Transforming Natural Language into Software Architecture Diagrams Using LLMs

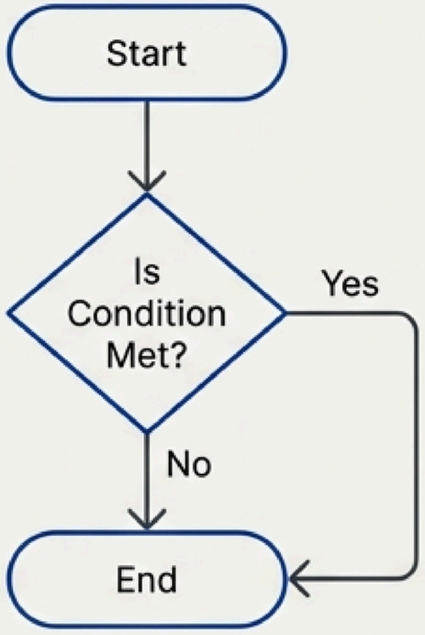
Abdul Qaadir
Student ID: SE/2020/036
BSc (Hons) Software Engineering
University of Kelaniya

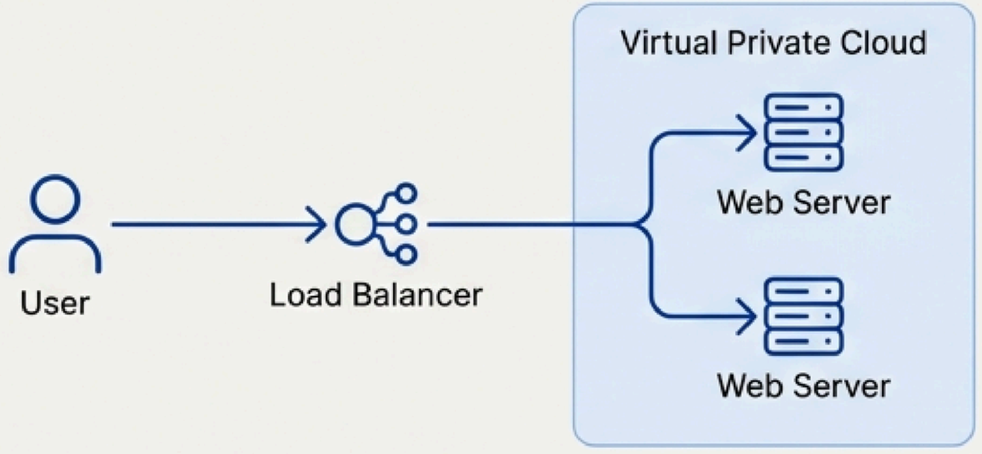
Supervisors
Dr. Isuru Hewapathirana | Professor, University of Kelaniya (Academic)
Dr. Srinath Perera | Chief Architect, WSO2 (Industrial)

Software diagrams are the essential language of system design.

Visual blueprints are essential for communication and planning in software engineering. They form the critical bridge between complex, abstract requirements and concrete technical implementation.

- Flowcharts
- Cloud Deployment Diagrams
- Entity-Relationship Diagrams (ERDs)
- Class Diagrams





The translation from human language to visual logic is a slow and error-prone process.

The Manual Bottleneck

- Creating diagrams manually from natural language requirements is time-consuming and prone to human error.
- Existing tools often involve steep learning curves and rely on tedious manual drag-and-drop interaction.
- This friction leads to inconsistencies and slows down development cycles.

Our Research Goal

- To automate the generation of accurate, editable, and standard-compliant diagrams using Artificial Intelligence.

Our journey involved testing three distinct hypotheses for automated generation.

Phase 1: Direct Image Generation

The initial idea of using Vision-Language Models directly.

Phase 2: Code Generation

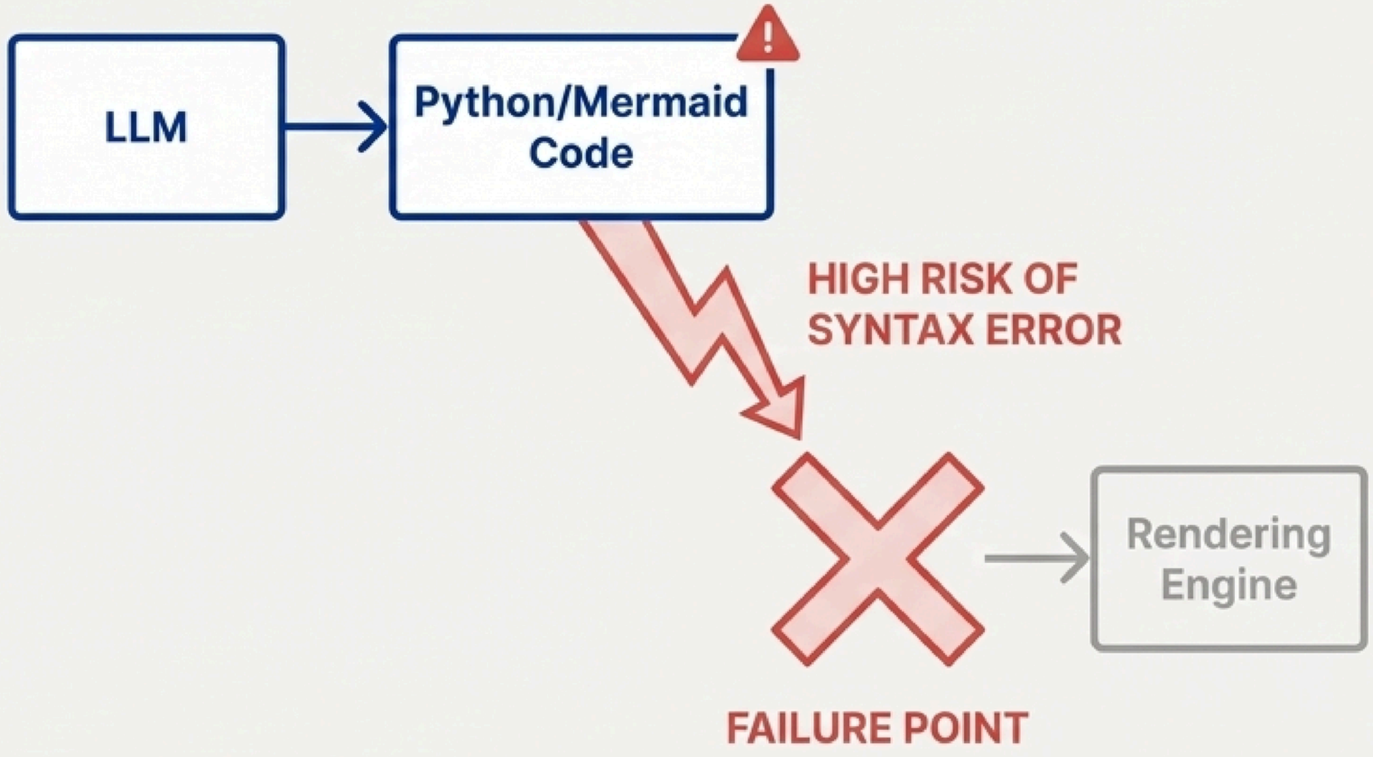
A more refined attempt to generate diagram-as-code.

Phase 3: Intermediate Representation

The successful approach focusing on a structured data format.

The core problem is syntax. LLMs excel at semantics, but brittle code wrappers demand perfection.

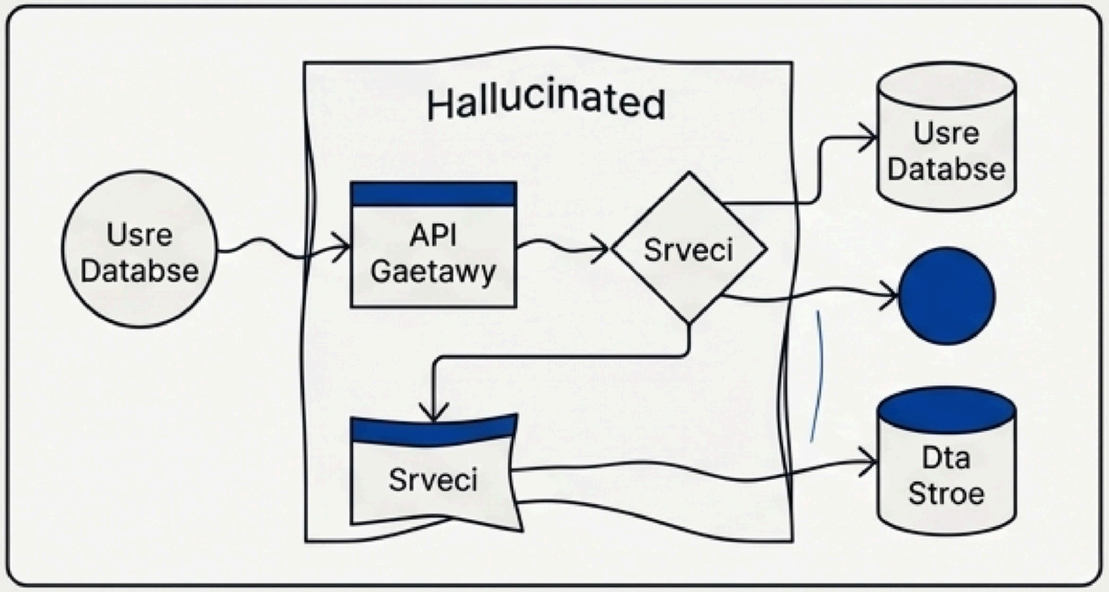
- Most AI diagramming tools rely on generating code wrappers for libraries like Mermaid.js, PlantUML, or Graphviz.
- LLMs frequently introduce minor syntax errors that cause the entire rendering engine to fail catastrophically.
- Furthermore, these libraries often lack support for specific, complex enterprise diagram components.



Initial approaches failed because VLMs/LLMs are not trained on large amounts of diagram images or diagram-generation code.

Path 1: Direct Image Generation

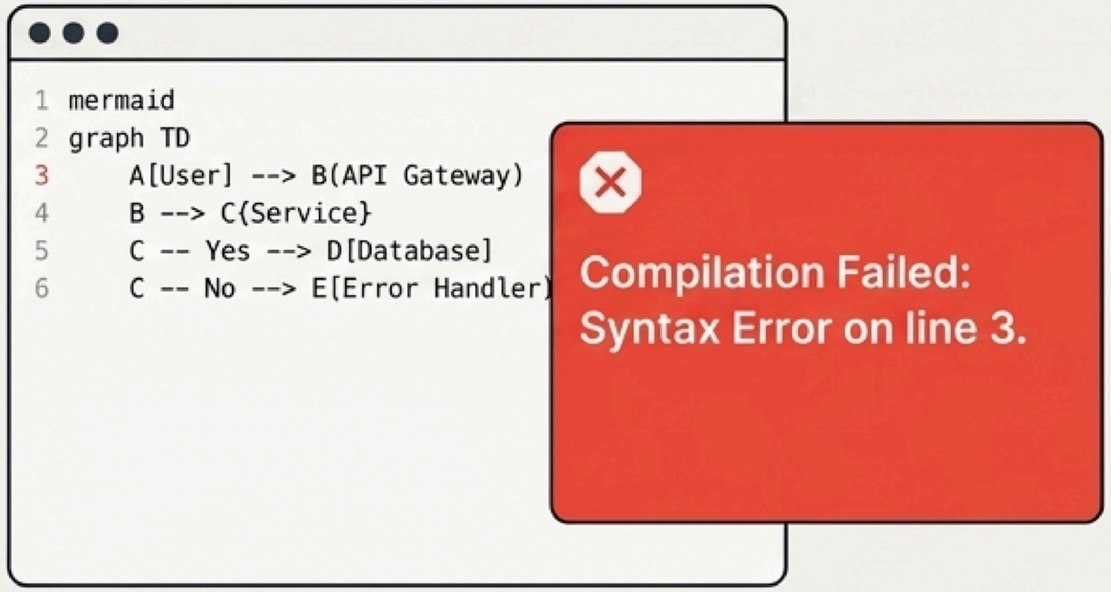
Hypothesis: Use Multimodal LLMs to generate diagram images directly.



⚠ Visually Unreliable

Path 2: Code Generation

Hypothesis: Use LLMs to generate diagram-as-code (e.g., Python, Mermaid).

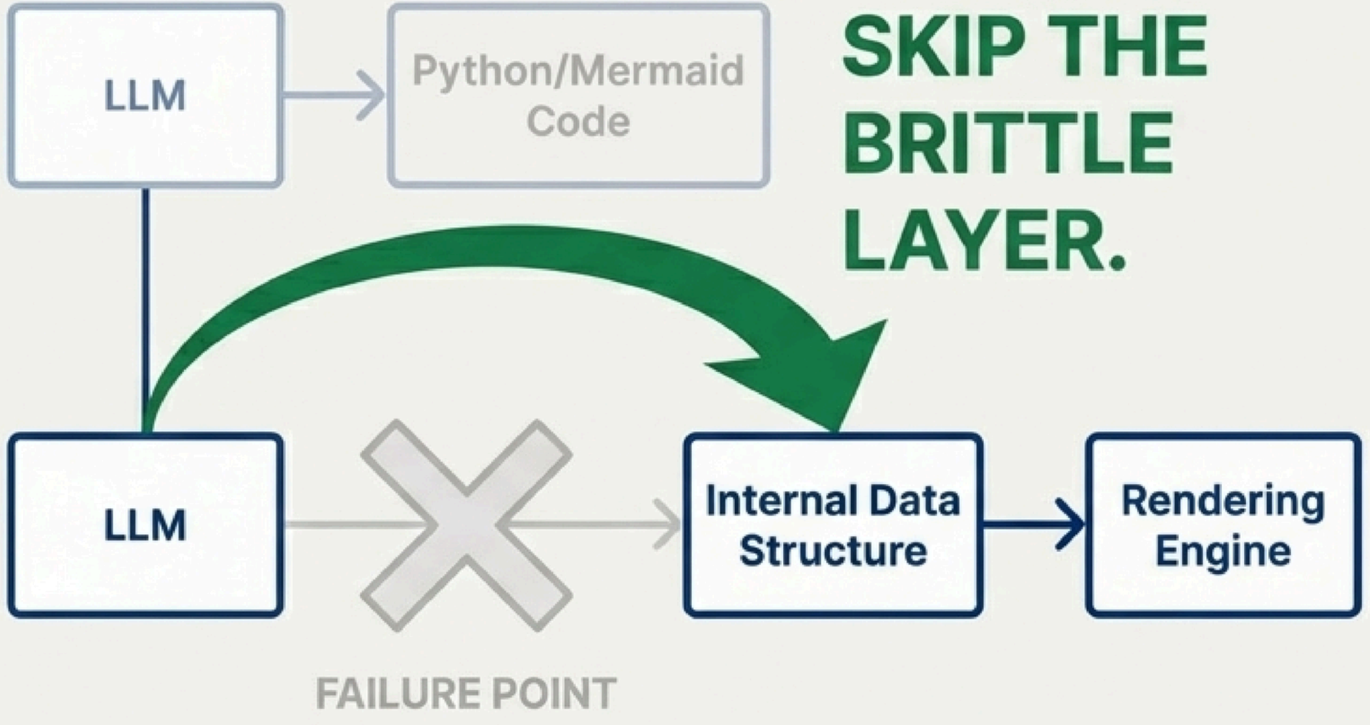


⚠ Syntactically Brittle

The breakthrough: Stop generating code. Start describing the system's structure.

The question became: Why force an LLM to generate complex Python/Mermaid syntax, only to have a rendering engine parse it back into a data structure?

The realization was that we could skip the brittle syntax layer entirely.

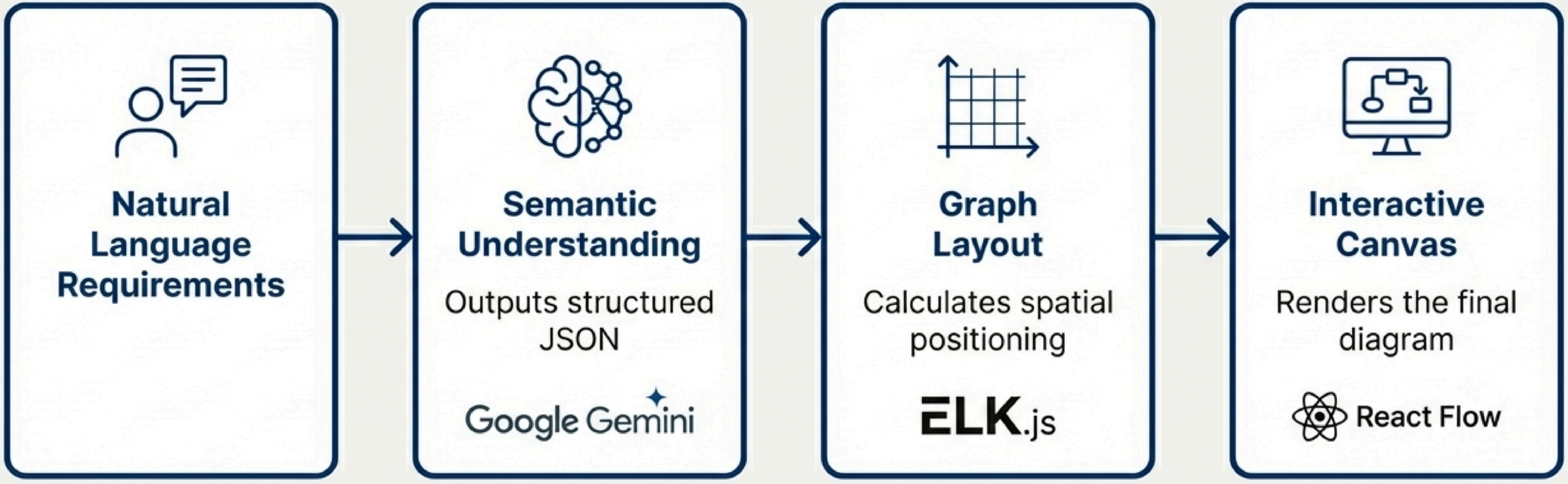


Designed a schema to represent any diagram as a simple data structure.

- Instead of generating code, we define a universal JSON schema.
- This structure remains consistent, using only Nodes and Edges, regardless of the diagram type—from a simple flowchart to a complex cloud architecture.

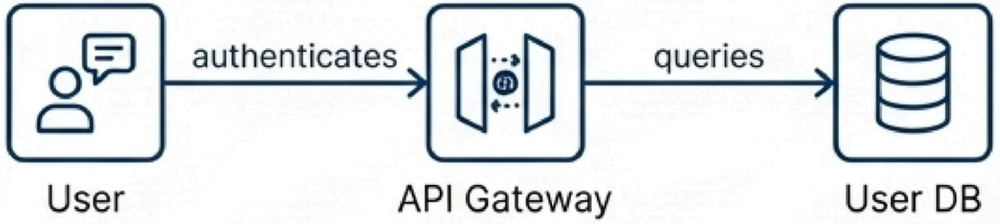
```
{
  "nodes": [
    { "id": "api_gateway", "label": "API Gateway",
     "type": "service" },
    { "id": "user_db", "label": "User Database",
     "type": "database" }
  ],
  "edges": [
    {
      "source": "api_gateway", "target": "user_db",
      "target": "user_db",
      "label": "queries"
  ] }
}
```

New pipeline decouples semantic understanding from visual rendering.



The JSON-first approach successfully generates interactive, editable diagrams.

A user authenticates through an API Gateway, which queries a User DB.



Live Demos

Experimental Build (NeuroFlow): https://abqaadir.github.io/NeuroFlow/

Production Candidate (SysArchitect): https://abqaadir.github.io/sysArchitect/

Our roadmap includes rigorous evaluation and enhancing human-AI collaboration.

Evaluation Strategy

Acknowledge the lack of a standardized dataset mapping natural language requirements to technical diagrams.

1. **Dataset Creation:** **Create** custom dataset of requirements and ground-truth diagrams.
2. **Automated Verification:** Use a superior Multimodal Model (e.g., Gemini 3 pro) to **Verify** generated diagrams against requirements.
3. **Human Verification:** Conduct expert reviews to assess accuracy and layout logic.

Future Work

- **Prompt Engineering:** Experiment with ‘Chain-of-Thought’ and ‘Few-Shot’ prompting for complex enterprise architectures.
- **Human-in-the-Loop Design:** Allow users to edit the generated JSON/diagram in real-time and feed corrections back to the model for session-based context retention.

Related Research Papers

01. Structuring the Output of Large Language Models

02. Executable Code Actions Elicit Better LLM Agents

03. Structured Chain-of-Thought Prompting for Code Generation

04. Rigorous Evaluation of Large Language Models for Code Generation

05. Synergizing Reasoning And Acting in Large Language Models

06. Towards Automatically Extracting UML Class Diagrams from Natural Language Specifications

07. Model Generation with LLMs: From Requirements to UML Sequence Diagrams

08. DiagrammerGPT: Generating Open-Domain, Open-Platform Diagrams via LLM Planning

09. AutomaTikZ: Text-Guided Synthesis of Scientific Vector Graphics with TikZ

Thank You