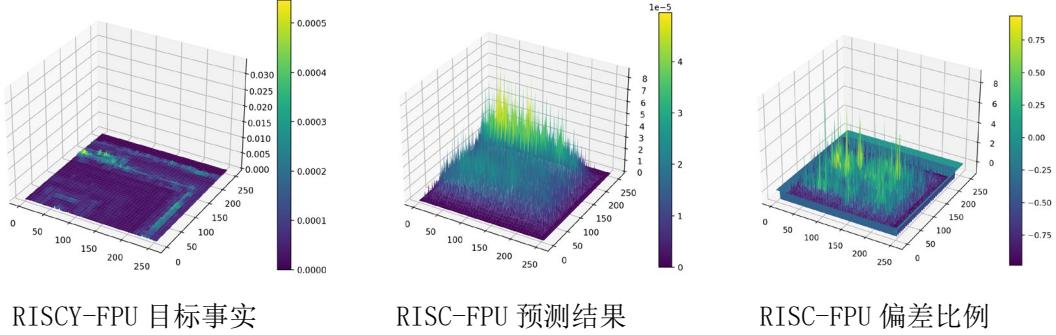


图 32 pix2pix 模型预测效果图

在使用 Pix2Pix 模型进行预测时，我们注意到虽然模型成功地将输入参数通过聚类转化为类似于目标的规律图形，但在数值层面上，预测结果与实际目标仍存在显著差异。为了解决这一问题，我们将 Pix2Pix 网络的输出作为一个中间步骤，而非最终结果。

为了提高数值准确性和波动幅度的一致性，我们设计并实施了一个名为 YogurtPyramid 的浅层神经网络。这个网络的目的是在中间预测结果和实际目标之间建立更精确的对应关系。通过这种方式，YogurtPyramid 网络能够调整和优化 Pix2Pix 模型的输出，使其更接近实际目标的数值和波动特性。这一步骤是对 Pix2Pix 模型的重要补充，有助于提升整体预测系统的准确性和可靠性。

3.3 YogurtPyramid 浅层神经网络

YogurtPyramid 是一个浅层神经网络，专门设计用于对深度神经网络模型的输出进行修正。深度网络虽然能够较好地拟合数据的微观波动细节，但在宏观上，如数值量级和波动幅度的变化，尚未得到充分体现。因此，我们引入 YogurtPyramidModel1256，这是一个新型的卷积神经网络架构，专门处理具有两个通道和 256x256 空间维度的输入张量。

3.3.1 YogurtPyramid 网络结构

YogurtPyramidModel 的网络结构设计理念是利用一系列不同尺寸的卷积层逐步捕捉输入数据的各种空间特征：

1. **多尺度卷积层：**该模型包含多个卷积层，每层具有不同尺寸的感受野（2x2 到 64x64）。这样的设计使得网络能够综合捕捉从微观到宏观的空间特征。
2. **卷积层配置：**各卷积层均配置有合适的步长和填充，确保输出张量尺寸与输入一致。所有层均采用偏置项，以增强模型的表达能力。
3. **激活与插值：**卷积后的输出通过 sigmoid 激活函数处理，再通过双线性插值调整至

原始输入尺寸，这有助于不同尺度的特征融合。

4. **特征融合**: 经处理的张量最终相加，实现了不同尺度特征的有效融合，从而提供了一个综合的输出。
5. **网络输出**: 经过一系列卷积、激活和插值操作后，模型输出与原始输入同尺寸的张量，表明其在保持空间结构的同时，对特征进行了有效的提取和转换。

通过 YogurtPyramid，我们能够在深度网络输出的基础上实现更全面的特征捕捉，从而提高整体模型的预测精度和波动幅度的一致性。根据不同输入矩阵大小（如 384*384、640*640）的要求，YogurtPyramid 网络有不同的实现方式。下面是其中一种实现 YogurtPyramidModel256 的结构图，输入和输出均为宽、高 256 的矩阵。

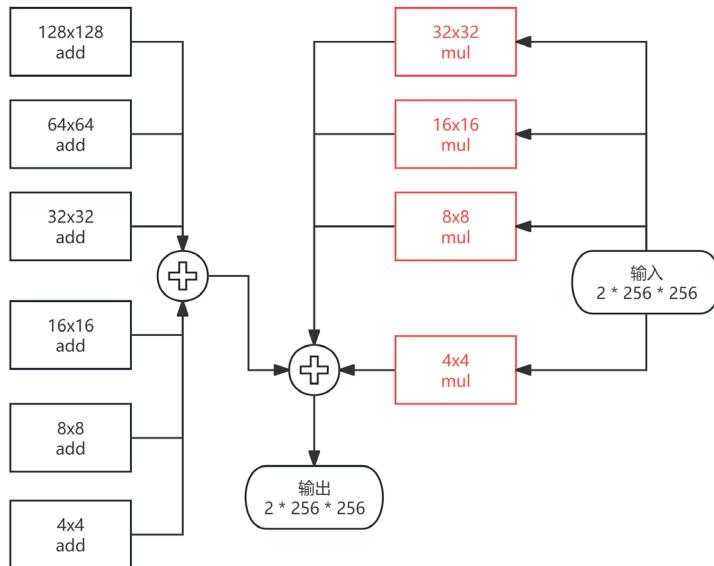


图 33 YogurtPyramidModel256 的网络结构图

参数量和网络结构层次：

```
(conv2x2_add): Conv2d(2, 2, kernel_size=(128, 128), stride=(128, 128))
(conv4x4_add): Conv2d(2, 2, kernel_size=(64, 64), stride=(64, 64))
(conv8x8_mul): Conv2d(2, 2, kernel_size=(32, 32), stride=(32, 32))
(conv8x8_add): Conv2d(2, 2, kernel_size=(32, 32), stride=(32, 32))
(conv16x16_mul): Conv2d(2, 2, kernel_size=(16, 16), stride=(16, 16))
(conv16x16_add): Conv2d(2, 2, kernel_size=(16, 16), stride=(16, 16))
(conv32x32_mul): Conv2d(2, 2, kernel_size=(8, 8), stride=(8, 8))
(conv32x32_add): Conv2d(2, 2, kernel_size=(8, 8), stride=(8, 8))
(conv64x64_mul): Conv2d(2, 2, kernel_size=(4, 4), stride=(4, 4))
(conv64x64_add): Conv2d(2, 2, kernel_size=(4, 4), stride=(4, 4))
)
Trainable parameters: 92820
```

图 34 以 256*256 的输入矩阵 YogurtPyramid 为例

3.3.2 YogurtNet 训练参数:

参数名	值	含义
batch_size	1	以 instance 为单位训练数据
Loss	L1_Loss, MSE	损失值评价指标, L1 范数、MSE
metrics	R_squares	R ² 评价指标
optimizer	Adam	优化器
Lr_scheduler	StepLR	学习率规划器
lr	0.000 000 01	最大学习率
Epoch	15	训练次数

表 5 所有 YogurtPyramid 网络的训练参数设置

3.3.3 训练结果分析

在我们的训练过程中，各芯片组的数据被分配为 90% 用于训练集，剩余 10% 作为验证集。这种划分策略旨在平衡模型训练的深度和泛化能力。根据训练结果，我们观察到模型的均方误差（MSE）值在约 0.003 到 0.004 的范围内。这个结果表明，模型对于训练数据和未见数据都有较好的拟合能力，证明了其良好的泛化性能。

MSE 值作为评估模型性能的关键指标，其较低的数值表明预测误差小，模型能够较准确地捕捉和预测目标变量。这一结果对于验证模型在 SoC 电源网络静态压降预测任务上的有效性至关重要，同时也为进一步的模型优化和应用提供了坚实的基础。

RISCY-FPU:

```
loss          : 0.00364947592917653
R_squares    : -8.476146697998047
val_loss     : 0.003904641788655716
val_R_squares : -8.052739143371582
```

zero-riscy:

```
loss          : 0.0021616594637481907
R_squares    : -2.268373966217041
val_loss     : 0.001697444704210024
val_R_squares : -2.6884546279907227
```

RISCY:

```

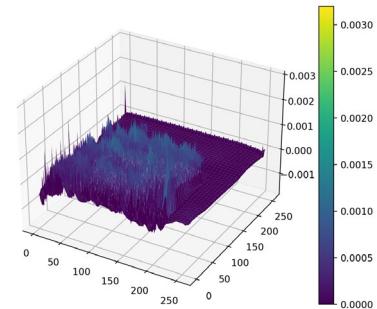
loss : 0.002513524955496623
R_squares : -1.781348466873169
val_loss : 0.0021611561168221375
val_R_squares : -1.689778208732605

```

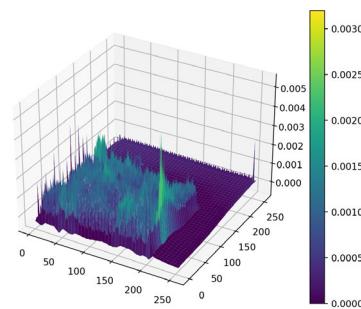
芯片组	L1 范数
RISCY-FPU	0.0297581
Zero-riscy	0.0182963
RISCY	0.0220029

表 6 不同芯片组的 L1 范数结果

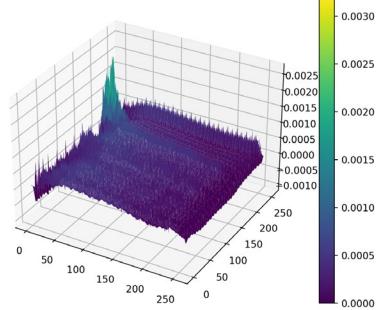
预测结果：



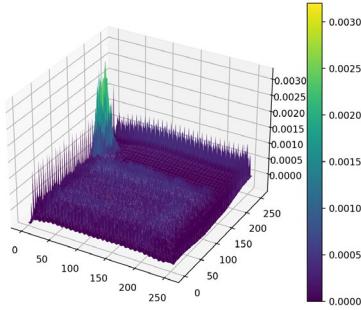
Zero-riscy VDD_drop 预测结果



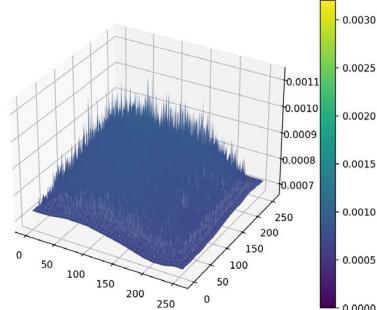
Zero-riscy GND_bounce 预测结果



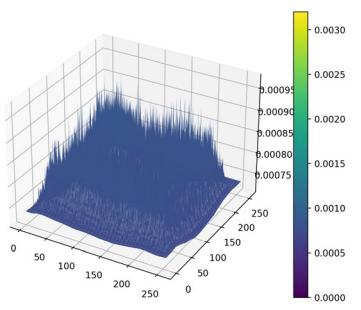
RISCY VDD_drop 预测结果



RISCY GND_bounce 预测结果



RISCY-FPU VDD_drop 预测结果



RISCY-FPU GND_bounce 预测结果

4 环境配置

4.1 硬件配置

本项目的仿真实验在高性能服务器上进行，以确保数据处理和模型训练的效率。服务器配备了 Intel(R) Xeon(R) Platinum 8368 CPU @ 2.40GHz 处理器，具有 152 个核心，配合 2 个 NVIDIA A100 80GB 图形处理单元，为深度学习提供了强大的计算能力。此外，服务器拥有 1.0Ti 的运行内存，保障了大规模数据集的处理能力。所有实验均在 Ubuntu 20.04.3 LTS 操作系统上进行，操作系统内核版本为 Linux 5.4.0，确保了仿真环境的稳定性与高效性。

4.2 软件配置

在软件层面，本实验采用 Python 3.8.10 作为主要的编程语言，借助 Anaconda 3 2022.10 包管理器管理依赖关系，保障了代码的可移植性和可复现性。为了实现深度学习算法，选择了 PyTorch 2.0.1 作为主要的机器学习框架，其与 CUDA 11.8 及 CUDNN 8.7.0 的兼容配置，充分利用了 NVIDIA A100 显卡的强大功能，显著提升了模型训练和推理的速度。

4.3 第三方库

实验中还使用了 gensim 4.3.0 和 nltk 3.8.1 两个第三方库，前者用于实现高效的自然语言处理，如 Word2Vec 模型的训练和推理；后者用于文本处理，提供了丰富的语言资源工具，为文本数据的预处理提供了强大支持。

5 赛题方指标结果：wall time, metric

为了显示我们的代码运行的通用性，我们选取了以下三个芯片的数据进行测试：

1. zero-riscy_freq_50_mp_1_fpu_55_fpa_1.5_p_6_hi_ap_
2. zero-riscy_freq_50_mp_1_fpu_60_fpa_2.0_p_4_hi_ar_
3. zero-riscy_freq_50_mp_1_fpu_65_fpa_1.0_p_3_hi_ar_

5.1 wall_time 评价指标

wall time 评价指标数据如下表所示：

项目	Wall time 时间
数据预处理（含 Word2Vec 推理）	191.65s

Pix2pix 推理	10.79s
YogurtPyramid 推理与最终数据生成	5.71s
总耗时（所有芯片数）	209.82s
单个芯片数平均耗时	69.94s

图 35 wall time 测试指标数据

细化的运行测试数据如下图所示：

```
Command being timed: "python run_preprocess_single.py"
User time (seconds): 191.65
System time (seconds): 0.66
Percent of CPU this job got: 113%
Elapsed (wall clock) time (h:mm:ss or m:ss): 2:49.80
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 219824
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 158227
Voluntary context switches: 646
Involuntary context switches: 19000
Swaps: 0
File system inputs: 0
File system outputs: 43752
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

图 36 数据预处理（含 Word2Vec 推理）运行时长

```
Command being timed: "python /home/stxianyx/code/eda/Week4/torchnet/core_icisctest_zero.py"
Percent of CPU this job got: 493%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:10.79
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 3686180
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 153
Minor (reclaiming a frame) page faults: 904680
Voluntary context switches: 1076
Involuntary context switches: 76023
Swaps: 0
File system inputs: 11488
File system outputs: 3096
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

图 37 pix2pix 运行测试数据

```

Command being timed: "python /home/stxianyx/code/eda/Week4/yogurtpyramid/test_icisc_pyramid_zero.py"
Percent of CPU this job got: 98%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:05.71
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 3671856
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 792522
Voluntary context switches: 246
Involuntary context switches: 784
Swaps: 0
File system inputs: 0
File system outputs: 3104
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0

```

图 38 YogurtPyramid 和文件输出运行测试数据

5.1 metric 评价指标

metric 评价指标使用了赛题方的官方 MAE 指标算法。结果如下所示：

项目	MAE 指标
zero-riscy_freq_50_mp_1_fpu_55_fpa_1.5_p_6_hi_ap_	3.2457
zero-riscy_freq_50_mp_1_fpu_60_fpa_2.0_p_4_hi_ar_	9.1438
zero-riscy_freq_50_mp_1_fpu_65_fpa_1.0_p_3_hi_ar_	3.4459
总 MAE (3 个)	15.8354
平均 MAE	5.2784

表 7 MAE 评价指标结果

6 作品的评价、改进和推广

6.1 作品创新点

本研究的创新之处主要体现在三个关键技术的应用上，这些技术的结合不仅提升了预测的准确性，也加速了处理过程。

首先，Word2Vec 算法的应用在本研究中具有创新性。通过将实例名称映射到二维坐

标系中，Word2Vec 充分利用了实例电气特性与名称的相关性。这一处理方法有效降低了物理坐标系中实例分布不均匀导致的电气特性波动问题，在另一个名称坐标系中实现了更加平稳的分布，为后续的类图像神经网络算法处理提供了便利。

其次，引入 pix2pix 模型来处理芯片电源网络静态压降（IR drop）预测任务，体现了创新性的思考。将这一电子设计自动化（EDA）问题视作类似图像处理的任务，不仅提高了预测速度，也为这一领域的问题解决提供了新的视角。

最后，结合浅层神经网络的线性拟合方法与深度学习算法，我们的模型在提高泛化能力方面取得了显著成效。这种混合方法的应用，有效地弥补了单纯使用深度学习算法时可能遇到的过拟合问题，同时保持了模型在处理复杂数据时的高效性。

综上所述，本研究在技术应用上的创新，不仅优化了处理流程，还在精度和效率方面取得了显著的提升，对芯片设计领域的研究和实践具有重要的启发意义。

6.2 作品遇到的问题和解决方法

6.2.1 数据预处理

1. 处理速度优化

在数据预处理阶段，我们首先遭遇的挑战是处理速度缓慢。为了提升效率，我们采用了多进程加上流程并行技术。这种方法显著加快了预处理步骤，使得大量数据集能够在合理的时间内被处理，从而为后续的模型训练节省了宝贵的时间。

2. 特殊字符识别问题

我们在使用 Word2Vec 模型时，发现系统无法识别特定字符，如'[' 和 ']'。为了解决这一问题，我们创造性地采用了'-' 和 '_' 组合的方式替换这些字符，即使用'-'代替'['，以及'_'代替']'。这一改动使得 Word2Vec 能够无障碍地处理文本数据，而不影响字符的原始含义。

3. 坐标向量重叠问题

在由 word2vec 模型推理名称坐标向量时，我们遇到了某些向量重叠的问题，这可能会导致信息的丢失。为了克服这一挑战，我们采用了螺旋搜索算法，通过将相邻的点赋予重叠向量，确保了所有的信息都能被妥善保存并用于后续分析。

6.2.2 模型训练

1. 数据拉伸参数设置

在模型训练过程中，合理设置数据拉伸参数是确保模型性能的关键。我们通过计算每

个芯片集所有通道的最大值，并取其 97.5% 的分位点作为参数设置的依据。这一策略帮助我们有效地处理了数据，使模型能够更好地适应不同的输入范围。

2. 边缘预测误差

我们注意到，在使用 numpy 矩阵进行计算时，边缘处的预测误差较大。为了解决这一问题，我们对边缘 3 像素内的数据应用了特定的激活函数处理，这样做的结果显著减小了边缘预测误差，提高了模型的整体准确度。

3. 平衡数据规律性与波动性

最关键的挑战是在进行聚类后如何平衡数据的规律性和波动性。为了达到最佳的预测精度，我们采用了中位数滤波处理方法。这种方法在平滑数据波动的同时，保留了数据的核心规律性，为我们在模型训练中取得了最大平衡点。

7 参考文献

- [1] Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125–1134).
- [2] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III* 18 (pp. 234–241). Springer International Publishing.
- [3] Akar, H., Abou Diab, J., Sbeity, F., Abou Ali, M., Kassem, A., & Hamawy, L. (2023, October). A comparative study for lung, colon and breast cancer diagnosis using different convolutional neural networks. In *2023 Seventh International Conference on Advances in Biomedical Engineering (ICABME)* (pp. 75–78). IEEE.
- [4] Zhou, D., Luo, Y., Zhang, Q., Xu, Y., Chen, D., & Zhang, X. (2023). A Lightweight Neural Network for Loop Closure Detection in Indoor Visual SLAM. *International Journal of Computational Intelligence Systems*, 16(1), 49.
- [5] Xie, Z., Li, H., Xu, X., Hu, J., & Chen, Y. (2020, November). Fast IR

drop estimation with machine learning. In *Proceedings of the 39th International Conference on Computer-Aided Design* (pp. 1–8).

[6] Xie, Z., Ren, H., Khailany, B., Sheng, Y., Santosh, S., Hu, J., & Chen, Y. (2020, January). PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)* (pp. 13–18). IEEE.

[7] Bernard, D., Biabiany, E., Cécé, R., Chery, R., & Sekkat, N. (2022).

Clustering analysis of the Sargassum transport process: application to beaching prediction in the Lesser Antilles. *Ocean Science*, 18, 915–930.

<https://doi.org/10.5194/os-18-915-2022>

[8] Shang, L., Liu, C., Tang, F., Chen, B., Liu, L., & Hayashi, K. Machine-Learning-Based Olfactometry: Odor Descriptor Clustering Analysis for Olfactory Perception Prediction of Odorant Molecules. *Entropy*, 20(12), 923.

<https://doi.org/10.3390/e20120923>

[9] Fang, Y. (2018). Feature Selection, Deep Neural Network and Trend Prediction. *Journal of Shanghai Jiaotong University (Science)*, 23(2), 137–142. <https://doi.org/10.1007/S12204-018-1938-5>

[10] Beyhan, S. (2022). Symbolic Regression Based Feature Extraction of Shallow Neural-Networks for Identification and Prediction. *Advances in Science and Technology*, 4(1), 1–13. <https://doi.org/10.33969/ais.2022040103>