Gen-UI-Lang: A Compact Language for LLM-Driven UI Generation

Praneeth Vadlapati
Independent researcher
praneethv@arizona.edu
ORCID: 0009-0006-2592-2564

Abstract: This paper introduces Gen-UI-Lang, a compact language designed to express user interfaces as concise abstract syntax trees that can be rendered to multiple targets such as HTML and React. The language prioritizes human readability and predictability for large language model (LLM) generation, reducing token usage while preserving structural clarity and thereby improving the reliability of model-produced UI descriptions. The design centers on a small set of composable factory functions that map to structured "Node" objects, enabling deterministic rendering, straightforward extensibility, and lightweight integration with interactive demo frameworks. A reference implementation demonstrates minimal renderer code, an optional LLM helper that biases outputs toward the language's syntax, and a demo script that exercises key functionality. Evaluation focuses on qualitative analyses of concision, LLM fidelity, and engineering overhead, showing that compact, predictable UI specifications reduce malformed model outputs and lower the development effort required for rapid prototyping. Limitations and avenues for future work include richer interaction semantics, formal verification of generated outputs, and pathways toward production-grade component generation.

The source code is available at github.com/Pro-GenAI/Gen-UI-Lang.

Keywords: Large language models, LLMs, Generative AI, Generative UI, user interfaces, UI synthesis, Artificial Intelligence, AI

I. Introduction

Rapid iteration of user interfaces (UIs) remains a challenge in early-stage design, prototyping, and LLM-assisted workflows because traditional UI authoring requires verbose markup or deep framework knowledge. This work proposes a compact, Python-first domain-specific language (DSL), Gen-UI-Lang, which represents UIs as small, composable node factories mapped to an AST, which can be deterministically rendered to multiple targets. The language is intentionally minimal—factories such as "ui", "row", "text", "btn", and "chart" capture common UI concepts while remaining amenable to generation and editing by language models. By narrowing the surface area of expressiveness to a predictable syntax, Gen-UI-Lang reduces the cost and error-proneness of asking LLMs to produce UI code and provides a pragmatic middle ground between free-form HTML and heavyweight component systems.

A. Disadvantages with current approaches

Existing approaches to UI generation span free-form HTML, component libraries, and JSON-based shorteners, and each presents trade-offs that hinder rapid, model-assisted workflows. Free-form HTML and framework-specific JSX are expressive but verbose and brittle when generated by models, leading to high token usage and fragile outputs. Compact formats that compress full component trees often sacrifice readability or require complex decoders that are unfriendly to human editing. Moreover, disparate render targets force duplication of intent across formats or reliance on heavy tooling, impeding quick prototypes and iterative experimentation.

B. Proposed system and its benefits

Gen-UI-Lang addresses these shortcomings by offering a terse, human-readable language that maps directly to an AST and small renderer implementations. The design emphasizes predictable token patterns, which improves the likelihood that LLMs will produce syntactically valid UI descriptions and makes small, local edits straightforward for developers. The implementation demonstrates that a single authored description can be converted to multiple targets with modest renderer code, reducing duplication and accelerating prototype delivery. Additional benefits include straightforward extensibility—new nodes and renderers can be added with minimal changes—and the ability to embed simple callables or metadata for interactive demos.

C. New use cases of the system

Gen-UI-Lang enables workflows that are difficult with current tooling, including LLM-driven UI sketching where an assistant returns a "ui(...)" snippet directly, automated transformation pipelines that convert a single spec into both a static preview and an interactive demo, and rapid experiment loops for HCI researchers exploring alternative layouts or interaction affordances. The language also supports lightweight integration into notebook-driven demos and rapid Gradio proof-of-concept builds, lowering the barrier for showcasing models and data with minimal engineering overhead.

D. Related work

The work draws inspiration from compact representation formats and template languages while differing from them by focusing on LLM compatibility and minimal implementation surface. Prior efforts to Generative UI [c] introduce the concept of using LLMs to generate interactive UI. However, those efforts lack focus on a short LLM-friendly language that models can use for simpler and faster generation of UI to answer user queries effectively. Unlike full-fledged UI frameworks, the goal of Gen-UI-Lang is not to replace production component libraries but to provide a compact, interoperable authoring layer for LLMs targeted at prototyping and model-centered workflows.

II. Methods

A. Language design

Gen-UI-Lang's syntax is intentionally small and compositional, and a small set of factory functions (for example "ui", "row", "text", "btn", "chart") construct "Node" objects that carry a "type", a "props" dictionary, and a list of children. This representation yields an AST that is both machine-friendly and readable, and common UI constructs are expressed in a single, compact expression rather than nested markup or verbose JSON structures. The choice to keep node factories and plain Python callables in the language supports interactive embedding and simple dynamic behaviors in demos while maintaining a small, parsable surface for model generation.

B. Implementation

The reference implementation is deliberately minimal and demonstrates the core ideas with a compact codebase that implements Node classes, factory helpers, and a lightweight renderer that converts node structures to HTML using straightforward templates. The implementation illustrates how additional renderers can be introduced by handling node types and mapping properties to target-specific constructs, and the resulting modularity reduces the engineering effort required to support new output formats. A demonstration program was employed to assemble representative node structures and to exercise the HTML rendering pathway as a baseline evaluation.

C. LLM integration

To evaluate LLM-centered workflows, the system incorporates an optional helper module that wraps chat-completion calls and biases prompts toward producing the compact UI syntax. The helper implements pragmatic checks for expected output patterns and a retry mechanism while leaving model configuration to the deployment environment. This integration emphasizes robust, provider-agnostic behavior and enables experiments in which language models directly synthesize UI descriptions for downstream rendering.

D. Evaluation protocol

Because this work focuses on representation design and practicality rather than claim-driven metrics, evaluation emphasizes qualitative analyses, measuring concision by comparing snippet lengths for representative UI examples, assessing LLM fidelity by the frequency of valid “ui(...)” responses in controlled prompts, and validating extensibility through small case studies where new nodes or renderers are added. Implementation overhead is measured by counting lines of renderer code and estimating developer effort for common extension tasks.

III. Results

A. Expressiveness and concision

Across representative examples the language produced markedly shorter authoring artifacts compared to equivalent JSX or verbose HTML snippets, and common layout and control patterns were expressible in single-line or short multi-line expressions that demonstrate the chosen primitives cover a useful design space without excessive verbosity. The compactness directly translates to lower token use in LLM-assisted generation and makes manual editing faster for humans.

B. LLM fidelity

In controlled experiments using the included prompt biasing, language models were more likely to emit syntactically valid Gen-UI-Lang snippets than free-form HTML when prompts requested the compact language, and while absolute success rates depend on model and prompt engineering the observed improvements indicate that restricting the target grammar benefits reliability in model outputs. The LLM helper's simple validation and retry logic reduced obvious format errors in practice.

C. Extensibility case studies

Small case studies implementing additional nodes, for example a “form” node and a “chart” node with specific property mappings, required minimal changes such as adding a factory and extending the renderer with a concise template branch. These modifications validated the project's claim that renderers and node types are easy to augment and highlighted a low barrier for community-driven extensions.

D. Implementation overhead

The reference renderer and AST utilities are intentionally compact, and the core implementation required only a few dozen lines to cover the most common nodes and a simple HTML renderer. This low overhead supports rapid prototyping and makes the system suitable for inclusion in research codebases and demonstrations without heavy dependency burdens.

IV. Discussion

Gen-UI-Lang demonstrates that a carefully scoped language can improve the practicality of LLM-assisted UI generation by reducing token overhead and increasing syntactic predictability. The trade-offs are deliberate because the language sacrifices the full expressiveness of target frameworks in exchange for human readability and model-friendliness, which is appropriate for prototyping, demos, and exploratory research. Limitations include the need for richer semantics for complex interactions and the absence of formal verification of rendered outputs, and future work should address richer bindings to stateful components, automated conversion to production-ready components, and user studies to quantify developer productivity gains.

V. Conclusion

This paper presented Gen-UI-Lang, a compact, Python-first language designed to bridge human authoring and LLM-assisted UI synthesis. The language's small set of composable primitives yields concise ASTs that render to multiple targets with minimal renderer logic, improving the reliability and efficiency of model-driven UI generation. Through implementation and qualitative evaluation the document shows benefits in concision, LLM fidelity, and extensibility while outlining avenues for extending semantics and productionization. Gen-UI-Lang is positioned as a lightweight tool for prototyping and research where rapid iteration and model compatibility are priorities.

REFERENCES

[1] <To be added>