



OMNI BUILD INTELLIGENCE

An LLM-Based Framework for Constraint-Aware Architectural Floor-Plan Generation, Visualization, and Interactive Project Management

Somen Poria

Generative Artificial Intelligence | Architectural Computing | Human-in-the-Loop Design Based on the implemented Omni Build Intelligence prototype

October 2026

How to Cite this Article:

Poria, S. (2026). OMNI_BUILD INTELLIGENCE An LLM-Based Framework for Constraint-Aware Architectural Floor-Plan Generation, Visualization, and Interactive Project Management. International Journal of Creative and Open Research in Engineering and Management, <i>02</i>(10), 1-9.
<https://doi.org/10.55041/ijcope.v2i10.034>

License:

This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.



OPEN ACCESS



<https://doi.org/10.55041/ijcope.v2i10.034>

The increasing capability of large language models (LLMs) creates an opportunity to redesign how users interact with computational design software. Architectural planning is a particularly relevant domain because early-stage requirements are often expressed in natural language, while the resulting design must ultimately be represented as geometry. This paper presents Omni Build Intelligence, an implemented full-stack prototype that explores an LLM-assisted workflow for generating preliminary architectural floor plans from structured user requirements. The system accepts building type, requested room count, plot width and length, and special requirements. A Flask backend constructs a domain-specific prompt and invokes a Llama 3.3 70B model through the Grok API. The response is expected in a structured textual format containing rooms, coordinates, dimensions, door relationships, and a summary.

The generated representation is parsed into easy for software to read objects and passed to a React interface. The interface provides a two-dimensional blueprint view and an isometric three-dimensional visualization. A conversion layer transforms generated room rectangles into editable wall and label primitives that can be associated with a project record. The broader platform also includes authentication, project management, plot listing and filtering, dashboard functionality, and persistence of drawing data. These components make Omni Build more than a standalone text-generation demonstration: it is a prototype for integrating generative AI into an interactive design workflow.

The research contribution of this paper is the formulation of a rigorous evaluation methodology for such a system. The proposed methodology defines an Architectural Requirement Prompt Dataset, meaningful baselines, formal geometric constraints, automated validation metrics, human evaluation criteria, ablation experiments, and statistical analysis procedures. Metrics include constraint satisfaction, boundary violations, room overlap, requirement accuracy, dimension validity, space use of space, valid-plan rate, and generation latency. A user guided study is proposed to evaluate perceived practicality and usefulness.

The study deliberately does not claim measured superiority because the repository provides implementation evidence but not a completed controlled benchmark. Instead, the paper establishes a reproducible experimental framework through which the system can be evaluated. This distinction prevents software functionality from being mistaken for research evidence. The work concludes that LLM-based architectural assistance is most appropriately positioned as an early-stage design aid whose natural-language flexibility is combined with explicit geometric validation and human review.

Keywords: large language models, generative AI, computer based architectural design, floor-plan generation, constraint based prompting, spatial reasoning, structured generation, user guided design, interactive visualization, project management



1. Introduction

1.1 Context and Motivation

Architectural design is a knowledge-intensive process in which requirements, constraints, geometry, aesthetics, and professional judgment interact continuously. A client may describe a desired building in terms of rooms, plot dimensions, budget, or special facilities, but those statements do not directly specify a geometric solution. The transformation from an informal requirement description into a spatial plan traditionally requires domain expertise and repeated manual drafting.

Digital drafting systems have reduced the cost of geometric editing and documentation, yet they do not necessarily solve the problem of translating natural-language intent into an initial arrangement.

Generative artificial intelligence changes this interaction model. A user can describe an intended building in a compact textual form and ask a generative model to propose a structured design. The model can act as an interface between human intent and computational representation.

The architectural domain also exposes an important weakness of general-purpose language models: language correctness does not imply geometric correctness. A model can produce a convincing description while placing a room beyond a boundary, creating overlaps, omitting required rooms, or producing dimensions that are not useful. Consequently, a serious architectural AI system needs an intermediate representation that can be validated independently of the model.

Omni Build Intelligence was developed around this hybrid idea. Its implemented generative-design module takes building type, room count, plot dimensions, and special requests as inputs. The backend builds a prompt that explicitly states boundary and overlap requirements and requests a fixed response format. The LLM response is parsed into room and door objects. The front end then visualizes those objects and can convert them into editable drawing primitives.

1.2 Research Problem

The research problem is therefore defined as follows: how can an LLM-driven architectural interface convert natural-language-oriented requirements into a structured floor-plan representation that is measurable, visualizable, and suitable for human refinement while preserving explicit site constraints?



This problem contains four sub-problems. First, requirements must be represented clearly enough for a generative model to interpret them. Second, the model must produce structured geometry rather than unconstrained prose. Third, the generated geometry must be evaluated independently of the model.

Fourth, the result must fit into a user workflow in which humans can inspect and modify the proposal.

1.3 Objectives

- Design and document an LLM-based pipeline for preliminary floor-plan generation.
- Represent generated rooms using explicit coordinates and dimensions.
- Render generated layouts in interactive 2D and isometric 3D views.
- Integrate generated layouts with an editable drawing and project-management workflow.
- Define a benchmark that can quantify geometric and semantic validity.
- Compare the proposed approach against rule-based and simpler LLM baselines.
- Establish reproducible hypotheses and statistical tests without fabricating results.

1.4 Research Questions

- RQ1: How reliably can the implemented LLM workflow transform architectural requirements into structured room geometry?
- RQ2: Does explicit constraint based prompting reduce geometric violations?
- RQ3: Does structured output improve parsing and requirement compliance?
- RQ4: What contribution does deterministic validation make to final plan validity?
- RQ5: Does an editable AI-generated starting point improve perceived usefulness for early-stage design?
- RQ6: What quality-latency trade-off emerges when validation and iterative refinement are introduced?

1.5 Scope

The scope is deliberately limited to preliminary architectural layout generation and software integration. The system is not treated as a structural engineering solver, building-code checker, professional BIM platform, or construction estimation engine. The repository README describes broader ambitions involving computer vision, machine learning, and augmented reality, but the implemented evidence inspected for this paper most strongly supports the LLM floor-plan generation, visualization, drawing, project, authentication, and plot management components. The paper therefore avoids presenting unimplemented capabilities as completed research contributions.

2. Background and Related Work

2.1 Floor-Plan Understanding

Automated floor-plan understanding has a substantial history in computer vision. The CubiCasa5K dataset provides 5,000 floor-plan images with annotations covering more than 80 object categories, creating a benchmark for recognizing rooms and architectural elements from images. Its importance to Omni Build is methodological: it shows the value of standardized room and object representations.

However, the current Omni Build prototype does not perform image-to-layout inference and does not claim to train on CubiCasa5K.

Floor-plan understanding and floor-plan generation are related but distinct problems. Understanding starts from an existing drawing and attempts to infer semantic or geometric information. Generation starts from requirements and creates a new arrangement. This distinction affects both datasets and metrics. A generation system needs measures of validity and requirement satisfaction, while an understanding system may focus on segmentation or object detection accuracy.



2.2 Graph-Based Generative Layouts

House-GAN formulates house-layout generation using relational graphs. Room categories and spatial adjacency relationships are encoded as a graph, and the generator produces axis-aligned room bounding boxes. This approach shows that explicit relational constraints are useful in architectural generation. It also provides a conceptual comparison for Omni Build: both systems seek structured room layouts, but they use different generative mechanisms.

Graph2Plan combines layout graphs with building boundaries and neural processing to generate floor plans under sparse user constraints. The method highlights the importance of separating semantic layout requirements from geometric boundaries. Omni Build currently encodes both types of information into an LLM prompt and then extracts coordinates from the model response. This creates a simpler interaction model but provides weaker formal guarantees than a dedicated constrained generator.

2.3 Human-in-the-Loop Design

Graph2Plan is also relevant because it explicitly considers user-in-the-loop design. Architectural generation is rarely a fully autonomous task: different clients and designers may prefer different spatial arrangements even when the same numerical requirements are satisfied. Omni Build adopts a similar philosophy at the application level. Its drawing workspace permits walls, doors, windows, labels, erasing, undo, zoom, pan, and export operations, enabling a generated concept to be edited.

2.4 Foundation Models and LLMs

The Llama 3 model family shows the broader capability of foundation language models across reasoning, coding, multilinguality, and tool-use tasks [4]. Omni Build uses a Llama 3.3 70B Versatile endpoint through Grok as the generative component. The research contribution is not model training; it is application-level orchestration: requirements are converted into a domain prompt, a structured response is requested, and the result is connected to a geometric visualization and editing system.

2.5 Research Gap

The literature shows strong results from image-based and graph-based floor-plan generation, but an LLM interface creates a different engineering and evaluation problem. Natural-language prompts are flexible and accessible, yet they are probabilistic and can violate numerical constraints. The gap addressed by OmniBuild is therefore the design and evaluation of an LLM-to-geometry workflow that retains the interaction simplicity of natural language while introducing explicit structure, validation, and human editing.

2.6 Positioning of Omni Build

Dimension	Traditional CAD	Graph-based generation	Omni Build prototype
Input	Manual geometry	Graph + boundary	Natural-language/form requirements
Primary engine	Deterministic drawing	Learned generator	LLM inference
Output	Editable geometry	Generated layout	Structured rooms + editable drawing
Constraint mechanism	Hard/manual	Graph + boundary	Prompt-level; validation proposed
Human refinement	Yes	Yes	Yes
Main strength	Precision	Formal spatial structure	Low-friction requirement to-concept workflow
Main limitation	Manual initial drafting	Model/data complexity	LLM geometric reliability

Table 1. Conceptual positioning of Omni Build against related design paradigms.



3. Problem Definition and Research Gap

3.1 Formal Problem Definition

Let the input requirement be represented by $R = \{B, W, L, N, S\}$, where B denotes building type, W and L denote plot width and length, N denotes the requested number or set of rooms, and S denotes special requirements. The generation function G maps the requirement to a structured plan P . $G(R) -$

$> P = \{\text{Rooms, Doors, Summary}\}$

Each room R_i is represented as $R_i = (T_i, x_i, y_i, w_i, h_i)$, where T_i is the room label and x_i, y_i, w_i, h_i define its rectangular geometry. A plan is useful only if its representation is internally consistent and satisfies the critical constraints.

3.2 Boundary Constraints

$0 \leq x_i, 0 \leq y_i, x_i + w_i \leq W, y_i + h_i \leq L$

These constraints ensure that the room remains within the rectangular plot. They are straightforward to evaluate deterministically once the LLM output has been parsed into numeric values.

3.3 Non-Overlap Constraints

For two axis-aligned rectangles R_i and R_j , the rooms are considered non-overlapping if at least one separation condition holds. This can be evaluated without any language model.

$(x_i + w_i \leq x_j) \text{ OR } (x_j + w_j \leq x_i) \text{ OR } (y_i + h_i \leq y_j) \text{ OR } (y_j + h_j \leq y_i)$

3.4 Semantic Constraints

Geometric validity is necessary but not sufficient. If the user requests two bedrooms, one kitchen, and one living room, the output must contain those room categories with the requested counts. Special requests should be converted into explicit predicates wherever possible. This creates a separation between semantic validation and geometric validation.

3.5 Research Gap as a Testable Claim

The proposed research does not assume that the LLM is inherently good at spatial reasoning. Instead, it tests a narrower claim: whether explicit constraints, structured outputs, and validation can make an LLM driven workflow sufficiently reliable for early-stage architectural concept generation. This claim is testable through controlled baselines and objective metrics.

3.6 Expected Scientific Contribution

The scientific value of the work lies in evaluating a practical hybrid system. The LLM is treated as a probabilistic requirement interpreter and proposal generator, while deterministic software is responsible for representation and validation. This division of responsibilities provides a clear path toward ability to repeat the experiment and failure analysis.



4. Omni Build Intelligence: System Overview

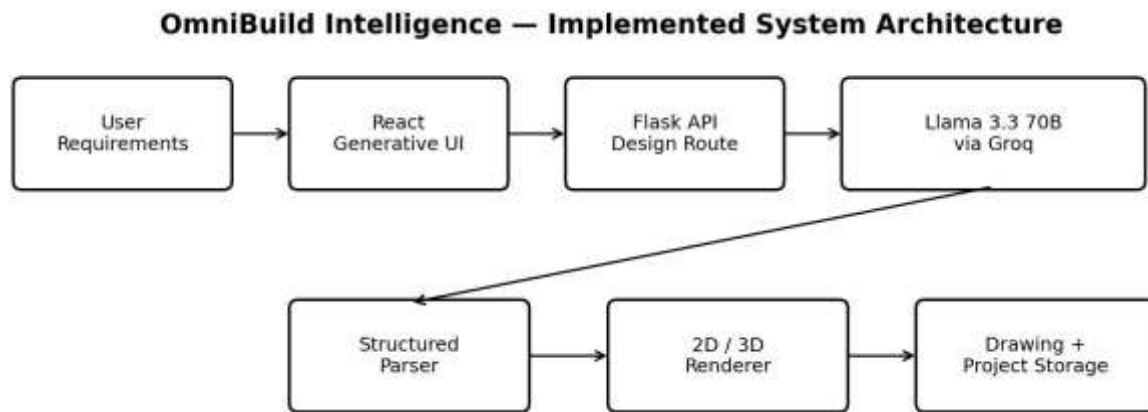


Figure 1. Implemented Omni Build architecture derived from the repository structure and application flow.

4.1 Front-End Layer

The front end is implemented using React 19 and react-scripts. The application includes pages for login, registration, dashboard, project creation, project detail, generative design, drawing, and marketplace browsing. Service modules communicate with the Flask API using HTTP requests and bearer tokens.

4.2 Backend Layer

The backend uses Flask with Flask-CORS. Application routes are organized into blueprints for users, plots, dashboard operations, design generation, and projects. This organization separates endpoint definitions from controller logic and provides a clear boundary between HTTP handling and application operations.

4.3 AI Design Controller

The design controller loads the Grok API key through environment configuration and creates a Groq client. The generation function reads building type, number of rooms, plot dimensions, and special requests. It then constructs a domain prompt and invokes the llama-3.3-70b-versatile model.

4.4 Structured Parsing

The returned text is divided into lines. The parser switches between ROOMS, DOORS, and SUMMARY sections. Room lines are split on the vertical bar delimiter and converted into objects with name, x, y, width, and height fields. Door lines are converted into source room, target room, and wall-direction fields. The final response also carries the original plot dimensions.

4.5 Front-End Generation State

The Generative Design component maintains form values, generation status, result state, view mode, rotation, project selection, and connection status. When generation succeeds, the result is displayed in 2D mode by default. A separate 3D canvas effect renders the same room objects using an isometric projection.



4.6 Project Connection

The generated rooms can be transformed into drawing walls and labels. The conversion uses a scale factor and an offset to map plot coordinates into canvas coordinates. The resulting drawing data is placed into a floor structure and sent to the project update endpoint. This provides an explicit bridge from AI-generated geometry to persistent project data.

4.7 Plot Marketplace

The platform also includes a land-plot module. Authenticated users can add plots, while plot retrieval supports filtering by minimum price, maximum price, and location. This feature is not part of the core LLM research experiment, but it shows the intended broader lifecycle: discover a site, create a project, generate a concept, and continue with an editable drawing.

4.8 Dashboard and Project Management

Project creation records project name, client, location, plot dimensions, building type, floors, budget, status, and notes. Project retrieval is scoped to the authenticated user. Updates can include drawing data, enabling generated designs to become persistent project artifacts.

5. LLM-Based Floor-Plan Generation Method

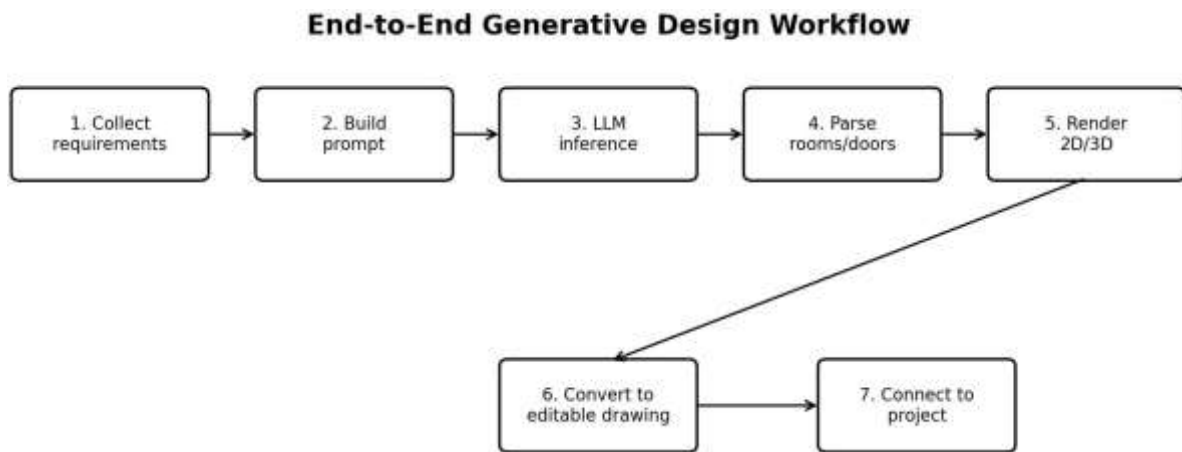


Figure 2. End-to-end generation and integration workflow.

5.1 Input Schema

Input	Type	Purpose	Prototype behavior
building_type	categorical text	Provides functional context	Defaults to residential
num_rooms	integer	Specifies requested room count	Default is 3 in backend; UI initializes 4
plot_width	numeric	Horizontal plot boundary	Default is 30 in backend; UI initializes 40



plot_length	numeric	Vertical plot boundary	Default is 40 in backend; UI initializes 50
special_requests	text	Additional user requirements	Defaults to none

Table 2. Input fields used by the implemented generation function. UI defaults and backend fallbacks differ and should be standardized in future work.

5.2 Prompt Strategy

The prompt frames the model as an expert architect and supplies the building type, room count, plot size, and special requests. It then requires an exact output format consisting of a ROOMS section, a DOORS section, and a SUMMARY section. Each room line must contain a name and four numerical attributes.

The prompt also states that the room geometry must remain within the plot, rooms should not overlap, dimensions should be realistic, and most of the plot should be covered.

This approach can be characterized as constraint based prompting rather than constrained decoding. The model is instructed to satisfy constraints, but the inference process itself does not mathematically enforce them. This distinction is central to the proposed experimental methodology.

5.3 Model Invocation

The implementation calls the Groq chat completion interface with the llama-3.3-70b-versatile model and a single user message containing the complete prompt. The current implementation does not specify a temperature, seed, structured JSON schema, function-calling contract, or automatic retry policy. These omissions are important experimental variables and should be controlled or documented in a research evaluation.

5.4 Parsing Algorithm

The parser is deliberately simple. It identifies section headers and processes bullet lines using delimiter based splitting. This design has an advantage for a prototype: the resulting object is easy to inspect and render. However, it is less robust than a formal JSON schema because malformed separators, unexpected text, missing fields, or non-numeric values can cause later issues.

5.5 Door Representation

Doors are represented as relationships between two room names plus a wall direction. The current visual conversion used for project import does not automatically translate the generated door relationships into drawing-door primitives; the imported floor data initializes doors as an empty array. This is a useful implementation limitation to document rather than hide.

5.6 2D Rendering

The 2D blueprint renderer maps plot coordinates into an SVG view box. A padding region is reserved around the plot, and separate scale factors are calculated for horizontal and vertical dimensions. Room rectangles are then positioned according to the parsed coordinates. The view includes a blueprint-style background, grid, boundary, room labels, and dimension annotations.

5.7 3D Visualization

The 3D mode uses an isometric projection. A helper function transforms x, y, and z coordinates using cosine and sine terms corresponding to a 30-degree projection. Rooms are rendered as top and wall faces, and a painter-style sort is applied before drawing. The resulting representation communicates depth but is not intended to be a physically accurate structural model.



5.8 Conversion to Editable Walls

Each room rectangle is converted into four wall segments. The transformation applies a scale of ten and a fixed canvas offset. A label is placed at the center of each room. This conversion is deterministic and therefore provides an important opportunity for testing whether the generated geometry survives the transition from AI output to editable drawing data.

6. Software and Data Architecture

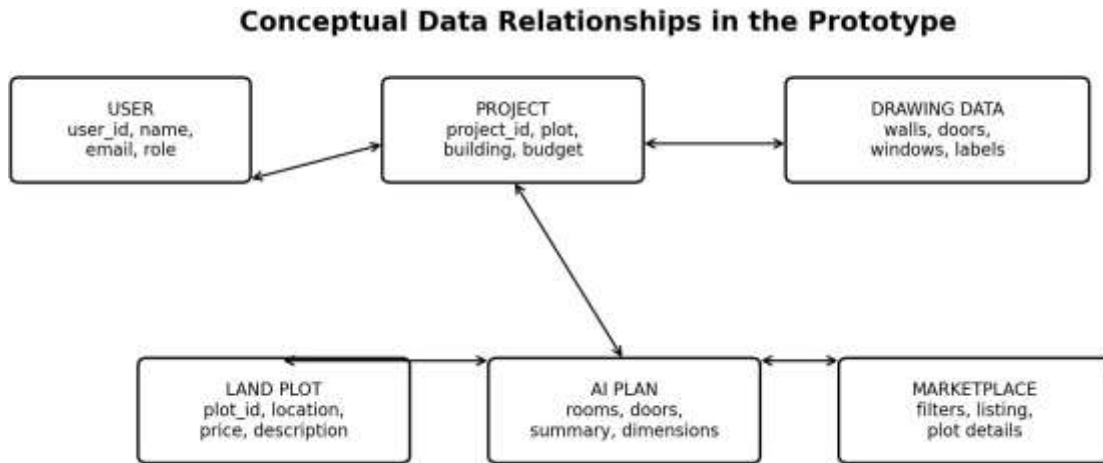


Figure 3. Conceptual data relationships represented by the implemented application modules.

6.1 Technology Stack

Layer	Technology	Evidence in repository
Client	React 19	frontend/package.json
Build tooling	react-scripts 5	frontend/package.json
Server	Flask	backend/app.py
Cross-origin requests	Flask-CORS	backend/requirements.txt and app.py
Database	MySQL connector	backend/db.py
Authentication	bcrypt + JWT	user_controller.py and auth.py
Generative model access	Groq client	design_controller.py
Model endpoint	llama-3.3-70b-versatile	design_controller.py
Visualization	SVG + Canvas	GenerativeDesign.js

Table 3. Technology stack identified from the project repository.

6.2 Authentication and Authorization

Registration hashes passwords using bcrypt before storing them. Login verifies the submitted password against the stored hash and creates a JWT containing the user identifier and a two-hour expiration.

Protected endpoints use a token_required decorator that checks the Authorization header and decodes the token.



The design-generation, project, and plot-add routes are protected. Public plot retrieval endpoints remain accessible without authentication. Project queries additionally filter by the authenticated user identifier, providing an important ownership boundary.

6.3 Database Persistence

The database connection uses MySQL Connector. Project records contain metadata such as client, location, plot dimensions, building type, floors, budget, status, notes, and drawing data. Drawing data is serialized as JSON before storage. This is a pragmatic design for a prototype, although a production system could normalize architectural entities or use a document-oriented representation depending on query requirements.

6.4 API Surface

Route group	Operations	Research relevance
/api/users	registration, login, users	Identity and access
/api/plots	add, list, retrieve	Site/land context
/api/design	generate, assist	Core generative AI
/api/projects	create, list, retrieve, update	Design persistence
/api/dashboard	dashboard data	Project overview

Table 4. Main API groups in the prototype.

6.5 Architectural Assistant Endpoint

The design blueprint includes an assist endpoint in addition to floor-plan generation. It accepts a user message and project context including project name, client, location, building type, plot dimensions, floors, and budget. The endpoint asks the same Llama model for short, practical architectural advice. This provides a second generative interaction mode, although it is not required for the primary floor-plan experiment.

6.6 Prototype Quality Observations

Repository inspection reveals several engineering details that should be addressed before a production deployment. The database connection contains a literal password, and authentication uses a literal placeholder secret key in both the controller and authentication utility. The Flask application runs with debug=True in the main block. The design controller prints a prefix of the API key when loaded. These are not research failures, but they are security and ability to repeat the experiment concerns that should be explicitly acknowledged.

6.7 Input Validation

The current generation path does not perform strong numeric validation before invoking the LLM. Plot dimensions and room count are accepted from the request and inserted into the prompt. A production-grade system should validate positive numeric values, upper bounds, string lengths, and special-request content before generation. This also improves the reliability of benchmark experiments because invalid inputs can be excluded before the model is evaluated.

7. Experimental Methodology

7.1 Research Design

The proposed evaluation follows a controlled comparative design. Every method receives the same architectural requirement record, and generated plans are evaluated using the same geometric validator. The key independent variable is the generation strategy: rule-based, unconstrained LLM, structured LLM, or the full proposed workflow.



7.2 Benchmark Construction

The primary dataset should be a requirement-level benchmark rather than an image-only dataset. Each sample contains a building type, plot dimensions, room requirements, special requirements, and a predefined constraint set. A reference floor-plan dataset such as CubiCasa5K can be used to inform room categories and realistic structural patterns, but it should not be described as OmniBuild training data unless the model is actually trained or fine-tuned on it.

Field	Description
Sample ID	Unique identifier
Building type	Residential, office, commercial, etc.
Plot width/length	Numerical dimensions
Room list	Requested rooms and counts
Special requirements	Parking, attached bathroom, etc.
Difficulty	Easy, medium, hard
Expected constraints	Boundary, overlap, presence, dimensions
Evaluation metadata	Seed/run and model configuration

Table 5. Proposed benchmark schema.

7.3 Dataset Difficulty

Easy cases should contain a small number of common rooms and generous plots. Medium cases should increase room count and introduce special requirements. Hard cases should stress the model through tight plots, multiple room types, and combinations of constraints. If irregular plot boundaries are not implemented, they should not be included as a claimed capability.

7.4 Data Splitting

A 60/20/20 split is recommended for prompt development, validation, and held-out evaluation. Because the system is not currently fine-tuned, these partitions should be described as development, validation, and test rather than training, validation, and test. Similar requirement templates should be grouped before splitting to reduce leakage.

7.5 Baseline 1: Rule-Based Layout

The first baseline should implement a deterministic placement strategy. It may use predefined room dimensions and sequential placement heuristics. The goal is to establish the performance of a simple non LLM method with strong geometric control.

7.6 Baseline 2: Unconstrained LLM

The second baseline should use the same LLM and input fields but omit the explicit boundary and overlap rules. It tests whether the observed validity comes from the model alone or from constraint based prompt engineering.

7.7 Baseline 3: Structured LLM

The third baseline should retain the output schema but reduce explicit constraint instructions. This isolates the effect of structured representation.

7.8 Proposed Configuration

The full configuration combines constraint based prompting, structured parsing, deterministic validation, visualization, and, if implemented for the experiment, regeneration. If regeneration is not enabled in the tested repository version, it should be reported as a proposed extension rather than a measured feature.

Method	LLM	Structured	Constraints	Validation	Repair
--------	-----	------------	-------------	------------	--------



Rule-based	No	Yes	Hard-coded	Yes	No
Single-shot LLM	Yes	Optional	No	No	No
Structured LLM	Yes	Yes	Limited	No	No
Omni Build	Yes	Yes	Prompt-level	Proposed/partial	Optional

Table 6. Experimental baselines.

7.9 Repeated Runs

LLM generation should be repeated for each test prompt when stochastic behavior is enabled. A recommended design is five independent runs for each of 100 held-out prompts, yielding 500 candidate layouts per method. The exact sample size should be reported according to actual execution.

7.10 Controlled Variables

- Use the same test prompts across all methods.
- Keep the model endpoint fixed within a comparison.
- Keep the output parser and geometric validator identical.
- Use the same definition of a valid plan.
- Archive raw model responses before manual inspection.
- Do not manually repair layouts before automated scoring.
- Blind human evaluators to the method identity.

8. Evaluation Metrics and Research Hypotheses

8.1 Constraint Satisfaction Rate

$CSR = \text{satisfied constraints} / \text{total constraints} \times 100$

Constraints can include boundary compliance, non-overlap, room presence, room count, dimension validity, and explicitly testable special requests.

8.2 Boundary Violation Rate

$BVR = \text{layouts with boundary violations} / \text{total layouts} \times 100$

Lower is better. A layout is boundary-invalid if at least one room violates the plot inequalities.

8.3 Room Overlap Rate

$OR = \text{overlapping room pairs} / \text{evaluated room pairs}$

The study should also report the percentage of layouts containing at least one overlap because pair-level averaging can conceal a small number of severely invalid plans.

8.4 Room Requirement Accuracy

$RRA = \text{correctly generated required room instances} / \text{total required room instances} \times 100$

8.5 Dimension Validity

$DV = \text{rooms satisfying predefined dimension rules} / \text{total rooms} \times 100$

Dimension thresholds must be fixed before testing. They should be justified using architectural references or a reference dataset and should not be adjusted after viewing test results.

8.6 Space Utilization

$SU = \text{total usable room area} / \text{plot area} \times 100$



Space use of space is not an independent definition of architectural quality. Excessively high use of space may reduce circulation or practicality, so it must be interpreted with other metrics.

8.7 Valid-Plan Rate

$$\text{VPR} = \text{fully valid plans} / \text{total generation attempts} \times 100$$

8.8 Latency

$$T_{\text{total}} = T_{\text{model}} + T_{\text{parse}} + T_{\text{validation}} + T_{\text{refinement}}$$

Latency should be separated into model response time, parsing time, validation time, and any refinement time. This allows quality-latency trade-offs to be studied.

8.9 Human Evaluation

Criterion	Scale	Question
Spatial organization	1-5	Does the arrangement appear coherent?
Requirement satisfaction	1-5	Does it address the requested spaces?
Practicality	1-5	Would it be a useful starting concept?
Visual clarity	1-5	Is the layout easy to understand?
Overall usefulness	1-5	How useful is the output for early design?

8.10 Hypotheses

- H1: Constraint-aware prompting increases constraint satisfaction compared with unconstrained LLM generation.
- H2: Constraint-aware prompting reduces boundary violations and room overlaps.
- H3: Structured output improves successful parsing and room requirement accuracy.
- H4: Deterministic validation increases the proportion of plans identified as valid.
- H5: Human evaluators assign higher usefulness ratings to validated, editable plans than to raw text generated plans.
- H6: Validation and iterative repair, if enabled, improve validity at the cost of increased latency.

8.11 Null and Alternative Hypotheses

For each directional hypothesis, the null hypothesis should state that the proposed method is not better than the comparison method. For example, H0 for H1 is $\text{CSR_OmniBuild} \leq \text{CSR_baseline}$, while H1 is $\text{CSR_OmniBuild} > \text{CSR_baseline}$. The final paper should report the statistical test, effect size, confidence interval, and practical difference rather than relying only on significance.



9. Experimental Design and Statistical Analysis

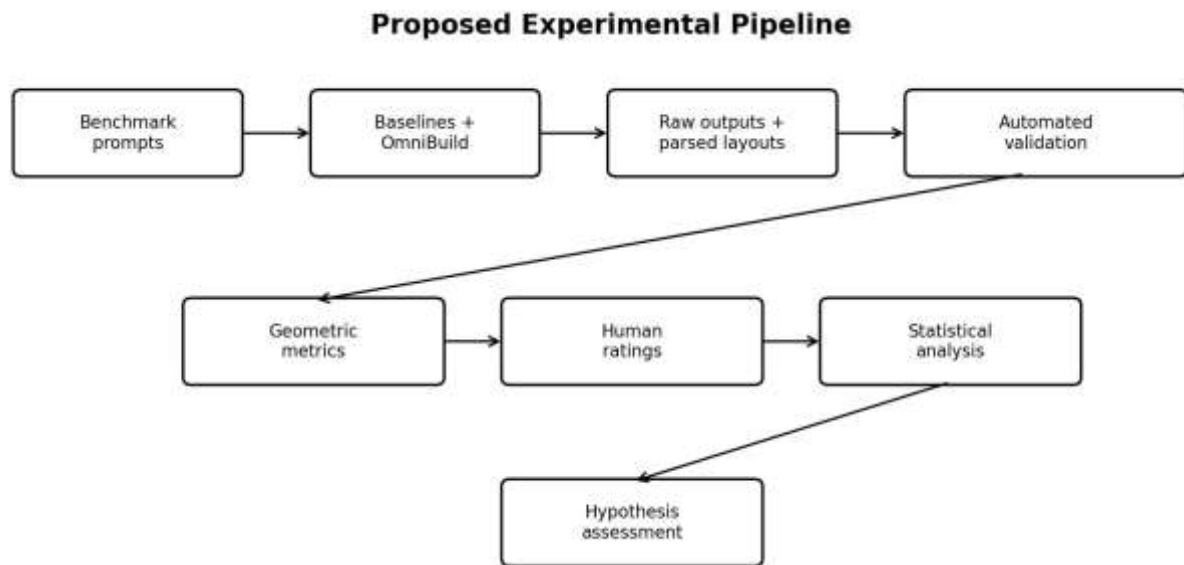


Figure 4. Proposed experimental pipeline for quantitative and human evaluation.

9.1 Experimental Procedure

1. Freeze the benchmark version and final prompt.
2. Generate outputs for every baseline using the same requirement records.
3. Store raw responses and timestamps.
4. Parse each response into the common room representation.
5. Run deterministic geometric and semantic validation.
6. Calculate primary metrics automatically.
7. Randomize and blind outputs for human evaluation.
8. Aggregate prompt-level results.
9. Run paired statistical tests and calculate effect sizes.
10. Perform error analysis on invalid outputs.

9.2 Statistical Tests

Paired tests are appropriate because the same requirement prompt is evaluated by multiple methods. If metric differences are about normal, a paired t-test may be used. If distributional assumptions are not met, the Wilcoxon signed-rank test provides a non-parametric alternative. For paired binary validity outcomes, McNemar's test can be considered.

The significance threshold should be defined before evaluation as $\alpha = 0.05$. However, significance alone does not establish practical usefulness. The final analysis should also report confidence intervals and an effect-size measure appropriate to the metric.

9.3 Multiple Comparisons

If many pairwise comparisons are performed across multiple metrics, the study should consider a correction procedure such as Holm adjustment. This reduces the risk of interpreting an isolated significant result as evidence of overall superiority.

9.4 Human Evaluation Protocol

Human evaluators should receive a short explanation of the task but should not be told which generation method produced each layout. They should see the same plot dimensions and requirement summary for



every evaluation item. A randomized order should be used. Evaluators can be drawn from architecture students, design professionals, or technically knowledgeable users, with participant composition reported transparently.

9.5 Inter-Rater Reliability

Because human judgments are subjective, inter-rater reliability should be calculated. Krippendorff's alpha is a flexible choice for multiple raters and ordinal ratings. The final paper should report the coefficient and explain how disagreements were handled.

9.6 Error Taxonomy

Code	Error type	Description
E1	Parser failure	Response cannot be converted to required fields
E2	Boundary violation	Room extends outside plot
E3	Room overlap	Two rooms intersect
E4	Missing room	Requested room absent
E5	Duplicate room	Unexpected duplicate
E6	Invalid dimensions	Room outside predefined range
E7	Door inconsistency	Invalid room/wall reference
E8	Low use of space	Area below benchmark threshold
E9	Ambiguous request	Cannot evaluate requirement reliably

Table 7. Proposed error taxonomy.

9.7 Reproducibility Checklist

- Model name and endpoint
- Experiment date
- Prompt version
- Dataset version
- Generation parameters
- Number of runs
- Raw outputs
- Parsed outputs
- Validation code/version
- Human evaluation form
- Statistical analysis script

10. Results Framework and Interpretation

No numerical results are reported in this version because the project repository does not contain a completed controlled benchmark with recorded ground-truth measurements. The tables in this section are therefore intentionally empty templates. This is a methodological choice rather than a missing section: reporting fabricated values would invalidate the research claims.

10.1 Main Quantitative Results

Method	CSR ↑	BVR ↓	Overlap ↓	RRA ↑	DV ↑	SU ↑	VPR ↑	Latency ↓
Rule based	-	-	-	-	-	-	-	-



Singleshot LLM	-	-	-	-	-	-	-	-
Structured LLM	-	-	-	-	-	-	-	-
Omni Build	-	-	-	-	-	-	-	-

Table 8. Main result template. Arrows show the desired direction.

10.2 Human Evaluation Results

Method	Organization	Requirements	Practicality	Clarity	Usefulness
Single-shot LLM	-	-	-	-	-
Structured LLM	-	-	-	-	-
Omni Build	-	-	-	-	-

10.3 Ablation Results

Config.	Constraint prompt	Structured	Validation	Repair	VPR	Latency
A	No	No	No	No	-	-
B	Yes	No	No	No	-	-
C	Yes	Yes	No	No	-	-
D	Yes	Yes	Yes	No	-	-
E	Yes	Yes	Yes	Yes	-	-

10.4 Interpretation Logic

If Omni Build produces higher CSR and lower BVR and overlap than the unconstrained baseline, this would support the claim that constraint based prompting contributes to geometric validity. If structured output improves parse success without improving geometric validity, the conclusion would be narrower: structure improves machine interoperability but not necessarily the model's spatial reasoning.

If validation improves VPR but increases latency, the result should be described as a quality-cost trade-off. If human usefulness improves despite modest geometric imperfections, the system may still be valuable as an ideation tool. Conversely, if human ratings remain low, future work should focus on layout quality rather than additional interface features.

10.5 Avoiding Unsupported Claims

- Do not state that Omni Build is more accurate without measured accuracy.
- Do not report a percentage improvement without a recorded baseline and denominator.
- Do not call the generated layouts architecturally correct without professional validation.
- Do not claim training on external datasets unless training was actually performed.
- Do not describe CV, AR, or ML modules as implemented research contributions unless corresponding code and experiments exist.



11. Security, Privacy, and Responsible AI

11.1 Credential Management

Repository inspection indicates that the prototype contains a literal database password in the database connection module and a placeholder JWT secret in authentication code. A production version should use environment variables or a managed secret service, rotate credentials, and ensure that secrets are excluded from version control. The design controller also prints a prefix of the LLM API key when the module loads; even partial credential disclosure should be removed.

11.2 Debug Mode

The Flask application starts with `debug=True` in its main execution path. Debug mode is useful during development but should not be enabled in a production deployment because it can expose internal information and create unsafe debugging behavior.

11.3 Authorization Boundaries

Project retrieval and updates are filtered by the authenticated user identifier, which is an important ownership check. Plot retrieval is public, while plot creation is protected. A future security review should verify authorization for every project-related resource and ensure that object identifiers cannot be used to access another user's data.

11.4 User and Client Privacy

Projects can contain client names, locations, budgets, and architectural drawings. Even when these fields are not formally regulated as sensitive data, they can be commercially or personally sensitive. Data minimization, access controls, audit logging, and retention policies should therefore be considered for deployment.

11.5 Prompt Injection

Special requests are incorporated into the LLM prompt. A robust deployment should treat user text as untrusted data and separate it from higher-priority system instructions. Length limits, sanitization, and schema validation can reduce the impact of malicious or malformed input.

11.6 Responsible Architectural Use

AI-generated floor plans should be explicitly labeled as conceptual assistance. They should not be used as a substitute for structural calculations, fire-safety planning, accessibility review, services design, local building regulations, or professional approval. Human review is not merely a user-experience feature; it is an important safety boundary.

12. Limitations and Threats to Validity

12.1 Construct Validity

The metrics proposed in this study capture measurable aspects of layout validity but cannot fully represent architectural quality. Circulation, daylight, ventilation, privacy, furniture fit, cultural preferences, structural feasibility, and aesthetics require additional analysis.

12.2 External Validity

The current generation interface is centered on rectangular plots and room rectangles. Results may not generalize to irregular sites, complex multi-floor buildings, commercial buildings, or different regional planning practices.

12.3 Model and Endpoint Dependence

Results depend on the selected model endpoint and inference configuration. LLM providers can update model behaviour over time. Exact endpoint identifiers, dates, and settings should therefore be recorded.



12.4 Prompt Dependence

Prompt wording can strongly influence model output. If prompt development is performed using test cases, the evaluation becomes optimistic. The final prompt must be frozen before the held-out test.

12.5 Parsing Dependence

The delimiter-based parser is less robust than formal structured output. A malformed model response can cause a valid semantic idea to be classified as a parser failure. Future experiments should separate generation errors from parsing errors.

12.6 Implementation Inconsistencies

The repository contains different default values between the backend generation function and the React form.

The backend defaults to three rooms and a 30-by-40-foot plot, while the UI initializes four rooms and a 40-by-50-foot plot. This does not prevent the UI from supplying explicit values, but it should be standardized for reproducible experiments.

12.7 Door Integration Limitation

The LLM output contains door relationships, but the current conversion to project drawing data initializes the imported door array as empty. Therefore, door semantics are generated but not fully preserved in the project import path. This should be treated as an implementation limitation.

12.8 No Ground-Truth Optimal Plan

Architectural requirements rarely determine a single optimal floor plan. Therefore, exact similarity to one reference drawing is not an ideal primary metric. Constraint satisfaction and human usefulness are more appropriate for this study.

12.9 Threats to Human Evaluation

Human ratings can be influenced by visual style, evaluator experience, and familiarity with floor plans. Blinding, randomization, clear instructions, and inter-rater reliability can reduce but not eliminate these effects.

13. Future Research Directions

13.1 Generate-Verify-Repair Loop

A direct extension is to place a deterministic validator between generation and final acceptance. When a room violates a boundary or overlaps another room, the validator can produce a structured error message. The LLM can then regenerate or repair only the affected parts. This architecture would convert soft prompt constraints into an explicit feedback loop.

13.2 Formal Constraint Solver

A stronger hybrid approach would let the LLM infer semantic requirements and adjacency relationships while a constraint solver determines coordinates. This separation would allow the model to focus on language and high-level planning while guaranteeing geometric conditions through optimization.

13.3 Graph-Based Intermediate Representation

The current representation is a collection of room rectangles and door relationships. Future work can introduce an adjacency graph in which nodes represent rooms and edges represent required relationships. This would connect Omni Build to graph-based research such as House-GAN and Graph2Plan.



13.4 Structured JSON or Schema-Constrained Output

Replacing delimiter-based parsing with a formal JSON schema would improve reliability and simplify validation. Required numeric fields could be type-checked before rendering. Invalid outputs could be rejected before reaching the visualization layer.

13.5 Building-Code and Accessibility Layer

Future versions can encode explicit rules for minimum dimensions, corridor widths, accessibility, door clearances, emergency exits, and local planning regulations. Such rules should be versioned and region specific.

13.6 Multi-Floor and Vertical Circulation

The project model already contains a floors field and drawing data supports multiple floor slots. A future generation model could produce coordinated plans across floors, including stairs, vertical shafts, and alignment constraints.

13.7 Preference Learning

The drawing editor can potentially provide preference signals. Repeated user modifications could reveal whether certain room arrangements are consistently preferred. With consent and privacy controls, these signals could support personalization or ranking of generated alternatives.

13.8 Cost and Performance Study

A deployment-oriented study should measure API cost, response time, validation time, retry count, failure rate, and user editing effort. A useful system is not only accurate; it must also be responsive and economically practical.

13.9 Professional Evaluation

Future validation should involve architecture professionals evaluating a blinded collection of generated layouts. Their assessment can address circulation, functional zoning, practicality, and overall design usefulness beyond the numerical geometry metrics.

14. Conclusion

This paper presents a detailed research-oriented analysis of Omni Build Intelligence, an implemented fullstack prototype that integrates large language model generation with architectural visualization and interactive project management. The core generative workflow accepts building type, room count, plot dimensions, and special requirements; constructs a constraint based prompt; invokes a Llama 3.3 70B model through Groq; parses the response into structured room and door objects; and renders the result through React-based 2D and isometric 3D interfaces.

The key technical idea is the structured intermediate representation. Instead of passing unconstrained model prose directly to the user, Omni Build extracts room geometry into explicit fields. This makes the output visualizable and provides a foundation for deterministic validation. The same representation can be converted into editable wall and label primitives and associated with persistent project data.

From a research perspective, the important question is not whether an LLM can generate something that looks like a floor plan. The stronger question is whether the system can satisfy measurable constraints reliably enough to be useful as an early-stage design assistant. The methodology proposed here addresses that question through a requirement-level benchmark, controlled baselines, geometric metrics, semantic compliance measures, human evaluation, ablation studies, and statistical analysis.

The current prototype should not be described as a fully autonomous architectural design system. It does not mathematically guarantee constraint satisfaction, and several broader capabilities mentioned in the project description are not supported by the inspected implementation. It also contains prototype-level



security and engineering issues that should be resolved before deployment. These limitations are not weaknesses to conceal; they define clear directions for future research.

The most promising architecture is a hybrid generate-verify-repair workflow. In such a system, the LLM would interpret requirements and propose a high-level arrangement, while deterministic geometry and domain rules would validate and repair the proposal. The human designer would remain responsible for final decisions. This division combines the flexibility of natural-language interaction with the reliability of explicit computational constraints.

Overall, Omni Build Intelligence provides a credible basis for studying natural-language-driven architectural concept generation. Its research value will ultimately depend on controlled experimental evidence. By keeping the implementation claims separate from measured findings, the proposed methodology provides a transparent foundation for such evaluation and for subsequent development of more reliable generative architectural tools.

15. Technical Discussion and Design Trade-offs

15.1 Natural-Language Flexibility versus Geometric Guarantees

The central architectural trade-off in Omni Build is between a flexible language interface and the strict requirements of geometric computation. Natural-language requirements allow users to express ideas without learning a formal layout language. However, the LLM is probabilistic, and its response can vary or violate numerical constraints. A deterministic geometry engine provides stronger guarantees but requires explicit rules. Omni Build therefore represents a hybrid design in which the LLM proposes a structured candidate and deterministic software is responsible for rendering and, in the proposed experimental extension, validation.

15.2 Structured Text versus Formal Schemas

The current parser uses delimiter-separated lines because this is easy to implement and visually inspect. Its weakness is sensitivity to formatting. A formal schema would make type checking and error reporting easier. This suggests a clear engineering progression: retain the semantic prompt but replace the delimiter contract with a schema-constrained response. The research comparison should then measure whether structured decoding reduces parser failures and improves valid-plan rate.

15.3 Generation Quality versus Response Time

Adding validation and regeneration can improve the probability that a returned layout is valid, but each additional model call increases latency and API cost. The system should therefore not optimize validity in isolation. A useful deployment objective can be framed as a quality-latency frontier, where the researcher compares the marginal improvement in validity against additional generation time.

15.4 AI Generation versus Human Editing

The drawing workspace changes the interpretation of model quality. An imperfect plan may still be valuable if it gives a designer a strong starting point and requires only minor edits. Conversely, a mathematically valid plan can be unhelpful if its functional organization is poor. For this reason, future experiments should measure editing effort, such as the number of wall movements, additions, deletions, or minutes required to reach an evaluator-approved state.

15.5 Prototype Scope versus Research Scope

The software contains several application modules beyond floor-plan generation, including authentication, dashboard operations, plot management, project persistence, and a marketplace interface. These features are useful for demonstrating an integrated product workflow, but they should not automatically be counted as research contributions. The research contribution is strongest where the



implementation introduces a testable interaction between language generation, structured geometry, visualization, and human refinement.

15.6 Failure Modes as Research Evidence

Invalid generations are not merely implementation defects; they are useful evidence about the limits of LLMbased spatial reasoning. A rigorous paper should preserve failed outputs, classify them, and analyze their causes. For example, repeated boundary violations may indicate that the prompt does not sufficiently anchor coordinate semantics, while repeated parser failures may indicate that a formal output schema is necessary. This turns failure analysis into a source of scientific insight.

15.7 Design Principles Derived from the Prototype

- Keep language understanding separate from geometric validation.
- Represent generated designs in a easy for software to read intermediate form.
- Preserve human editing after AI generation.
- Record raw outputs for ability to repeat the experiment.
- Use deterministic checks for constraints that can be expressed mathematically.
- Treat broader product features as supporting infrastructure unless independently evaluated.
- Expose uncertainty and validation failures instead of silently accepting invalid geometry.

16. Reproducibility and Submission Protocol

16.1 Versioning the Experimental Artifact

Before running the final benchmark, the repository should be tagged or copied into a fixed experimental version. The model endpoint, prompt text, frontend defaults, backend defaults, validation code, and benchmark file should all be versioned. This prevents later changes from making the reported experiment impossible to reproduce.

16.2 Run-Level Logging

Each generation should receive a unique run identifier. The log should contain the prompt identifier, method, model endpoint, timestamp, raw response, parser result, validation result, latency, and metric values. Human-evaluation records should use a separate anonymous identifier so that evaluator identity is not mixed with method identity.

16.3 Reproducible Metric Calculation

Metrics should be calculated from structured outputs using one shared evaluation script. Manual spreadsheet calculations should be avoided for primary results because they make it harder to audit individual failures. The evaluation script should export a row-level dataset from which the tables and statistical tests are generated.

16.4 Recommended Experimental File Structure

- experiments/benchmark.csv - requirement-level test cases
- experiments/prompts/ - frozen prompt versions
- experiments/raw_outputs/ - model responses
- experiments/parsed/ - normalized room/door objects
- experiments/metrics/ - row-level metric calculations
- experiments/human_eval/ - blinded evaluation records
- experiments/results/ - final tables and statistical outputs
- experiments/config.json - model and run configuration



16.5 Acceptance Criteria

A generated layout should be considered fully valid only when it passes all critical constraints defined before the experiment. If a weighted score is preferred, the weights and minimum threshold must be frozen before the test. Any layout failing the parser should be retained as a failure rather than manually repaired.

16.6 Final Paper Reporting Checklist

- Report the exact number of prompts and generations.
- Report the exact model endpoint and experiment date.
- Describe the prompt-development process without tuning on the test set.
- Provide the baseline definitions.
- Report primary and secondary metrics separately.
- Report confidence intervals and effect sizes.
- Describe invalid outputs and representative failure categories.
- Separate prototype features from research findings.
- State all limitations and unsupported capabilities.
- Archive enough artifacts for another researcher to reproduce the analysis.

References

- [1] A. Kalervo, J. Ylioinas, M. Häikiö, A. Karhu, and J. Kannala, "CubiCasa5K: A Dataset and an Improved Multi-Task Model for Floorplan Image Analysis," arXiv:1904.01920, 2019.
- [2] N. Nauata, K.-H. Chang, C.-Y. Cheng, G. Mori, and Y. Furukawa, "House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation," arXiv:2003.06988, 2020.
- [3] R. Hu, Z. Huang, Y. Tang, O. van Kaick, H. Zhang, and H. Huang, "Graph2Plan: Learning Floorplan Generation from Layout Graphs," in Proc. ACM SIGGRAPH, 2020, doi: 10.1145/3386569.3392391.
- [4] A. Grattafiori et al., "The Llama 3 Herd of Models," arXiv:2407.21783, 2024.
- [5] R. Hu et al., "House-GAN++: Generative Adversarial Layout Refinement Networks," arXiv, 2021.
- [6] J. Duggempudi et al., "Text-to-Layout: A Generative Workflow for Drafting Architectural Floor Plans Using LLMs," arXiv, 2025.
- [7] S. Qin, R. E. Weber, and X. Lu, "Tokenization Allows Multimodal Large Language Models to Understand, Generate and Edit Architectural Floor Plans," arXiv, 2026.
- [8] L. Lara et al., "Generative Floor Plan Design with LLMs via Reinforcement Learning with Verifiable Rewards," arXiv, 2026.
- [9] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
- [10] R. Szeliski, Computer Vision: Algorithms and Applications, 2nd ed. Springer, 2022.

Appendix A. Suggested JSON Representation

A future structured output contract can use a representation similar to:

```
{ "plot": { "width": 40, "length": 50 }, "rooms": [ { "name": "Bedroom", "x": 0, "y": 0, "width": 12, "height": 10 }, "doors": [ { "room 1": "Bedroom", "room 2": "Hall", "wall": "right" } ], "summary": "...". This is a methodological example, not an observed model response.
```

Appendix B. Project Evidence Map

Research claim	Repository evidence	Status
LLM floor-plan generation	backend/controllers/design_controller.py	Implemented



Constraint-aware prompt	design_controller.py	Implemented as prompt instructions
Structured room parsing	design_controller.py	Implemented
2D visualization	frontend/src/pages/GenerativeDesign.js	Implemented
Isometric 3D visualization	GenerativeDesign.js	Implemented
AI-to-wall conversion	GenerativeDesign.js	Implemented
Project persistence	project_controller.py + project routes	Implemented
Authentication	user_controller.py + auth.py	Implemented
Land plot filtering	plot_controller.py + Marketplace.js	Implemented
Computer vision	No corresponding core implementation identified	Not established
Augmented reality	No corresponding core implementation identified	Not established
Automated geometric validation	No complete validator identified	Research extension

Table 9. Implementation-to-research evidence map based on repository inspection.

Appendix C. Research Integrity Statement

The final submitted paper should preserve the distinction between implementation evidence and experimental evidence. Any number inserted into the Results section must be traceable to an experiment log. External datasets should be cited and used according to their licenses. Prior research should be paraphrased and cited rather than copied. The presence of a reference does not justify claiming that Omni Build implements the referenced method.