Name: Shu Gu
Matriculation number: S2094833
MEng Honours Project Phase 1 Report
Intelligent Software for Circuit Acceleration
17 January 2025

Mission Statement Template

Project Title: ***Intelligent Software for Circuit Acceleration***

Student: ***Shu Gu # S2094833***

Academic Supervisor: ***Dr. Danial Chitnis***

Project Definition:

The project focuses on accelerating circuit simulation, a critical field demanding substantial computational resources due to the complexity and size of modern electronic circuits. It aims to harness advanced computing technologies—CPUs, GPUs, and FPGAs— to enhance efficiency and innovation across various applications, including scientific research, machine learning, and data analytics. The objectives are to benchmark and optimize these technologies for faster simulation times, improve accuracy, and reduce energy consumption, thereby enabling more innovative and sustainable electronic design practices. This initiative seeks not only to advance circuit simulation methodologies but also to equip participants with hands-on experience in cutting-edge computational technologies, fostering collaboration with industry and academia.

Main tasks:

- *Related Studies: Detailed studies demonstrating the current application of accelerated simulation techniques to real-world electronic design challenges, showcasing potential improvements in design cycle times, accuracy, and overall project efficiency.*
- *Enhanced Simulation Tools: Development of optimized simulation software that leverages CPUs, GPUs, and FPGAs for faster execution times.*
- *Benchmarking Reports: Comprehensive benchmarking reports that detail the performance gains achieved through the optimization of circuit simulation on various hardware platforms.*

Scope for extension:

- *Cross-platform Optimization: Explore optimization techniques across a broader range of computing platforms to ensure wide applicability and accessibility of simulation acceleration technologies.*
- *Standardization: Work on standardizing the benchmarks and methodologies used for circuit simulation acceleration, facilitating comparison, and adoption across the industry.*

Background knowledge:

- *Understanding of computer architecture*
- *C++ Programming*
- *Python Programming*
- *Understanding of circuit simulation theory*
- *Knowledge of parallel computing and optimization*

Resources:

- *Intel SYCL development tools and (or) Nvidia CUDA development tools*

- *High performance CPU and GPU*
- *Dataset of generated circuits*

Location/s:

- *No specific location required*

The supervisor and student are satisfied that this project is suitable for performance and assessment in accordance with the guidelines of the course documentation.

Signed

Student: Shu Gu Shu Gu.....

Academic Supervisor: Daniel Chitnis.....

Date: 12/03/2024

Abstract

The complexity of modern electronic circuits demands innovative approaches to accelerate circuit simulation processes, which are critical for verifying and optimizing designs in various industries. This project introduces advanced methodologies for circuit matrix generation and solver optimization by leveraging state-of-the-art machine learning and computational tools. The proposed framework encompasses diverse matrix generation methods, benchmarking, and machine learning-based solver selection to enhance performance. Key contributions include a novel matrix generator and a classification-based solver selection model, demonstrating significant improvements in simulation efficiency and accuracy. The outcomes of this work not only advance the field of circuit simulation but also provide valuable insights for broader applications in electronic design automation (EDA), contributing to the development of sustainable and scalable technologies.

Declaration of Originality

I declare that this thesis is my
original work except where stated.

Statement of Achievement

This project has successfully implemented a machine learning-based framework for the selection of intelligent solvers in circuit simulation, addressing a critical need for efficiency and scalability in electronic design automation (EDA). The project created a diverse and representative dataset that mirrors real-world circuit scenarios by developing robust matrix generators, including topology-based and purely random methods. This dataset enabled comprehensive benchmarking and evaluation of solvers, contributing valuable insights into solver performance under varying conditions.

A key achievement of this project is the integration of machine learning classifiers: Random Forest, K-Nearest Neighbors (KNN), Bayesian Networks, and Support Vector Machines (SVM), into the solver selection pipeline. The results demonstrated exceptional classification accuracy, with Random Forest and KNN achieving over 94%, and significant reductions in computational runtime through accurate solver prediction. Developing and refining the Average Runtime Error (ARE) metric further ensured precise evaluation of the framework's effectiveness.

Furthermore, the project successfully profiled and compared the performance of the leading sparse matrix solvers, including KLU, NICSLU, PARDISO, and GLU, offering valuable benchmarks for future research and industrial applications. By combining domain-specific knowledge with cutting-edge machine learning techniques, this work establishes a scalable, adaptive, and efficient solution for circuit simulation.

The outcomes of this research contribute to advancing EDA methodologies, fostering innovation in circuit design, and laying the groundwork for future enhancements in solver prediction and matrix generation. The project's results demonstrate technical achievements and highlight the potential for societal and commercial impact, setting a strong foundation for further exploration and application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The complexity of modern electronic circuits continues to challenge Moore's Law, increasing the demand for efficient computational methods in circuit simulation [1]. Circuit simulation, essential for verifying and optimizing designs before production, faces growing demands for accuracy and speed that often exceed existing tools' capabilities. Sparse matrix solvers like KLU, NICSLU, and PARDISO [1]–[4] are widely used in scientific computing but lack optimizations specific to circuit matrices, limiting their effectiveness.

A key challenge in circuit simulation lies in the variability of solver performance during the matrix-solving phase, especially in LU decomposition [5]. Solver efficiency is highly dependent on matrix properties such as sparsity, connectivity, and dimensionality. Addressing this challenge requires advanced methods to intelligently select the best solver for a given matrix, reducing simulation time while maintaining accuracy [6].

Building on Phase 1, which identified areas for improvement, this Phase 2 report introduces new tools and methodologies for matrix generation and solver optimization. Phase 1 demonstrated the potential of leveraging sparsity and structural patterns in matrices to improve solver performance and applying machine learning to predict suitable solvers based on matrix characteristics [5]. This phase builds on those insights by implementing two new matrix generation methods: a constrained random generator and a neural network-based generator that mimics real-world circuit matrices. These methods enhance training dataset diversity and provide robust benchmarks for evaluating solver performance.

The central objective of this work is to develop a machine-learning framework capable

of predicting the optimal solver for a given circuit matrix. This involves creating a robust training dataset from diverse matrices and benchmarking solver performance under various conditions. The methodologies introduced in this phase aim to accelerate the circuit simulation process while providing scalable and intelligent tools for addressing the complexities of modern electronic circuits.

The following chapters outline the theoretical background, methodologies, and experimental results, concluding with an analysis of the impact and potential applications of these advancements. This report aims to contribute to the development of efficient and adaptive simulation tools that align with the evolving demands of circuit design.

Chapter 2

Background

The foundational role of circuit simulation in modern electronic design is providing a cost-effective and efficient alternative to physical prototyping. By simulating the behavior of circuits, engineers can identify potential issues and performance in a virtual environment, significantly accelerating the design cycle [7].

This chapter builds upon the Phase 1 background by revising key topics such as circuit simulation methods, matrix-solving techniques, and the integration of machine learning. Phase 2 introduces further explorations into circuit generation methods and advanced machine learning applications tailored to circuit simulation challenges. These extensions aim to deepen the understanding of the state-of-the-art and provide a solid foundation.

2.1 Circuit Simulation and Matrix Solving

2.1.1 LU Decomposition

Circuit simulation involves transforming circuit netlist descriptions into mathematical representations, and then analysing to predict the circuit's behaviour under various conditions. Modern SPICE-like simulators support a wide range of analyses, including DC, AC, transient, and noise analyses [9]. The Computational core of circuit simulation lies in solving the large, sparse linear system of equations generated from the netlist by MNA [10]. These systems are typically expressed in the form:

$$Y_v = J$$

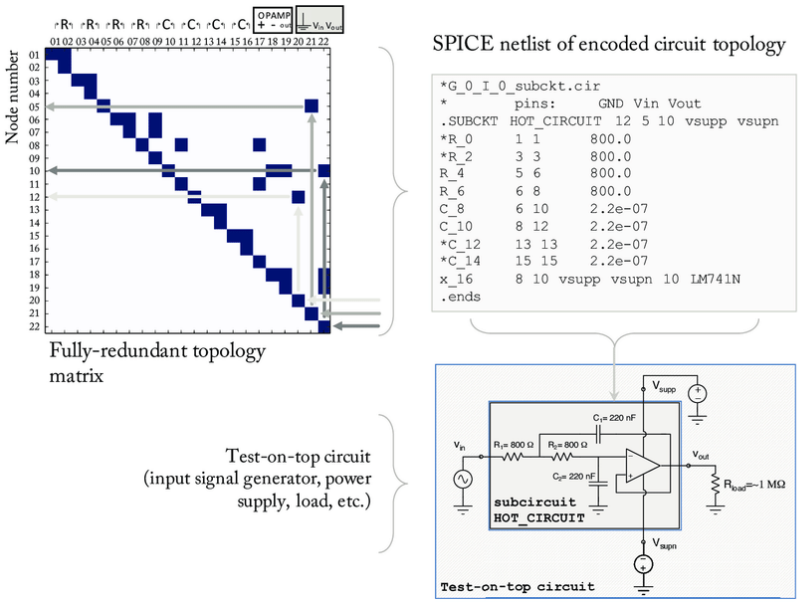


Figure 2.1: Circuit's Transform Among Netlist, Schematic, and Matrix [8]

Where Y is the nodal admittance matrix, v is the vector of node voltages, and J represents current sources. Solving these equations efficiently is critical, as circuit matrices are often highly sparse and exhibit specific structural characteristics.

LU decomposition is commonly employed to decompose the matrix into lower and upper triangular forms for efficient solution as shown in example matrix 2.1. However, the computational complexity of LU decomposition scales cubically with the matrix size, which can become a bottleneck for larger circuits [11]. Furthermore, the optimal reordering of sparse matrices is an NP-hard problem [12], making it computationally infeasible to always find the best permutation for a large system.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \tag{2.1}$$

2.1.2 Brief Review on Solvers: KLU, GLU, NICSLU, and PARDISO

In the realm of circuit simulation, efficient and reliable solvers are essential for handling the large, sparse matrices that arise from Modified Nodal Analysis (MNA). This section provides a concise overview of four prominent solvers: KLU, NICSLU, GLU, and PARDISO, each offering unique approaches towards sparse LU factorization.

KLU [1] is a sequential solver specifically optimized for circuit matrices, which are typically sparse and unsymmetric. It employs the Gilbert-Peierls (G/P) left-looking algorithm without supernodes, making it well-suited for the irregular structures of circuit matrices. KLU has been integrated into tools like MATLAB and the Xyce circuit simulator [13], demonstrating high performance in circuit simulations.

NICSLU [4] builds upon the foundations of KLU, NICSLU introduces parallelism to enhance performance on multicore processors. It utilizes a two-mode scheduling algorithm that combines cluster and pipeline modes to exploit parallelism effectively.

GLU [14] is designed to leverage the massive parallelism of Graphics Processing Units (GPUs) for LU factorization. According to the Peng et al. [14], unlike traditional solvers that rely on dense matrix operations, GLU adapts to the sparse and irregular nature of circuit matrices. This can substantially accelerate the parallel computing of circuit matrix LU factorization.

PARDISO [15] is a high-performance, parallel sparse direct solver that utilizes a combination of left- and right-looking Level-3 BLAS operations to exploit parallelism. It is designed for a wide range of sparse matrices and is known for its efficiency in solving large-scale linear systems. PARDISO has been integrated into various scientific computing environments and has demonstrated robust performance across different applications [16].

Previous study done by Razik et al. [17] comparing KLU, NICSLU, and GLU in the context of power system simulations found that KLU and NICSLU achieved similar performance for total execution times on individual matrices. However, according to their experiment, GLU, despite its massive parallelization for GPUs, could not compete with current CPU-based implementations in certain scenarios. The study suggests that while GPU-based solvers like GLU offer potential, CPU-based solvers such as KLU and NICSLU currently provide more consistent performance for power system simulations.

2.2 Circuit Generation and Solver Prediction

2.2.1 Tradition Circuit Generation

Circuit generation serves as a cornerstone of electronic design, enabling the synthesis of circuit layouts that adhere to electrical and functional constraints.

Traditional circuit topology generation relies on mathematical modeling and algorithmic techniques to construct and analyze circuit configurations [9]. Graph theory forms the foundation of these methods, where circuits are represented as graphs, with nodes corresponding to electrical components and edges denoting the connections between them. Central to this representation is the use of node incidence matrices and loop matrices, which encode the connectivity of circuit elements and facilitate the application of Kirchhoff's Voltage Law (KVL) and Kirchhoff's Current Law (KCL) [18].

For example, Wasynczuk et al. [19] proposed an automated state-space model generation algorithm for power circuits, which utilized standard node incidence matrices and elementary branch data to derive system state equations.

Another seminal contribution to topology-based methods was made by Lin et al. [20], who developed an algorithm for constructing voltage multiplier circuits. Their method systematically generated n-fold voltage multipliers using n capacitors and n diodes, ensuring that the generated topologies satisfied key electrical constraints such as maintaining tree structures for capacitors and diodes.

2.2.2 State-of-the-Art: Machine Learning Based Generation

Recent advancements in ML have redefined circuit topology generation by automating and optimizing processes that were traditionally manual and time-intensive. Reinforcement Learning and Transformer-based architectures have emerged as key technologies, offering unprecedented flexibility and efficiency in both analog and digital circuit design [21].

One of the example is AutoCkt [22], a deep reinforcement learning framework designed to address the challenges of analog circuit design, particularly in optimizing post-layout circuit parameters. Developed by Settaluri et al. [22], AutoCkt employs an agent that mimics the iterative thought process of human designers to explore and converge on circuit parameters that meet specific design specifications. Unlike traditional optimization-based approaches, AutoCkt learns a generalized understanding of the design space through

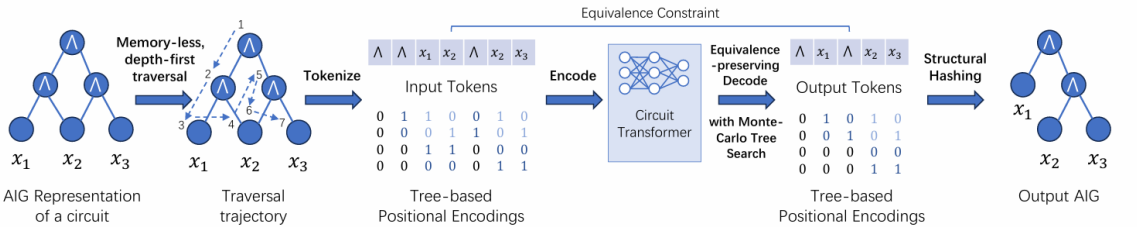


Figure 2.2: Topology Based And-Inverter Graph Generation Using Transformer[23]

sparse subsampling, enabling it to reliably generate solutions for a wide range of target specifications with significantly improved efficiency.

Transformer-based architectures have also demonstrated remarkable capabilities in digital circuit synthesis. The Circuit Transformer [23], for instance, employs a neural model to predict the next logic gate in a circuit while preserving design equivalence. This end-to-end approach ensures that generated circuits adhere to predefined constraints, offering a robust solution for logic synthesis and optimization as shown in figure 2.2.

2.2.3 Solver Selection as Classfication Problem

Machine learning has emerged as a transformative tool for addressing the limitations of traditional circuit simulation techniques. Building on these foundations, both traditional classification methods like KNN and SVM, and neural network architectures have been investigated for their ability to offer a solution in capturing the relationships inheret in matrices, and optimizing time-step selection in transient analysis and predicting convergence rates for iterative solver [24]–[26]. These applications underscore the versatility of machine learning in enhancing the computational efficiency of circuit simulation workflows. Recent advancements have transformed solver selection into a classification task. Researchers have employed neural networks to predict the optimal solver for sparse linear systems, leveraging features derived from matrices. For example, Sun et al.[27] proposed an efficient Krylov solver prediction strategy that combines computationally inexpensive feature selection and a classifier, achieving over 90% accuracy. Similarly, Funk et al.[28] utilized neural networks to classify Krylov solvers, emphasizing sparsity pattern features

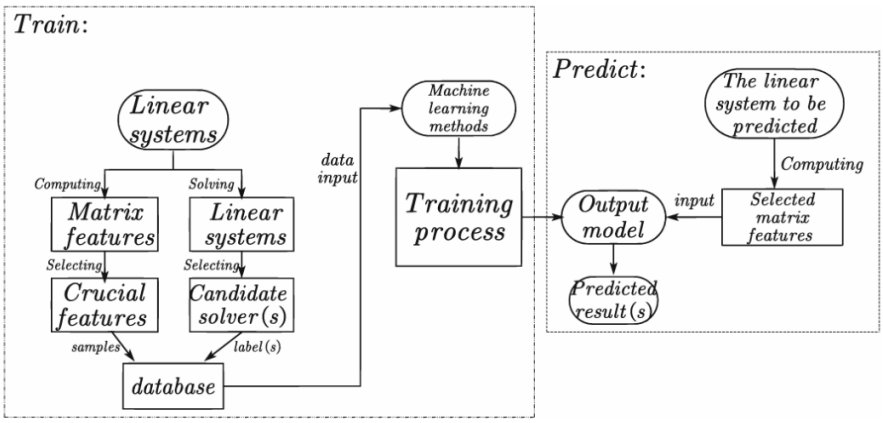


Figure 2.3: Matrix Solver Classification Framework Proposed by Sun et al. [27]

and obtaining top-1 accuracy of 60%. These approaches highlight how intelligent solver picking accelerates computation by avoiding trial-and-error methods.

But nevertheless of these previous work are applied towards predicting circuit matrix or linear sparse matrix, but general scientific problem-based dense matrices. Usually sparse matrices and circuit matrices have different properties such as high sparsity, symmetry, and specific patterns of diagonal dominance, which distinguish them from general sparse or dense matrices.

In this context, developing a specialized framework that leverages matrix-specific features such as sparsity patterns, connectivity information, and spectral properties can bridge the gap between generic ML applications and circuit-oriented solver prediction. By incorporating domain knowledge into the feature engineering process, such frameworks could significantly improve the prediction accuracy and computational efficiency of solver selection tasks in circuit simulations.

Chapter 3

Impact and Exploitation

The integration of advanced machine-learning techniques into circuit simulation has significant potential to transform various industries, benefit society, and advance academic research. This chapter explores the societal benefits, commercial potential, and pathways for broader adoption of this innovative approach.

3.1 Societal Impact

The development of intelligent simulation tools has far-reaching implications for society, addressing critical challenges in technological accessibility, environmental sustainability, and public infrastructure [21]. By reducing computational costs and energy consumption, the project democratizes access to advanced simulation tools, enabling small and medium-sized enterprises (SMEs) to innovate without substantial financial constraints. Furthermore, the energy efficiency achieved through these advancements aligns with global sustainability goals, supporting the design of low-power electronics and the optimization of renewable energy systems.

In addition, the tools developed in this project can be integrated into educational curricula, equipping students and professionals with skills in computational efficiency, machine learning, and electronic design. This fosters a new generation of engineers capable of addressing complex problems in industries ranging from telecommunications to healthcare [29]. Enhanced circuit simulation accuracy also contributes to the development of more reliable infrastructure, such as resilient power grids and efficient communication

networks, ultimately improving the quality of life for communities worldwide [21].

3.2 Business Potential

The economic impact of this project is equally significant, with the electronic design automation (EDA) market projected to grow from 17.2 billion USD in 2024 to 31.2 billion USD by 2030 at a compound annual growth rate (CAGR) of 10.5% [30]. This growth is fueled by the increasing complexity of semiconductor designs and the adoption of AI-driven tools that accelerate development cycles and reduce costs. Companies such as Synopsys and Cadence Design Systems are already embedding AI into their EDA platforms, achieving substantial revenue growth and highlighting the demand for such innovations [31], [32]. The flexibility and scalability of these tools make them particularly attractive for businesses seeking to enhance design efficiency and achieve faster time-to-market. Furthermore, collaborations between academic institutions and industry players could lead to the development of proprietary solutions tailored to specific domains, fostering innovation and competitive advantage.

3.3 Pathways to Broader Adoption

To ensure widespread adoption, efforts must focus on establishing industry standards and fostering interoperability with existing tools. Standardized benchmarks and methodologies for circuit simulation would enable fair comparisons and encourage the integration of intelligent solvers into industrial workflows. Additionally, the transition to cloud-based platforms presents an opportunity to make these advanced tools accessible to a wider audience.

Open-source contributions can also accelerate the adoption of these technologies by fostering collaboration among researchers and developers. By sharing parts of the framework with the wider community, this project can stimulate further advancements and encourage novel applications across diverse fields. Finally, targeted educational initiatives and training programs will be essential to equip professionals with the skills necessary to harness the full potential of AI-driven EDA tools.

3.4 Challenges and Future Directions

Despite the promising opportunities, challenges remain in scaling these tools for extremely large systems and ensuring the interpretability of machine learning models in industrial contexts. Addressing these issues will require continued research into hybrid approaches that combine domain-specific physical models with advanced machine learning techniques [33]. Expanding datasets to capture a broader range of circuit scenarios and incorporating more sophisticated feature engineering methods will further enhance the robustness and reliability of predictions.

The potential of this project to influence society, business, and academia is immense, offering innovative solutions to longstanding challenges in electronic design and setting a foundation for future advancements. With strategic implementation and collaboration, the outcomes of this research can drive significant progress in technology and beyond.

Chapter 4

Random Matrix Generator

4.1 Matrix Generation

Sparse matrices play a pivotal role in a wide range of computational fields, including circuit simulation, large-scale optimization, and machine learning. In the context of circuit simulation, sparse matrices arise naturally due to the limited connectivity of circuit components, where each node is typically connected to only a few others. This sparsity leads to matrices that are computationally expensive to solve, particularly as the system size grows. Matrix-solving techniques, such as LU decomposition, are critical for numerical solutions but often constitute the primary bottleneck in simulation workflows.

The performance of sparse matrix solvers varies significantly depending on matrix properties such as sparsity, structure, and condition number [34]. To address this variability, there is an increasing interest in using machine learning to predict the optimal solver for a given matrix. However, such an approach requires a substantial and diverse dataset of matrices as training data. Accessing real-world circuit matrices is challenging due to intellectual property constraints, as organizations often guard their designs to protect proprietary information. Consequently, there is a critical need for synthetic matrix generation methods to produce diverse datasets that mimic the properties of real-world circuits. These generated matrices enable robust benchmarking of solvers and increase the coverage and precision of machine-learning models for solver selection.

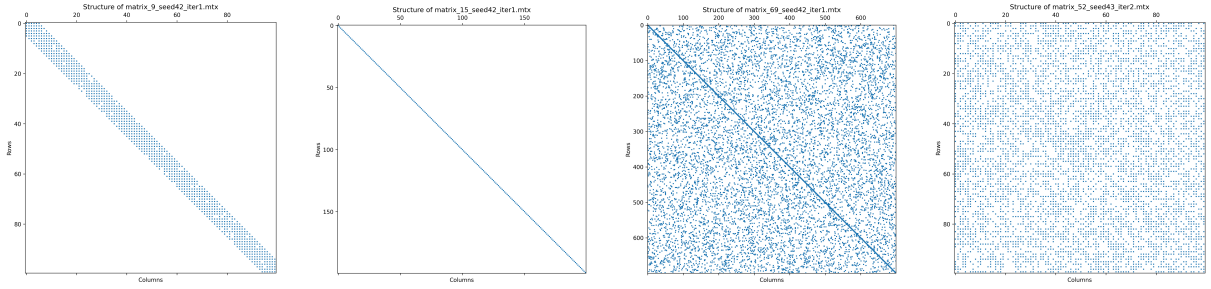
Based on this, a random matrix generator was developed and implemented in this project. It has three major modules: pure random matrix generation, topology-based

circuit matrix generation, and neural-network generation.

4.1.1 Novel Purely Random Matrix Generator

The matrix generator is designed to address the dual objectives of dataset diversity and computational robustness. The generator employs a highly configurable framework, allowing users to control key matrix attributes such as size, sparsity, numerical range, and structural type. The core methodology involves constructing matrices based on user-defined configurations, validating them to ensure numerical properties, and storing them in a widely accepted format for subsequent analysis.

Matrix generation begins with the selection of a structural type, which can include diagonal, banded, sparsely random, or positive-definite matrices.



(a) Banded **(b)** Diagonal **(c)** Positive Definite **(d)** Sparsely Random

Figure 4.1: Shape of Generated Matrices

For each matrix, sparsity is determined as the fraction of nonzero elements, which is either directly specified or adaptively adjusted based on the matrix size. Numerical values are assigned to matrix entries within a user-defined range, with additional steps to enforce properties such as diagonal dominance or positive-definiteness when required. This ensures the matrices are not only stable for computational purposes but also exhibit features commonly observed in real-world circuit systems.

Validation is a critical component of the generation process. Each matrix is checked for rank, condition number, and adherence to specified sparsity levels. Matrices that fail these checks are discarded, and the generator attempts new configurations until a valid matrix is produced or a predefined maximum number of attempts is reached. This iterative approach ensures that only matrices meeting rigorous numerical criteria are included in

the final dataset. Once validated, matrices are saved in the Matrix Market format, making them compatible with popular solver libraries and machine learning frameworks.

4.2 Topology Based Circuit Matrix Generator

4.2.1 Enhancements Over previous Work

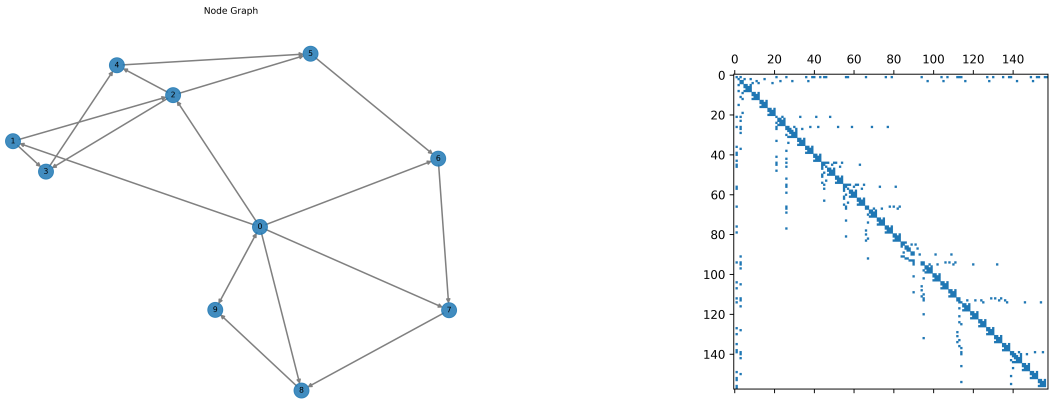
This topology-based circuit matrix generator significantly improves upon earlier methods proposed by previous student Huo Lang, and improved in this work by introducing greater flexibility and realism. It incorporates multiple connection methodologies, including random, fully connected, scale-free, and small-world networks. These options allow for a diverse range of circuit topologies, closely resembling real-world connectivity patterns. The generator ensures all nodes are interconnected by introducing a directed cycle as the backbone of the graph. This guarantees that each node has at least one incoming edge while enforcing user-defined constraints, such as a maximum in-degree.

Another key improvement lies in the module assignment process, which matches circuit components to nodes based on the number of required inputs. This ensures that each node is assigned a component consistent with the desired circuit structure, enhancing the realism of the generated topologies. Additionally, the generator includes robust visualization and output features.

4.2.2 Methodology and Implementation

The topology-based circuit matrix generator is designed to create diverse and realistic circuit topologies while ensuring numerical stability. The process begins with generating a directed graph G of n nodes, each represented by a custom `MyNode` object as shown in figure 4.2. To guarantee basic connectivity, the algorithm first constructs a directed cycle that connects all nodes. This step ensures no node is isolated and provides a foundation for further structural refinement.

Depending on the selected connection method, additional edges are introduced while adhering to user-defined constraints, such as maximum in-degree. Random connectivity adds edges stochastically, while fully connected topologies link all permissible nodes. Scale-free and small-world models emulate real-world networks with preferential attach-



(a) Topology Connection Among Nodes **(b)** Topology Based Circuit Matrix

Figure 4.2: Generated Circuits Visualization: (a) Topology graph of generated circuit, the nodes represent analogue or digital devices connected; (b) Sparse matrix created after MNA analysis of the generated connection of devices

ment and clustering, respectively, while ensuring compliance with in-degree limits. Successor and predecessor relationships are updated for all nodes after edge construction to maintain graph consistency.

Module assignment follows, where each node is matched with a circuit component from a predefined library. The assignment considers the in-degree of each node to ensure compatibility with the number of required inputs for each module. Nodes without input constraints, such as start nodes, are exempt from this requirement, further enhancing flexibility. Once assigned, each node instantiates its module and prepares for matrix assembly.

Matrix construction involves several steps. A depth-first search traversal assigns indices to nodes, ensuring consistent mapping to matrix rows and columns. Voltage source integration, handled through a dedicated `Vs` class, establishes global power rails and appends corresponding entries to the global matrices A and b in $Ax = b$. Each node contributes local sub-matrices, which are aggregated to form the final global matrix system. The completed matrices are stored in Matrix Market format, with a complementary visual representation of the graph saved as an image.

4.2.3 Limitations

Despite its advancements, the generator has some limitations. While it captures basic connectivity and numerical stability, it does not fully account for all electrical or domain-specific constraints. For example, the generator does not model intricate physical behaviours, such as transistor-level parasitics or detailed layout rules, limiting its use for highly specialized simulations.

Randomization in module assignment and topology generation may result in networks that lack the hierarchical or modular structure found in large-scale real-world circuits. Moreover, while the generator handles large circuits effectively, extremely dense topologies or very high node counts may push memory and runtime constraints.

Another limitation is the reliance on numerical validation over domain-specific checks. While properties such as rank and condition number are verified, physical constraints like Kirchhoff's current laws are not universally enforced. Consequently, the matrices may not fully reflect the nuances of real-world electrical networks.

4.3 Neural Network Matrix Generator

This is a proposed neural-network–based approach to generating synthetic circuit matrices using diffusion models in this chapter. The method treats matrices as 2D images, applying a diffusion process followed by a learned denoising step. This approach, leveraging a Denoising Diffusion Probabilistic Model (DDPM) [35] with a lightweight U-Net architecture, generates realistic matrices for circuit simulations while maintaining their sparse structural characteristics. But due to its high complexity, it was not implemented to be fully functional at the time point of this report finished, which will be mentioned later in the Limitation and Challenge section.

4.3.1 Diffusion Models

Diffusion models offer a powerful mechanism for data generation by progressively adding noise to input data during a forward pass and learning to reverse this process through denoising. This iterative process enables the creation of realistic samples that closely resemble the training data distribution [36].

In the current implementation, circuit matrices are treated as 2D grids, which allows the use of image-based architectures such as CNNs. Although Graph Neural Networks (GNNs) are more suited for explicitly capturing graph topology [37], diffusion models provide a simpler yet effective alternative by leveraging convolutional operations to model matrix structure indirectly [38]. The DDPM framework used here applies a series of Gaussian noise perturbations during the forward process, creating intermediate noisy states. The reverse process is learned by training the U-Net [39] model to predict and remove the noise.

4.3.2 Framework and Implementation

The neural network matrix generator comprises three main components: the diffusion process, the U-Net architecture, and the training and sampling procedures as shown in figure 4.3. The implementation is described in detail below.

4.3.2.1 Diffusion Process

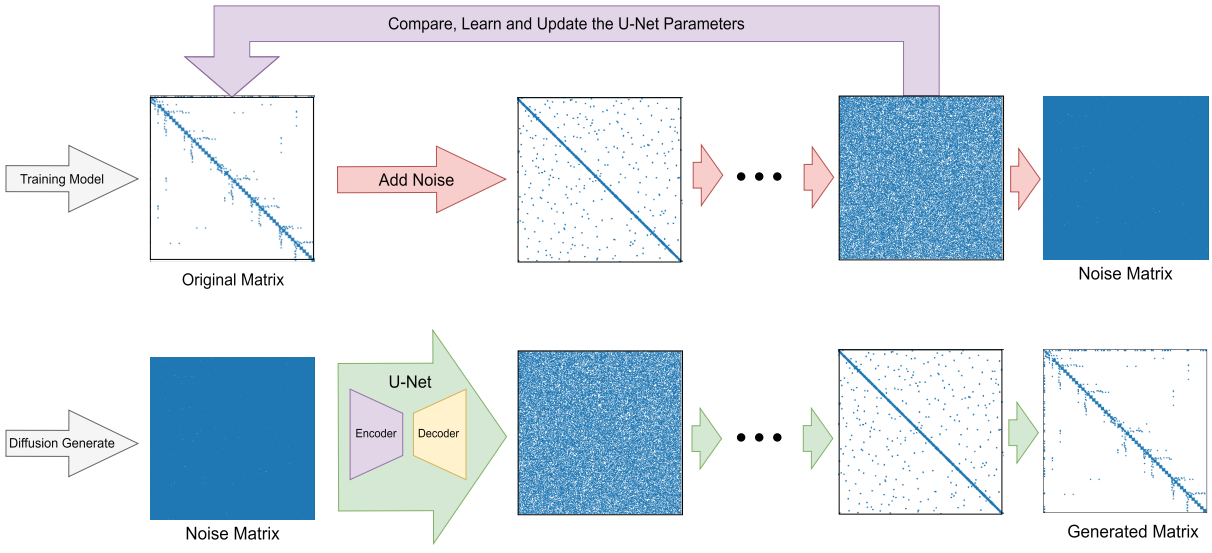


Figure 4.3: Illustration of the Training and Denoising Diffusion Processes in a U-Net-Based Diffusion Model

The forward diffusion process progressively adds Gaussian noise to the input matrix x_0 over T time steps. The noisy matrix at time step t, x_t, is computed as:

$$x_t = \sqrt{\alpha_{\text{bar},t}} \cdot x_0 + \sqrt{1 - \alpha_{\text{bar},t}} \cdot \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, I)$ is Gaussian noise, and $\alpha_{\text{bar},t}$ is the cumulative product of noise reduction factors $\alpha_t = 1 - \beta_t$, with β_t linearly scheduled between 10^{-4} and 2×10^{-2}.

The reverse diffusion process denoises x_t iteratively, predicting ϵ using a U-Net model. At each step, the predicted noise is used to approximate x_{t-1} as:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \alpha_{\text{bar},t}}} \cdot \hat{\epsilon}_\theta(x_t, t) \right) + \sigma_t z,$$

where $\hat{\epsilon}_\theta(x_t, t)$ is the noise predicted by the U-Net, $\sigma_t = \sqrt{\beta_t}$, and $z \sim \mathcal{N}(0, I)$ is Gaussian noise added for stochasticity (only for $t > 0$).

4.3.2.2 U-Net Architecture

The U-Net [39] serves as the backbone for noise prediction. It follows an encoder-decoder structure with skip connections to preserve spatial details as shown in figure 4.4. The architecture consists of:

- **Encoder:** Sequential convolutional blocks with 32 and 64 filters, followed by max-pooling layers that downsample the input, capturing multi-scale features.
- **Bottleneck:** Two convolutional layers with 128 filters at the lowest spatial resolution, capturing high-level latent features.
- **Decoder:** Upsampling layers that restore spatial resolution, concatenating skip connections from the encoder to retain spatial context. Convolutional blocks with 64 and 32 filters reconstruct the denoised output.
- **Output Layer:** A 1×1 convolutional layer producing a single-channel output, predicting the added noise (ϵ) at each timestep.

This architecture balances simplicity with effectiveness, offering sufficient capacity for sparse matrix generation while remaining computationally efficient.

4.3.2.3 Training and Sampling

During training, a random timestep t is selected for each batch, and noise is added to the input matrix x_0. The U-Net predicts the noise, $\hat{\epsilon}_\theta(x_t, t)$, and the model is trained to minimize the mean squared error (MSE) loss:

$$\mathcal{L} = E_{x_0,t,\epsilon}\left[\|\epsilon - \hat{\epsilon}_\theta(x_t, t)\|^2\right].$$

Sampling starts from pure Gaussian noise x_T and iteratively denoises it using the learned reverse diffusion process. The final output x_0 represents a generated matrix.

4.3.3 Limitations and Challenges

While effective, the model assumes that matrices can be treated as 2D images, potentially overlooking graph-specific topological information. This simplification limits the ability to capture detailed electrical constraints or domain-specific characteristics inherent in circuit matrices. Moreover, the iterative nature of the diffusion process results in significant computational overhead, particularly for large-scale or high-resolution matrices.

The reliance on binarization for preprocessing may further obscure important numerical relationships in the original data, potentially affecting the fidelity of generated matrices. These limitations highlight the need for more sophisticated models that integrate graph-specific insights and domain-aware constraints.

Also, currently, the implemented diffusion model can only generate fixed-size small matrices (128×128), which reduces its ability to generate and rebuild the structure of large-scale matrices. Therefore, this type of matrix generator is not used in the later experiments as a matrix source.

Future improvements could include incorporating Graph Neural Networks (GNNs) into the diffusion pipeline to better capture topological relationships. Implement the method of slicing the matrix into subblocks during the training and generating steps will also help further increase the ability to capture structural details and enable the generation of matrices of various sizes.

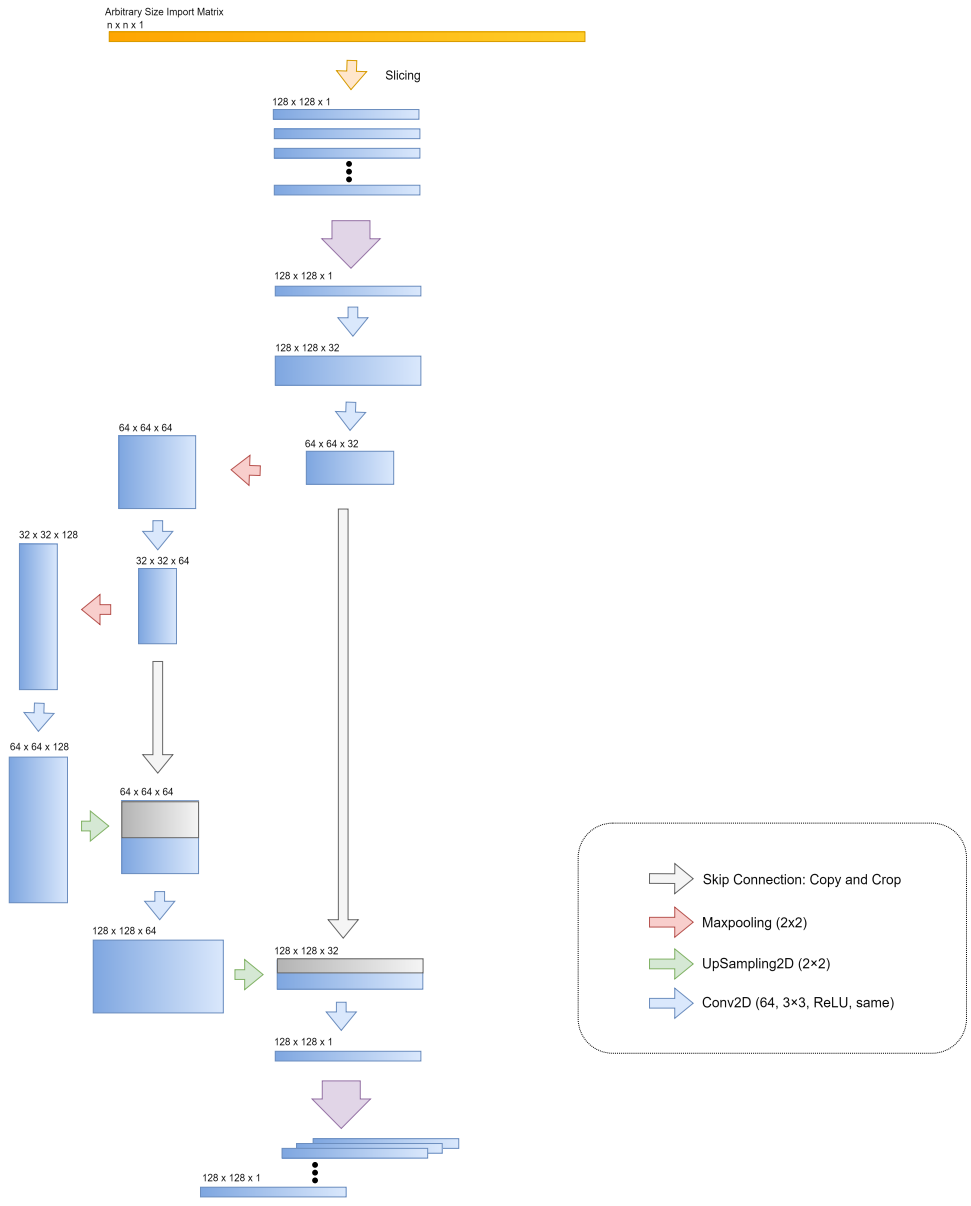


Figure 4.4: A schematic of the lightweight U-Net architecture used for diffusion noise prediction, showing the downsampling encoder, bottleneck, upsampling decoder, and skip connections, all culminating in a 1×1 convolution layer for predicting the noise.

Chapter 5

Benchmarking and Profiling of Solvers

This chapter describes the benchmarking framework for evaluating the performance of various sparse matrix solvers, including KLU, NICSLU, GLU, PARDISO, on synthetic generated and real-world matrices. The implementation, analytical methods, and performance metrics are discussed in detail, supported by insights from the provided code and sample outputs. This framework enables systematic performance evaluation, aiding the selection of solvers tailored to specific computational workloads.

5.1 Structure and Implementation

The benchmarking process is structured around Python scrips as a central orchestrator. The script begins by processing command-line arguments to determine the source of the matrices. Users can choose to use pre-downloaded matrices from SuiteSparse Matrix Collection [40], [41], generate matrices using a random circuit generator, or employ a purely random matrix generator. Once the data source is specified, the script organizes the matrices into designated directories, ensuring compatibility with subsequent benchmarking tasks.

For SuiteSparse matrices, the `SuiteSparseManager` handles downloading and organizing the files. In contrast, the random circuit and purely random generators create synthetic matrices with customizable parameters, such as size, sparsity, and structure. The generated or organized matrices, stored in Matrix Market (`.mtx`) format, are then passed to the benchmarking routines for evaluation.

Each solver benchmark, such as KLU, NICSLU, GLU, PARDISO, operates independently on the prepared matrices. For instance, the KLU benchmark, implemented in `run_klu_kernel.py`, identifies all `.mtx` files in the specified directory and processes them sequentially. The benchmark invokes the solver executable through system commands, capturing key performance metrics like analyze and factorization times. The results are logged into a structured CSV file for later analysis.

5.2 Formulas and Analytical Approach

Sparse solvers, such as those evaluated in this framework, aim to efficiently solve systems of linear equations of the form

$$A\mathbf{x} = \mathbf{b},$$

where $A \in R^{n \times n}$ is a sparse matrix. These solvers typically perform the following steps:

1. **Symbolic Factorization:** The matrix A is analyzed to determine an efficient fill-reducing ordering. The goal is to minimize the number of non-zero elements introduced during numerical factorization. This process is represented as:

$$A = P\,L\,U,$$

where P is a permutation matrix, and L and U are lower and upper triangular matrices, respectively.

2. **Numerical Factorization:** The actual entries of L and U are computed. The computational cost is proportional to the number of non-zero entries in L and U, denoted as $\text{nnz}(L+U)$. The cost can be approximated as:

$$\text{Cost} \sim k \cdot \text{nnz}(L+U),$$

where k depends on the solver's specific implementation and hardware parallelization.

3. **Solution Phase:** Once the matrix is factorized, the system is solved for $\mathbf{x}$ using forward and backward substitution.

The benchmarking framework records performance metrics for the symbolic factorization and numerical factorization stages, providing insights into the solver's computational efficiency.

5.3 Performance Metrics

The benchmarking framework evaluates solver performance based on key metrics:

- **Analyze Time** (t_{analyze}) measures the time spent on symbolic factorization and preprocessing.
- **Factorization Time** (t_{factor}) represents the time required for numerical factorization.
- **Total Runtime** (t_{total}) is the sum of analyze and factorization times:

$$t_{\text{total}} = t_{\text{analyze}} + t_{\text{factor}}.$$

Additional metrics, such as speedup and efficiency, can be derived for multi-threaded solvers. Speedup compares the performance of the solver under test to a baseline, while efficiency evaluates the ratio of speedup to the number of threads used. And in this work, we will mainly focus on the factorization time. As the analysis phase in sparse solvers is essential for determining an efficient ordering to minimize fill-in during factorization, its computational cost is generally lower compared to the factorization phase. Additionally, the analysis phase is typically performed once for a given matrix structure and can be reused if multiple systems with the same sparsity pattern but different right-hand sides are solved. In contrast, the factorization must be recomputed if the matrix values change, further emphasizing its impact on performance. [42]

The benchmark results are stored in CSV files, capturing attributes such as the matrix name, number of non-zero elements (nnz), and runtime metrics.

Chapter 6

Machine Learning-Based Solver Picking

6.1 Methodology

This chapter builds upon the foundational Phase 1 methodology, which introduced a machine learning-based approach for classifying sparse matrices to select the optimal solver. The three-stage framework shown in figure 6.1 comprises pre-analysis, solving, and post-analysis stages.

- Pre-analysis Stage: Sparse matrices, derived from circuit netlists, are analyzed to extract manually selected and latent features such as locality, sparsity, and structural patterns. These features, combined with matrix labels, form the training data for classification models.
- Solver Selection: The trained classification model predicts the most efficient solver for a given matrix, prioritizing reduced pre-analysis time to maximize solving efficiency.
- Post-analysis: Analysis of the profiling results and finding potential insightful information.

Building on this, the Phase 2 focuses on implementing and evaluating the machine learning framework for solver selection. The process involves leveraging a diverse dataset of synthetic and real-world-problem based matrices to predict the optimal solver kernel based on matrix characteristics. Runtime errors and the implications of solver mis-classification are analyzed to assess prediction accuracy and error mitigation strategies.

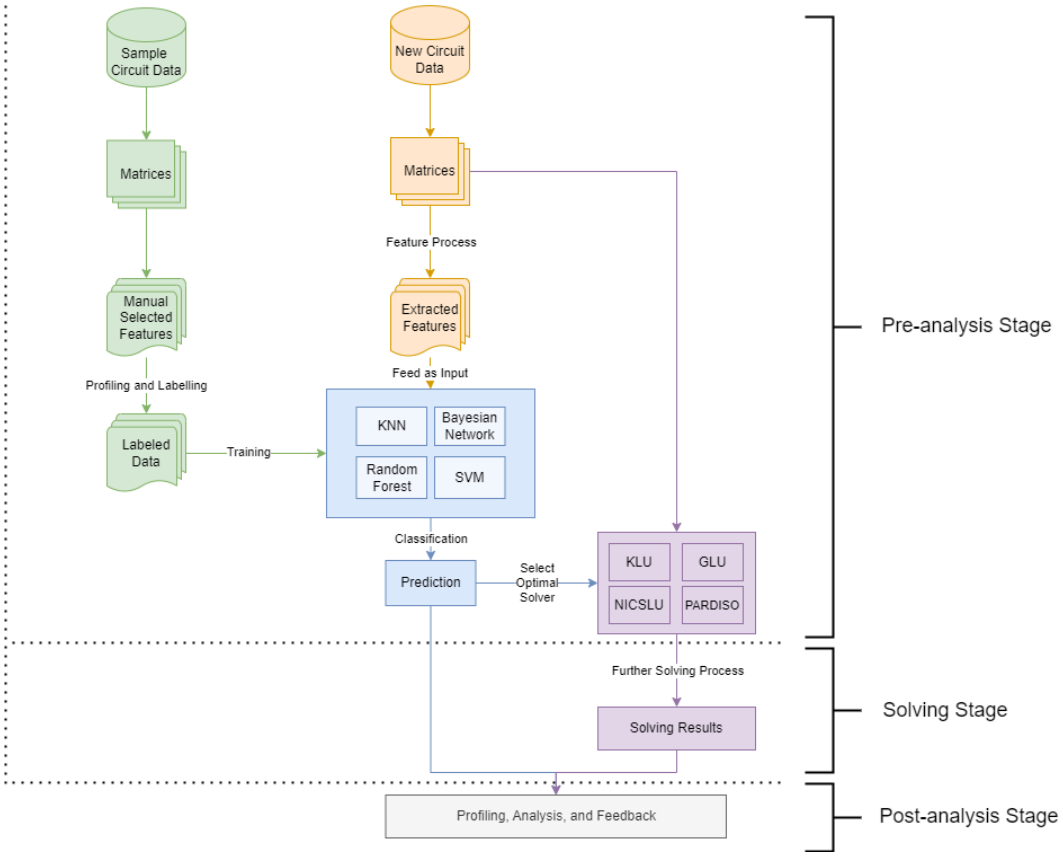


Figure 6.1: Framework for Sparse Matrix Classification and Solver Selection: A Visual Overview of the Pre-analysis, Solving, and Post-analysis Stages

6.2 Experiment

6.2.1 Experiment Setup

The experiments were conducted on a high-performance computational setup with the following specifications:

- **Processor:** AMD Ryzen Threadripper PRO 5995WX with 64 cores and 128 threads, operating at a maximum frequency of 7.02 GHz with a base frequency of 1.8 GHz. The system includes a total of 256 MB L3 cache.
- **Memory:** 128 GB DDR4 RAM, ensuring ample capacity for memory-intensive matrix computations.
- **GPU:** NVIDIA GeForce RTX 4090 with 24 GB GDDR6X VRAM, running on CUDA 12.2 with driver version 535.154.05, enabling accelerated matrix processing and neural network training.
- **Storage:**
  - Samsung SSD 990 PRO 2TB NVMe drive, serving as the primary disk for fast read/write operations during matrix data handling and solver execution.
  - Samsung SSD 870 EVO 2TB for additional storage of datasets and logs.
- **Operating System:** Ubuntu 22.04.4 LTS (Jammy Jellyfish) with kernel version 6.5.0-26-generic.
- **Software Environment:**
  - **Compilers:** GCC 12.2 with OpenMP support for parallelized computations.
  - **Matrix Solvers:** GLU, KLU, NICSLU, and PARDISO libraries, optimized for sparse matrix operations.
  - **Machine Learning Frameworks:** Python 3.9 with essential libraries including NumPy, SciPy, scikit-learn, and TensorFlow.
  - **Visualization Tools:** Matplotlib and Seaborn for generating correlation plots and performance comparisons.

- **Networking:** Dual 10G Ethernet connections for high-speed data transfer during remote access and dataset acquisition.
- **Runtime Configuration:** Docker containerization was utilized for consistent experimental environments across all solvers and machine learning models.

This setup was chosen to ensure reproducibility, scalability, and efficiency in handling the computationally intensive tasks associated with matrix processing and machine learning pipeline execution.

6.2.2 Matrices' Features Selected

Below are the selected features for the linear sparse matrices. Total 33 computational features, most of the features are referenced from the work proposed by Yeom, et al. (2016) [43] and Chen, et al. (2023)[44]. Inside these 33 features, 8 features are omitted due to inconsistent of high memory and computation time required for large-scale matrices. These omitted features are labeled "*(Omitted)" in the following table, and are excluded in the machine learning tasks.

Feature	**Meaning**
Matrix	The identifier for the matrix being analyzed.
Locality	The property of matrix elements being close to each other, often related to the physical proximity in the underlying domain or graph.
Sparsity	The presence of many zero elements in a matrix. Formula: sparsity $= \frac{\text{nnz}}{n^2}$.
Ncol	Number of columns in the matrix.
Nrow	Number of rows in the matrix.
NNZ	Number of non-zero elements, nnz(A).
NNZL	Number of non-zero elements in the lower triangle of the matrix.
NNZU	Number of non-zero elements in the upper triangle of the matrix.

Feature	**Meaning**
NNZmax	Maximum number of non-zero elements per row. Formula: NNZmax = $\max_j(\text{nnz}(A_{j,*}))$.
NNZavg	Average number of non-zero elements per row. Formula: NNZavg = $\frac{\text{nnz}}{n}$.
DEmax	Largest element in magnitude along the diagonal. Formula: DEmax = $\max_j \lvert a_{j,j}\rvert$.
DEmin	Smallest non-zero element in magnitude along the diagonal. Formula: DEmin = $\min_{j \text{ where } a_{j,j}\neq 0} \lvert a_{j,j}\rvert$.
bwL	Bandwidth of the lower triangle. Formula: bwL = $\max_j(j-i)$ for $a_{i,j} \neq 0$ and $i > j$.
bwU	Bandwidth of the upper triangle. Formula: bwU = $\max_j(i-j)$ for $a_{i,j} \neq 0$ and $i < j$.
NO	Number of ones in the matrix.
NDDrow	Number of rows that are strictly diagonally dominant, satisfying $2 \cdot \lvert a_{j,j}\rvert - \sum_{i\neq j} \lvert a_{i,j}\rvert > 0$.
OneNorm	1-norm of the matrix. Formula: $\lVert A\rVert_1 = \max_j \sum_i \lvert a_{i,j}\rvert$.
FroNorm	Frobenius norm of the matrix. Formula: $\lVert A\rVert_F = \sqrt{\sum_{i,j} \lvert a_{i,j}\rvert^2}$.
Ndim *(Omitted)	Number of spatial dimensions derived from the mesh.
Meshdo *(Omitted)	Finite element discretization order.
Meshrl *(Omitted)	Mesh refinement level.
Kappa_hmin	Approximation to the condition number term $\sqrt{\kappa}$, where $\kappa_{h_{\min}} = (1/\min(h))^2$.
Kappa_hmax	Approximation to the condition number term $\sqrt{\kappa}$, where $\kappa_{h_{\max}} = (1/\max(h))^2$.
sp_all	Structural sparsity. Formula: sp_all = $\frac{\text{nnz}}{n^2}$.
n2_structural *(Omitted)	Normalized structural sparsity.
sp_nz	Numerical sparsity. Formula: sp_nz = $\frac{\text{nnz}-\text{nz}}{n^2}$.

Feature	Meaning
n2_numerical *(Omitted)	Normalized numerical sparsity.
psym *(Omitted)	Structural symmetry. Formula: psym $= \frac{p_{\text{nnz}}}{\text{nnz}}$.
nnz_structural_symmetry *(Omitted)	Symmetry of the non-zero structure.
nsym_numerical_symmetry *(Omitted)	Numerical symmetry of the matrix.
bandwidth	The average of the lower and upper bandwidths. Formula: Bandwidth $= \frac{\text{bwL}+\text{bwU}}{2}$.
bandwidth_total	Total bandwidth of the matrix. Formula: Bandwidth Total $=$ bwL $+$ bwU.
coefficient_of_variation	Average coefficient of variation across rows and columns. Formula: $\frac{1}{2n}\sum_{j=0}^{n}[\text{cv}(r_j)+\text{cv}(c_j)]$, where $\text{cv}(x) = \frac{\sigma(x)}{\mu(x)}$.

6.2.3 Dataset Composition

The dataset, comprising 1,267 matrices from four sources (randomly generated, random topology, small-world topology, and real-world matrices from the SuiteSparse Matrix Collection [40], [41]), provides broad coverage of matrix characteristics and solver behaviours:

1. **Circuit Random**: 100 Topology-based randomly generated matrices.
2. **Circuit SmallWorld**: 100 Topology-based matrices reflecting small-world connectivity patterns.
3. **Pure Random**: 200 Matrices with no structured topology.
4. **SuiteSparse**: 892 Real-world matrices from 36 domains, sourced from the SuiteSparse Matrix Collection [40], [41].

This diversity enables the training of robust machine-learning models. A visual examination of the dataset reveals significant trends of the relationship between NNZ and factorization time, which is illustrated in Figure 6.3. Furthermore, the distribution of

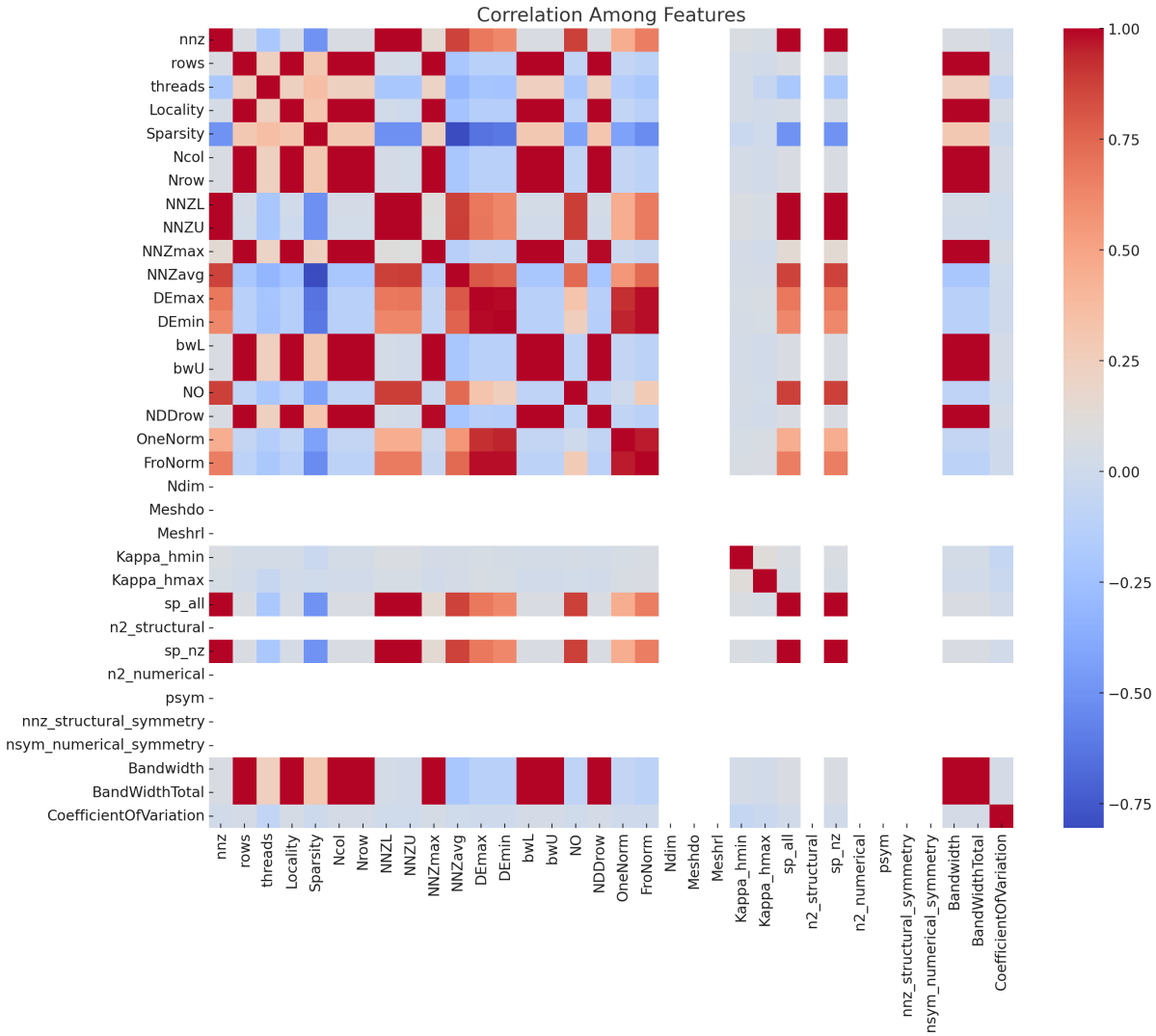


Figure 6.2: Correlation Plot of All Selected Features (See table 6.2.2). Missing blocks indicate features omitted due to the high memory and computation time required to convert sparse matrices to dense format.

solver runtimes is depicted in Figure 6.4. The plot shows their unstable performance under different matrices, with potential to utilize the machine learning tool to pick the optimal solver.

The dataset incorporates benchmarks of all above matrices for four solvers—GLU, KLU, NICSLU, and PARDISO[2]–[4], [45]. Special labeling was introduced for cases where NICSLU fails, defined as having a factorization time below 100 milliseconds for matrices with NNZ $> 10^3$. These cases were labeled as `nicslu_failed` and excluded from being considered the fastest kernel, as NICSLU cannot properly decompose certain matrices that are too large in size and leading to failure of factorization. The failure of NICSLU in these scenarios arises due to several inherent limitations in its approach to matrix decomposition. The primary reason NICSLU fails for certain matrices is the phenomenon of fill-in, which occurs during LU decomposition. Fill-in refers to the introduction of non-zero elements in positions that were previously zero within the matrix. This drastically increases the memory requirements and computational workload, especially for matrices with high sparsity or complex structural patterns [4]. Consequently, NICSLU's ability to efficiently handle such matrices is significantly compromised, leading to unacceptably long factorization times or outright failure.

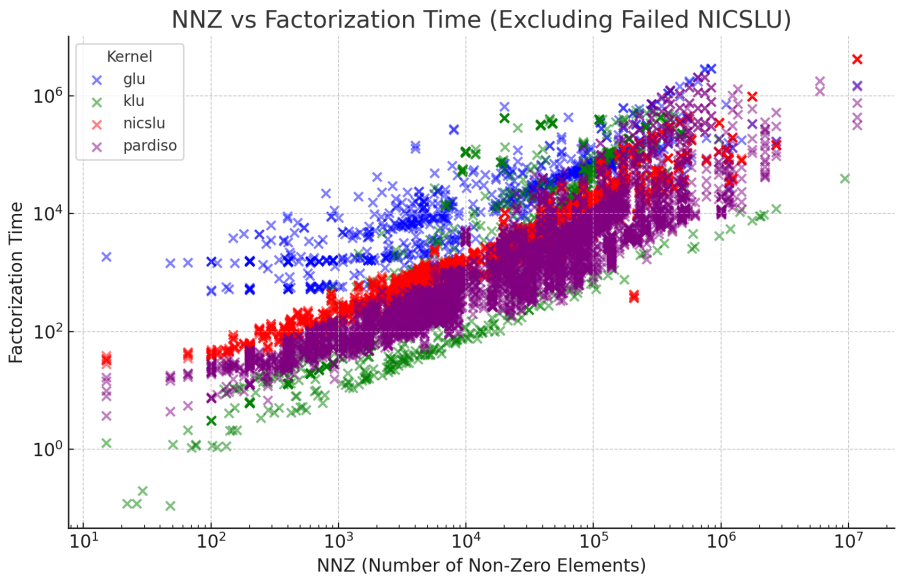


Figure 6.3: Scatter Plot of All Solver's NNZ over Factorization Time in Milliseconds

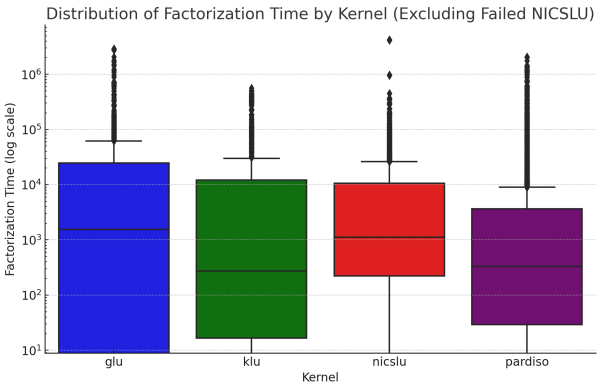


Figure 6.4: Distribution of Solver's Performance Solving Matrices over Factorization Time in Milliseconds

6.2.4 Machine Learning Algorithms Selected

The machine learning-based framework for solver selection employs four classification algorithms: Random Forest, K-Nearest Neighbors (KNN), Bayesian Network, and Support Vector Machine (SVM). These algorithms were selected for their complementary strengths in handling diverse data characteristics and their applicability to sparse matrix classification.

Random Forest Classifier [46] is an ensemble learning method that constructs multiple decision trees like figure 6.5 during training and outputs the mode of their predictions. Its robustness against overfitting and ability to handle high-dimensional datasets make it a strong candidate for this application. The algorithm effectively captures non-linear relationships and feature interactions [47], which are critical for identifying complex patterns in sparse matrices.

K-Nearest Neighbors (KNN) [48] is a non-parametric algorithm that classifies data points based on the majority label of their nearest neighbors. Its simplicity and intuitive approach to decision-making make it a baseline model for comparison.

Bayesian Networks [49] are probabilistic graphical models that represent the conditional dependencies among variables. They are particularly useful for datasets with inherent uncertainty and sparse features.

Support Vector Machine (SVM) [50] is a supervised learning algorithm that finds the optimal hyperplane to separate data points into distinct classes. Its strength lies in handling linearly separable data and adapting to non-linear problems through kernel

functions.

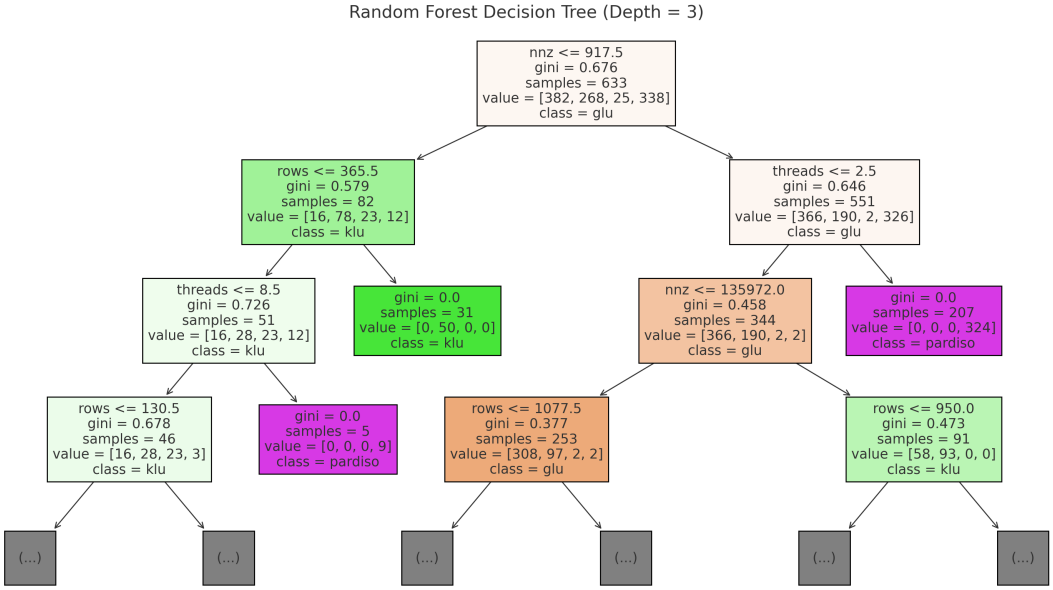


Figure 6.5: Part of the Random Forest Decision Tree Model

6.2.5 Model Performance Analysis

For the machine learning models performing the classification tasks, their performance metrics includes: Accuracy, Precision, Recall, F1-Score, and Average Runtime Error. As the models are trained on training dataset, these metrics will be evaluated on validation dataset to avoid overfitting and showing model's real performance on un-seen data.

Accuracy, defined as the proportion of true results (both true positives and true negatives) among the total number of cases examined, is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives [51], [52].

Precision, the proportion of true positive results in the total predicted positives, is given by:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Recall (Sensitivity), the proportion of true positive results in the total actual positives, is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

The **F1-Score**, which provides a balance between precision and recall, is the harmonic mean of the two and is computed as:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The **Average Runtime Error (ARE)** metric was used to evaluate the runtime performance of the selected solver kernel compared to the best kernel. ARE is defined as:

$$\text{ARE} = \frac{1}{d} \sum_{i=1}^{d} \frac{\text{SM}_{\text{runtime}}(i) - \text{BM}_{\text{runtime}}(i)}{\text{BM}_{\text{runtime}}(i)},$$

where $\text{SM}_{\text{runtime}}(i)$ is the runtime of the solver selected by the model, $\text{BM}_{\text{runtime}}(i)$ is the runtime of the best solver, and d is the number of observations in the dataset.

During the evaluation, it was observed that some predictions misaligned with the available solvers for certain matrices, leading to invalid or infinite runtime values in the Average Runtime Error (ARE) calculations. Misalignment occurred when the model predicted a solver not tested for a particular matrix. To address this, the ARE calculation was adjusted to exclude invalid predictions and ensure accurate reflection of runtime deviations.

6.3 Model Prediction Results

The ARE calculation and additional refinements yielded significant improvements in model evaluation. Random Forest and KNN models achieved the highest accuracy, both at 98.7%, with low ARE values of 24.735 ms, indicating their ability to reliably select near-optimal solvers. Bayesian Network demonstrated moderate performance, with an accuracy of 81.8% and an ARE of 29.840 ms, reflecting challenges in handling class im-

balances. The SVM model performed the least effectively, with an accuracy of 50.6% and an ARE of 48.202 ms, highlighting difficulties in generalizing to the dataset.

These results are summarized in Table 6.2. The results demonstrate that while most models except for SVM can achieve high classification accuracy, even small misclassification rates can result in significant runtime penalties for certain matrices, particularly those with extreme sparsity or large sizes. And among these four models, Random Forest and KNN model has shown the best performance

Table 6.2: Performance Metrics of All Models with ARE

Model	Accuracy	Precision	Recall	F1-score	ARE (ms)
Random Forest	**98.7%**	**99.2%**	**96.4%**	**97.8%**	**24.735**
KNN	94.5%	98.5%	94.2%	96.3%	25.891
Bayesian Network	81.8%	40.9%	50.0%	45.0%	29.840
SVM	50.6%	63.5%	69.8%	49.6%	48.202

The results illustrate the robustness of Random Forest and KNN in predicting solver performance, with minimal runtime penalties. Addressing prediction misalignment and refining the ARE metric were crucial to achieving reliable evaluations. The insights into feature importance and model performance indicate potential areas for further optimization.

Index	Matrix Name	KLU	NICSLU	PARDISO	GLU	True Optimal	Predicted Optimal
0	random_circuit_30_1	140.23	441.83	139.22	1523.80	pardiso	klu
1	matrix_181_seed45_iter4	293800.0	13.33	5940.90	Failed	nicslu	nicslu
2	matrix_86_seed42_iter1	58.81	246.67	165.62	1077.98	klu	klu
3	bcsstk27	9532.08	1928.00	653.80	12811.23	pardiso	pardiso

Table 6.3: Comparison of Factorization Times (Unit: ms) and Kernel Predictions

The example table 6.3 highlights the performance of different solvers on specific matrices, showcasing variability in runtimes and the accuracy of model predictions. For random_circuit_30_1, PARDISO was the true optimal solver, but KLU was predicted as optimal, reflecting a small runtime penalty due to the misclassification. For matrix_181_seed45_iter4, NICSLU emerged as both the true and predicted optimal solver, demonstrating accurate alignment between model predictions and actual performance. However, for the more challenging matrix bcsstk27, PARDISO was correctly identified

as the true optimal solver due to its ability to handle dense and ill-conditioned matrices effectively. These results emphasize the importance of accurate predictions, as even minor mis-classifications can result in significant computational inefficiencies for certain matrices.

In conclusion, this methodology establishes a strong foundation for adaptive solver selection, demonstrating the practical utility of machine learning in circuit simulation workflows. Future research could explore reducing runtime variability by incorporating additional matrix features such as bandwidth and diagonal dominance. Expanding the solver library and utilizing advanced feature engineering methods, such as auto-encoders, could further improve predictions. Additionally, active learning techniques and online adaptation of models would enable dynamic updates to accommodate evolving circuit designs.

Chapter 7

Conclusion

The experiments conducted in this study validate the efficacy of using machine learning-based approaches for solver selection in sparse matrix computation. The results demonstrate that the proposed framework effectively classifies matrices and identifies the optimal solver with a high degree of accuracy, significantly enhancing computational efficiency in circuit simulations.

Among the models tested, Random Forest and K-Nearest Neighbors (KNN) emerged as the most robust classifiers, achieving classification accuracies of 98.7% and 94.5%, respectively. These models also exhibited low Average Runtime Error (ARE) values, with Random Forest achieving an ARE of 24.735 ms and KNN an ARE of 25.891 ms, indicating their capability to reliably predict solvers with minimal runtime penalties. In contrast, the Bayesian Network and Support Vector Machine (SVM) models demonstrated moderate to poor performance, highlighting their limitations in handling imbalanced datasets and complex feature interactions.

The analysis of solver-specific performance further emphasized the variability in runtime efficiency across different solvers. For instance, PARDISO consistently outperformed other solvers on dense and ill-conditioned matrices, whereas NICSLU was more effective for smaller, less sparse matrices. However, NICSLU's inherent limitations in handling large-scale matrices were evident, as seen in its frequent failures during factorization. These insights underline the importance of accurate solver prediction, as even minor misclassifications, such as predicting KLU instead of PARDISO for certain matrices, can lead to significant computational inefficiencies.

Additionally, the visual analysis of solver performance trends, illustrated through scatter and box plots, provided valuable insights into the relationships between matrix characteristics, such as NNZ (number of non-zero elements), and solver runtimes. These observations highlight the potential for further refinement of the feature set to improve classification performance.

Chapter 8

Plan of Future Research

Future research can build on these results by addressing several key areas:

- Enhanced Feature Engineering: Incorporating additional matrix features, such as dynamic bandwidth and diagonal dominance, may improve model accuracy and runtime predictions.
- Solver Library Expansion: Including a broader range of solvers, particularly domain-specific options, would enable more granular optimization for diverse circuit matrices.
- Advanced Machine Learning Techniques: Utilizing deep learning methods, such as graph neural networks (GNNs), could capture intricate relationships within matrix structures, further improving classification performance.

Future advancements in this domain can drive further progress, contributing to more sustainable and scalable computational practices, establish a strong foundation for integrating machine learning into circuit simulation workflows.

Acknowledgements

I would like to express my sincere gratitude to all those who have supported me. First and foremost, I am grateful to my family and friends for their unwavering support, and understanding. And a special thanks to Dr. Danial, my supervisor, whose guidance and expertise have been invaluable. His insightful feedback and continuous support have greatly contributed to the completion of this work. To everyone who has helped me in any way, thank you, your support has always been necessary for the successful completion of this work.

References

[1] Davis, T. A. and Palamadai Natarajan, E., "Algorithm 907: KLU, a direct sparse solver for circuit simulation problems," *ACM Trans. Math. Softw.*, vol. 37, no. 3, Sep. 2010, ISSN: 0098-3500. DOI: 10.1145/1824801.1824814. [Online]. Available: https://doi.org/10.1145/1824801.1824814.

[2] Davis, T. A., "Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 2, 2004.

[3] Schenk, O. and Gartner, K., "Solving sparse linear systems with pardiso," *Computational Science and Engineering*, vol. 5, no. 3, 2004.

[4] Chen, X., Wang, Y., and Yang, H., "NICSLU: An adaptive sparse matrix solver for parallel circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, 2015.

[5] Jin, Z., Li, W., Bai, Y., Wang, T., Lu, Y., and Liu, W.: "Machine learning and gpu accelerated sparse linear solvers for transistor-level circuit simulation: A perspective survey (invited paper)," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 96–101. DOI: 10.1109/ASP-DAC58780.2024.10473846.

[6] Chen, X., Wang, Y., and Yang, H.: 'Parallel Sparse Direct Solver for Integrated Circuit Simulation' (Engineering, Engineering (R0)), 1st ed. (Springer Cham, 2017), pp. IX, 129, 8 b/w illustrations, 43 illustrations in colour, ISBN: 978-3-319-53429-9. DOI: 10.1007/978-3-319-53429-9.

[7] Cadence PCB Solutions. 'IC Design and Manufacturing Process'. Accessed: 2024-07-16. (2023), https://resources.pcb.cadence.com/blog/2023-ic-design-and-manufacturing-process.

[8] Rojec, Ž., Olenšek, J., and Fajfar, I., "Analog circuit topology representation for automated synthesis and optimization," *Informacije MIDEM*, vol. 48, pp. 29–40, Jan. 2018.

[9] Paul, C.: 'Fundamentals of Electric Circuit Analysis', 1st. (Wiley, 2001).

[10] Ho, C., Ruehli, A., and Brennan, P., "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504–509, 1975.

[11] Davis, T. A.: 'Direct Methods for Sparse Linear Systems', 1st. (Society for Industrial and Applied Mathematics, 2006).

[12] Liu, J. W., "The role of elimination trees in sparse factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 1, pp. 134–172, 1990. DOI: 10.1137/0611010. eprint: https://doi.org/10.1137/0611010. [Online]. Available: https://doi.org/10.1137/0611010.

[13] Verley, J. C., Keiter, E. R., and Thornquist, H. K.: "Xyce: Open source simulation for large-scale circuits," in *Proceedings of Workshop on Open-Source EDA Technology*, (ACM, 2018), p. 3. DOI: 10.1145/nnnnnnn.nnnnnnn. [Online]. Available: https://doi.org/10.1145/nnnnnnn.nnnnnnn.

[14] Peng, S. and Tan, S. X.-D., "GLU3.0: Fast GPU-based parallel sparse lu factorization for circuit simulation," *IEEE Design & Test*, vol. 37, no. 3, pp. 78–90, 2020. DOI: 10.1109/MDAT.2020.2974910.

[15] Schenk, O., Gärtner, K., Fichtner, W., and Stricker, A., "Pardiso: A high-performance serial and parallel sparse linear solver in semiconductor device simulation," *Future Generation Computer Systems*, vol. 18, no. 1, pp. 69–78, 2001, I. High Performance Numerical Methods and Applications. II. Performance Data Mining: Automated Diagnosis, Adaption, and Optimization, ISSN: 0167-739X. DOI: https://doi.org/10.1016/S0167-739X(00)00076-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X00000765.

[16] Schenk, O. and Gärtner, K.: "Pardiso," in *Encyclopedia of Parallel Computing*, Padua, D., Ed. (Springer US, 2011), pp. 1458–1464, ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_90. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_90.

[17] Razik, L., Schumacher, L., Monti, A., Guironnet, A., and Bureau, G.: "A comparative analysis of LU decomposition methods for power system simulations," pp. 1–6. DOI: 10.1109/PTC.2019.8810616.

[18] Zhou, Y., Ren, H., Zhang, Y., Keller, B., Khailany, B., and Zhang, Z.: "Primal: Power inference using machine learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6.

[19] Wasynczuk, O. and Sudhoff, S., "Automated state model generation algorithm for power circuits and systems," *IEEE Transactions on Power Systems*, vol. 11, no. 4, pp. 1951–1956, 1996. DOI: 10.1109/59.544669.

[20] Lin, P. and Chua, L., "Topological generation and analysis of voltage multiplier circuits," *IEEE Transactions on Circuits and Systems*, vol. 24, no. 10, pp. 517–530, 1977. DOI: 10.1109/TCS.1977.1084273.

[21] Chen, L., Chen, Y., Chu, Z., *et al.*: *The dawn of ai-native eda: Opportunities and challenges of large circuit models*, 2024. arXiv: 2403.07257 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2403.07257.

[22] Settaluri, K., Haj-Ali, A., Huang, Q., Hakhamaneshi, K., and Nikolic, B.: "Autockt: Deep reinforcement learning of analog circuit designs," in *2020 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 490–495. DOI: 10.23919/DATE48585.2020.9116200.

[23] Li, X., Li, X., Chen, L., Zhang, X., Yuan, M., and Wang, J.: *Circuit transformer: End-to-end circuit design by predicting the next gate*, 2024. arXiv: 2403.13838 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.13838.

[24] Bhowmick, S., Eijkhout, V., Freund, Y., *et al.*: "Application of machine learning in selecting sparse linear solvers." [Online]. Available: https://api.semanticscholar.org/CorpusID:17403761.

[25] Sood, K.: "Automated selection of numerical solvers." [Online]. Available: https://www.cs.uoregon.edu/Reports/DRP-201510-Sood.pdf.

[26] Dufrechou, E., Ezzatti, P., Freire, M., and Quintana-Ortí, E. S., "Machine learning for optimal selection of sparse triangular system solvers on gpus," *Journal of Parallel and Distributed Computing*, vol. 158, pp. 47–55, 2021, ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2021.07.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731521001593.

[27] Sun, H.-B., Jing, Y.-F., and Xu, X.-W., "A new matrix feature selection strategy in machine learning models for certain krylov solver prediction," *Journal of Classification*, 2024, Accepted 19 June 2024, Published 06 July 2024. DOI: 10.1007/s00357-024-09484-0. [Online]. Available: https://doi.org/10.1007/s00357-024-09484-0.

[28] Funk, Y., Götz, M., and Anzt, H.: "Prediction of optimal solvers for sparse linear systems using deep learning," in *Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing (PP)*, pp. 14–24. DOI: 10.1137/1.9781611977141.2. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611977141.2. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611977141.2.

[29] Ren, H. and Hu, J., Eds.: 'Machine Learning Applications in Electronic Design Automation', 1st ed. (Springer Cham, 2023), pp. XII, 583, ISBN: 978-3-031-13074-8. DOI: 10.1007/978-3-031-13074-8. [Online]. Available: https://doi.org/10.1007/978-3-031-13074-8.

[30] 'EDA tools market size & share analysis - growth trends & forecasts (2025 - 2030)', 2025, Accessed via Mordor Intelligence website. https://www.mordorintelligence.com/industry-reports/electronic-design-automation-eda-tools-market.

[31] Khera, V.: *How will EDA benefit from the AI revolution?* Accessed 26 October 2023, 2023. [Online]. Available: https://cadence.com/content/dam/blog/articles/2023/10/how-will-eda-benefit-from-the-ai-revolution.html.

[32] Cherney, M. A. and Singh, J., "Synopsys expects upbeat Q4 on firm demand for chip design software," *Reuters*, Aug. 2024, Accessed 21 August 2024. [Online]. Available: https://www.reuters.com/technology/synopsys-expects-upbeat-q4-on-firm-demand-for-chip-design-software-2024-08-21/.

[33] Rai, R. and Sahu, C. K., "Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus," *IEEE Access*, vol. 8, pp. 71 050–71 073, 2020. DOI: 10.1109/ACCESS.2020.2987324.

[34] Davis, T. A. and Natarajan, E. P.: "Sparse matrix methods for circuit simulation problems," in *Scientific Computing in Electrical Engineering SCEE 2010*, Michielsen, B. and Poirier, J.-R., Eds. (Springer Berlin Heidelberg, 2012), pp. 3–14, ISBN: 978-3-642-22453-9. DOI: 10.1007/978-3-642-22453-9_1. [Online]. Available: https://doi.org/10.1007/978-3-642-22453-9_1.

[35] Ho, J., Jain, A., and Abbeel, P.: *Denoising diffusion probabilistic models*, 2020. arXiv: 2006.11239 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2006.11239.

[36] Yang, L., Zhang, Z., Song, Y., *et al.*: *Diffusion models: A comprehensive survey of methods and applications*, 2024. arXiv: 2209.00796 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2209.00796.

[37] Zhou, J., Cui, G., Hu, S., *et al.*: *Graph neural networks: A review of methods and applications*, 2021. arXiv: 1812.08434 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1812.08434.

[38] Si, C., Huang, Z., Jiang, Y., and Liu, Z.: *Freeu: Free lunch in diffusion U-Net*, 2023. arXiv: 2309.11497 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2309.11497.

[39] Ronneberger, O., Fischer, P., and Brox, T.: *U-Net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[40] Davis, T. A. and Hu, Y., "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, 1:1–1:25, Nov. 2011, Now known as the SuiteSparse Matrix Collection. DOI: 10.1145/2049662.2049663. [Online]. Available: https://doi.org/10.1145/2049662.2049663.

[41] Kolodziej, S. P., Aznaveh, M., Bullock, M., *et al.*, "The suitesparse matrix collection website interface," *Journal of Open Source Software*, vol. 4, no. 35, pp. 1244–1248, Mar. 2019. DOI: 10.21105/joss.01244. [Online]. Available: https://doi.org/10.21105/joss.01244.

[42] Gould, N. I. M. and Scott, J. A., "A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations," *ACM Transactions on Mathematical Software*, vol. 30, no. 3, pp. 300–325, 2004. DOI: 10.1145/1024074.1024075. [Online]. Available: https://doi.org/10.1145/1024074.1024075.

[43] Yeom, J.-S., Thiagarajan, J. J., Bhatele, A., Bronevetsky, G., and Kolev, T.: "Data-driven performance modeling of linear solvers for sparse matrices," in *2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pp. 32–42. DOI: 10.1109/PMBS.2016.009.

[44] Chen, Q., Ye, Y., Li, M., Yan, H., and Shi, L., "Optimized matrix ordering of sparse linear solver using a few-shot model for circuit simulation," *Integration*, vol. 93, p. 102 062, 2023, ISSN: 0167-9260. DOI: 10.1016/j.vlsi.2023.102062. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926023001049.

[45] Shao, M., Li, P., *et al.*, "A novel high performance sparse linear solver for circuit simulation on GPU platform," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, 2014.

[46] Breiman, L., "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[47] Liaw, A. and Wiener, M., "Classification and regression by randomforest," *Forest*, vol. 23, Nov. 2001.

[48] Cover, T. and Hart, P., "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. DOI: 10.1109/TIT.1967.1053964.

[49] Fix, E. and Hodges, J. L., "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989, ISSN: 03067734, 17515823. [Online]. Available: http://www.jstor.org/stable/1403797, (visited on Jan. 15, 2025).

[50] Cortes, C. and Vapnik, V., "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995, ISSN: 1573-0565. DOI: 10.1007/BF00994018. [Online]. Available: https://doi.org/10.1007/BF00994018.

[51] Powers, D. M., "Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, 2011.

[52] Han, J., Kamber, M., and Pei, J.: 'Data Mining: Concepts and Techniques'. (Elsevier, 2011).