International Journal of Web of Multidisciplinary Studies
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
website: *https://ijwos.com* OPEN ACCESS E-ISSN : 3049-2424
Volume 03, Issue 10,October 2026 DOI: https://doi.org/10.71366/ijwos03102693971

FlowCAD: A Verification-Gated, Library-Grounded Framework for Natural Language-Driven PCB Design Automation

Prof. SAGAR BIRJE[1], SHIVALINGSARJ DESAI[2], NIVEDITA AKKIMARADI[3], MAITHRI H M[4], SIMRAN MAMDAPUR[5]

[1,2,3,4,5] *Department of Artificial Intelligence and Data Science, Angadi Institute of Technology and Management, Savagaon Road, Belagavi 590009, Karnataka, India*

Article Info

Article History:
Published: 31 Oct 2026

Publication Issue:
Volume 03, Issue 10
October 2026

Page Number:
642-653

Corresponding Author:
Prof. SAGAR BIRJE

Abstract:

Designing a printed circuit board (PCB) from a written description involves several connected tasks. An engineer must interpret the requirements, choose suitable components, prepare the schematic, arrange the components, route the connections, check the design rules and generate the files needed for fabrication. These tasks are usually handled across a number of engineering tools and still require substantial manual work. This paper presents FlowCAD, a prototype that explores how natural-language input can be connected to parts of this workflow. The system uses a language model to interpret requirements, suggest a functional architecture and choose from components retrieved from an index containing more than 22,000 KiCad library entries. The remaining stages use structured and deterministic processes: SKiDL is used to create the netlist, KiCad's pcbnew API is used for rule-based placement, and a verification stage checks library references, netlist information and board design rules. The prototype can also export Gerber X2, drill, bill-of-materials and GLB files. There are important limits to its current state. Routing is only partially implemented, and the browser-based 3D viewer is still being developed. Generating fabrication-format files should therefore not be confused with producing a fully routed, functionally verified or fabrication-ready PCB. We also describe an evaluation plan covering three circuit-complexity levels, repeated runs, stage-level measurements, baseline comparisons, ablation studies and tests using deliberately introduced faults. Those experiments have not yet been carried out. No quantitative performance results are claimed in this paper.

Keywords: Electronic Design Automation, Printed Circuit Boards, Large Language Models, Netlist Generation, Design Rule Checking, KiCad, Human-in-the-Loop Design.

1. INTRODUCTION

A printed circuit board provides the electrical and mechanical foundation for most electronic products. Turning an idea for an electronic device into a working PCB is not a single design task. It involves understanding what the device must do, selecting components that can perform those functions, connecting the components in a schematic, arranging them on the board, routing the copper traces and checking the design before fabrication. Tools such as KiCad, Altium Designer and Cadence Allegro support much of this work [1]–[3]. Even with these tools, engineers still need to make many decisions and move information between stages. This can be challenging for students and small hardware teams, and it can also slow down early design iterations for experienced engineers.

Recent progress in large language models (LLMs) has created interest in describing electronic circuits in ordinary language and using software to turn those descriptions into designs. Several research projects have explored this idea, mainly at the schematic-generation stage. For example, pcbGPT reports pass@1 scores of 0.90 on a 20-task embedded-schematic benchmark and 0.72 on its hard tasks [4]. SchGen reports an 82% valid-circuit rate and 60.5% expert-verified functional correctness for its code-style representation; the corresponding valid-circuit rate for raw KiCad-

Copyright © 2026 The Author(s) : This is an open access article under the CC BY license (http://creativecommons.org/licenses/by-nc/4.0/)

file generation was 32% [5]. PCBSchemaGen combines language-model code generation with structured verification across 23 circuits [6]. These results are useful points of comparison, but the systems are primarily concerned with schematic generation. PCBSchemaGen also includes a layout-file demonstration, so it would be inaccurate to say that it has no layout capability. Commercial products address other parts of the process, including natural-language schematic capture and placement or routing of an existing design [7]–[10].

A central challenge is that language models generate likely responses, while PCB design must satisfy precise electrical and geometric constraints. A board can look convincing on screen and still contain incorrect connections or rule violations. Work such as PCBWorld shows the value of allowing design agents to interact with the KiCad engine and use its design-rule feedback [11]. FlowCAD follows a similar practical principle: the language model helps propose a design, but software tools perform the checks. These checks can reveal specific classes of errors; they should not, however, be treated as proof that a circuit will work as intended.

FlowCAD is a prototype that takes a natural-language description and uses it to create a placed KiCad board, together with several design and fabrication-format outputs. The work described here makes five contributions:

(C1) A verification-gated workflow that keeps language-model generation separate from deterministic synthesis, placement and checking. The system is designed to run its checks after generation and after each supported edit.

(C2) A component-selection process that restricts the model to candidates retrieved from a KiCad library index. Library grounding has also been explored in pcbGPT [4], so we do not present it as a new idea on its own. The contribution here is its use within a wider workflow that includes board placement, board-level DRC, export and re-verification after supported edits.

(C3) Programmatic netlist generation using SKiDL, rule-based placement through KiCad's pcbnew API, and a verification stage combining library-integrity checks, static netlist analysis and KiCad DRC.

(C4) A conversational editing interface that recognises a limited set of resize, move, replace and add commands. These edits are applied through pattern matching and are followed by another verification run, without requiring another LLM call.

(C5) An evaluation plan with defined metrics, baselines, ablation studies and a seeded-fault test for the verifier. The paper also distinguishes clearly between features that are implemented, partly implemented and still in development.

The rest of the paper is organised as follows. Section 2 discusses related work, Section 3 describes the problem and design principles, and Sections 4 and 5 explain the architecture and implementation. Section 6 sets out the evaluation plan. Sections 7–9 discuss the current prototype, its limitations and the conclusions that can be drawn at this stage.

2. RELATED WORK

2.1 LLM-Based Schematic and Netlist Generation

pcbGPT converts natural-language requirements into editable KiCad schematics using a Python circuit DSL, component-library search, datasheet-based knowledge extraction and several validation steps [4]. SchGen uses a code representation designed to preserve circuit meaning, including relative component placement and pin-name-based wiring [5]. PCBSchemaGen combines LLM-based code generation with reward-guided structured verification [6]. These projects provide relevant foundations for language-driven circuit design. Their main focus is schematic generation; FlowCAD explores what happens when the workflow continues into board placement, DRC, fabrication-format export and repeated checks after supported conversational edits.

2.2 Engine-Grounded Agents

ChatEDA uses an LLM agent to coordinate existing electronic-design-automation tools [13]. PCBWorld provides an environment in which agents can operate on KiCad designs and receive feedback from the design-rule engine [11]. These systems demonstrate the value of connecting language-model agents to established engineering software instead of asking a model to produce every design artefact directly. PCBWorld is centred on board-level tasks rather than the full path from a written requirement to a board. Research on placement and routing benchmarks also points to the difficulty of geometric reasoning and preserving electrical constraints. These observations support FlowCAD's decision to use native EDA tools for operations that require precise geometry and rule checking.

2.3 Commercial and Open-Source Tools

Flux provides a browser-based electronic-design environment with an AI assistant [8]. ProtoFlow and other commercial products describe natural-language or requirements-driven schematic workflows [7], [14]–[16]. Quilter and DeepPCB focus more directly on placement or routing for designs that already exist [9], [10]. The capability descriptions in this comparison are taken from public product documentation. They are not the result of independent testing, and some details may change as products are updated. Any uncertain entry in Table 1 should therefore be checked against the relevant provider's current documentation before publication.

Table 1. Capabilities of related systems as stated in public documentation (mid-2026); not independently benchmarked. Reqs. = structured requirements as input.

System	NL input	Schematic	Place / route	Verification	Mfg. output
Flux [8]	Yes	Copilot	AI-assisted	Partial	Yes
ProtoFlow [7]	Yes	Yes	Manual	ERC / DRC	KiCad export
Quilter [9]	No	No	Autonomous	Physics-based	Yes
DeepPCB [10]	No	No	Autonomous	Physics-based	Yes
Circuit Mind [15]	Reqs.	Yes	Limited	Partial	BOM
CELUS [16]	Reqs.	Yes	Early stage	Partial	Export
pcbGPT [4]	Yes	Yes (KiCad)	No	Validation	No
SchGen [5]	Yes	Yes	No	Evaluation only	No
PCBSchemaGen [6]	Yes	Yes	No	Structured	No
FlowCAD	**Yes**	**Yes (SKiDL)**	**Placement; routing partial**	**DRC + static checks**	**Gerber X2, BOM, 3D**

2.4 Gap

The sources reviewed for this paper describe a range of tools: some generate schematics from written requirements, while others focus on placement or routing once a design already exists. Within the material reviewed, we did not find independent validation of one system that combines natural-language input, library-grounded component selection, board placement, deterministic checks after supported edits and fabrication-format export in a single workflow. This is a conclusion limited to the sources examined; it is not a claim that FlowCAD is the first system of its kind. FlowCAD explores this combination, but its routing, functional validation and fabrication-readiness capabilities are not yet complete.

3. PROBLEM FORMULATION AND DESIGN PRINCIPLES

We represent the user's natural-language prompt by p and the available component-library index by L. The index contains the symbol and footprint entries that FlowCAD can retrieve and use. A design is represented as D = (S, N, B), where S is the structured requirement specification, N is the netlist built from parts in L, and B is the PCB layout. A deterministic validator V examines the design and assigns one of three outcomes—pass, warning or fail—to each of K check categories. The objective is to generate a design D from p that satisfies the conditions expressed in:

$$max\ Pr[V(D)\ contains\ no\ fail]\ \ subject\ to\ \ parts(N)\ \subseteq\ L \quad (1)$$

Since some stages use a language model, the same prompt may not produce exactly the same output every time. The probability in (1) should therefore be estimated by running the system repeatedly, as described in Section 6. This leads to five design principles.

P1. Keep generation and checking separate. The language model can suggest design information, but it does not decide whether the design is valid. That decision is based on the reports produced by deterministic checks.

P2. Select components from an actual library. FlowCAD restricts component choices to entries available in L so that the corresponding KiCad symbols and footprints can be resolved.

P3. Pass information through structured interfaces. Each stage exchanges schema-validated JSON or standard KiCad files. This makes it easier to inspect intermediate results and test individual stages.

P4. Use a code-oriented intermediate representation. The system generates SKiDL code from structured JSON rather than asking the language model to write raw KiCad files, which are a less suitable generation target [5].

P5. Check the design again after it changes. Verification is run after generation and after each supported conversational edit. This makes it possible to see whether a change introduces a new issue.

4. SYSTEM ARCHITECTURE

Figure 1 gives an overview of the FlowCAD pipeline. The language model is involved in understanding the requirements, creating the functional architecture and choosing from retrieved component candidates. Netlist synthesis, board placement, verification and the supported editing operations are handled by deterministic software. The system is intended to run verification after the initial design and again after every supported edit. One part of the workflow is not yet complete: net classes can be assigned, but a full topological autorouter has not been integrated.

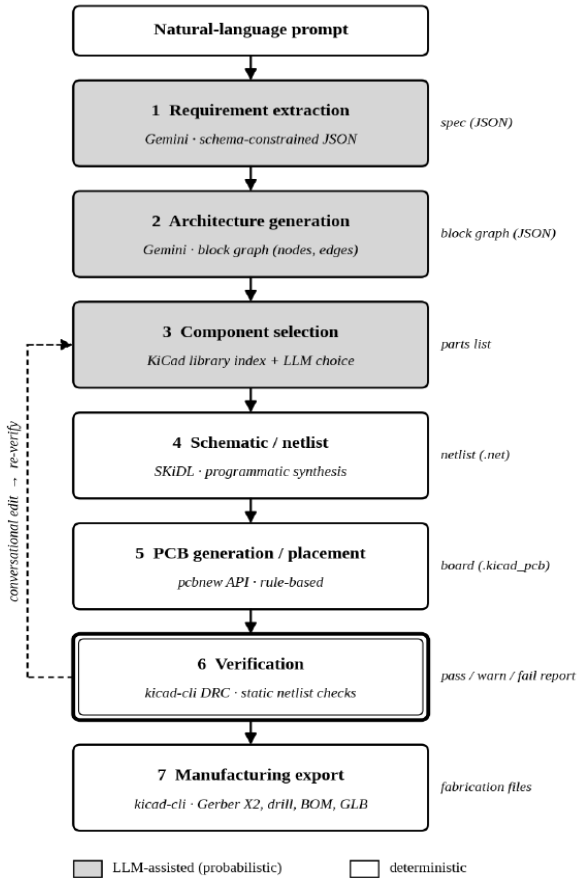


Figure 1. The FlowCAD pipeline. Shaded stages are LLM-assisted (probabilistic); unshaded stages are deterministic. Verification runs after generation and after every conversational edit.

4.1 Requirement Extraction

The user prompt is sent to a Gemini model [19]. The model is asked to return data in a defined JSON structure covering items such as the microcontroller, sensors, actuators, interfaces, power requirements and board constraints. The response is checked against the schema before the next stage can use it. If the response is malformed or the service returns a rate-limit error, the implementation can retry up to five times, using an exponential back-off between attempts.

4.2 Architecture Generation

The structured specification is then represented as a block graph G = (V, E). Each node represents a functional part of the design, and each edge represents a connection between functional blocks. For example, the graph may describe a power stage supplying a microcontroller, with the microcontroller connected to sensors and controlling a relay.

4.3 Library-Grounded Component Selection

When the service starts, it reads the KiCad library files (.kicad_sym and .kicad_mod), parses their S-expression data and builds an in-memory index. The index contains more than 22,000 standard library entries, with information such as the part name, library, pin count, category and keywords. For each block in the architecture, category and keyword filters are used to retrieve a smaller set of possible components. Only these candidates are passed to the language model, rather than the full library. This keeps the model's context manageable and limits its choices to parts that exist in the indexed libraries. The model then selects one of the retrieved candidates and provides a short explanation for its choice.

4.4 Schematic and Netlist Synthesis

The selected components are created as SKiDL [12] part objects and linked to their symbol and footprint references. The system then uses the connections in the block graph to build pin-level connections and exports the result as a KiCad netlist (.net). An important implementation detail still needs to be documented more fully: the exact rule used to translate each functional connection into component pins. That mapping should cover power, ground, buses and GPIO connections, and should be checked against a canonical pin-function table. Until this information is available, netlist-accuracy measurements cannot be interpreted with confidence.

4.5 PCB Generation and Placement

FlowCAD runs scripts through KiCad's bundled Python interpreter because the pcbnew bindings are tied to that environment. The scripts are launched as subprocesses and are responsible for creating the board, importing footprints and nets, and placing components according to the implemented rules. Components are grouped by type. Connectors are placed close to a board edge, decoupling capacitors are positioned near the power pins of their associated integrated circuits, and footprint bounding boxes are used to reduce overlap. Wider net classes are assigned to power and ground connections than to signal connections. A separate service reads board geometry for display in the web interface when KiCad is available. If KiCad is unavailable, the interface may show a rule-based fallback layout. That fallback is only a visual aid: it is not a KiCad board, has not passed KiCad verification and must not be counted in evaluation results. A complete topological autorouter is not part of the current implementation.

4.6 Verification

The verification process is shown in Figure 2. It combines three types of checks. First, library-integrity checks confirm that the selected symbols and footprints can be resolved. Second, static netlist analysis examines connectivity and pin usage. Third, the KiCad command-line tool, kicad-cli, runs PCB design-rule checks and returns the findings in JSON format. FlowCAD combines these outputs into a report that marks each check category as pass, warning or fail. In the KiCad CLI version used for this prototype, netlist-level electrical-rule checking is not available through the command line, so static netlist analysis is used instead. Because routing is incomplete, unconnected-item findings are expected; they should be listed separately from other error-level DRC violations. Passing the available checks does not show that the circuit performs its intended function. The authors must also confirm what the application actually does when the combined report contains a failure—for example, whether it blocks export, displays the report or attempts regeneration.

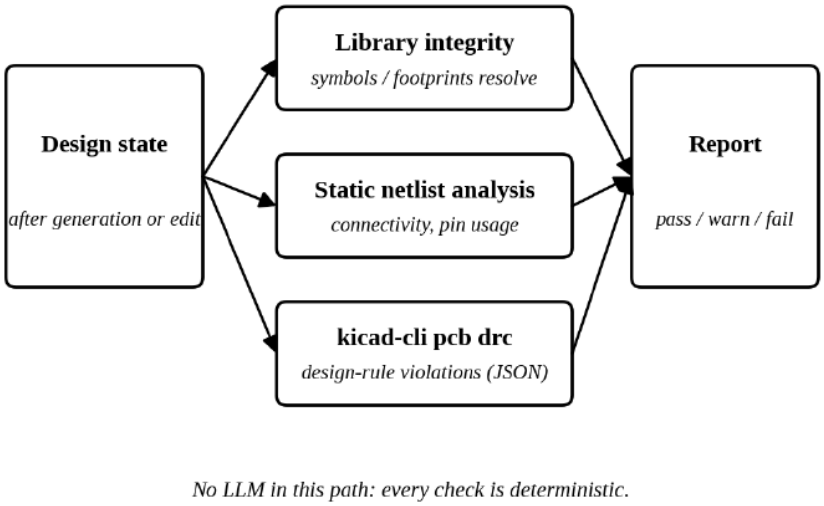


Figure 2. The verification gate applied after generation and after every edit.

4.7 Manufacturing Export

The prototype uses kicad-cli to generate Gerber X2 files, drill files, a CSV bill of materials and a GLB 3D model. In this paper, a successful export means that the requested files have been created and can be opened independently. It does not mean that the board is ready to send to a manufacturer. Since routing is only partly implemented, the copper layers do not yet describe a fully connected board. The design has also not been checked by a fabricator, manufactured or tested as a physical PCB.

4.8 Conversational Editing

The editing interface accepts follow-up instructions such as “make the board smaller”, “move the connector to the left”, “replace the sensor” and “add a status LED”. The prototype looks for patterns corresponding to four supported operations: resize, move, replace and add. When a command matches one of these patterns, the stored design is updated and the verification process runs again. If the command does not match a supported pattern, the system returns an error rather than guessing what the user meant. The interface is therefore limited to the command patterns implemented in the current prototype. Its success rate has not yet been measured.

5. IMPLEMENTATION

5.1 Software Stack

Table 2. Software stack of the FlowCAD prototype.

Layer	Technology
Web client	React, Vite, TanStack Router, Tailwind CSS; GLB model viewer; hosted on Vercel
Backend	Python, FastAPI (one endpoint per stage plus model serving); hosted on Render
LLM	Gemini (Flash-class model) through the Google Gemini API [19]
EDA engine	KiCad [1] (pcbnew API, kicad-cli); SKiDL 1.2.3 [12]
Data	Supabase (PostgreSQL and authentication): projects and design state

5.2 Algorithms

Algorithm 1: Library-grounded component selection.

Input: block graph $G = (V, E)$, library index L. *Output:* parts list P.

$P \leftarrow \emptyset$

for each node $v \in V$ **do**

$C \leftarrow$ Filter(L, category(v), keywords(v)), truncated to K candidates

$c \leftarrow$ LlmChoose(v, C) (answer must be an entry of C)

if $c \notin C$ **then** retry (up to the attempt limit); on failure report v as unresolved

$P \leftarrow P \cup \{(v, c)\}$

return P

Algorithm 2: Rule-based placement and net classes.

Input: netlist N, board outline. *Output:* placed board.

classify parts: connector, integrated circuit, decoupling capacitor, other passive

compute the bounding box of every footprint

place integrated circuits as layout anchors

place connectors on the nearest board edge

place each decoupling capacitor adjacent to the power pin of its integrated circuit

place remaining passives near the parts they connect to, keeping bounding boxes disjoint

assign wider net classes to power and ground nets than to signal nets

Algorithm 3: Verification and conversational edit.

Input: design state D, optional command x. *Output:* updated D and report R.

if x is given **then** match x against the resize, move, replace and add patterns; if none matches, return an error message; otherwise apply the matched mutation to D

$R_1 \leftarrow$ library-integrity check of D

$R_2 \leftarrow$ static netlist analysis of D

$R_3 \leftarrow$ kicad-cli pcb drc on D, parsed from JSON

return D and the aggregate of R_1, R_2, R_3 as pass/warning/fail per category

5.3 Implementation Status

Table 3 summarises what is currently available in the prototype. A status of "implemented" means that the corresponding feature is present in the described workflow; it does not mean that its accuracy or reliability has been established through the experiments proposed in this paper. Routing is still partial, and the browser-based 3D viewer remains in development. Since test evidence has not been provided for this manuscript, no stage is described as both implemented and experimentally validated.

Table 3. Implementation status of the pipeline.

Stage	Technique	Status
Requirement extraction	Gemini, schema-constrained JSON, retry with back-off	Implemented; not experimentally evaluated
Architecture generation	Gemini, block graph	Implemented; not experimentally evaluated
Component selection	KiCad library index (22,000+ parts) and LLM choice	Implemented; not experimentally evaluated
Schematic / netlist	SKiDL 1.2.3, .net export	Implemented; not experimentally evaluated
PCB generation / placement	pcbnew via subprocess; rule-based placement	Implemented; not experimentally evaluated
Routing	Net-class assignment; no topological autorouter	Partial
Verification	kicad-cli DRC, library and netlist checks	Implemented; not experimentally evaluated
Export	Gerber X2, drill, BOM CSV, GLB model	Implemented; not experimentally evaluated
Conversational editing	Pattern-based parser, automatic re-verification	Implemented; not experimentally evaluated
Web 3D viewer	Integration of the exported GLB model	In progress

6. EVALUATION PROTOCOL

The following section explains how FlowCAD should be evaluated. The experiments have not been completed for this version of the paper. For that reason, Table 5 is provided as a template for recording results, not as a summary of measurements.

6.1 Test Circuits

The three circuit tiers in Table 4 should each be described using several versions of the prompt: a short version, a detailed version and an under-specified version. Together, these prompts form the task set T. Since language-model responses can vary between runs, each task should be executed five times. The exact model identifier and its sampling settings should be recorded for every experiment so that the results can be interpreted and, where possible, reproduced.

Table 4. Circuit tiers used in the evaluation.

Tier	Circuit
T1	ESP32 LED indicator with current-limiting resistor, powered over USB-C through a 3.3 V regulator
T2	ESP32 with a DHT22 temperature and humidity sensor, powered over USB-C
T3	Smart irrigation controller: ESP32-WROOM-32E, DHT22, soil-moisture input through a 3-pin JST-XH header, SRD-05VDC relay driven by a ULN2003A and broken out to a 2-pin screw terminal, USB-C 5 V input, AMS1117-3.3 regulator

6.2 Metrics

Let Ggen represent the netlist produced by FlowCAD and Gref represent the reference netlist. Each net is represented as a set of nodes, where a node identifies a component by its role and a pin. Using the netlist-accuracy approach described in SchGen [5] as a starting point, precision, recall and the Jaccard index are calculated as follows:

$$J = \frac{|G_{gen} \cap G_{ref}|}{|G_{gen} \cup G_{ref}|} \qquad (2)$$

The component resolution rate measures the proportion of selected components that can be resolved to valid library entries:

$$CRR = \frac{|\{c \in P : c \in L \text{ without } fallback\}|}{|P|} \qquad (3)$$

Requirement-field accuracy (RFA) measures how many of the fields in the reference specification are correctly recovered by the system. Placement-stage clean pass (PCP) is the proportion of generated boards with no error-level DRC violations, excluding unconnected-item findings that are expected while autorouting is incomplete. DRC findings should also be reported by category, with unconnected items shown separately. End-to-end success means that a run completes the specified software stages and generates the listed output files without manual intervention. It should not be interpreted as evidence that the circuit works or that the PCB can be manufactured successfully.

$$pass@k = \frac{1}{|T|}\sum_{t \in T} 1[\exists\, i \le k : success(t,i)] \qquad (4)$$

The time taken by each stage should be reported using the mean and standard deviation. The edit-success rate (ESR) is the proportion of commands in a fixed, pre-declared test set that the system applies correctly and then verifies again. Two outcomes should be kept separate: whether the available checks pass, and whether the generated design matches the reference design. A board may pass the checks that are currently implemented and still perform the wrong function.

6.3 Baselines and Ablations

The first baseline, B0, asks the same language model to generate a netlist in a single prompt, without library grounding, schema constraints or verification. In ablation A1, library grounding is removed and the model can name components freely. In A2, schema constraints and schema validation are removed. Further experiments should examine the effect of how retrieval candidates are prepared, the placement rules and the enforcement of the verification gate. Every baseline and ablation should use the same tasks and comparable model settings so that the results can be interpreted fairly.

Table 5. Results template; populate only with measured values from the stated protocol. Use "n/a" for metrics that cannot validly be computed.

Configuration	RFA	CRR	J	PCP	pass@1	pass@5
B0 (single prompt)	TBD	TBD	TBD	TBD	TBD	TBD
A1 (no grounding)	TBD	TBD	TBD	TBD	TBD	TBD

Configuration	**RFA**	**CRR**	**J**	**PCP**	**pass@1**	**pass@5**
A2 (no schema)	TBD	TBD	TBD	TBD	TBD	TBD
FlowCAD (full)	**TBD**	**TBD**	**TBD**	**TBD**	**TBD**	**TBD**

6.4 Results

There are no quantitative results to report in this version because the planned evaluation has not been run. The empty cells in the results table are intentional. They should be replaced only after the experiments have been completed and the resulting data have been checked and analysed.

6.5 Implementation Observations

Three implementation issues became clear during development. First, KiCad's pcbnew Python bindings depend on the Python environment bundled with KiCad. For that reason, FlowCAD runs the relevant scripts as subprocesses instead of importing pcbnew directly into the web service. Second, the version of KiCad used in the project does not expose netlist-level electrical-rule checking through its command-line interface, so the prototype performs static netlist analysis for that part of verification. Third, the Gemini service can return malformed responses or impose rate limits. The extraction stage therefore uses a limited retry process with exponential back-off.

7. DISCUSSION

One of FlowCAD's main design choices is to keep language-model output separate from the checks performed by engineering tools. This makes it possible to inspect the model's proposals and see which checks pass or fail, rather than allowing the model to judge its own work. The checks are still limited by what they cover. KiCad DRC and static netlist analysis cannot, on their own, establish functional correctness, signal integrity, thermal safety or electromagnetic compatibility.

The language model is used for three tasks: interpreting the requirements, preparing the functional architecture and selecting from the retrieved component candidates. Netlist generation, placement, verification and the supported editing operations are handled by deterministic software. This division keeps the model away from tasks that require exact pin assignments or geometric operations and makes intermediate files available for inspection. At present, this is an architectural design choice. Its practical advantages still need to be demonstrated through the proposed evaluation.

FlowCAD should be viewed as a tool for preparing an initial engineering design, not as a replacement for engineering review. Before a board is fabricated or used, a qualified engineer should check the component ratings, pin assignments, power paths, protection circuits, clearances and intended operation. This review is particularly important for designs involving safety-critical functions, high-speed signals, significant heat or high power.

8. LIMITATIONS AND THREATS TO VALIDITY

(L1) Routing is not complete. The current prototype assigns net classes but does not include a full topological autorouter. As a result, the generated board is not fully routed. Its exported copper files must not be described as representing a finished or manufacturable PCB.

(L2) The checks cover only part of the design. FlowCAD currently checks library references, performs static netlist analysis and runs KiCad DRC. These checks can identify problems within their scope, but they do not demonstrate that the circuit will function correctly. Signal integrity, thermal behaviour, electromagnetic compatibility and the complete electrical behaviour of the circuit are not modelled. A seeded-fault study is also needed to find out which errors the verifier detects and which it may miss.

(L3) The quality of component selection depends on the available libraries and retrieval process. Some parts, such as microcontroller modules and sensor breakout boards, may not be present in the standard KiCad libraries or in the supplementary libraries being indexed. If a suitable part is missing from the candidate set, the model cannot select it. Library coverage and retrieval recall should therefore be measured separately from the model's ability to choose among the candidates it receives.

(L4) The proposed evaluation has limitations that need to be considered. The task set is relatively small and was prepared by the development team. Language-model outputs can vary, and results may change when model or software versions are updated. This version does not include measured results or an independent expert review. To reduce bias, the reference designs and pin-function mappings should be prepared or checked independently before the evaluation is run.

(L5) The current prototype is intended for single-board designs of low to moderate complexity. It does not retrieve live component prices. User prompts are sent to a cloud-based model service, and the availability or version of the hosting platform, component libraries and external APIs may affect whether another team can reproduce the same results.

9. CONCLUSION AND FUTURE WORK

This paper has described FlowCAD, a prototype that connects a natural-language prompt with component selection from a KiCad library index, SKiDL-based netlist generation, rule-based board placement, deterministic verification and several export formats. It also includes a limited conversational editing interface that re-runs verification after supported changes. The central idea is to use a language model where interpretation is useful and established engineering tools where exact operations and checks are required. The current implementation should not be overstated: its main stages are described as implemented, but they have not yet been evaluated experimentally; routing is partial, and the browser-based 3D viewer is still in development. The contribution of this paper is therefore a description of the architecture, prototype and planned evaluation—not evidence of measured performance, functional correctness or manufacturing readiness. Future work will need to resolve the implementation questions listed below, complete the proposed experiments, integrate and test routing, obtain independent engineering review, and fabricate and test physical boards before stronger claims can be made.

PRE-SUBMISSION AUTHOR CHECKLIST

The points below need to be confirmed by the authors using the source code, development logs and running implementation. They are deliberately left open where the available project information does not provide a reliable answer.

1. Verification-gate behaviour: Confirm what the application does when the combined verification report contains a failure. Does it prevent export, show the report without blocking export, or attempt to regenerate the design?

2. Pin-level netlist generation: Explain exactly how each connection in the block graph is converted into component-pin connections. Include the rules for power, ground, buses and GPIO, and provide a reference table of the relevant pin functions.

3. Component retrieval: State the candidate limit K, how candidates are ordered and truncated, and how search keywords are produced. Measure library coverage and retrieval recall separately.

4. Reproducibility information: Record the exact Gemini model identifier, temperature, top-p value, token limit and dates of use. Also report the versions or hashes of KiCad, kicad-cli, SKiDL, Python and the indexed component libraries.

5. Placement and verification settings: Document the default board outline, placement grid, clearances, rule for positioning decoupling capacitors, net-class widths, static-netlist checks and mapping from findings to severity levels.

6. Data storage and interface: Confirm which part of the application reads from and writes to Supabase, which artefacts are stored, how board-layout data is extracted, when the fallback display is used, and which edit commands are supported.

7. Tests and evaluation evidence: Identify any unit or integration tests that already exist. Prepare fixed reference specifications, netlists and pin maps, and ask a second reviewer to check them independently. Run the evaluation only after these references are agreed, then replace every TBD entry with a measured value or mark it n/a where appropriate.

8. Figures and publication details: Replace each placeholder figure with a diagram that reflects the running implementation. Add the corresponding author's confirmed email address, funding information and acknowledgements where applicable. Check the target journal's formatting requirements and verify every reference and access date.

This revision intentionally leaves unknown implementation details and experimental values unresolved rather than filling them with assumptions. Until the evaluation has been completed, the paper is best positioned as a system-description paper. It should not be presented as an experimentally validated performance study.

References

[1] KiCad Project, "KiCad EDA Suite." [Online]. Available: https://www.kicad.org.

[2] Altium Limited, "Altium Designer," 2026. [Online]. Available: https://www.altium.com/altium-designer.

[3] Cadence Design Systems, "Allegro X Design Platform," 2026. [Online]. Available: https://www.cadence.com.

[4] T. King, S. Kehrberg, M. Beigl and T. Röddiger, "pcbGPT: Automatic PCB schematic synthesis from natural language requirements," arXiv:2606.01188, 2026.

[5] Q. Luo, R. Ma, X. Zhang, and L. Qiu, "SchGen: PCB schematic generation with semantic-grounded code representations," arXiv:2605.30345, 2026.

[6] H. Zou, P. Han, E. Nazerian, M. Zhang, Z. Guo and A. Q. Huang, "PCBSchemaGen: Reward-guided LLM code synthesis for printed circuit boards (PCB) schematic design with structured verification," arXiv:2602.00510, 2026.

[7] ProtoFlow, "Best AI PCB design software in 2026 (tested & compared)," 4 June 2026. [Online]. Available: https://www.protoflow.ai/compare/best-ai-pcb-design-software-2026.

[8] Flux, "Flux: browser-based electronic design," 2026. [Online]. Available: https://www.flux.ai.

[9] Quilter, "Quilter: autonomous PCB placement and routing," 2026. [Online]. Available: https://www.quilter.ai.

[10] DeepPCB, "DeepPCB: autonomous PCB routing," 2026. [Online]. Available: https://www.deeppcb.ai.

[11] H. Song, J. Park, W.-S. Choi, S. Bae, H.-S. Jeong, Y. Park and S. Lee, "PCBWorld: A benchmark environment for engine-grounded PCB design automation," arXiv:2607.05915, 2026.

[12] SKiDL Project, "SKiDL: A Python package for describing electronic circuits," GitHub repository. [Online]. Available: https://github.com/devbisme/skidl.

[13] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng and B. Yu, "ChatEDA: A large language model powered autonomous agent for EDA," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 43, no. 10, pp. 3184–3197, 2024.

[14] ProtoFlow, "AI PCB design in 2026: What's real and what's hype," 6 June 2026. [Online]. Available: https://www.protoflow.ai/blog/ai-pcb-design-2026-guide. Vendor source; verify claims and access date.

[15] Circuit Mind, "Circuit Mind: automated schematic and BOM generation," 2026. [Online]. Available: https://www.circuitmind.io.

[16] CELUS, "CELUS: automated electronics design," 2026. [Online]. Available: https://www.celus.io.

[17] Removed pending independent verification. The repository attribution and content were not confirmed in the review; restore only after inspection.

[18] B. Benchoff, "OrthoRoute: a GPU-accelerated autorouter for KiCad," GitHub repository, 2025.

[19] Gemini Team, Google, "Gemini: A family of highly capable multimodal models," arXiv:2312.11805, 2023.

[20] Ucamco, "The Gerber Layer Format Specification." [Online]. Available: https://www.ucamco.com/en/gerber.

[21] IPC, "IPC-2221A: Generic standard on printed board design," 2003.