A APPENDIX A: CROSS-SOURCE GENERALIZATION ANALYSIS

To evaluate robustness across different data distributions and verify generalizability, we conduct leave-one-source-out (LOSO) experiments. Our training corpus comprises three sources: MG-Verilog (9,239 modules), verilog_github (7,791 modules), and VeriGen (3,923 modules). For each experiment, we train RocketPPA on two sources and test on the held-out third source using identical hyperparameters from Section 4.1.

Table 9 presents LOSO results at 10% threshold averaged across all four synthesis conditions (15nm/45nm $\times$ area-opt/delay-opt). When testing on held-out sources, average pass rates are 68.4% for area, 54.8% for delay, 52.4% for total power, and 53.9% for static power. Compared to training on all sources (Table 2: 71.6%, 57.2%, 55.0%, 56.7%), this represents degradation of only 3.2, 2.4, 2.6, and 2.8 percentage points respectively across the four metrics.

Table 9. Leave-one-source-out pass@10% averaged across synthesis conditions.

Train Src	Test Src	Area	Delay	Tot. Pwr	Stat. Pwr
All (Main)	VEval	71.6%	57.2%	55.0%	56.7%
MG + GH	VG	68.3%	54.8%	52.1%	53.9%
MG + VG	GH	69.1%	55.4%	53.2%	54.6%
GH + VG	MG	67.9%	54.1%	51.8%	53.2%
Average LOSO	–	68.4%	54.8%	52.4%	53.9%

MG=MG-Verilog, GH=verilog_github, VG=VeriGen, VEval=VerilogEval.

The modest degradation proves that RocketPPA generalizes effectively across diverse data sources and learns source-invariant circuit representations rather than memorizing source-specific patterns. Performance remains consistent across the three LOSO experiments (within $\pm$1.5 points), demonstrating that no single source dominates the learned features and confirming robust generalizability. Area prediction shows the strongest cross-source generalization (68.4%), likely because structural features transfer better than power patterns which may reflect source-specific coding practices.

The LOSO degradation (2.4–3.2 points) is comparable to our leave-one-regime-out results in Table 5 (typically $<$2.5 points), proving that our model generalizes equally well across data sources as it does across synthesis conditions. These results validate that our contrastive learning framework enables robust generalization to designs from new organizations or coding styles without retraining, which proves the practical generalizability necessary for real-world deployment. This cross-source robustness, combined with cross-regime results, establishes that RocketPPA learns fundamental circuit properties that generalize beyond the training distribution.

B APPENDIX B: HDL CODE REPAIR FRAMEWORK

Due to the significant number of syntactically incorrect or non-synthesizable Verilog modules in our initial dataset aggregated from MG-Verilog, verilog_github, and VeriGen repositories, we developed an automated LLM-driven repair pipeline to ensure data quality. Figure 8 illustrates this iterative framework, which takes faulty Verilog modules from our dirty dataset and attempts to correct them through multiple repair cycles. The system employs both a Verilog parser for syntax validation and synthesis tools to verify that corrected modules can be successfully synthesized. When errors are detected, the framework captures detailed error descriptions and leverages an LLM to generate targeted repairs.

The repair process operates through iterative refinement, with a maximum limit to prevent infinite loops. In each iteration, the LLM receives the current Verilog module along with specific error messages or warnings from the parser and synthesis tools. The LLM is instructed to analyze the error, propose corrections while preserving the original design intent, and return only the corrected Verilog code without explanatory text. Successfully repaired modules are added to our clean dataset, which ultimately comprised 20,953 synthesizable modules across all three sources. This cleaning pipeline was essential for creating the high-quality training corpus needed for RocketPPA, as it eliminated syntax errors and synthesis failures that would otherwise introduce noise during model training.

C APPENDIX C: TOKEN SIZE AND PPA METRIC CORRELATIONS

Figure 9 presents a comprehensive visualization of the relationship between Verilog code token size and the three primary PPA metrics (power, delay, and area) across our training dataset. The 2D heatmaps use color intensity to represent the frequency of designs within specific token size intervals and metric value ranges. This visualization reveals several important patterns in our dataset distribution: larger token sizes generally correlate with higher gate counts and greater power consumption, though the relationship is not strictly linear. The heatmaps demonstrate substantial variance in PPA metrics even within narrow token size bands, indicating that code length alone is insufficient for accurate PPA prediction and motivating our use of deep semantic representations from the LLM backbone.

The three heatmaps collectively illustrate the multi-dimensional complexity of the hardware design space. For power consumption (left panel), we observe a dense concentration of designs in the lower token size and power ranges, with a long tail extending toward higher values spanning

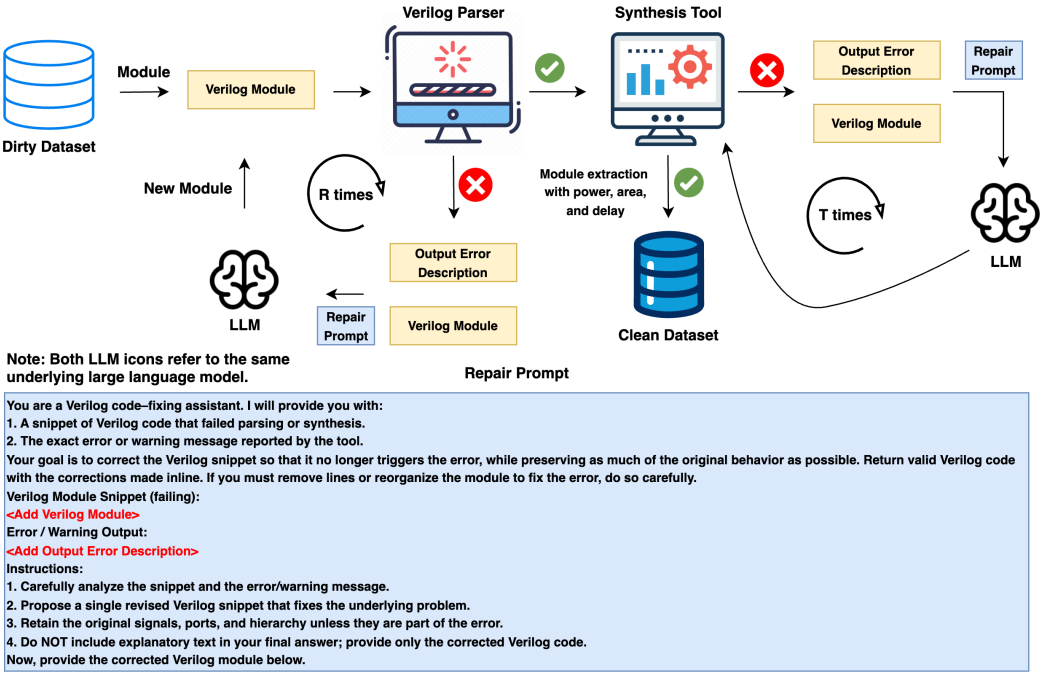

Figure 8. The proposed HDL code repair framework.

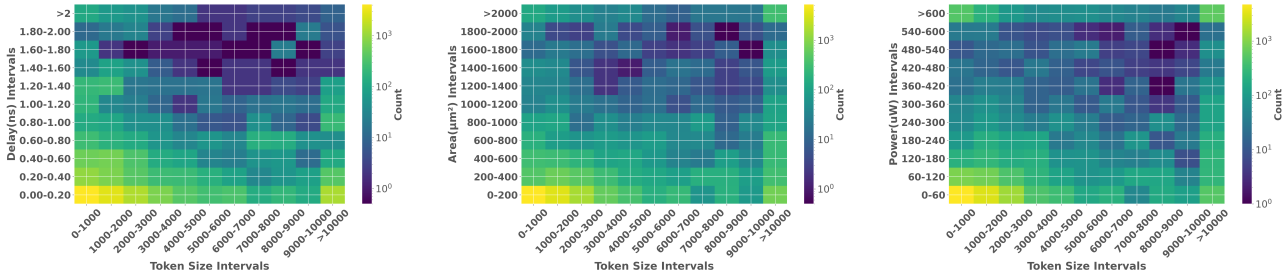

Figure 9. 2D Heatmap of Token Size vs. Power, Delay, and Area

several orders of magnitude. The delay heatmap (center panel) shows a more uniform distribution across token sizes, reflecting that timing characteristics depend heavily on critical path structures rather than design size. The area heatmap (right panel) exhibits the strongest correlation with token size, as expected given that larger modules typically instantiate more logic gates. These distribution patterns validate our decision to apply logarithmic transformation and z-score normalization (Equation 9) during preprocessing, as the raw PPA values span multiple orders of magnitude and would otherwise destabilize gradient-based training.

D APPENDIX D: TRAIN AND VALIDATION LOSS

Figure 10 shows the training and validation Huber loss curves for RocketPPA. The model converges stably, with the training loss reaching approximately 0.18 and the validation loss plateauing near 0.22. The validation curve does not increase after convergence, indicating minimal overfitting.

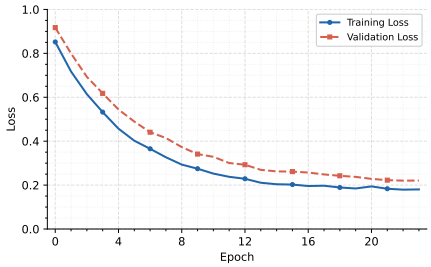

Figure 10. Training and validation Huber loss over 24 epochs for LLaMA-3.1-8B-Instruct with MoE. The training loss converges to 0.18 and the validation loss plateaus at 0.22, with a stable gap of 0.04.