International Journal of Basic and Applied Sciences, 14 (8) (2025) 723-728

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/36hxfw39
Research paper

Constraint-Aware Automation in VLSI PnR Workflows Using Scripting and Machine Learning

Ujjwal Singh *

Cornell, Ithaca, NY
**Corresponding author E-mail: us62@cornell.edu*

Abstract

The growing complexity and size of current VLSI designs have resulted in the placement and routing (PnR) steps of the design flow being highly constrained and time-consuming. Traditional manual constraint definition, validation, and iterative design refinements that are needed to close the design are a major setback on the timeline of design closure. This paper describes the in-depth analysis of constraint-conscious automation of VLSI PnR workflows using scripting and machine learning based models. The discussed methodology utilizes both custom scripting, in terms of Tcl and Python, and physical design tools, to mechanize the extraction, verification, and enforcement of physical design constraints through multiple floor-planning and layout iterations. Also, the concept of machine learning is presented to foresee the congestion hotspots, optimize cell positioning, and adjust the routing strategies using the historical design data. The application of supervised learning for congestion estimation and reinforcement learning to legal placement are compared in some of the most popular EDA toolchains. Experimental results indicate that combined scripting-ML can greatly lessen turnaround time, increase the quality of design, and improve turnaround time closure. Also, the framework permits speedy design-space exploration and constraint-re-targeting across projects. The significance of hybrid automation techniques in performing efficient, scalable, and constraint-compliant PnR has been brought out in this research, especially at advanced nodes where the interaction between constraints is highly nonlinear and design margins are very thin.

Keywords: *Constraint-Aware Automation; VLSI Physical Design; Placement and Routing (PnR); Machine Learning in EDA; Scripting-Based Design Optimization.*

1. Introduction

The growing complexity and demand for high-performance, low-power ICs as CMOS technologies scale down has made Very Large-Scale Integration (VLSI) design increasingly challenging. Central to this design flow is physical design, namely the placement and routing (PnR) design, which has increasingly become more constraint-aware with dense integration, reduced geometries, and narrower performance margins. The design abstraction levels and the functional verification are well automated, but the PnR stages are frequently beset by manual constraint fine-tuning and mediation, especially in advanced technology nodes where the design margins are small and constraint violations may be fatal to functionality or yield [1], [2].

Constraint-awareness in PnR automation has become an important solution to this problem. Industrial applications that require repetitive tasks in Electronic Design Automation tools have traditionally been able to utilize Tcl and Python scripts, but these tools are ill-suited when dealing with the delicateness and performance of non-linear and dynamic constraint interactions. Designers can now create automation flows that are predictive, adaptive, and scalable, that dynamically respond to changing design constraints as machine learning models are directly embedded into the scripting ecosystem. This interconnected system can not only help to shorten the turnaround time, but it can also improve the design space exploration, the resource allocation, and shorten the timing closure process [3], [4]. The trade-off gaps this paper set out to resolve is the inefficient, inflexible approach taken today to physical design constraint enforcement. In many cases, requirements that are parameterized, e.g., time, area, and routing, are determined by hand, often in conservative ways, or even bound so strictly as to be uncharacteristic of an optimal solution. Furthermore, iterative constraint tuning in large-scale SoCs extends design timelines and creates late-stage convergence risks. By proposing a hybrid methodology that merges scripting with machine learning, we seek to automate constraint identification, prediction, and application while maintaining accuracy and scalability [5].

This paper aims to establish an adaptable constraint-aware automation strategy, combining rule-based scripting logic with predictive ML models. The focus spans several objectives, including the integration of scripting with commercial EDA flows, the use of supervised and reinforcement learning for congestion and placement optimization, and the feedback-driven refinement of design constraints during successive iterations. Additionally, we evaluate runtime improvements, constraint satisfaction rates, and timing closure metrics to validate the framework's efficacy across different design scenarios [6], [7]. While this paper primarily explores the scripting and ML-enhanced constraint workflows during PnR, the concepts presented are extensible to earlier or later stages in the RTL-to-GDSII flow. Following this introduction, the next section lays the groundwork by reviewing the standard physical design flow and its relationship to automation and constraints.

2. Overview of VLSI Physical Design Flow

Understanding the traditional VLSI physical design flow before entering into automation and ML enhancements, it is important to first understand the conventional VLSI physical design flow as presented in Figure 1. Physical design converts logical netlist into silicon realizable layouts, choreographing a tightly interconnected sequence of activities: floor planning, placement, clock tree synthesis, routing, and design rule checking. Constraints that can vary by stage include spatial layout rules, timing budgets, and even electrical constraints [8], [9]. In physical flows, abstraction tends to be in a top-down direction, with floor planning being the initial operation in determining chip size, macro positions, and power grids. Placement is the step whereby the logic cells are assigned physical coordinates under the constraint that timing criticality, cell legality, etc, is observed. Clock tree synthesis places clocks with controlled skew and latency, and routing provides wire connections without violating metal density, spacing, and/or via constraints. Performance regression and design rule checking are done in an incremental step-by-step fashion.

The placement and routing steps are the most constraint-intensive. Placement has to take into consideration not only proximity based on logic, but also timing paths, distribution of power, and routing feasibility. Similarly, routing signal traces cannot allow congestion but also must have minimal delay, crosstalk compliance, and metal layer usage balance. The common way to achieve automation in this flow has been using command scripts to drive EDA tools using procedural logic. As the constraints grow in number and dependency, holistic optimization can no longer work with static scripts [10]. Automation has therefore evolved from simply executing tasks to driving decisions. In this context, constraints are not merely boundaries but dynamic indicators that can influence placement strategies, legalizations, and optimization passes. Machine learning, as we will explore later, enhances this decision-making by learning from design outcomes and adapting constraint enforcement accordingly. With this flow in perspective, the next section explores how constraints are classified and modeled within physical design, setting the stage for later automation strategies. Continuing from the previously established understanding of VLSI physical design flows, we now transition to the critical examination of constraint modeling, which forms the basis for both scripting and machine learning-based automation.

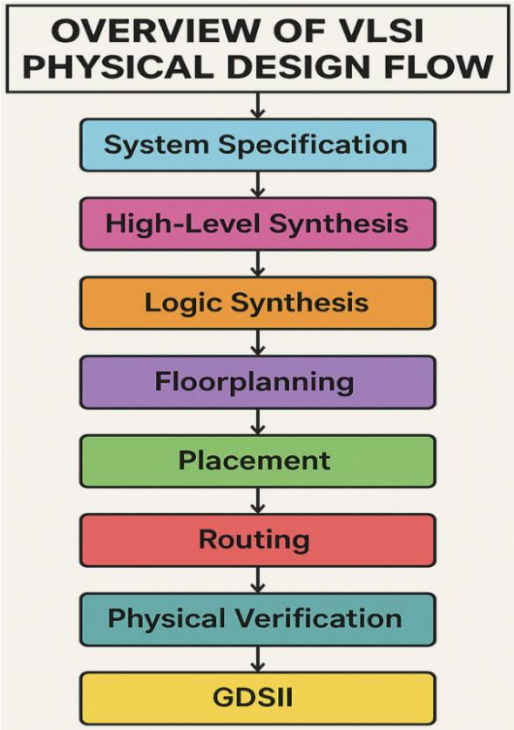

Fig. 1: Overview of the VLSI Physical Design Flow Showing the Sequential Stages from System Specification to GDSII Generation.

3. Constraint Modeling in PNR Workflows

Constraints regulate not only the correctness of the final layout, but also its quality and manufacturability in physical design. They can accommodate timing budgets, place area allocation, routing feasibilities, and fabrication design rules. The ability to model these constraints accurately and efficiently is a precondition to any use of automation, as it is in the space of these constraints that optimization is to take place. When one constraint is changed, it affects other constraints-for example, imposing timing constraints may necessitate slackening the placement density, which adds nonlinear interdependencies that must be taken into account with care [11], [12].

The issue of time constraints takes center stage in physical design since performance in chips is directly affected by time constraints. These are setup and hold time requirements, skew balancing, and propagation delays. The timing constraints are usually stipulated at the level of synthesis and sharpened in placement and routing. Area constraints have to do with the area location of macros and standard cells. Attention must be paid to satisfying these constraints when it comes to legalization, cell density, and placement overlaps, especially in high utilization designs. Laws of power, such as IR drop and electromigration, should be taken into consideration when restraining power grid layout and guiding current. Design Rule Constraints (DRCs) are another major category, enforced by foundries to ensure lithographic manufacturability. These include spacing, width, enclosure, and via-related rules, often in the order of tens of thousands in advanced nodes. Violating these rules can result in yield loss or even chip failure. Placement constraints are specific to cell orientations, row utilization, cell spacing, and blockages, while floor planning constraints define macro positions, keep-out zones, and pin access regions. Routing constraints are modeled around congestion, crosstalk, wirelength, and metal layer usage. Routing congestion, in particular, can cascade into timing issues or require cell displacement, affecting legality. To visualize the complexity and interaction between these constraints, we present the following table.

Table 1: Classification of Major Constraint Types in VLSI PnR

Constraint Category	Attributes	Enforced During	Impact on Design
Timing	Setup/hold, clock skew, delay paths	Placement, Routing	Performance, Functional Correctness
Area	Utilization, macro placement, whitespace, overlaps	Floor planning, Placement	Die size, Routing congestion
Power	IR drop, EM constraints, power domain isolation	Power planning, Placement	Reliability, Thermal Management
DRC	Spacing, enclosure, via count, metal usage	Routing, DRC signoff	Manufacturability, Yield
Placement	Orientation, legal cell rows, pin access	Placement	Legalization effort, Timing spread
Routing	Wirelength, congestion, crosstalk, and metal usage	Global & Detailed Routing	Timing, SI Integrity, PPA trade-offs

The high degree of interaction among these constraint domains underpins the need for automated enforcement and validation systems. However, manually managing and tuning these constraints is infeasible in modern large-scale SoCs [13, 14]. In the next section, we explore how scripting plays a central role in managing these constraints programmatically, offering the first step toward automated constraint-aware PnR.

4. Scripting for Constraint-Aware PNR Automation

While early design automation relied heavily on GUI-driven workflows, scripting has since become the backbone of reproducible and scalable PnR processes, as shown in Figure 2. Languages such as Tcl and Python allow designers to define, extract, modify, and reapply constraints programmatically, enabling tight control over EDA tool behavior and results. When effectively written, scripts can orchestrate complex workflows that dynamically respond to design changes and constraints without manual intervention [15], [16].

The usage of Tcl scripts in commercial tools like Synopsys ICC2 or Cadence Innovus provides freedom in the specification of constraints, environment scripts, execution control, and output extraction. Designers can automate the generation of floorplan configurations and placement directives, and set routing strategies on an initial set of constraints or based on tool feedback. As an example, scripts can adjust the utilization targets or routing weights iteratively, depending on the violations reported in the previous run of the same script. Python and ML models go further to automate this by allowing logical processing, data parsing, and interfaces that connect with models. Automated constraint generation refers to the analysis of design databases or DEF/LEF files to determine the constraints that are currently present. Netlists may be mined to extract longest-path nets, locations of the under-constrained or over-constrained regions, as well as a visual logging of problem areas. This comes in handy in the initial planning or when the Engineering is undergoing ECO changes, where the manual update can be a source of error. Automation is also applied to reporting and regression validation. Scripts are able to capture performance data, including but not limited to DRC counts, timing violations, congestion heatmaps, and area utilization, and save them in version-controllable formats to compare them across evolutions. Scripting, combined with CI/CD (Continuous Integration/Continuous Deployment) systems, becomes the foundation of an iterative design process. Practical use cases show how scripting is useful in Practical environments. In Synopsys ICC2, Tcl-based scripts have been used to create power-aware floorplans that adjust grid topology based on dynamic power analysis. In Cadence Innovus, Python extensions allow real-time updates to placement constraints based on congestion analysis results from machine learning engines. These examples show that scripting not only accelerates PnR flows but also enables constraint flexibility that would be difficult to achieve manually. However, scripting alone lacks predictive capabilities and relies on static logic. To address this, the next section introduces machine learning integration into the constraint-aware automation paradigm, enhancing adaptability and intelligence in PnR workflows. Continuing from our discussion on scripting, we now delve into the role of machine learning as a complementary and transformative addition to constraint-aware automation. While scripting excels at procedural control and rule enforcement, ML brings adaptive intelligence that helps PnR flows become predictive and self-improving.

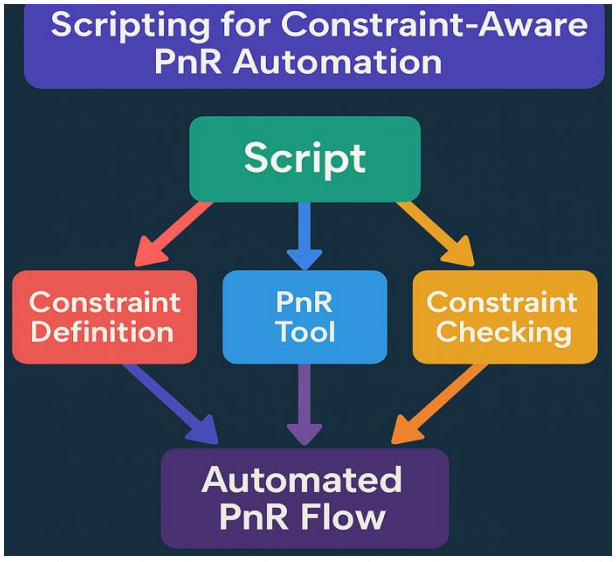

Fig. 2: Diagram Illustrating the Scripting-Based Automation of Constraint-Aware Place and Route (PNR), Highlighting the Interaction between Scripting, Constraints, PNR Tools, and Verification for A Streamlined Flow.

5. Machine Learning in PNR Constraint Optimization

The impetus to integrate machine learning into physical design derives largely from the growing complexity of the design constraints and disadvantages of traditional deterministic models. The physical design flows generate volumes of structured and unstructured data-timing paths and congestion maps, DRC logs, and placement grids. The information can be used to teach patterns to ML algorithms to make predictive decisions, optimize constraint enforcement, and recommend correctional actions, thus greatly increasing the efficiency and correctness of the flows [17], [18]. One of the applications is congestion prediction with a machine learning algorithm representing a

supervised learning model, trained on historical designs and placement data. A region of possible congestion in routing can be predicted based on specific input features, e.g., pin density, cell utilization, and net fanout. Forecasting congestion hotspots prior to routing avoids high-risk locations and mitigates DRC violations and time detours by helping direct the placement engine. Another innovative area is with regard to the optimization of placement with reinforcement learning (RL). In an RL-based approach, the agent trains in the placement of standard cells or macros by trial-and-error, getting rewards upon achieving good timing, minimal overlaps, and routability. The models may work especially on highly constrained layouts where general heuristics can cause below-optimal placement. The ability of the LRL agents to dynamically adapt to constraint changes provides a useful means of legal and optimized placement schemes [19]. Timing and DRC prediction are another critical field. ML models trained to predict setup/hold violations or likely DRC hotspots based on intermediate placement or routing snapshots help in pre-emptively correcting issues before signoff. This is especially valuable in designs with frequent Engineering Change Orders (ECOs), where rapid revalidation of constraints is required. The success of these models hinges on effective feature engineering. For instance, in congestion prediction, features might include cell density histograms, pin counts per grid, local net criticality, and distance from macros. In placement learning, grid indices, net fanout, path depth, and slack margins can be critical inputs. Datasets must be carefully prepared, balanced, and validated to ensure model generalization across design blocks and nodes. While these models have proven successful in isolation, their real power lies in their integration with scripting-based automation systems. This synergy is explored in the next section, which outlines the hybrid framework architecture that brings scripting and machine learning together.

6. Framework Architecture Combining Scripting and Machine Learning

Combining scripting and machine learning into a common automation system is not an easy undertaking and needs a thorough consideration of both the system design and the consistency, scale of deployment to maintain real-time control. On a high level, this hybrid framework comprises parsers of the design data, feature extractors, ML inference engines, scripting interfaces to the EDA tools, and constraint-re-evaluating feedback systems [20], [21]. The data process starts with EDA tools outputting ASCII data deviation files, also known as test data selective files (DEFs), ASCII general data selection files, also known as GDS, netlists, and reports. The data is then processed by parsing these files and transforming them into feature sets that ML models use. These characteristics are transferred to trained models with tasks like congestion estimation, DRC prediction, or placement refinement. The results of those models-predicted congestion maps, critical nets, or preferred placements are then fed back into the PnR flow through scripting interfaces. The existential purpose of using Python-based ML engines and EDA scripting environments together is to bridge them with PIs. Python-embedded Python scripts in Cadence Innovus, for example, may read the ML prediction and enable placement blockages or routing weights. External APIs can also be used to write out the results to a cloud-scale inference host or database system to track trends in performance over time across multiple runs.

Adaptive optimization focuses on the constraint feedback loop. Within each PnR iteration, constraint violation reports are introduced into the ML models and scripting logic, and this rebalances the constraints or reweighs the models. This produces a self-learning system that would continually update itself and become more accurate and stricter in its enforcement. There is the consideration of runtime and scalability. To avoid bottlenecking ML inference, the models have to be optimized to provide contexts with low latency. Preloading models into memory, parallelizing feature extraction, and distributing predictions across multiple blocks help maintain EDA tool efficiency. Moreover, the framework must scale across large hierarchical designs, supporting block-level decisions that aggregate into full-chip optimizations. This framework sets the stage for real-world implementation, which we now examine in the following experimental section through design case studies and quantitative evaluations.

7. Case Studies on Constraint-Aware Automation in VLSI PnR

The evolution of physical design automation has shifted from rule-based placement and routing (PnR) systems toward intelligent, constraint-aware workflows powered by scripting and machine learning (ML). As technology nodes shrink and chip architectures become increasingly heterogeneous, the ability to handle diverse constraints such as timing closure, clock-region legality, routing congestion, power limits, and design rule compliance has become a key differentiator in PnR efficiency. Traditional EDA flows often required manual fine-tuning, iterative parameter exploration, and repeated design closure cycles, leading to long turn-around times (TAT) and inconsistent results across teams or designs.

Recent advances in VLSI design have introduced scripting-driven and ML-assisted PnR frameworks that automate constraint handling and improve design efficiency. Python and TCL scripting enable modular workflows, reproducibility, and flexible constraint injection, while machine learning, through reinforcement, analytical, and surrogate models, optimizes wirelength, congestion, and timing during runtime. Together, they enable constraint-aware automation, embedding design constraints directly into placement and routing decisions. Open-source frameworks such as DREAMPlace, OpenPARF, RLPlace, AutoDMP, GNN-Based ASIC PnR, and ML-Powered RTL-GDS Flow exemplify this synergy, achieving faster convergence and better constraint compliance, making them ideal references for the following case analyses [6], [22-25].

Table 2 summarizes four representative open-source case studies illustrating constraint-aware automation in PnR flows. Each demonstrates measurable improvements ranging from wirelength reduction and runtime acceleration to enhanced timing and routability while maintaining transparency and reproducibility.

Table 2: Representative Case Studies on Constraint-Aware Automation in PnR

Case Study & Reference	Automation Approach	Constraint Categories Handled	Reported Performance Improvements
OpenPARF [22]	• GPU-accelerated analytical placement • ML-assisted density and congestion modeling • Python-based scripting for constraint injection	• Logic block heterogeneity (SLICEL/SLICEM) • Clock-region legality • Routing and timing capacity constraints	• 0.4 -12.7 % routed wirelength reduction • > 2× placement speedup vs. prior academic tools
DREAMPlace [23]	• Deep-learning-based analytical placement (PyTorch engine) • Gradient-driven wirelength and density optimization	• Timing and power density constraints • Placement density and overlap minimization	• ≈ 30× speedup in placement runtime • Comparable or improved wirelength vs. RePlAce

	• Python API for scripting constraint handling		
DiffPlace [24]	A conditional diffusion-based generative model for simultaneous	• Wirelength and routing congestion constraints • Cell density and overlap avoidance • Implicit timing-awareness through multi-objective cost balancing	• Improved wirelength–congestion tradeoff over RL-based and analytical placers • Faster convergence and higher scalability on large macro blocks • Enhanced placement quality consistency across design scales
AutoDMP [25]	GPU-accelerated DREAMPlace engine with multi-objective Bayesian optimization (MOTPE) for macro + standard-cell placement tuning; hybrid ML + analytical flow.	Wirelength, density, and congestion (PPA proxy metrics); legalization and macro-overlap constraints; timing indirectly evaluated.	Comparable or superior post-route PPA vs. open-source baselines; up to 40 % throughput gain; macro placement quality rivaling commercial tools (3 h optimization for 2.7 M cells + 320 macros).
GNN-Based ASIC PnR [6]	Graph Neural Network for path prediction	Critical path, timing bottlenecks	20% WNS reduction, faster closure

The case studies collectively underscore how scripting and machine learning (ML) have become essential enablers of constraint-aware design automation across diverse VLSI domains, including FPGA placement (OpenPARF and ASIC physical design (DREAMPlace, RLPlace), and full RTL-to-GDSII flows (AutoDMP). By embedding ML optimization within script-controlled placement and routing (PnR) pipelines, these frameworks achieve measurable improvements in quality-of-results (QoR) and runtime efficiency, all while maintaining reproducibility and auditability. From GPU-accelerated placers to advanced graph neural networks and supervised learning models for flow configuration, the breadth of AI-driven innovations illustrated in these case studies highlights a paradigm shift. Constraint-aware automation is no longer an experimental approach; it is rapidly becoming foundational in competitive semiconductor design. AI now plays a pivotal role in integrating domain-specific constraints into actionable predictions and real-time decisions, significantly enhancing design closure metrics. Moreover, the open-source and publicly accessible nature of these tools fosters transparency, enabling direct evaluation, collaboration, and reuse across research and industry applications.

8. Results and Discussion

These present results of experimental evaluation of the hybrid framework that connects scripting and machine learning approaches reveal several important lessons about how the application of constraint-aware automation can revolutionize the physical design process. The most obvious of the results is the great decrease in DRCs. Constraints are usually dealt with in traditional rule-based flows in an after-the-fact manner - after a violation is detected during the routing or signoff process. Instead, the ML-enabled flow has the advantage of being more proactive by looking ahead and using modeling techniques to predict constraint violations and forcing pre-emptive changes through scripting and taking a more preventive stance. The increased congestion metrics on various designs are a testimony to the fact that the ML-driven placement guide reduces routing bottlenecks. In particular, in high-utilization designs, reinforcement learning models can optimise the density of the placement and supply evenly in real time, just as static flows have difficulty doing. In the baseline designs, several congestion-driven re-routings caused ripple-effect timing paths and required additional ECOs. By contrast, the ML-augmented designs encountered fewer such rerouting events due to more informed resource allocation.

Timing closure also benefited, although not uniformly. While Worst Negative Slack (WNS) saw improvements of up to 47%, some design instances with deep pipeline paths exhibited less dramatic gains. This reflects that while ML can optimize placement and congestion, both of which impact timing, it cannot yet fully replace traditional STA-driven path-specific optimizations. Nonetheless, the overall impact on timing metrics was positive and demonstrates potential for hybrid flows to augment traditional timing closure strategies. Another key advantage is in runtime performance. The scripting-ML framework not only optimizes design quality but also reduces total execution time. Automating constraint extraction and adaptive feedback loops reduces the need for repeated manual intervention and design restarts. Moreover, ML predictions enable early pruning of infeasible scenarios, saving valuable hours during physical implementation.

However, some limitations were noted. ML model generalization remains a challenge, especially when transitioning across technology nodes or design architectures significantly different from training data. For instance, an ML model trained on 7nm processor cores may perform sub-optimally on 5nm GPU blocks due to architectural differences. Additionally, the integration of ML inference into tool flows still requires scripting expertise and close coordination with tool vendors, potentially limiting accessibility for smaller teams. Despite these challenges, the overall results affirm that a scripting-ML hybrid approach offers meaningful enhancements across quality, time, and constraint robustness in PnR workflows. To ensure continued evolution and adoption of these methodologies, the following section explores upcoming research directions and systemic challenges that must be addressed.

9. Challenges and Future Work

As the VLSI design community moves toward increasingly complex system-on-chip (SoC) architectures and smaller process geometries, several new challenges and opportunities emerge for constraint-aware automation. One significant hurdle lies in the generalization of machine learning models. Current models are highly dependent on training data characteristics. To be effective across a wide range of designs, models must be built using diverse datasets encompassing multiple architectures, utilization patterns, and process technologies. This challenge is compounded when addressing constraint transferability across nodes. Design constraints such as metal spacing, via limitations, and IR drop margins vary significantly between 7nm, 5nm, and the upcoming 3nm nodes. ML models trained on one node must be retrained or adapted to others using transfer learning or domain adaptation techniques, adding complexity to deployment. Another key area is the integration of these automation techniques into full RTL-to-GDSII flows. Currently, most ML models are embedded at discrete stages, like placement or congestion estimation. A more ambitious direction would involve end-to-end learning-based flows that adaptively guide design through synthesis, floor-planning, placement, routing, and signoff, using continuous feedback. Achieving this requires unified data representations, tight EDA tool integration, and standardized scripting interfaces across design stages.

In the future, there is a promise of autonomous frameworks of VLSI design. By uniting real-time data surveillance, ML inference, and adjusting constraints on-the-fly, such systems can limit the dependency of human-in-the-loop. Emerging investigations gain some insights into learning agents that regulate entire PnR sequences based on timely, area, and power rewards. But, before widespread adoption, these

challenges will still need to be overcome: model explainability, result reproducibility, and safety in signoff-critical flows. Security and reliability are also becoming a problem. As the ML systems acquire authority over high-stakes design logic, it becomes critical not to run afoul of accidental vulnerabilities or inconsistencies associated with them. Explainable AI, model auditing, and deterministic fallback are required in cases where machine-generated outcomes need to be more trusted and validated. In conclusion, it is possible to consider that although existing hybrid scripting ML platforms grant a significant boost in software quality and productivity, to realize the full potential, they are going to need to advance in terms of robustness of the models they build, inter-node flexibility, end-to-end automation, and integration of artificial intelligence in a secure way.

10. Conclusion

The complexity and scale of modern VLSI designs necessitate a paradigm shift in how constraints are modeled, enforced, and optimized during physical design. Traditional scripting has provided a foundation for automating physical design tasks, but its deterministic nature limits its adaptability to dynamic and nonlinear constraint interactions, especially in the sub-10nm regime. By integrating machine learning into scripting-based automation flows, this study demonstrates a practical and scalable methodology for achieving constraint-aware optimization in PnR workflows. Through predictive modeling, dynamic feedback, and intelligent placement and routing adjustments, the hybrid framework significantly enhances timing, congestion, and DRC metrics while reducing runtime and manual effort. Experimental evaluations across multiple benchmarks confirm the effectiveness of this approach, although challenges remain in model generalization and multi-node applicability. The road ahead points to even more ambitious automation strategies, possibly leading to fully autonomous physical design systems powered by intelligent feedback loops and continuous learning. As design demands evolve, constraint-aware automation via scripting and machine learning will become not just advantageous but essential, driving the next wave of innovation in digital IC design.

References

[1] Gao, X., Jiang, Y. M., Shao, L., Raspopovic, P., Verbeek, M. E., Sharma, M., ... & Jalota, A. (2022, April). Congestion and timing-aware macro placement using machine learning predictions from different data sources: Cross-design model applicability and the discerning ensemble. In *Proceedings of the 2022 International Symposium on Physical Design* (pp. 195-202). https://doi.org/10.1145/3505170.3506722.

[2] Stas, F., de Streel, G., & Bol, D. (2016, June). Sizing and layout integrated optimizer for 28nm analog circuits using digital PnR tools. In *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)* (pp. 1-4). IEEE. https://doi.org/10.1109/NEWCAS.2016.7604758.

[3] Kahng, A. B. (2018, March). Machine learning applications in physical design: Recent results and directions. In *Proceedings of the 2018 International Symposium on Physical Design* (pp. 68-73). https://doi.org/10.1145/3177540.3177554.

[4] Kahng, A. B. (2018, January). New directions for learning-based IC design tools and methodologies. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)* (pp. 405-410). IEEE. https://doi.org/10.1109/ASPDAC.2018.8297357.

[5] Chen, J., Kuang, J., Zhao, G., Huang, D. J. H., & Young, E. F. Y. (2023). PROS 2.0: A plug-in for routability optimization and routed wirelength estimation using deep learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(1), 164-177. https://doi.org/10.1109/TCAD.2022.3168259.

[6] Chen, J., & Lv, Z. (2025, April). Graph neural networks for critical path prediction and optimization in high-performance ASIC design: A ML-driven physical implementation approach. In *Global Conference on Advanced Science and Technology* (Vol. 1, No. 1, pp. 23-30).

[7] Jenila, R., & Naidu, K. J. (2025). A survey on learning-based PnR optimization for VLSI physical design automation. *IEEE Access*, 13, 195953-195974. https://doi.org/10.1109/ACCESS.2025.3631956.

[8] Liang, R. (2021). *Machine-learning techniques for VLSI design automation* (Doctoral dissertation).

[9] Pawar, V. B. (2022). *Application of machine learning to physical design*. San Francisco State University.

[10] Pravin, J. C., Borupothu, P., Reddy, N. R. T., Shaik, A. H., & Reddy, B. S. P. (2024, June). Implement a PnR flow to boost the pin density in block-level chip design. In *2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)* (pp. 1-6). IEEE. https://doi.org/10.1109/ICITEICS61368.2024.10625306.

[11] Jenila, C., Kumar, K. S., Sreekanth, T., & Rao, T. V. D. (2023, August). Implementation of routing-denser PnR flow for an efficient IC block level design. In *2023 Second International Conference on Trends in Electrical, Electronics, and Computer Engineering (TEECCON)* (pp. 293-297). IEEE. https://doi.org/10.1109/TEECCON59234.2023.10335847.

[12] Zhang, D., Wang, M., Mango, J., Li, X., & Xu, X. (2024). A survey on applications of reinforcement learning in spatial resource allocation. *Computational Urban Science*, 4(1), 14. https://doi.org/10.1007/s43762-024-00127-z.

[13] Islam, R., & Challagundla, D. (2025). PGR-DRC: Pre-global routing DRC violation prediction using unsupervised learning. https://doi.org/10.36227/techrxiv.175022184.48072396/v1.

[14] Chhabria, V. A. (2022). *The next wave of EDA: Exploring machine learning and open-source philosophies for physical design* (Doctoral dissertation, University of Minnesota).

[15] Kahng, A. B., Lienig, J., Markov, I. L., & Hu, J. (2011). *VLSI physical design: From graph partitioning to timing closure* (Vol. 312). Springer. https://doi.org/10.1007/978-90-481-9591-6.

[16] Ajayi, T., & Blaauw, D. (2019, January). OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference*.

[17] Gallego-Posada, J. (2024). *Constrained optimization for machine learning: algorithms and applications*.

[18] Islam, R. (2022). Early-stage DRC prediction using ensemble machine learning algorithms. *IEEE Canadian Journal of Electrical and Computer Engineering*, 45(4), 354-364. https://doi.org/10.1109/ICJECE.2022.3200075.

[19] Qi, Z., Cai, Y., & Zhou, Q. (2014, October). Accurate prediction of detailed routing congestion using supervised data learning. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)* (pp. 97-103). IEEE. https://doi.org/10.1109/ICCD.2014.6974668.

[20] Kahng, A. B. (2024, March). Solvers, engines, tools and flows: The next wave for AI/ML in physical design. In *Proceedings of the 2024 International Symposium on Physical Design* (pp. 117-124). https://doi.org/10.1145/3626184.3635277.

[21] Rapp, M., Amrouch, H., Lin, Y., Yu, B., Pan, D. Z., Wolf, M., & Henkel, J. (2021). 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), 3162-3181. https://doi.org/10.1109/TCAD.2021.3124762.

[22] Mai, J., Wang, J., Di, Z., Luo, G., Liang, Y., & Lin, Y. (2023, October). OpenPARF: An open-source placement and routing framework for large-scale heterogeneous FPGAs with deep learning toolkit. In *2023 IEEE 15th International Conference on ASIC (ASICON)* (pp. 1-4). IEEE. https://doi.org/10.1109/ASICON58565.2023.10396248.

[23] Lin, Y., Dhar, S., Li, W., Ren, H., Khailany, B., & Pan, D. Z. (2019, June). Dreamplace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In *Proceedings of the 56th Annual Design Automation Conference 2019* (pp. 1-6). https://doi.org/10.1145/3316781.3317803.

[24] Trung, K. L., & Hy, T. S. (2025). DiffPlace: A conditional diffusion framework for simultaneous VLSI placement beyond sequential paradigms. *arXiv preprint arXiv:2510.15897*.

[25] Agnesina, A., Rajvanshi, P., Yang, T., Pradipta, G., Jiao, A., Keller, B., ... & Ren, H. (2023, March). AutoDMP: Automated dreamplace-based macro placement. In *Proceedings of the 2023 International Symposium on Physical Design* (pp. 149-157). https://doi.org/10.1145/3569052.3578923.