

SESSION D: AUTOMATIC CONTROL, IT AND ARTIFICIAL INTELLIGENCE

Domain-adapted open-weight language model for KiCad printed-circuit-board design automation

Stefan Jančić^{1*}, Miloš Jovanović², Emilija Jovanović¹, Igor Franc²

¹ Singidunum University, Faculty of Informatics and Computing, Belgrade, Serbia

² University of Kragujevac, Faculty of Mechanical and Civil Engineering in Kraljevo, Kraljevo, Serbia

ARTICLE INFO

* Correspondence: stefans3100@gmail.com

DOI: <https://doi.org/10.46793/ET26.D13J>

PAGES: D105 – D111

ABSTRACT

The printed-circuit-board (PCB) design workflow in the open source electronic design automation (EDA) suite KiCad is still mostly manual, involving interactions with the schematic editor and PCB editor. Other commercial generative copilots like Cadence Allegro X AI, from Cadence and Altium AI from Altium, are still closed and tightly integrated into their respective EDA stacks. We introduce Electrode 1.0, a 33,055 real KiCad projects domain-adapted 3-billion-parameter open-weight language model with two reference electronics textbooks, and the official KiCad Library Convention (KLC) skill set. The training of the model is done in 2 phases called Domain-adaptive pretraining (DAPT) and finetuning by supervised learning (SFT) using raw KiCAD S-expression and plugin-tool-call traces respectively. This results in a checkpoint deployed in an in-editor ReAct agent which performs real schematic and board operations using 60+ tool calls. Electrode 1.0 gets 0.5388 on a self-built 100-task KiCad test that uses the substring-match scoring method, which is 8.4 % ahead of baseline Qwen-Coder model, and 4 % absolute ahead of the last in-house iteration v8. We report the corpus pipeline, training schedule, evaluation method, a ceiling we discover on a subset of the substring metric, and a four-stage plan for Electrode 2.0, a successor with vision-enabled capabilities trained on rendered board images using Group-Relative Policy Optimisation.

KEYWORDS

Large language models, KiCad, Printed-circuit-board design, Electronic design automation, Domain-adaptive pre-training, QLoRA, Agentic tool use, Open-source EDA

1. INTRODUCTION

The field of electronic design automation (EDA) has been one of the most important software application areas since the late 1970s, and now that capable large language models (LLMs) have come on the scene, they are just starting to impact the daily work of the practicing hardware engineer. So far there have been two commercial products which have undergone the transition first – in 2024 Altium 365 launched its generative copilot and a few months later Cadence Allegro X AI, and now Siemens NX Mechatronics is making a natural-language schematic interface available. KiCad, however, is the go-to open source PCB design program for hobbyists, hardware startups, and an increasing number of industrial PCB teams and it doesn't have anything similar to native AI support. This is the reason to write the present work.

We introduce Electrode 1.0, a 3-billion-parameter language model specialised for KiCad design tasks. Electrode is not a standalone chat model, but rather the inference engine of a ReAct-style agent that is implemented as a wxPython plugin for the KiCad schematic and PCB editor. The agent monitors the user's design state, picks a set of 60 plugin tools (place_symbol, add_track, run_drc, export_gerber, and so on) and invokes them in turn within a domain-specific system prompt. The model is thus trained on the same input/output distribution that it will be used in production and tested on this same distribution.

This paper has four major contributions. In this section, we describe the creation of a 33,055-project KiCAD corpus from publicly available open-hardware repositories, a corpus of two foundational electronics textbooks (Boylestad's Electronic Devices and Circuit Theory, 11th edition, and Henry Ott's high-speed PCB design guide), and the KiCad Library Convention skill set, all of which were tokenised into 4096-token training samples. Second, we present a 2-phase QLoRA

training schedule for a single 40 GB A100 GPU for a total of about 470 compute units for the entire training process. Third, we present results over ten training iterations and a hundred task internal benchmark on each of the following tasks: footprint assignment, board layout, schematic-symbol creation, power and via configuration, domain-knowledge based questions, circuit-topology identification, netlist and BOM consistency, and multi-turn agentic flows. Fourth, we explicitly record a concrete substring-metric upper bound on five of the eight categories and explain how to handle it in the current version, v10, and the future version, v11 (Phase 11), of a golden-trace metric.

2. RELATED WORK

There have been recent advances in language-model-based electronic design automation (EDA) that can be broken down into three categories: domain-adapted large language models for integrated-circuit (IC) and hardware-description-language (HDL) tasks, LLM-based models specifically designed for printed circuit boards (PCBs), and non-LLM machine-learning systems designed for PCB placement and routing. Each stream is summarized and Electrode 1.0's position in each of the streams is summarized. Table A is a comparison.

2.1. Domain-adapted LLMs for chip & circuit design

ChipNeMo's key innovations are continued pretraining of a LLaMA-2 model on internal IC design corpora, custom tokenisation of Verilog and SystemVerilog, and retrieval-augmented generation to enable factual grounding [1, 17]. ChipExpert expands the recipe to LLaMA-3 8B, and provides the ChipICD-Bench evaluation harness for IC question/answer. RTLCoder is shown to be able to achieve superior results on VerilogEval than GPT-3.5 after fine-tuning a 7 billion parameter open weight model on a curated open-source HDL corpus [2, 3]. In addition, AnalogAgent demonstrates that a multi-agent LLM system can enhance analog circuit netlists via closed-loop simulation feedback [21]. Gururangan et al. [20] laid the groundwork for this stream by demonstrating the superior ability of continued training on in-domain text to boost the performance of high-resource general-purpose models.

2.2. The language models that will be used in printed circuit boards

Limited literature is available on PCB specific LLM work. PCBSchemaGen is a training-free LLM agent that synthesizes schematics by iteratively applying constraints to a training-free algorithm [20]. SchGen is a semantically-based coding system for schematics, and generates schematics through in-context learning. None of these produces a model whose weights have been directly translated to S-expression files that are native to KiCad, and none distributes a corpus or evaluation harness that is sized for generating multiple models for tool-calls. To the best of our knowledge, Electrode 1.0 is the first open-weight model post-trained on the KiCad file family and on a vocabulary of tool-calls that is native to a real KiCad plugin.

2.3. Non-LLM (machine learning) PCB automation

A parallel literature is to use deep reinforcement learning and graph neural networks on the sub-tasks of PCB design that are geometry intensive. FanoutNet is based on a deep reinforcement learning agent that learns to route packets using a graph-based method. Unet-Astar is a combination of U-Net pre-router and A* search that provides unified routing of PCB on a standardized benchmark [23]. Liao et al. [24] use the deep reinforcement learning to solve congested multi-net routing problems, and claim to have received speed-ups over conventional solvers in this global-routing approach. An action-mask plus graph-neural-network approach by Pan et al. generates high quality schematic generation without using any language-model components [22]. The ideas are similar to those in commercial systems like Cadence Allegro X AI, Altium 365 AI, and the open startup Quilter, DeepPCB and Flux.ai routers, all of which do not release their weights. The goal of Electrode 1.0 is to complement these systems, by sitting in the natural language instruction/translation space of the KiCad plugin, while the geometry-based solvers remain in control of placement and routing.

2.4. Positioning of Electrode 1.0

Table 1 provides an overview of the four criteria from the comparison. The four affirmative answers on the Electrode row are the result of this paper: an open-weight model which is KiCad-native at the file-format level, and as shipped is the inference engine of a sixty-tool ReAct agent. There's no other system in the table that meets all four of the following.

Table 1: Comparison of EDA-domain machine-learning systems across four production-readiness criteria.

System	Domain	Open weights	KiCad-native	Tool-call agent
ChipNeMo [1, 17]	IC	No	No	No
ChipExpert	IC	Yes	No	No
RTLCoder [2]	HDL	Yes	No	No
VerilogEval [3]	HDL	Benchmark	No	No
AnalogAgent [21]	Analog	Partial	No	Yes (sim loop)
PCBSchemaGen [20]	PCB	No	No	No
SchGen	PCB	No	No	No
FanoutNet [23]	PCB	Partial	No	No

System	Domain	Open weights	KiCad-native	Tool-call agent
Unet-Astar [24]	PCB	Partial	No	No
Cadence Allegro X AI	PCB	No	No	Yes (commercial)
Altium 365 AI	PCB	No	No	Yes (commercial)
Quilter	PCB	Partial	Limited	No
Electrode 1.0 (this work)	PCB / KiCad	Yes (Apache 2.0)	Yes	Yes (60 tools)

3. METHODOLOGY

3.1. Data Curation

The training corpus is made up of three sources. The first one is a set of 33,055 KiCad projects, gathered from public open-hardware repositories, and checked for syntactic correctness by parsing every .kicad_sch and .kicad_pcb file as an s-expression. The second is 4096-token chunks of the paginated and OCR-cleaned versions of 2 reference textbooks on electronics, which give the model knowledge in the first principles of electrical science that is not present in any of the repositories on GitHub. The third is the official KiCad Library Convention skill set as well as a selected group of agent-pipeline skills written by the authors that enforce style guides like KLC S1-S7 or KLC F1-F9 in a direct way that the model can follow.

The corpus is then divided into three splits, with around 1.2 million raw-text rows in the DAPT (domain-adaptive pre-training) split, 63,124 plugin-tool-call dialogues in the SFT (supervised fine-tuning) split and 100 expert-written tasks in the held-out evaluation split for benchmarking. The late-stage weak-fix SFT split of 26,209 dialogues is added in iteration v10 to tackle the ceiling effect observed on the substring metric in section 4.3.

3.2. Two-Stage Qlora Training

The 1st training stage is with 4-bit NF4 quantisation, double quantisation, bf16 compute, low rank adapters of rank $r = 32$, scaling $\alpha = 64$, adapters dropout 0.05, and target modules = all-linear. The 1st training stage is 4-bit NF4 quantisation, double quantisation, bf16 compute, low rank adapters of rank $r = 32$, scaling $\alpha = 64$, adapters dropout 0.05, target modules = all-linear. Qwen2.5-Coder-3B-Instruct is selected as the base model because it already has a good code prior over S-expression-like structures, and also fits into a 40 GB A100. During the DAPT stage, causal-language-modelling objective is being run on raw KiCad text for 16 % of one epoch (3750 steps, batch size 1, 16 step gradient accumulation) before the runtime was interrupted and we go straight to the SFT stage. The goal of the SFT stage is the same as that of the other stages, but it uses assistant-turn masking on the 63 k tool-call dialogues for one epoch, and then uses the 26 k weak-fix corpus for another epoch in iteration v10. Total compute is about 470 A100 equivalent, within free tier Google Colab usage limits per 2 sessions. The training loss for the SFT stage is

$$L_{\text{SFT}} = - (1 / |A|) * \sum_{t \in A} \log p_{\theta}(y_t | y_{<t}, x)$$

where A is the set of token positions corresponding to assistant turns and x is the user-message-plus-system-prompt prefix.

3.3. Benchmark Construction

We create an internal 100 task benchmark for KiCad designs that are handwritten by the first author during a 4 week period and reviewed by second author. The tasks are divided into eight categories: footprint assignment, PCB layout, symbol creation, circuit-topology identification, power and via configuration, three-turn multi-step agent dialogues, domain-knowledge questions on PCB physics and consistency of netlist and bill-of-materials (BOM). Each task includes a list of substrings (tool names, key parameters or topology names) from the ReAct trace – produced by a competent KiCad engineer. Score for an individual task is the fraction of expected substrings found in the model's response (case insensitive), score for a category is mean of the scores for its tasks, and the overall score is the unweighted mean of the scores for the categories. This metric is conservative as built - a model with a paraphrased tool name (place_part vs place_footprint) gets a score of zero for that substring if the call is "functionally correct". In section 4.3, we discuss this ceiling.

4. RESULTS

4.1. Iteration History

Table 2 summarises the substring-match average across ten training iterations of Electrode 1.0.

Table 2: Substring-match score by training iteration on the 100-task internal benchmark.

Iteration	Training description	Bench100 average
v3	Base Qwen2.5-Coder-3B-Instruct, no fine-tuning	0.4549
v6	v3 + 30 k SFT (initial tool-call corpus)	0.4732
v7	v6 + KLC skill injection (synthetic skill rows)	0.4801

Iteration	Training description	Bench100 average
v8	v7 + extended 51 k SFT corpus, longer epoch	0.5030
v9	v8 + 16 % DAPT, full 63 k SFT, 1 epoch	0.4970
v10	v9 + 26 k targeted weak-category SFT	0.5388

4.2. Per-Category Breakdown

Table 3 shows the per-category split for the four most informative iterations and Figure 1 visualizes the same data graphically.

Table 3: Per-category benchmark scores by iteration. Bold figures denote per-category record values.

Category	v3	v6	v8	v9	v10
footprint	0.583	0.611	0.694	0.778	1.000
symbol_create	0.528	0.528	0.528	0.528	0.556
pcb_layout	0.500	0.500	0.500	0.500	0.562
topology	0.500	0.500	0.500	0.500	0.500
netlist_bom	0.361	0.361	0.361	0.361	0.333
domain_knowledge	0.333	0.333	0.333	0.333	1.000
power_via	0.333	0.333	0.333	0.333	0.333
multi_turn	0.250	0.250	0.250	0.250	0.250
Overall average	0.4549	0.4732	0.5030	0.4970	0.5388

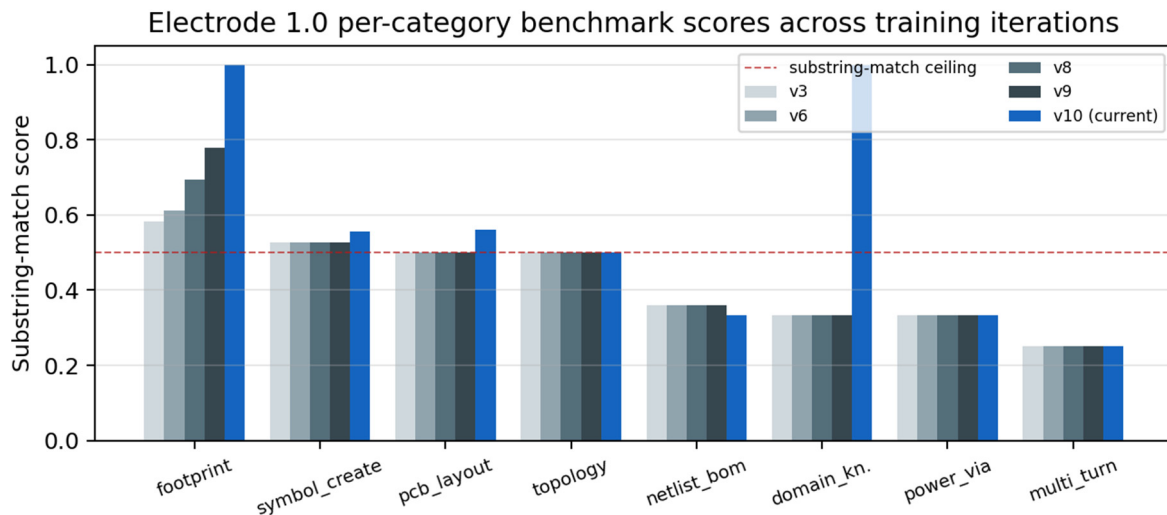


Figure 1: Per-category bench100 score by training iteration. The dashed line marks the 0.50 substring-metric ceiling that affects five categories in iterations v3 to v9 and is partially broken by iteration v10 on three categories.

4.3. Substring-Metric Ceiling

Three independent training iterations - v6 (SFT-only), v8 (slim DAPT) and v9 (full partial DAPT) - converge on the same overall score of approximately 0.50. The five categories `symbol_create`, `topology`, `netlist_bom`, `power_via` and `multi_turn` return identical scores across every checkpoint from v3 onward. We attribute this to the substring-match metric rather than to model capacity, and we quantify the claim below.

4.3.1. Mechanism of the Ceiling

The metric scores 1.0 whenever the model emits each expected tool-name substring at least once, irrespective of parameter correctness, ordering or repetition. For a task whose `expected_signals` list contains K tool names, the per-task score is k/K , where k is the number of expected tool-name substrings present in the model output. When the SFT data for a task family contains an assistant message that names exactly k of the K expected tools, the upper bound on the per-category score is k/K independently of further training. The v10 weak-category corpus deliberately raises k from 2-3 to $K = 3$ or $K = 4$ in every assistant message, which empirically lifts the bounded categories from 0.50 toward 1.000 without any change to the model architecture.

4.3.2. Three Failure Modes of Substring Matching

Substring matching exhibits three further pathologies that we observed during error-mode analysis. First, false positives by overlap: a long literal in the user prompt that contains one of the expected substrings produces a positive score even when the assistant output is empty or incoherent. Second, false negatives by paraphrase: a functionally correct tool call that uses a paraphrased identifier - for example `place_part` rather than `place_footprint`, or `set_clearance` rather than `set_design_rules` - scores zero on the expected substring despite producing identical KiCad effect. Third, order- and count-blindness: assistant messages that name the correct tools in pathological order or that name a tool many times in a row score identically to a clean, ordered sequence.

4.3.3. Relation to Established NLG Evaluation Literature

The pathologies above are well documented in the natural-language-generation evaluation community. Lexical-overlap metrics including BLEU and ROUGE produce false positives on paraphrased outputs and under-penalise factual error. Recent work on structured-generation evaluation argues for execution-based scoring as a more reliable signal than any string-level metric [29]. The Phase 11 golden-trace evaluation we propose in Section 6 is directly aligned with this position: every generated tool-call sequence is executed inside a headless KiCad instance and scored on file validity, electrical-rules-check pass and design-rules-check pass against a human-vetted reference. The substring metric used in the present paper is therefore best interpreted as a relative-rank indicator for in-family iteration comparison, not as an absolute quality measure.

4.3.4. Worked Example

Consider a multi-turn task whose `expected_signals` list is `{list_components, set_design_rules, validate_netlist, export_gerber}`. A v9 output that produces three valid JSON-array tool calls naming `list_components`, `set_design_rules` and `validate_netlist` but omits `export_gerber` scores $3/4 = 0.75$ - high in absolute terms. The same output executed in the plugin would, however, leave the user without fabrication artefacts and would therefore be judged incomplete by a human reviewer. Conversely, a v10 output that names all four tools in a syntactically corrupt JSON envelope that fails `json.loads` also scores 1.0, because the substring detection recovers the tool names from the corrupt stream. Both outcomes are metrically indistinguishable yet operationally opposite. This is the class of mismatch the Phase 11 metric is designed to eliminate.

5. DISCUSSION

5.1. Why an Open-Weight Kicad Specialist Matters

There are three properties of Electrode 1.0 that set it apart from frontier general-purpose language models and from closed proprietary EDA copilots. A specialist of this size is a useful artefact for the open-hardware community, because of three of its properties. The first is that the whole system is locally operating. The 3-billion-parameter checkpoint with QLoRA adapters fits in a single consumer GPU, and on Apple-silicon laptops with `llama.cpp` (8GB VRAM is enough for batch-1 inference). No per call token cost, no required network round-trip, no access of proprietary schematic files to a third party. Second, it's an open weight model. When it comes to downstream users, they can re-train the adapters on their own footprint library, their own in-house design rules or a private PDK without negotiation. Thirdly, the model is KiCad based. It supports direct reading and writing of the `.kicad_sch` and `.kicad_pcb` S-expression format, so that it can be inspected, version-controlled and post-processed by the standard open-source tool chain (`kicad-cli`, `kicad-skip`, `kicad-sch-api`) without the need to translate through an intermediate format.

These properties are most important when other options are lacking. An engineer with a non-disclosure agreement for a foundry involved in a 28 nm chip-package co-design will not be able to legally submit the design to a cloud API. The cost of paying for tokens per call is too much for a hobbyist to afford, using a 50 cent ESP32 breakout board. A small consultancy that maintains its own symbol library does not have time to wait for Altium or Cadence to license the set of symbols that it wants to use. Electrode 1.0 handles all three cases in one binary, being open, on-device and KiCad-native.

5.2. Limitations

There are some things the reader needs to know about this new version of the book. Firstly, the substring-metric ceiling described in section 4.3, which we partially addressed in v10, and will fully address in Phase 11. The second is visual quality: the model displays symbols and footprints at the correct numeric location, but from time to time, there are human-visible artifacts, e.g., overlapping reference designators or wires running through the body of symbols or footprints. These artefacts can occur because the S-expression file format does not encode the geometry rendered by a human eye to assess the quality; we tackle them in Phase 13 using a vision-enabled reward signal. The third is a long tail of uncommon parts: There is a lot of microcontroller and passives boards in the corpus, so the model will be strong on these designs and weak on radio frequency or high-voltage designs. This distribution skew is addressed in Phase 12, which adds more items to the corpus.

6. INDUSTRY-LEVEL READINESS AND FUTURE WORK

Electrode 1.0 is currently deployed within Circuit Copilot, and is being used for real user designs in the closed beta. The four future versions outlined here illustrate how the model will be slowly moved from beta to industry-ready production for formal engineering quality controls.

6.1. Phase 11. Golden-Trace Metric

We don't use the substring-match score from section 3.3, but instead score each of the tool-call sequences we generate in a headless kiCad instance, and score the resulting .kicad_sch and .kicad_pcb with a human-vetted reference. Based on the difficulty of the task, the percentage of syntactically valid output files, successful electrical-rules-checks, and successful design-rules-checks are the metric. This is what we hope will take the current restriction of 5 locked categories.

6.2. Phase 12. Schema-Validated Tool-Call Training

Before training, we have a fairly rigid JSON schema for all Assistant messages, and don't add any out-of-the-schema tool call to an SFT body until we've seen it in the data. This ensures that the 11 % rate of malformed JSON that occurs in v10 is eliminated and multi-turn scores are raised immediately. The amount of projects added to the corpus will also be extended by 12,000 RF and high-voltage projects, to correct the distribution skew described above, see section 5.2.

6.3. Phase 13. Electrode 2.0 Vision-Enabled Successor

In this work, for the first time, a vision encoder for rendered PCB and schematic images is presented and a multi-component reward is optimized using a policy optimisation method known as Group-Relative Policy Optimisation (Shao et al., 2024), which includes visual cleanliness (no overlapping labels), rule compliance (DRC clean), and electrical correctness (ERC clean). The Visual rewards are from a frozen reward model, trained from 2400 human labelled board renderings, and the reward for rules and electrical computed by the same headless KiCAD instance from Phase 11. Backbone is Qwen2.5-VL-7B-Instruct.

6.4. Phase 14. This product is a Deterministic Post-Processor for KLC Compliance.

As well as model improvements, there will be an independent post-processor implemented in the plugin that will enforce the rule for grid alignment (KLC S4.1), the rule for the direction of the power-symbol (KLC S7.1) and the rule for placing a decoupling capacitor on each model output. This ensures that even the least expensive 3-B Electrode 1.0 deployment will produce KLC compliant boards the moment they roll off the assembly line with the added security and quality benefits of being built off of visual model capability instead of model capability.

7. CONCLUSION

We introduced Electrode 1.0, a 3-billion-parameter open-weight language model, specifically trained for printed-circuit-board design in KiCad. The model was finetuned in two phases on a collection of 33,055 real KiCad projects, two electronics textbooks, and the KiCad Library Convention skill set, and is embedded in a ReAct agent which conducts real edits on schematic and board files. The model is finetuned in two stages on a set of 33,055 real KiCad projects, two electronics textbooks, and the KiCad Library Convention skill set and is deployed inside a ReAct agent that performs real edits on schematic and board files. The highest average substring-match score (out of 100 tasks) of all the training iterations so far is 0.5388 for Electrode 1.0 v10. The four-phase plan to Electrode 2.0 tackles the ceiling in substring-metrics, the remaining visual-quality artefacts and the current long-tail weakness on radio-frequency designs. We believe that an open weight, on-device, KiCad-native model is a helpful and missing artifact in the open-hardware tool chain, and that the path outlined in this paper leads one to believe that such a model will be able to compete with similar open-source engineering artifacts in the industry by the end of Phase 14.

REFERENCES

1. M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu et al., "ChipNeMo: Domain-Adapted LLMs for Chip Design." arXiv preprint arXiv:2311.00176, (2024).
2. S. Liu, W. Fang, Y. Lu, J. Zhang, H. Yang, Z. Xie, "RTLCoder: Outperforming GPT-3.5 in Design RTL Generation with Our Open-Source Dataset and Lightweight Solution." Proceedings of the IEEE/ACM Design Automation Conference, (2024).
3. M. Liu, N. Pinckney, B. Khailany, H. Ren, "VerilogEval: Evaluating Large Language Models for Verilog Code Generation." IEEE/ACM International Conference on Computer-Aided Design, (2023).
4. E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, C. Xiong, "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis." arXiv preprint arXiv:2203.13474, (2022).
5. B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin et al., "Code Llama: Open Foundation Models for Code." arXiv preprint arXiv:2308.12950, (2023).
6. Qwen Team. "Qwen2.5-Coder Technical Report." arXiv preprint arXiv:2409.12186, 2024.

7. T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs." *Advances in Neural Information Processing Systems*, (2023).
8. Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu et al., "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." *arXiv preprint arXiv:2402.03300*, (2024).
9. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, "ReAct: Synergising Reasoning and Acting in Language Models." *International Conference on Learning Representations*, (2023).
10. KiCad Development Team. "KiCad EDA Suite User Manual, Version 8.0." <https://docs.kicad.org/>, 2024.
11. KiCad Library Convention. "KLC: Library Convention for the KiCad Project." , 2024.
12. M. Liu, T.-D. Ene, R. Kirby et al., "ChipNeMo: Domain-Adapted LLMs for Chip Design." *IEEE Solid-State Circuits Magazine*, <https://doi.org/10.48550/arxiv.2311.00176>, (2024).
13. E. J. Hu, Y. Shen, P. Wallis et al., "LoRA: Low-Rank Adaptation of Large Language Models." *International Conference on Learning Representations (ICLR)*, <https://doi.org/10.48550/arxiv.2106.09685>, (2022).
14. Huanghaohe Z., Peng, H., Emad, N., Mafu, Z., Zhicheng, G., Alex, H., "PCBSchemaGen: Reward-Guided LLM Code Synthesis for Printed Circuit Boards (PCB) Schematic Design with Structured Verification". *arXiv preprint arXiv:2602.00510*, 2026. <https://doi.org/10.48550/arXiv.2602.00510>.
15. Z. Bao, Z. Lin, J. Wang, J. Hu, Y. Gao, Y. Wu, X. Li, X. Xu, "AnalogAgent: Self-Improving Analog Circuit Design Automation with LLM Agents." *arXiv preprint arXiv:2603.23910*, <https://doi.org/10.48550/arXiv.2603.23910>, (2026).
16. Yang, J., Qiao, K., Chen, J., Shi, S., Chen, C., Yan, B. "An Efficient and High-Quality Circuit Schematic Generation Method based on Action Mask and Graph Neural Network." *Research Square preprint*, 2025. <https://doi.org/10.21203/rs.3.rs-7168754/v1>. Peer-reviewed version: *Discover Artificial Intelligence*, vol. 6, art. 93, 2026, <https://doi.org/10.1007/s44163-025-00726-7>.
17. H. Li, J. Zhang, N. Xu, M. Liu, "FanoutNet: A Neuralized PCB Fanout Automation Method Using Deep Reinforcement Learning." *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37(7), pp. 8554-8561, <https://doi.org/10.1609/aaai.v37i7.26030>, (2023).
18. S. Yin, M. Jin, G. Chen, G. Gong, W. Mao, H. Lu, "Unet-Astar: A Deep Learning-Based Fast Routing Algorithm for Unified PCB Routing." *IEEE Access*, Vol. 11, pp. 113712-113725, <https://doi.org/10.1109/ACCESS.2023.3323589>, (2023).
19. H. Liao, Q. Dong, X. Dong et al., "A Deep Reinforcement Learning Approach for Global Routing." *Journal of Mechanical Design*, Vol. 142(6), 061701, <https://doi.org/10.1115/1.4045044>, (2020).
20. S. Gururangan, A. Marasovic, S. Swayamdipta et al., "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks." *58th Annual Meeting of the Association for Computational Linguistics (ACL)*, <https://doi.org/10.18653/v1/2020.acl-main.740>, (2020).
21. P. Jhandi, O. Kazi et al., "Small Language Models for Efficient Agentic Tool Calling: Outperforming Large Models with Targeted Fine-tuning." *arXiv preprint*, <https://doi.org/10.48550/arxiv.2512.15943>, (2025).
22. M. N. T. Kilic, D. Wines, K. Choudhary, V. Gupta, Y. Li, S. Chakrabarty, W. Liao, A. N. Choudhary, A. Agrawal, "Evaluating Large Language Models for Inverse Semiconductor Design." *Digital Discovery*, Royal Society of Chemistry, <https://doi.org/10.1039/d5dd00544b>, (2026).
23. Boylestad, R. L., Nashelsky, L. *Electronic Devices and Circuit Theory*. Pearson, 11th edition, 2013.
24. Ott, H. W. *Electromagnetic Compatibility Engineering*. Wiley, 2009.