Enabling Risk Management of Machine Learning Predictions for FPGA Routability

Andrew David Gunter, Maya Thomas, Nikhil Pratap Ghanathe, Steven Wilton
The University of British Columbia
Vancouver, British Columbia, Canada
{agunter,mayabj,nikhilghanathe,stevew}@ece.ubc.ca

Abstract

Machine Learning (ML) models sometimes make inaccurate predictions for the routability of field-programmable gate array (FPGA) circuit designs. This risks time wasted attempting to route an unroutable design or the premature termination of a routable design's compilation. While improving model accuracy is beneficial, we explore a complementary approach to mitigate the risk of inaccurate predictions by assessing the confidence of ML models. This approach could allow individuals to customize their own trade-off for the competing risks of wasted time and premature compilation termination. In this paper, we introduce a novel mixture of experts ML system for FPGA routability prediction and further quantify the confidence calibration of this system to determine its suitability as a risk management tool. We evaluate our prediction system for the purpose of enabling user risk management in FPGA routability prediction, comparing against a baseline inspired by prior work. Our evaluation finds our approach to achieve almost 2× the precision in risk trade-off between time wasted on unroutable designs and premature termination of routable designs.

CCS Concepts

• **Hardware** → *Reconfigurable logic and FPGAs*; **Physical design (EDA)**; • **Computing methodologies** → **Machine learning**.

Keywords

ML, confidence, uncertainty, CAD, FPGA, routing, early exit, risk

ACM Reference Format:
Andrew David Gunter, Maya Thomas, Nikhil Pratap Ghanathe, Steven Wilton. 2024. Enabling Risk Management of Machine Learning Predictions for FPGA Routability. In *2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD '24), September 9–11, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3670474.3685969

1 Introduction

Recent years have seen significant work on machine learning (ML) techniques for computer-aided design (CAD) ([1, 4, 7, 8, 10, 15, 19–21, 25, 27, 29, 30] among many others). In many cases, ML models make accurate predictions which improve CAD results but this accuracy is not always guaranteed. When a significant cost is associated with an inaccurate prediction, it becomes pragmatic to take a risk management approach by minimizing costs while still maximizing the benefits of accurate predictions. To this effect, we take interest in understanding how *ML model confidence* can be leveraged in an ML-aided CAD system.

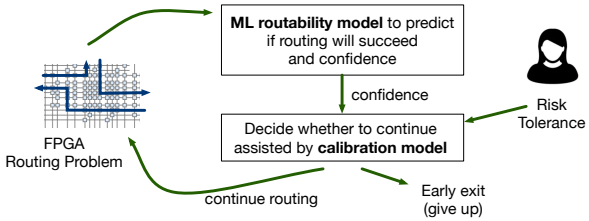

Figure 1: Overall ML-Assisted FPGA Routing Flow

To explore this idea, we focus on the routing task for Field-Programmable Gate Arrays (FPGAs). FPGAs contain fixed logic segments joined by programmable routing switches arranged in a complex topology. This makes the FPGA routing problem difficult and time-consuming; for a large FPGA, the routing task can take days and there is no guarantee of success. Our previous work presented an ML-based approach to predict whether a legal solution will eventually be found [11]. Using such a prediction, users can abort doomed attempts early (i.e. "early exit"), allowing them to modify the design, buy a bigger FPGA, or simply try again with a different random seed.

In this paper, we argue that predicting success/failure is not always enough; it is also important to estimate the *confidence* of the routability prediction. Knowing the confidence of a prediction may allow a user to take their own risk tolerance into consideration as they decide whether to abort a run. A user on a tight timeline may prefer to be biased towards exiting early to avoid wasted runtime, while a user with tight resource constraints may be content spending more time to find an optimized solution on a small FPGA.

In this paper, we consider both the confidence estimation problem and how this confidence information can be used in FPGA routing (see Figure 1). We make the following contributions:

(1) We present *a mixture of experts ML system which predicts design routability and reports its confidence that the prediction is correct* (top box in Figure 1). More precisely, our system outputs the probability that routing will complete within a specified maximum number of iterations.
(2) We present *a method to use confidence information to automatically make early exit decisions* (bottom box in Figure 1).

MLCAD '24, September 9–11, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0699-8/24/09
https://doi.org/10.1145/3670474.3685969

Recognizing that different users have different risk tolerances, we provide a "knob" which allows the user to trade-off runtime for average completion rate. A key challenge is to convert a user's request into a parameter that can be used with confidence information to determine whether to continue or early exit. Our method achieves this.

(3) *We compare the achievable route time / completion rate curve to a baseline* inspired by our previous work [11] and show that our approach can provide a better trade-off between these two quantities.

Overall, we show that our technique for directly estimating confidence information leads to better decisions during the FPGA routing CAD flow. Although our exposition is focused on our specific problem domain, we expect that the use of confidence information has broader applicability across many MLCAD prediction scenarios.

This paper is organized as follows. Background is in Section 2. Our novel confidence estimator is in Section 3 and its accuracy is evaluated in Section 4. Our approach for using confidence to make better decisions is in Section 5. Section 6 concludes the paper. Additional experimental details and artifacts are in the appendix.

2 Background

2.1 Related Work

Machine learning (ML) has been shown to improve solutions for difficult CAD problems through techniques such as predictive modelling, automation, and more recently conversational hardware design [4]. They have been applied to various stages of CAD like congestion estimation [1, 10, 19, 25] during routing, design space exploration [15, 20, 27, 30] for HLS designs, placement [7, 8, 21, 29] and so on. Since ML models in CAD typically perform predictions/suggest actions [26] in an iterative CAD process, an overfit/underfit model will lead to poor quality of results [10]. For example, [21] encounters overfitting in MLP for delay-modelling and mitigates it through ensemble-learning using a random-forest algorithm. Similarly, [10] demonstrates that a random forest of simple regressors are less prone to overfitting, even outperforming a sophisticated MLP-regressor. Therefore, overfit/underfit leads to poor performance, severely restricting usability of the model in CAD. Under this scenario, model reliability can be improved if the model predictions are accompanied with a well-calibrated *confidence* value that aligns with its actual accuracy. Model/confidence calibration is a well studied problem in ML [9, 12–14, 16, 28], which helps align predicted probabilities with actual likelihood of correctness, thereby enhancing the reliability of model predictions. For example, if a well-calibrated ML model confidently predicts maximum routing congestion during routing, it might be used to advise users to refrain from routing further, thereby saving valuable time. In contrast, a low-confidence estimation may encourage the user to proceed with routing. There have been some prior works in CAD that employ bayesian techniques like gaussian process that can naturally quantify uncertainty [3, 18, 32]. However, they suffer from the *curse of dimensionality* and do not scale well, are computationally-inefficient and are sensitive to variations in data and kernel selection.

2.2 Machine Learning Model Calibration

When a ML model's self-reported *confidence* values align with its actual accuracy over many samples, that model is said to be well-*calibrated*. Common metrics to measure the calibration quality are:

Expected calibration error (ECE) [12]: An easily interpreted score which measures the difference between prediction confidence and actual accuracy across different confidence intervals in the range $[0.0, 1.0]$. Confidence values are binned into discrete intervals and confidences are compared to accuracy within each bin:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{N} \cdot |acc_{B_m} - conf_{B_m}| \tag{1}$$

$$acc_{B_m} = \frac{1}{|B_m|} \sum_{\forall i \in B_m} [\, y_i = \hat{y}_i \,] \qquad conf_{B_m} = \frac{1}{|B_m|} \sum_{\forall i \in B_m} p_i$$

where M is the number of intervals, $\frac{1.0}{M}$ is the size (width) of each interval, N is total number of predictions across all intervals, B_m is the m^{th} bin spanning the interval $(\frac{m-1}{M}, \frac{m}{M})$, and $|B_m|$ is the number of samples in B_m. acc_{B_m} and $conf_{B_m}$ are the accuracy and confidence of m^{th} bin. y_i is the true label and $\hat{y}_i$ is the predicted label with associated prediction confidence p_i. A low ECE score is desirable as it indicates less disparity on average between confidence and accuracy across all intervals.

Brier Score (BS) [5]: BS measures the mean-squared difference between predicted probabilities and actual outcomes.

$$BS = \frac{1}{N} \sum_{i=1}^{N} (\pi_i - y_i)^2 \tag{2}$$

where π_i is predicted probability of the positive class and y_i is the actual outcome (e.g., 0/1 in binary classification).

Negative log-likelihood (NLL): NLL measures how close predictions are to the ground truth. A lower NLL indicates that the model assigns high confidence to correct outcomes.

$$NLL = -\frac{1}{N} \sum_{i=1}^{N} (y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)) \tag{3}$$

3 Constructing a confidence estimator

FPGA routing is typically done using a negotiated-congestion algorithm [22, 24]. Each iteration, congested resource costs are increased to try resolve congestion while optimizing for timing. The algorithm stops when either a legal solution has been found (success) or a predetermined maximum number of iterations finish (failure). Routing is time-consuming and state-of-the-art techniques do not guarantee a legal routing solution, even if one exists. This motivates early exit during routing, in which the routing process is terminated when it is believed that a legal routing solution will not be found.

Our previous work employs a mixture of experts (MoE) system composed of regressors to predict, during routing, how many more iterations will be required to achieve a solution [11]. Routing is exited early if the prediction exceeds a user's specified maximum iteration tolerance. In this paper, we instead directly predict the probability (confidence) that routing will complete within the user's tolerance. This section describes our MoE architecture for this task.

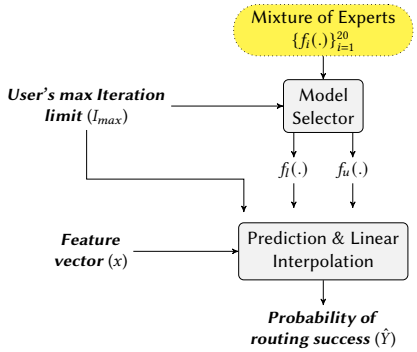

Figure 2: Overview of the proposed approach.

3.1 Mixture of Experts Architecture

We seek to construct a predictor which receives two inputs:

(1) The maximum number of routing iterations allowed.
(2) A feature vector describing the state of routing at the end of a routing iteration. We replicate the features in [11].

This predictor must output the probability that routing will complete within the specified iteration limit. The iteration limit parameter (I_{max}) enables users to define a "worst-case" routing scenario, typically between 50 and 1000; [11] established that relatively few designs are routable beyond this range. Iteration limits lower than 50 may have been worth modelling, but we are focused on challenging routing scenarios which rarely resolve within 50 iterations and thus we would have insufficient data for smaller iteration limits.

We implement a simple mixture of experts (MoE) system composed of 20 binary classification models $\{f_i\}_{i=1}^{20}$. Each model f_i is trained to make predictions for the routing problem at a unique decision threshold $Z_i = i*50, i \in [1..20]$. If the required iteration limit $I_{max} \in [50..1000]$ is a multiple of 50 such that $I_{max}/50 = i$ then $\hat{Y} = f_i(x)$. Otherwise the closest matching "upper" model, f_u with $u = ceil(I_{max}/50)$, and "lower" model, f_l with $l = floor(I_{max}/50)$, are found. Then linear interpolation is performed as:

$$\hat{Y} = \frac{(Z_u - I_{max}) * f_l(x) + (I_{max} - Z_l) * f_u(x)}{(Z_u - I_{max}) + (I_{max} - Z_l)}. \quad (4)$$

where, $*$ denotes multiplication of scalars and $\hat{Y}$ is the probability that the design will complete routing within I_{max} iterations. For any given inference, at most two models within the system are queried. This system is visualized in Figure 2.

3.2 Model Selection

We evaluated several ML model types to be used in our ML confidence estimation system for the twenty f_i. We find decision tree-based ML models to perform best, as is often the case in tabular data analysis. Full results are found in Appendix C, but here we compare gradient-boosted decision trees (GBDT) against multi-layer perceptrons (MLP) as other model types yield inferior performance. As captured in Table 1, GBDTs achieve both higher accuracy and higher Matthew's correlation coefficient (MCC) [6] than MLPs when both have their hyperparameters tuned and they are evaluated on a 5-fold cross-validation of our training dataset. GBDTs also have the benefits of being quicker to train i.e., 15× faster than MLP on average, have negligible inference time even without hardware acceleration, and are white-box models with interpretable predictions. Evaluation metrics, dataset, and hyperparameter details are provided in Appendices A & B & D respectively. We also considered a neural network for model selection in the MoE system but opted out, as choosing f_l and f_u based on I_{max} is straightforward.

Table 1: gradient-boosted decision trees outperform neural networks

Model	Accuracy (↑)	MCC (↑)
GBDT	86%	0.60
MLP	79%	0.30

4 System-level Evaluations

4.1 Setup

In this section, we evaluate the prediction quality and confidence calibration of our tuned MoE classifier system on our final test dataset (detailed in Appendix B and our artifacts from Appendix E). We compare against the architecture from [11] which combines 4 binary classifiers and 4 regressors to predict the number of iterations required to solve FPGA routing problems. Each regressor is trained to make predictions for different iteration ranges and the classifiers are used to select which regressor to query for a given inference. We also compare against an enhanced version of [11] which contains 10 classifiers and 10 regressors for the sake of parity with our prediction system of 20 classifiers. This enhanced architecture partitions the prediction problem such that one regressor is trained to make predictions for iteration range [1, 100] then another for range [101,200], etc. The classifiers select which regressor to query in the same manner as [11].

The prior work architectures make "expectation-based" predictions where they predict the number of iterations remaining and compare this against the routing iteration limit in order to determine the routability of a circuit. There is no associated estimation for probability of success, they predict routability likelihood as either 0% or 100%. It might be possible to estimate success probability based on the difference between predicted iterations remaining and the iteration limit, but this was not part of the original implementation. The crux of our investigations is to determine the utility of estimating such probabilities, a full treatment of the various techniques to accomplish this is beyond our scope. One possible way to obtain *confidence* estimates for regressors is through gaussian process regression. However, it is non-trivial, often requiring multiple inference passes [9], which inflates routing time. In contrast, our MoE classifier system is the most direct approach to estimating success probabilities, as its binary classification trees' raw prediction confidences are already probabilities based on training data.

4.2 Calibration Comparison

Since our goal is to estimate the probability of success in routing, not to enhance routing itself, having a well-calibrated prediction system is crucial. We assess calibration by running inference on the entire test set.

Table 2: Confidence Calibration Study

	Prediction System	ECE (↓)	BS (↓)	NLL (↓)
train	Expectation-based (Prior work)	6.4%	0.065	2.23
	Enhanced Expectation-based	5.4%	0.055	1.89
	Confidence-based (Ours)	5%	0.03	0.12
test	Expectation-based (Prior work)	20.8%	0.208	7.18
	Enhanced Expectation-based	20.9%	0.209	7.22
	Confidence-based (Ours)	5.8%	0.128	0.39

The "Expectation-based" results in Table 2 correspond to the architecture of our prior work. The "Confidence-based" results correspond with our MoE classifier system from Section 3.1. The "enhanced" version of prior work is slightly worse in all metrics compared to the original, but the differences are insignificant. This suggests that merely partitioning the problem space isn't enough to improve predictions. Our confidence-based MoE system outperforms both approaches. The expected calibration error of the prior work on test data is 20.8%, compared with 5.8% from our confidence-based system, showing that simply taking the decision of greater likelihood based on model predictions leaves a high risk exposure.

Furthermore, understanding how calibration changes across use cases is crucial. We assess this across various routing iteration limits. Figure 3 shows the expected calibration error (ECE) of the confidence-based system compared to the regressors across many decision thresholds. The confidence-based system achieves significantly lower ECE compared to prior work in all cases without any overlapping, despite the expectation-based approach performing strongly at low iteration limits.

Table 3 compares the prediction quality of the three systems in terms of accuracy and Matthew's correlation coefficient (MCC). As the test data is imbalanced with respect to the two classes (27% routable to 73% unroutable), the MCC is important since it rebalances the importance of the classes. Although the differences are small, our confidence-based MoE has slightly higher prediction quality but this is not a major finding. The key takeaway is that the differences between the systems in Table 2 and the results to follow in Section 5 are not simply due to prediction quality discrepancies. Rather, the prediction qualities are roughly equal and other aspects of the systems are important for their usage as risk management tools by estimating routability probabilities.

Table 3: Prediction Quality Summary

Prediction System	Accuracy (↑)	MCC (↑)
Expectation-based (Prior work)	79.2%	0.599
Enhanced Expectation-based	79.1%	0.602
Confidence-based (Ours)	80.4%	0.605

4.3 Ablation Study

We chose to use 20 models in our MoE classifier system because this was the most we could reasonably use with our available resources, as each model requires a slightly different training data set to be curated and they are trained separately. We study the performance of this system with models removed in an ablation study.

Figure 4 depicts the variation of system performance and calibration as models are removed. In each stage of ablation, models are removed as uniformly as possible. For example, the data point corresponding with 20 models has models for decision thresholds {50, 100, 150, .., 1000}. Each data point represents an average across all routing iterations from all circuit designs in our evaluation set (see appendix) evaluated at user iteration limits in the range [400, 600]. We restrict evaluation to this range to make the experiments computationally feasible. The results show that system performance is unaffected as long as there are more than 2 models in the system. The data point for 2 models evaluates a system which is only trained for decision thresholds at 50 and 950. Our system of 20 classifiers may be unnecessarily large but this is not harmful to our results.

5 FPGA Routing Risk Management Evaluations

5.1 Setup

Routing Replay: While the evaluation described in Section 4 is diagnostically useful, the most important metrics can only be captured from routing. Now we evaluate the scenario where a routing

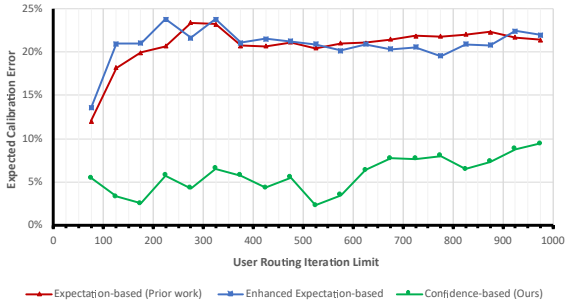

Figure 3: Our confidence-based approach to predicting routability likelihood achieves much lower expected calibration error at all routing iteration limits compared with expectation-based approaches. Y axis plots average value of ECE for the test data.

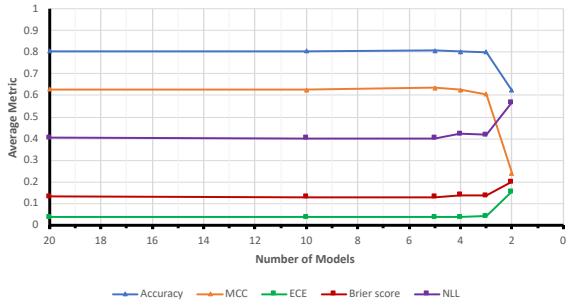

Figure 4: System performance is unaffected as long as at least 3 models are part of MoE. Y-axis is average value of accuracy, MCC, ECE, Brier score and NLL across all test data. For Accuracy and MCC *higher is better* and for ECE, BS and NLL *lower is better.*

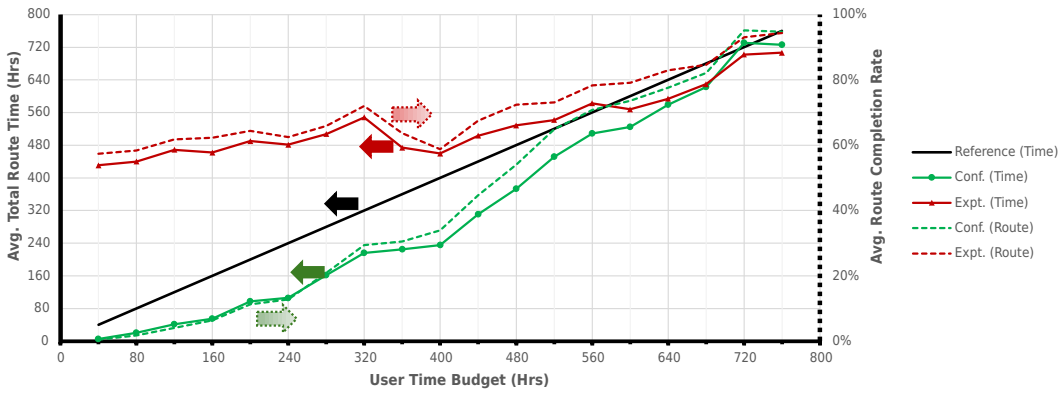

Figure 5: Confidence-based routing early exit is better for risk management than a regression-based approach.

attempt early exits whenever a ML system predicts the design is unroutable. Such routing experiments are typically very slow, but in our prior work [11], we introduce routing simulation to accelerate early exit experimentation. We adopt this idea by recording routing results between iterations during data collection and "replaying" them with various early exit conditions. In this process, routing proceeds almost exactly as it normally would until an early exit trigger is activated to terminate routing.

During routing replay, we collect two metrics: (1) *total time spent in routing* – the sum of time spent on completed designs, early exited designs, and unroutable designs, and (2) *the route completion rate* – the fraction of routable designs which are not exited early. In practice, a trade-off emerges; being quick to early exit generally results in lower routing time but also reduces the route completion rate (and vice versa). The risk management we pursue is about enabling the possibility to control this trade-off by intelligently tuning the early exit trigger to best suit individual users.

User-tuned Early Exit: We investigate two types of early exit triggers: (1) Minimum prediction confidence from our MoE classifier system, and (2) A number of consecutive unroutable predictions from the expectation-based prediction system. We wish to be able to control the trade-off of routing time versus routing completion with an appropriately selected trigger value to accommodate different users' risk sensitivities. However, there is no intrinsic one-to-one mapping from these trigger values to the resultant routing time and completion rate. Considering the prediction confidence trigger as an example, the route completion rate depends on both the confidence trigger and the ML models generating the corresponding confidence values. A "pessimistic" ML model that never predicts above 70% routability likelihood would achieve a route completion rate of 0% if the early exit confidence threshold were set to 75%; an "optimistic" ML model would achieve 100% route completion for the same threshold if it never predicted below 80% routability likelihood. Due to this lack of intrinsic mapping, the relationships between the early exit triggers and the two outcome metrics must be empirically characterized.

Evaluation Scenario: We target a theoretical scenario where a highly experienced user has some fixed amount of time that they are willing to spend on routing a suite of designs and they also roughly know (1) their designs rarely complete routing after some number of routing iterations, i.e. a max iteration limit (I_{max}) for stopping the router, and (2) how many of the designs could reasonably be completed with the time that they have. This user then inputs their required route completion rate and iteration limit into our risk management tools, which set the routing early exit trigger to a value that will most precisely achieve the user's requested route completion rate (and therefore respect the user's time budget).

5.2 Risk Management Characterization

Characterization Step: We split our training dataset into training and characterization subsets. For each early exit trigger type, relevant models are trained on design data from the training subset. Characterization is performed by "replaying" routing on the characterization subset while sweeping the routing iteration limit in {50, 75, .., 1000} and user-requested route completion rate in {5%, 10%, 95%}. The resultant actual route completion rate across all of the designs is then recorded. We create a lookup table where the keys are a desired route iteration limit and route completion rate and the values are the minimum early exit trigger value that will result in exceeding the desired completion rate. For our MoE confidence estimator, the values are minimum confidences; for the expectation-based predictor, the values are the minimum consecutive unroutable predictions needed to trigger early exit.

Evaluation Step: We evaluate each relevant prediction system and its associated characterization lookup table as risk management tools by performing routing replay with them on our test set designs (detailed in Appendix B and our artifacts from Appendix E). This is done for the same range of iteration limits and completion rates as used in the characterization step.

In Figure 5 we present results from a perspective considering the user's original time budget. This figure compares the results from evaluating the two early exit trigger types (prediction confidence and consecutive unroutable predictions) with their corresponding prediction systems, confidence-based in green and expectation-based in red. Each data point represents an average of results across the full range of tested routing iteration limits. Points below the black reference line process all designs for routing in less time than the user's budget while those above exceed the budget. An ideal

Table 4: Our confidence-based risk management performs best

Risk Manager	Root Mean Squared Error (↓)	
	Time Control	Routing Control
Enhanced Expectation-based	273 hours	38%
Confidence-based (Ours)	143 hours	22%

risk management tool would adhere tightly to the black reference line. The confidence-based approach generally uses less time than budgeted while the expectation-based approach generally exceeds the budget. Correspondingly, the expectation-based approach also has higher route completion rates but this is actually not ideal as it would require far more time than the user has available to achieve this. Overall, the confidence-based approach does a better job as a risk management tool because it is more precisely able to control the trade-off between routing time and route completion.

Table 4 summarizes the results from Figure 5. The "Time Control" column represents the same time-centric perspective taken in Figure 5. The corresponding values are the root mean squared errors for the two risk management approaches, where the "true" values are the user's time budget. The errors are absolute differences between the time budget and the actual routing time, i.e. error is unchanged whether actual routing time is above or below the time budget, as either case represents an inability to precisely manage risk. The "Routing Control" column corresponds with the error associated with the route completion rate values input to the lookup tables for the risk management tools. Overall, **our confidence-based approach is able to manage the risk trade-off in routing time and route completion rate almost twice as precisely as the expectation-based approach from prior work.**

5.3 Future Improvement: Calibration Across Iterations

Here we investigate the potential source of error in our results from Section 5. Figure 6 shows a breakdown of the expected calibration error (ECE) of the confidence-based and expectation-based prediction systems as routing progresses for train and test data sets. Each data point averages across all predictions for an iteration across runs. While calibration is good in early iterations, it degrades later on. It is very likely that this was a significant cause of error for both prediction systems in managing risk. However, the training set calibration actually improves in the later stages of routing, which is what we expect as the models have better information to make predictions at these points. This suggests at least one of two possibilities: either the models are somehow overfit to the training data for late-stage routing but not early-stage routing, or the training data and testing data have divergent characteristics in late-stage routing. We have already tuned the hyperparameters of our models to avoid overfitting, so we investigate the latter possibility and results are shown in Figure 7.

Figure 7 shows the ECE results when we change the train-test split. The original train-test split used in our evaluations is extremely challenging, featuring relatively small designs in the training set and relatively large, industrial-complexity designs in the test set. In Figure 7, the train-test split is changed so that designs

Figure 6: Expected Calibration Error (ECE) with original train/test split

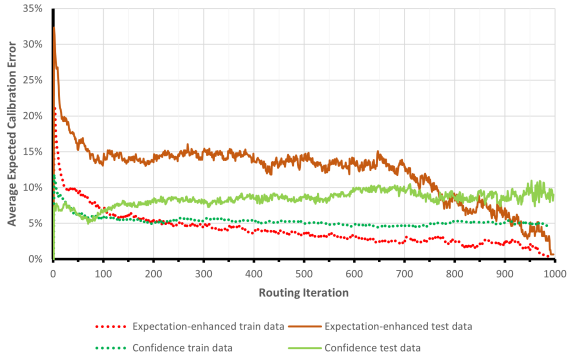

Figure 7: Expected Calibration Error (ECE) with modified train/test split

are randomly assigned between train and test rather than splitting by design size. In this case, the test set ECE is more aligned with the training set, suggesting that further work needs to consider more high-quality circuit designs in order to improve model calibration.

6 Conclusion

In this paper, we explored a novel problem in the form of risk management for ML model predictions as applied to FPGA routability for early exit. We have introduced an approach for enabling this risk management by leveraging the self-reported confidence values of tree-based ML classifiers. We compare this approach in a risk management evaluation against an alternative approach based on prior work. We find that our confidence-based approach is roughly twice as precise in managing the trade-off between time spent in routing and the route completion rate across a suite of realistic benchmark designs. Although further exploration of advanced techniques is needed to make this type of risk management framework practical, our findings focus on the underlying prediction mechanisms which will be the core driver of such a framework in the future.

References

[1] Mohamed Baker Alawieh, Wuxi Li, Yibo Lin, Love Singhal, Mahesh A. Iyer, and David Z. Pan. 2020. High-Definition Routing Congestion Prediction for Large-Scale FPGAs. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 26–31. https://doi.org/10.1109/ASP-DAC47756.2020.9045178

[2] Aman Arora, Andrew Boutros, Daniel Rauch, Aishwarya Rajen, Aatman Borda, Seyed Alireza Damghani, Samidh Mehta, Sangram Kate, Pragnesh Patel, Kenneth B. Kent, Vaughn Betz, and Lizy K. John. 2021. Koios: A Deep Learning Benchmark Suite for FPGA Architecture and CAD Research. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 355–362. https://doi.org/10.1109/FPL53798.2021.00068

[3] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2021. BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[4] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. 2023. Chip-chat: Challenges and opportunities in conversational hardware design. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 1–6.

[5] Glenn W Brier. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review* 78, 1 (1950), 1–3.

[6] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21 (01 2020). https://doi.org/10.1186/s12864-019-6413-7

[7] Mohamed A. Elgamma, Kevin E. Murray, and Vaughn Betz. 2020. Learn to Place: FPGA Placement Using Reinforcement Learning and Directed Moves. In *2020 International Conference on Field-Programmable Technology (ICFPT)*. 85–93. https://doi.org/10.1109/ICFPT51103.2020.00021

[8] Mohamed A. Elgammal, Kevin E. Murray, and Vaughn Betz. 2022. RLPlace: Using Reinforcement Learning and Smart Perturbations to Optimize FPGA Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 8 (2022), 2532–2545. https://doi.org/10.1109/TCAD.2021.3109863

[9] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 1050–1059. https://proceedings.mlr.press/v48/gal16.html

[10] Pingakshya Goswami and Dinesh Bhatia. 2021. Congestion prediction in fpga using regression based learning methods. *Electronics* 10, 16 (2021), 1995.

[11] Andrew David Gunter and Steven J.E. Wilton. 2023. A Machine Learning Approach for Predicting the Difficulty of FPGA Routing Problems. In *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 63–74. https://doi.org/10.1109/FCCM57271.2023.00016

[12] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. *CoRR* abs/1706.04599 (2017). arXiv:1706.04599 http://arxiv.org/abs/1706.04599

[13] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M. Dai, and Dustin Tran. 2021. Training independent subnetworks for robust prediction. arXiv:2010.06610 [cs.LG]

[14] José Miguel Hernández-Lobato and Ryan P. Adams. 2015. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. arXiv:1502.05336 [stat.ML]

[15] Jihye Kwon and Luca P. Carloni. 2020. Transfer Learning for Design-Space Exploration with High-Level Synthesis. In *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*. 163–168. https://doi.org/10.1145/3380446.3430636

[16] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6405–6416.

[17] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, Kenneth B. Kent, Jason Anderson, Jonathan Rose, and Vaughn Betz. 2014. VTR 7.0: Next Generation Architecture and CAD System for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 7, 2, Article 6 (jul 2014), 30 pages. https://doi.org/10.1145/2617593

[18] Yuzhe Ma, Ziyang Yu, and Bei Yu. 2019. CAD Tool Design Space Exploration via Bayesian Optimization. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*. 1–6. https://doi.org/10.1109/MLCAD48534.2019.9142051

[19] Dani Maarouf, Abeer Alhyari, Ziad Abuowaimer, Timothy Martin, Andrew Gunter, Gary Grewal, Shawki Areibi, and Anthony Vannelli. 2018. Machine-Learning Based Congestion Estimation for Modern FPGAs. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 427–4277. https://doi.org/10.1109/FPL.2018.00079

[20] Anushree Mahapatra and Benjamin Carrion Schafer. 2014. Machine-learning based simulated annealer method for high level synthesis design space exploration. In *Proceedings of the 2014 Electronic System Level Synthesis Conference (ESLsyn)*. 1–6. https://doi.org/10.1109/ESLsyn.2014.6850383

[21] T. Martin, G. Grewal, and S. Areibi. 2021. A Machine Learning Approach to Predict Timing Delays During FPGA Placement. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 124–127. https://doi.org/10.1109/IPDPSW52791.2021.00026

[22] Larry McMurchie and Carl Ebeling. 1995. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '95)*. Association for Computing Machinery, New York, NY, USA, 111–117. https://doi.org/10.1145/201310.201328

[23] Kevin Murray, Scott Whitty, Suya Liu, Jason Luu, and Vaughn Betz. 2015. Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD. *ACM Trans. Reconfigurable Technol. Syst.* 8, 2, Article 10 (mar 2015), 18 pages. https://doi.org/10.1145/2629579

[24] Kevin E. Murray, Oleg Petelin, Sheng Zhong, Jia Min Wang, Mohamed Eldafrawy, Jean-Philippe Legault, Eugene Sha, Aaron G. Graham, Jean Wu, Matthew J. P. Walker, Hanqing Zeng, Panagiotis Patros, Jason Luu, Kenneth B. Kent, and Vaughn Betz. 2020. VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling. *ACM Trans. Reconfigurable Technol. Syst.* 13, 2, Article 9 (may 2020), 55 pages. https://doi.org/10.1145/3388617

[25] Chak-Wa Pui, Gengjie Chen, Yuzhe Ma, Evangeline F. Y. Young, and Bei Yu. 2017. Clock-aware ultrascale FPGA placement with machine learning routability prediction: (Invited paper). In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 929–936. https://doi.org/10.1109/ICCAD.2017.8203880

[26] Martin Rapp, Hussam Amrouch, Yibo Lin, Bei Yu, David Z. Pan, Marilyn Wolf, and Jörg Henkel. 2022. MLCAD: A Survey of Research in Machine Learning for CAD Keynote Paper. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 10 (2022), 3162–3181. https://doi.org/10.1109/TCAD.2021.3124762

[27] Atefeh Sohrabizadeh, Cody Hao Yu, Min Gao, and Jason Cong. 2021. AutoDSE: Enabling Software Programmers Design Efficient FPGA Accelerators. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 147. https://doi.org/10.1145/3431920.3439464

[28] Mattias Teye, Hossein Azizpour, and Kevin Smith. 2018. Bayesian Uncertainty Estimation for Batch Normalized Deep Networks. arXiv:1802.06455 [stat.ML]

[29] Huimin Wang, Xingyu Tong, Chenyue Ma, Runming Shi, Jianli Chen, Kun Wang, Jun Yu, and Yao-Wen Chang. 2022. Cnn-inspired analytical global placement for large-scale heterogeneous fpgas. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 637–642.

[30] Zi Wang and Benjamin Carrion Schafer. 2020. Machine Leaming to Set Meta-Heuristic Specific Parameters for High-Level Synthesis Design Space Exploration. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218674

[31] Saeyang Yang. 1991. Logic Synthesis and Optimization Benchmarks User Guide Version 3.0.

[32] Shuhan Zhang, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. 2022. LinEasyBO: Scalable Bayesian Optimization Approach for Analog Circuit Synthesis via One-Dimensional Subspaces. In *2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD)*. 27–34. https://doi.org/10.1109/MLCAD55463.2022.9900105

A Machine Learning Metrics

In our system-level evaluations, we evaluate model performance using Accuracy and Matthews Correlation Coefficient (MCC), which are given by,

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

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

where, TN denotes true negatives, FN denotes false negatives, TP denotes true positives, and FP denotes false positives.

Furthermore, since the datasets we use contain more negative (unroutable) samples than positive (routable), we use the MCC as our primary metric as it is more robust to class imbalance. Unlike accuracy, MCC takes the distribution of error between classes into consideration, which helps provide a more realistic picture of the classifier's performance. Accuracy is still provided in some cases for context, but it is possible to score well on accuracy on our imbalanced datasets while only ever predicting negative. The range for accuracy is $[0\%, 100\%]$, and MCC is in the range $[-1, +1]$. For both, *higher values are better*. A MCC score of +1 is a perfect binary classifier, -1 is a perfect binary classifier with inverted predictions, and a score of 0 is the expected result from a random classifier. In addition, we measure the calibration quality of the model using the calibration metrics described in Section 2.

B Data Gathering

We extract our data by replicating [11] through real FPGA routing with the FPGA CAD tools in the open-source Verilog-to-Routing (VTR) project [24]. Training data are extracted from routing the digital circuit designs found in the MCNC, VTR, and "Titan-other" (from VTR) benchmark suites [17, 31]. Testing data are extracted from the Koios and Titan23 benchmark suites [2, 23].

As we pursue a supervised learning approach and FPGA routing is iterative, we extract one data sample from each iteration for a given routing problem. Each sample consists of a feature vector paired with a regression label. The regression label for an iteration is the number of iterations remaining until the routing problem will be successfully solved. These labels are technically computed at the end of a routing problem then assigned to corresponding iterations' feature vectors. The features are a replication of the 79 features described in [11] using the information available, encompassing features such as resource overuse/underuse, FPGA architectural details, wirelength utilization and fanout, among others. While our regression models are trained to predict routing iterations remaining, our classification models are trained to predict whether the total number of iterations (Y) required to solve a routing problem is greater than some threshold (Z_i), i.e. if $Y > Z_i$. The corresponding boolean classification labels for a given Z are thus easily derived from the regression labels. We stop unsolved routing problems after 1000 iterations have completed.

C Detailed Model Selection

We performed experiments with simple classical ML models to justify our choice of gradient boosted decision trees for our main experiments. The results in this section come from training the models on our train set and testing on our test set. These results were not used in making this decision, as that would mean to use test set information in decision-making which is a potential form of data leakage. However, we present these results here to show the superiority of gradient-boosted decision trees (GBDT) compared to other classical ML model types. Table 5 reports the average performance metrics on the test set for the ML models we evaluate. For brevity, we do not include results of non-test set data. Overall, GBDT model outperforms all other model types as judged by the MCC values, which supports its use in our main experiments.

Table 5: Comparison of various classical model types. Results averaged across all decision thresholds.

Model	Accuracy (↑)	MCC (↑)
Gradient-boosted Decision Tree classifier with depth=6	**78%**	**0.56**
K-Nearest Neighbours model (with 10 neighbors)	71%	0.38
Gaussian Naive Bayes with priors=(0.5, 0.5)	63%	0.37
Support Vector Machine-based model	56%	0.02

D Hyperparameter Tuning

All models evaluated are trained with the default scikit-learn configurations. The MLP models are trained with early-stopping enabled. For the expectation-based systems, we replicated the setup of [11] and find that the default scikit-learn 1.0.1 model hyperparameters are best for that architecture. For our twenty classification ensemble models, we tune the max depth of individual trees and the number of trees in the ensemble. This tuning is done to maximize the Matthews correlation coefficient (MCC) of the models on a 5-fold cross validation of train data, as suggested by [6] for imbalanced data like ours. We also aim to maximize the range of their confidence estimations. The number of unique confidence values a classification tree can yield (per class) is bounded by the number of its leaf nodes and therefore depth, i.e. $\#confidences \leq leaves \leq 2^{depth}$. As we desire to use the decision trees' confidences directly for decision-making, it is ideal to maximize the tree depths without incurring training overfit. We initially found a depth of 6 or 7 to be suitable then tested these values in a 5-fold cross-validation of our training data while varying the number of trees simultaneously, finding 30 trees with depth 6 to be optimal. Table 6 provides an apt summary for classification hyperparameter tuning

Table 6: Classifier Hyperparameter Tuning. Results averaged across all decision thresholds.

Max depth	Number of Trees	Avg Accuracy across decision thresholds (Test) (↑)	Avg MCC across decision thresholds (Test) (↑)
6	5	81.53%	0.3133
6	10	85.76%	0.5567
6	20	85.77%	0.5847
6	**30**	**85.74%**	**0.5919**
6	50	85.47%	0.5917
6	70	85.24%	0.5887
7	5	82.27%	0.3629
7	10	85.39%	0.5498
7	20	85.63%	0.5838
7	30	85.58%	0.5896
7	50	85.38%	0.5896
7	70	85.06%	0.5841

Tuning of Multi-layer Perceptron

For our neural network model, we used a multilayer perceptron (MLP) architecture with 2 hidden layers of size n and $n/2$ neurons respectively. We evaluated the MCC and the calibration metrics for all $n \in \{100, 200, ..., 800\}$ as reported in Table 7. Since our dataset is imbalanced, the MLP runs the risk of becoming biased towards the majority class resulting in misrepsentation of the classifier's performance. Hence, we consider both the MCC and the calibration metrics to determine the best n. We tested these values in a 5-fold cross validation of our training data. As shown in Table 7, we find that $n = 400$ configuration provides a balanced performance in terms of MCC and calibration quality compared to other configurations. Therefore, we set $n = 400$ in our evaluations.

Table 7: MLP Hyperparameter Tuning. Results averaged across all decision thresholds.

Hidden Layer1 (n)	Avg MCC (Test) (↑)	Avg ECE (Test) (↓)	Avg Brier score (Test) (↓)	Avg NLL (Test) (↓)
100	0.29	0.07	0.16	0.77
200	0.23	0.06	0.17	0.71
300	0.21	0.08	0.18	1.14
400	0.3	0.07	0.17	0.97
500	0.31	0.07	0.16	1.26
600	0.31	0.08	0.17	1.57
700	0.3	0.07	0.16	1
800	0.33	0.08	0.16	1.24

E Artifact Appendix

E.1 Abstract

Artifacts include "raw" CSV data, "intermediary" CSV data, and Python scripts to process them. The raw data are those which are used from the beginning of an experiment. Intermediary data are provided where full experimentation on the raw data would be too slow for reproduction, these serve to accelerate the experimental process. The tables in the paper can generally be reproduced (unformatted) while the figures cannot be directly reproduced. Unplotted figure data can generally be reproduced. Exact reproduction of tables and plotted graphs requires manual work which is described in the artifacts.

E.2 Artifact check-list (meta-information)

- **Program:** Included, publicly available.
- **Model:** Necessary models trained from scratch or included.
- **Data set:** Included, approximately 10GB.
- **Run-time environment:** Conda 24.3 or newer, Python 3.10 or newer.
- **Hardware:** CPU.
- **Output:** Console log, some CSV files. Described in artifact README.txt.
- **Experiments:** Refer to artifact README.txt.
- **How much disk space required (approximately)?:** 10GB.
- **How much time is needed to prepare workflow (approximately)?:** 30 minutes.
- **How much time is needed to complete experiments (approximately)?:** 1 day.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Creative Commons Attribution 4.0 International.
- **Data licenses (if publicly available)?:** Creative Commons Attribution 4.0 International.
- **Archived?:** https://zenodo.org/doi/10.5281/zenodo.13118004

E.3 Description

E.3.1 How to access. Download and unzip from Zenodo at https://zenodo.org/doi/10.5281/zenodo.13118004

Will require about 10GB of disk space unzipped.

E.3.2 Software dependencies. Conda 24.3 or newer, Python 3.10 or newer.

E.3.3 Data sets. Included.

E.3.4 Models. Included or trained by artifact scripts.

E.4 Installation

Download and unzip the file hosted on Zenodo. Then use a terminal to navigate to the unzipped project top-level directory. Perform the commands:

conda env create -f paper87.yml

conda activate paper87

E.5 Evaluation and expected results

After following installation steps, all Python scripts in the "src" directory which correspond with a table or figure (indicated in the filename) should be executed. Figures will not be produced, but their unplotted underlying data will be. The artifacts contain descriptions of how to reproduce figures manually. Tables will generally be reproduced (unformatted) by the scripts and the artifacts' documentation (README.txt) notes where manual work is required otherwise.

Results will be output to console logs and CSV files. Results should match those reported in the paper to the extent of reduced precision due to rounding errors.