CS-433 Machine Learning - Project 2 Report: TITRE

Yasmine Tligui, Rim El Qabli, Lily Gilibert
Departement of Computer Science, EPFL Lausanne

***Abstract*—Field Programmable Gate Arrays (FPGAs) are reconfigurable hardware circuits designed to perform complex computations on signals in applications such as medical imaging, real-time control systems, and industrial automation. To efficiently route signals while avoiding congestion, algorithms like PathFinder determine optimal paths through iterative processes. However, these methods are computationally expensive and slow to converge.**

In this work, we propose a Machine Learning (ML) approach to predict what the n-th iteration of PathFinder would have produced for an FPGA, allowing us to avoid running the costly PathFinder algorithm for every new FPGA. By running a few initial iterations of the PathFinder algorithm and identifying main features that capture both static architectural constraints and evolving congestion trends, this method aims to reduce runtime while maintaining high-quality routing solutions.

I. Introduction

Field Programmable Gate Arrays (FPGAs) are reconfigurable hardware circuits that provide unparalleled flexibility for performing diverse computational tasks. Unlike fixed-function hardware, FPGAs can be programmed and reprogrammed to execute specific algorithms by dynamically routing signals between different sources and destinations depending on the desired functionality. This adaptability allows FPGAs to efficiently handle tasks ranging from signal processing to controlling complex real-time systems [1].

This reconfigurability comes with challenges, particularly in solving the routing problem, where signal paths must be determined between logic block pins within a limited and fixed network of resources. Routing involves determining how signals navigate an FPGA while avoiding congestion, minimizing delays, and adhering to timing constraints. Algorithms like PathFinder address this by iteratively assigning paths, detecting conflicts, and re-routing signals until a congestion-free solution is found. The iterative nature of these algorithms, while effective, can become computationally expensive. For large and complex FPGA designs, traditional methods which rely heavily on heuristic-based approaches, like PathFinder, may require hundreds of iterations to resolve all conflicts, making it a bottleneck in the design process. This limitation often results in suboptimal solutions or necessitates extensive re-routing to resolve conflicts, increasing the runtime further [2].

Recent advancements in machine learning have introduced new possibilities for improving FPGA routing efficiency. ML models have demonstrated the ability to predict congestion costs and guide signal paths more effectively than heuristic-based approaches [3]. However, most existing ML methods rely primarily on static features derived from the FPGA architecture and initial placement of logic blocks. This limits their ability to adapt dynamically to the congestion data generated during the routing process, reducing their potential impact.

Moreover, exploring the integration of dynamic congestion data into ML predictions remains a relatively uncharted direction in FPGA routing research. Within our limited timeframe, we have taken a broad exploratory approach, examining a variety of features derived from previous iterations' costs, and implementing a handful of preliminary ML models in different configurations. Rather than focusing solely on polished applications, we emphasize the research aspect of this work, considering it as a stepping stone toward future refinement. Our aim is to illuminate how these features and design choices may guide more specialized strategies for modeling FPGA routing complexity.

II. FPGA structure

FPGAs are ordered as grids of repeating tiles, as illustrated in Figure 1. Each tile contains several components that collectively enable the FPGA's flexibility and computational power. The image below provides a visual representation of a typical FPGA tile-based architecture, structured by the following key elements :

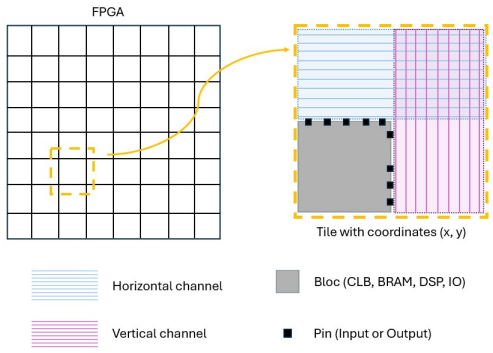


Figure 1: FPGA Tile-Based Architecture.

- **Horizontal and Vertical Channels:** Wires that allow signals to travel across the FPGA in horizontal and vertical directions.
- **Logic Blocks (CLBs, BRAMs, DSPs, IOs):** The computational units of the FPGA.
- **Input and Output Pins:** They connect the internal logic blocks to the routing signals.

Each tile in the FPGA grid serves one of three roles for signal routing: it can act as a source where a signal originates, a destination where the signal terminates, or a pass-through conduit for signals allowing them to travel across the grid. Signals that require computation or data manipulation interact with logic blocks through input/output pins, which connect the routing channels to the internal components of the blocks. The channels include fixed-length wires of 1, 2, 4, or 12 units (L1, L2, L4 and L12), designed to handle both local and long-distance connections. Shorter wires ensure efficient routing for nearby tiles, while longer wires reduce delays for signals traveling across larger sections of the FPGA. This modular architecture, while efficient, introduces challenges such as congestion and resource contention, which the ML framework seeks to address through predictive modeling.

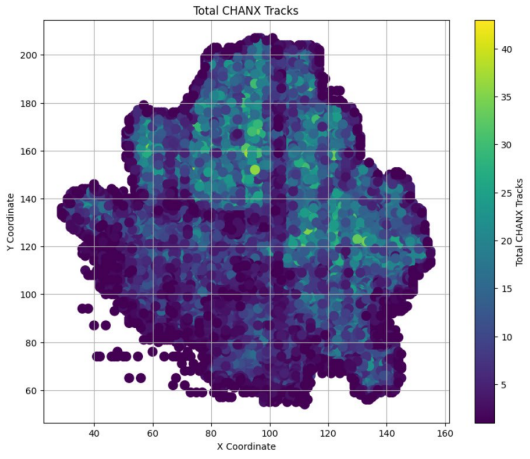


Figure 2: Channel usage for horizontal channels

Figure 2 illustrates the channel usage for the first iteration of the initial benchmark. The color bar on the right represents the number of tracks (or wires) used in each horizontal channel, with brighter colors indicating higher usage. As expected, the edges of the FPGA exhibit minimal or no usage, which aligns with the behavior of routing

algorithms prioritizing the shortest paths. This pattern confirms the routing algorithm's efficiency in optimizing track utilization while avoiding unnecessary traversal of unused peripheral regions.

III. Our contributions

Existing machine learning approaches to FPGA routing predominantly rely on static architectural features [3], derived from the FPGA structure and the initial placement of logic blocks. Although these methods provide useful insights, they fail to capture the dynamic nature of congestion as it evolves during routing, limiting their ability to make accurate predictions.

In this work, we address this limitation by introducing a machine learning hybrid approach that incorporates both static architectural constraints and dynamic congestion patterns and past-node cost predictions extracted from early iterations of the PathFinder algorithm. By combining these complementary sources of information, our method bridges the gap between static models and the evolving demands of FPGA routing, to predict the outcomes of later PathFinder iterations without compromising the quality of routing solutions.

Our contributions can be summarized as follows:

- **Dynamic Feature Integration:** Beyond static architectural data, we leverage dynamic congestion patterns collected during initial PathFinder iterations, providing a temporal perspective on routing trends.
- **Adaptability to FPGA Architectures:** Our approach is flexible and scalable, making it applicable to various FPGA architectures and easily integrable into existing routing workflows.

IV. Methodology

A. Introduction

To predict routing costs on FPGAs using Machine Learning (ML), our approach consists of three main steps: data structuring, feature extraction, and modeling. The initial data includes accumulated costs and paths traversed by signals. From this data, we extract specific features that capture both structural characteristics and congestion dynamics. Finally, the data is partitioned across several scenarios to evaluate the robustness and performance of the models. Let's see this in detail!

B. Data Structuring

The FPGA benchmark files are organized as follows:

- **Accumulated Costs:** Reflect congestion trends observed during the early iterations. Historical cost values are logged for each node during the initial routing iterations performed by the PathFinder algorithm to quantify the congestion level and resource utilization. A node may correspond to an horizontal or vertical channel, a logic block, or an input/output pin.

- **Signal Paths:** Provide detailed insights into the routing trajectories established by the PathFinder algorithm. For each net, the routing data captures the coordinates of the source, the intermediate nodes traversed, the sinks, and any logic blocks used.

This structured data forms the foundation for the ML-based cost prediction framework. The accumulated costs and signal paths represent the point where the PathFinder algorithm completes its initial iterations, allowing the ML models to take over. It ensures that both static and dynamic characteristics of the routing process are effectively captured, fostering an efficient integration of heuristic and ML methodologies in FPGA routing.

C. Feature Extraction

To enable ML-based predictions for FPGA routing, we derive a set of meaningful features that effectively capture the state of the routing process, the utilization of resources across the FPGA, and its physical architecture. The objective is to provide the ML models with a detailed and nuanced view of the routing landscape, incorporating metrics that reflect tile-level activity, routing channel usage, and historical congestion patterns.

By focusing on localized and broader network characteristics, these features allow the ML framework to make informed predictions about routing costs and congestion trends, bridging the gap between heuristic methods and data-driven approaches. The first two features are the statics features used in [3]. Below is a detailed explanation of the extracted features:

- [3]**Input Pin Density:**

$$\frac{\text{Number of signals connected to input pins of the tile}}{\text{Total input pin capacity of the tile}}$$

This feature captures the load on the input pins of each tile, representing its connectivity demand.

- [3]**Output Pin Density:**

$$\frac{\text{Number of signals connected to output pins of the tile}}{\text{Total output pin capacity of the tile}}$$

Similar to input pin density, this metric measures the load on output pins, providing insight into how actively a tile is driving signals to the network.

- **Local Tile Density Usage:**

$$\frac{\text{Number of wires used in the tile}}{\text{Total wire capacity in the tile}}$$

This feature dynamically captures congestion in the immediate vicinity of the tile, evolving with each routing iteration as signals are redefined to avoid resource overload.

- **Global Tile Density Usage:**

By extending the scope of the local tile density function, this feature captures congestion trends on a larger scale (5x5 tile grid), reflecting system-wide utilization patterns and the interplay between neighboring tiles. This broader perspective provides insights into regional congestion hotspots and enables the ML model to account for interactions that may influence routing decisions beyond the immediate vicinity of a tile.

- **Local Channel Bias:**

$$\frac{\text{CHANX used - CHANY used}}{\text{CHANX used + CHANY used}}$$

This feature indicates the directional preference for horizontal (CHANX) or vertical (CHANY) routing channels within a tile. It was chosen as a feature because understanding channel bias helps capture imbalances in resource usage, which can highlight potential congestion points and inform routing decisions to optimize signal distribution across the FPGA.

- **Global Channel Bias:**

Extended to the immediate neighborhoods, this feature identifies directional preferences and congestion trends within a broader 5x5 tile grid. By analyzing channel usage over a larger area, it provides insights into system-wide imbalances in routing resource utilization, capturing the cumulative effect of routing decisions and helping the model predict how global congestion patterns evolve. This perspective is essential for addressing potential bottlenecks that arise from interactions across multiple tiles.

- **Historical costs:**
  To accurately analyze and predict routing costs, we also computed the historical costs of each tile. These costs were specifically calculated for the first three iterations of the routing process, obtained using the Pathfinder algorithm.
  These historical costs indeed provide a temporal view of congestion patterns and help identify problematic tiles during the routing process.

  For this matter, we explored several approaches to calculate the historical costs:

– **Mean cost per tile:** We began by simply averaging the costs of all nodes within each tile. This approach provided a simple representation of overall tile congestion.

Problem with averages: Averaging the costs masked critical high-cost values (e.g., 15, 16) as they were diluted by the majority of low-cost values (e.g., 1, 2) (cf Table I). This imbalance caused the models to overlook essential hotspots contributing to severe congestion. As a result, the features carried limited variation, making it difficult to distinguish between tiles that were truly problematic and those that were not. Such a loss of granularity undermined the primary objective of capturing the dynamic and uneven nature of routing costs.

Table I: Distribution of Historical Cost Values

Cost values	Count
1	13,474,866
2	742,150
3	166,316
4	47,442
5	13,443
6	4,214
7	1,432
8	522
9	189
10	82
11	17
12	7
13	3
14	2
15	2
16	1

– **Weighted average per channel (two costs per tile, one for each CHANX and CHANY):** To address the imbalance in cost values, we introduced weights to amplify the impact of the minority high-cost values while maintaining the average as the central metric.

Challenges with weighted average per channel: Although weighting increased the prominence of minority high-cost values, they remained underrepresented. The weighted averages still smoothed out essential congestion spikes, which prevented the model from fully recognizing and reacting to serious bottlenecks.

– **Weighted average per tile:** To simplify the representation and improve model performance, we calculated a single weighted average cost per tile, aggregating data from both channels and all components. This method provided a more representative and cohesive view of each tile's congestion status, while producing high precision decimal values often clustered around 1.
– **Max cost per channel:** As another approach, we assigned each channel's cost to the maximum wire cost within that channel. By focusing on the maximum cost, we ensured that rare but significant high-cost values were not overlooked. This approach improved the distribution and variability of feature values, but it failed to represent the overall congestion context of the tile adequately.
– **Summed costs per channel:** To address the limitations of prior methods, we computed the sum of all wire costs within each channel. This method effectively emphasized the influence of higher-cost wires while maintaining a broader representation of the channel's overall congestion. This approach provided integer-valued costs over a wider range, making it more suitable for capturing variability and distinguishing between tiles.

Ultimately, we retained two balancing methods as the most effective: the **weighted average per tile** and the **summed costs per channel**. These approaches offered the best trade-off between representativeness and feature variability.

D. Target Extraction

The targets for our predictions are the costs, per tile or per channel depending on which of the two balancing methods we used, this being at **50%**, **80%**, and **100%** completion of the routing process. These milestones were selected to attempt an analysis of how prediction performance evolves as the routing progresses. We also recalculated the historical cost features per tile/per channel to align the features with this goal. By predicting the costs at these completion stages, we validate whether the model's performance decreases due to the diminishing availability of early iteration information.

E. Data Partitioning

To evaluate model performance, the data is partitioned systematically across benchmarks and iterations:

– **Feature Extraction:** All benchmarks are processed, and the extracted features are combined into a single dataset. This ensures that the models are exposed to a comprehensive and diverse dataset representing a variety of routing scenarios.
– **Shuffling and Splitting:** The combined data is shuffled and split into **70%-30%** for training and testing. This systematic partitioning ensures fair evaluation and prevents overfitting to specific benchmarks or iterations.

F. Machine Learning Methods

We evaluated the following ML methods to capture both regression and classification aspects of routing cost prediction:

– **Linear Regression:** A simple baseline to capture linear relationships.
– **Ridge Regression:** An enhanced version of Linear Regression with a regularization parameter to manage multicollinearity and reduce overfitting.
– **Random Forest Regressor:** A non-linear model capable of capturing complex dependencies between features and costs.
– **Logistic Regression:** Considered for classification tasks but abandoned due to poor convergence.
– **Neural Networks:** Initially explored for their capacity to handle complex classification tasks but similarly excluded due to convergence issues, even with extended training iterations.

For each combination of preprocessing method (weighted average or summed costs) and model (Linear Regression, Ridge Regression, and Random Forest), we applied an additional layer of balancing via sampling weights. Sampling weights aimed to address residual imbalances in the data, ensuring fair representation of underrepresented classes. However, sampling weights were not supported by Random Forest, reducing the total number of tested combinations to **10**: 2 preprocessing methods $\times$ 3 models $\times$ 2 sample weight settings (applied or not) -2 due to Random Forest limitations.

V. Results and Discussion

In this section, we discuss the results obtained from the different modeling approaches and preprocessing methods described previously. We focus on analyzing the Mean Squared Error (MSE) and the coefficient of determination (R^2) for each combination of models, targets, and feature extraction strategies.

Table II: Results for the averaged cost method

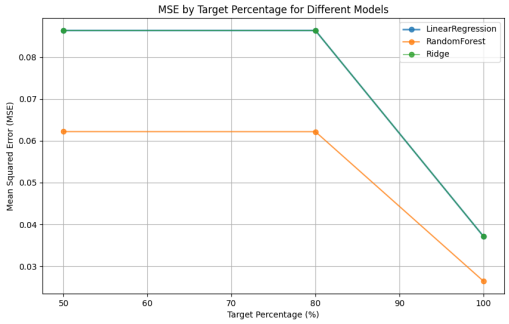
Weight. Avrg. Cost Per Tile	Target	Linear Reg		Ridge Reg		Random F	
		MSE	R^2	MSE	R^2	MSE	R^2
Non-weighted	50%	0.086	0.14	0.086	0.14	0.062	0.38
	80%	0.086	0.14	0.086	0.14	0.062	0.38
	100%	0.037	0.083	0.037	0.083	0.026	0.34
Weighted	50%	0.83	-7.31	0.83	-7.31	0.062	0.38
	80%	0.83	-7.30	0.83	-7.30	0.062	0.38
	100%	0.72	-16.9	0.72	-16.9	0.026	0.34

A. Overall Trends

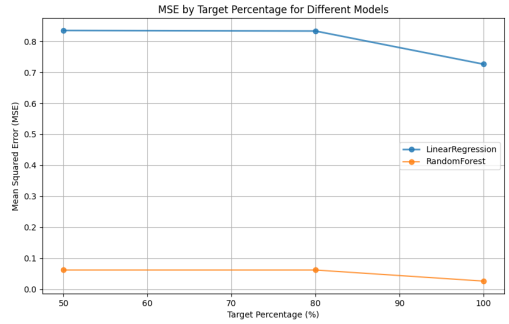
When comparing the models (Linear Regression, Ridge Regression, and Random Forest) across different target completion percentages (50%, 80%, and 100%), two primary observations emerge:

Table III: Results for the summed cost method

Sum. Cost per Channel	Target	Linear Regression		Ridge Regression		Random Forest	
		MSE	R^2	MSE	R^2	MSE	R^2
Non-weighted CHANX CHANY	50%	1.50e-27	1	7.48e-09	0.99	0.0091	0.99
		4.86e-27	1	7.94e-09	0.99	0.00074	0.99
	80%	1.50e-27	1	7.48e-09	0.99	0.0091	0.99
		4.86e-27	1	7.94e-09	0.99	0.00074	0.99
	100%	1.50e-27	1	7.48e-09	0.99	0.0091	0.99
		4.86e-27	1	7.94e-09	0.99	0.00074	0.99
Weighted CHANX CHANY	50%	3.00e-25	1	7.48e-09	0.99	0.0091	0.99
		1.41e-24	1	7.94e-09	0.99	0.00074	0.99
	80%	3.00e-25	1	7.48e-09	0.99	0.0091	0.99
		1.41e-24	1	7.94e-09	0.99	0.00074	0.99
	100%	3.00e-25	1	7.48e-09	0.99	0.0091	0.99
		1.41e-24	1	7.94e-09	0.99	0.00074	0.99



(a) MSE by target for averaged costs



(b) MSE by target for sample weighted averaged costs

Figure 3: Comparison of MSE by target for averaged costs and sampled weighted averaged costs.

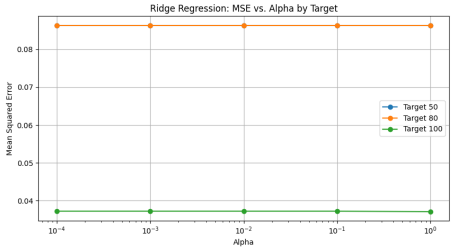


Figure 4: Tuning of alpha for Ridge Regression for averaged costs

- **Model robustness:** Random Forest consistently outperforms Linear Regression and Ridge Regression, particularly in scenarios involving non-trivial feature transformations. This is apparent in both non-weighted and weighted settings, where Random Forest maintains relatively low MSE and higher R^2 values.
- **Influence of preprocessing:** The choice of preprocessing method (averaged costs per tile or summed costs per channel) heavily influences the difficulty and richness of the learning task. Averaged cost features offer a more nuanced challenge, allowing differences between models to surface. In contrast, the summed cost method often yields results that are suspiciously close to perfect predictions, suggesting a more direct relationship between features and targets.

B. Averaged vs. Summed Costs

Averaged Costs per Tile: When using averaged cost features, we see that Linear and Ridge Regression models exhibit moderate MSE and limited improvement when moving from 50% to 80%. At 100%, both models slightly improve, but their R^2 values remain modest. The flat response of Ridge Regression across different regularization strengths (alpha values) indicates limited sensitivity to regularization, suggesting that the extracted features do not interact meaningfully with this parameter.
In these averaged-cost scenarios, Random Forest provides a clear advantage, achieving lower MSE and higher R^2. Its ability to capture non-linearities and interactions appears beneficial for these more complex, less directly correlated features.
Summed Costs per Channel: With the summed cost features, the predictions for all models (Linear Regression, Ridge Regression, and Random Forest) yield extremely high R^2 values, often close to 1, and very low MSE. These near-perfect results occur consistently across all target percentages and remain stable even when weights are introduced. This suggests that the summed cost approach may be oversimplifying the problem, effectively handing the models a set of features that already approximate the target extremely well. Such a scenario could arise from data leakage or from the target being a near-linear transform of the features. As a consequence, the models appear to excel trivially, offering limited insights into their actual predictive capabilities.

C. Impact of Weighting and Regularization

Weighting: Introducing sample weights substantially affects the models relying on averaged-cost features. In particular, Linear and Ridge Regression deteriorate to the point of negative R^2 values, indicating worse-than-baseline predictions. This suggests that the weighting strategy may have amplified data imbalance or outliers in a way that these linear models could not accommodate. Conversely, Random Forest remains relatively stable, reinforcing its robustness and flexibility.
Regularization (Ridge Regression): Adjusting the alpha parameter in Ridge Regression does not significantly alter the performance. The absence of noticeable changes across a wide range of alpha values implies that the chosen features do not benefit from the form of shrinkage offered by Ridge Regression. Either the linear relationship is too simplistic or the model's complexity is not the limiting factor.

D. Interpretation and Limitations

These results highlight that the predictive task's complexity is highly dependent on the selected feature engineering approach. The averaged-cost features yield more realistic scenarios for model comparison, exposing differences in model adaptability and robustness. On the other hand, the summed-cost approach leads to overly simplistic outcomes where even linear models perform near-perfect predictions, suggesting that the target information is already embedded, almost directly, in the features. Additionally, the failure of certain weighting strategies and the minimal impact of regularization point toward the need for more refined feature extraction. Future work may explore alternative approaches that strike a balance between revealing meaningful congestion patterns and avoiding trivial solutions.

E. Interpretation at 50%, 80%, and 100% Completion

At 50% completion of the routing iterations, the FPGA's congestion profile typically begins to stabilize, allowing the model to form initial predictions on routing patterns. By 80%, most channels have established a relatively consistent flow of signals, indicating that the model's forecasts are aligning closely with the actual resource utilization. At the final stage, 100% completion, the predictions should ideally converge with the PathFinder results, offering near-identical routing patterns. This progression highlights how initial partial insights can guide subsequent

routing decisions and ultimately reach a high-fidelity approximation of the final configuration. In practice, these intermediate interpretations enable adaptive strategies that reduce the need for full iterative exploration.

VI. Conclusion

In this work, we explored machine learning models for predicting FPGA routing costs based on features derived from early PathFinder iterations. Our findings underscore the importance of careful feature engineering. When employing averaged cost features, Random Forest emerges as a more resilient and capable predictor, clearly outperforming linear methods. In contrast, the summed cost approach results in near-perfect predictions for all models, suggesting that essential target information is overly embedded in these features, limiting the models' capacity to demonstrate meaningful learning or differentiation.

Regularization with Ridge Regression fails to provide notable improvements, indicating that the extracted features and the formulated prediction task do not align well with linear shrinkage strategies. Moreover, certain weighting approaches exacerbate performance issues for linear models, highlighting the complexity of dealing with imbalances or outliers in the data.

In summary, while our preliminary results suggest that integrating dynamic congestion data can enhance predictive accuracy and potentially streamline FPGA routing, further refinement is needed. Future directions include exploring more granular feature extraction methods, refining weighting strategies, and investigating alternative modeling techniques or regularization schemes. Such efforts could yield a more balanced approach, offering both robust predictions and non-trivial insights into the routing process.

VII. Ethics

We did not identified any ethical risks. Here is our process below.

1) Stakeholders Considered:

- **Direct Stakeholders:** Engineers and researchers who utilize FPGA routing optimization tools.
- **Indirect Stakeholders:**
  - The environment, due to the energy consumption associated with training and running machine learning models.

2) Ruling Out Ethical Risks:

- **Fairness:** The project leverages publicly available datasets without apparent biases, ensuring fair applicability across technical users.
- **Privacy:** The project does not involve personal or sensitive data. It relies solely on routing benchmarks.
- **Autonomy:** The solution is fully transparent and remains under the control of technical users, with no impact on individual agency.
- **Sustainability:** We did not conduct any research, however, we recognize the carbon footprint associated with training machine learning models. This potential impact is mitigated by the long-term benefits of optimizing FPGA routing efficiency.

3) Conclusion: Although no significant ethical risks were identified, the environmental impact of ML model training warrants further reflection. We recommend that these findings be reviewed by ethics specialists to ensure comprehensiveness and reliability.

VIII. Acknowledgements

We want to thank the PARSA laboratory for the opportunity to work with them on such an interesting research.

A special thank you to Ph.D student Shashwat Shrivastava who guided and helped us throughout this project.

References

[1] M. Iida, "What is an FPGA?" in *Principles and Structures of FPGAs*. Singapore: Springer Singapore, 2018, pp. 23–45.

[2] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for fpgas," in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[3] U. Siddiqi, T. Martin, S. Van Den Eijnden, A. Shamli, G. Grewal, S. Sait, and S. Areibi, "Faster fpga routing by forecasting and pre-loading congestion information," in *2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD)*, 2022, pp. 15–20.