DRAWING

WITH LLMS

Context of the problem

Goal: Generate SVG images from text prompts that describe a scene or object.

Input: Natural language prompt (e.g., "A red circle inside a blue square").

Output: Valid SVG code that visually represents the prompt.

System objective

Build a model that

✓ Receives a text prompt

✓ Outputs SVG code that visually represents the described image

✓ Prioritizes semantic and visual alignment with the description

Input/Output constraints

Input:

- A single, natural language prompt describing an image
- 200 characters max

Output:

- A self-contained SVG string (valid XML format)
- 10000 byte limit
- No external dependencies

SVG fidelity score (Evaluation metrics)

- Assesses how closely the generated SVG resembles the intended image described by the text prompt
- Visual accuracy (does the image “look right”)
- Structural correctness of SVG elements (shapes, positions, relationships)
- Semantic alignment with the original text prompt

Sensitivity and chaos

⚠️ Sensitivity Factors

* Small changes in the prompt wording can cause large shifts in SVG layout or shape count

- Model's tokenization may emphasize irrelevant words, altering output

🌪️ Chaos Factors

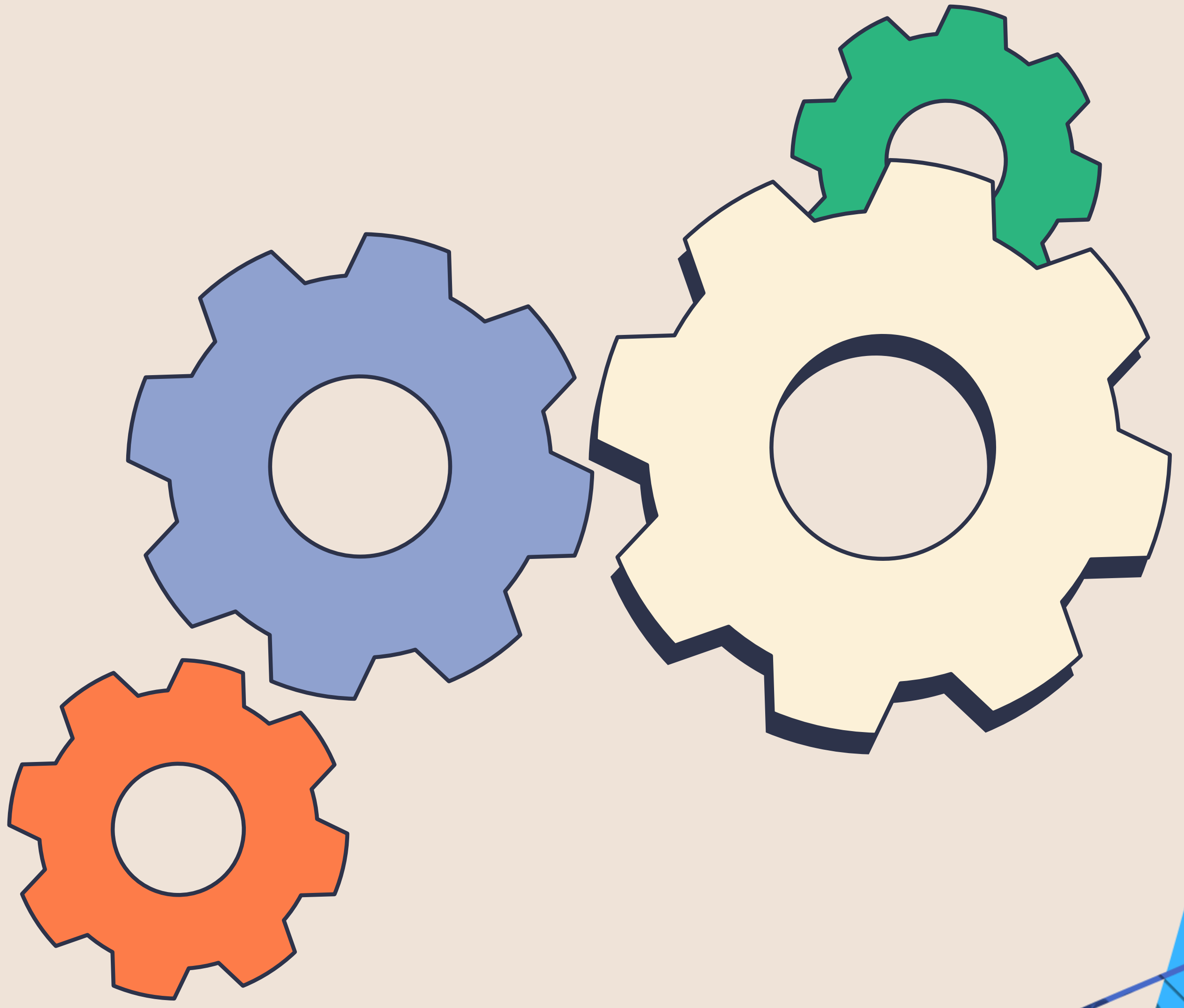
- LLMs may hallucinate SVG tags or structure (e.g., invalid or unrenderable code)
- Overlaping.

Functional requirements

The system must respect the contest restrictions and generate functional SVG images from text.

Non-functional requirements

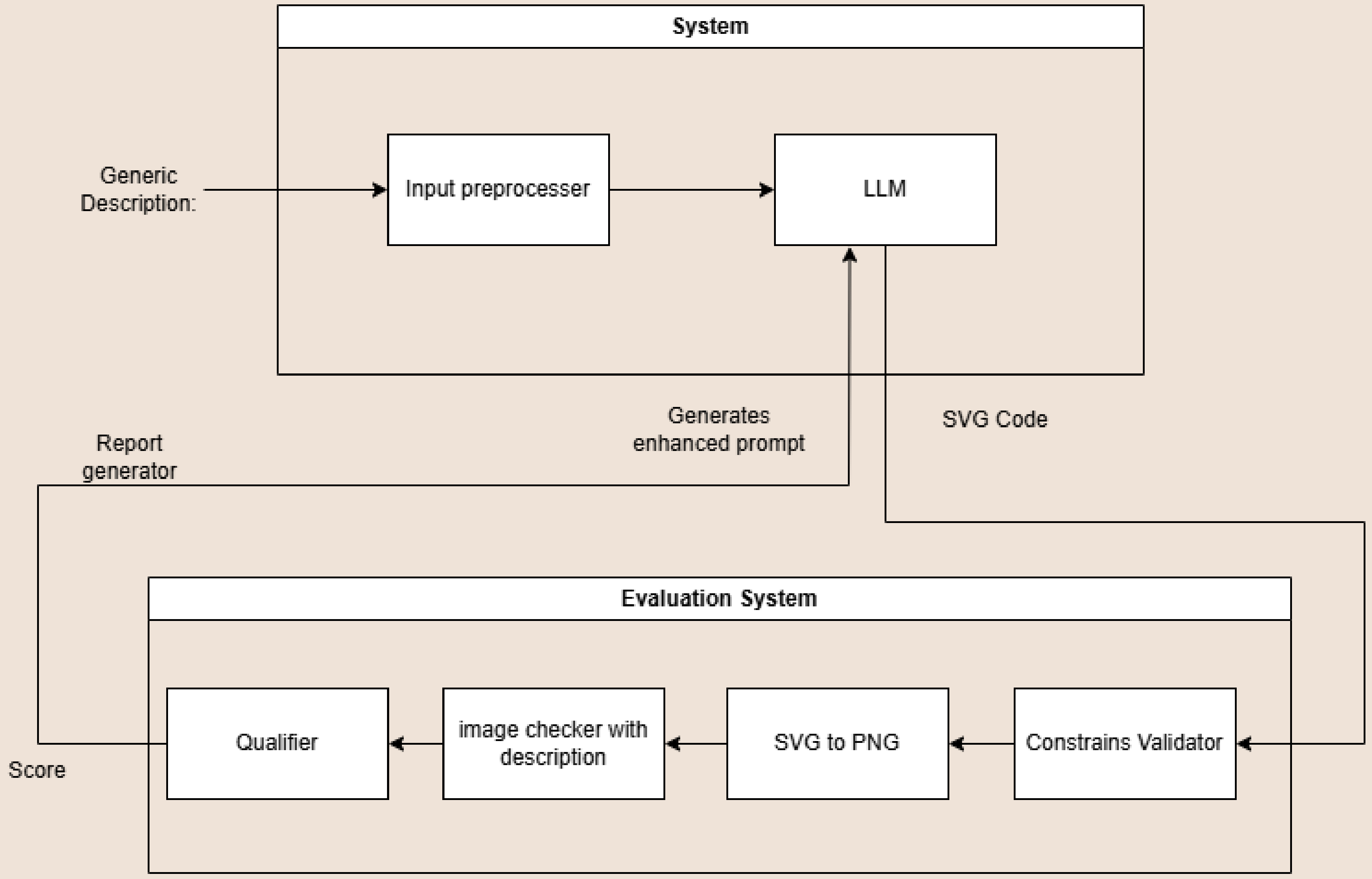
The system must be fast, reliable, efficient and modular.



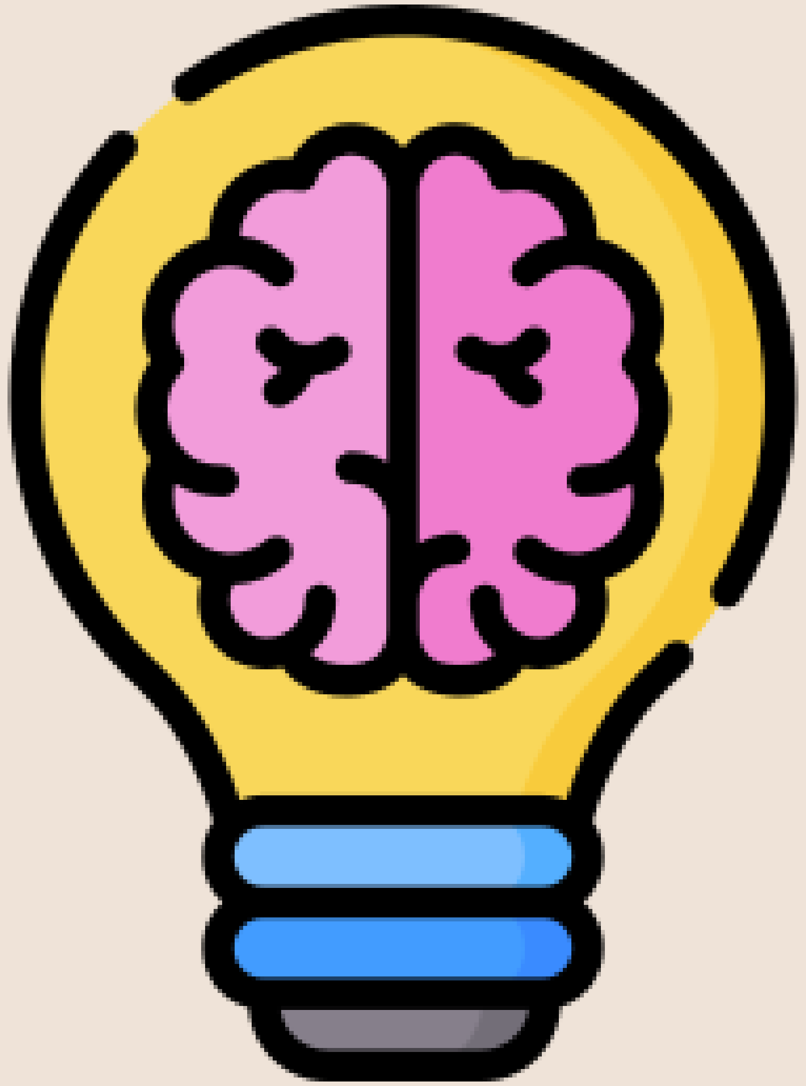
Main components

We divide the system into four parts: Input Handler, Prompt Engineer, LLM Generator and a Feedback Loop. Each performs a key function and communicates with the others.

Architecture diagram and flow



Principles applied

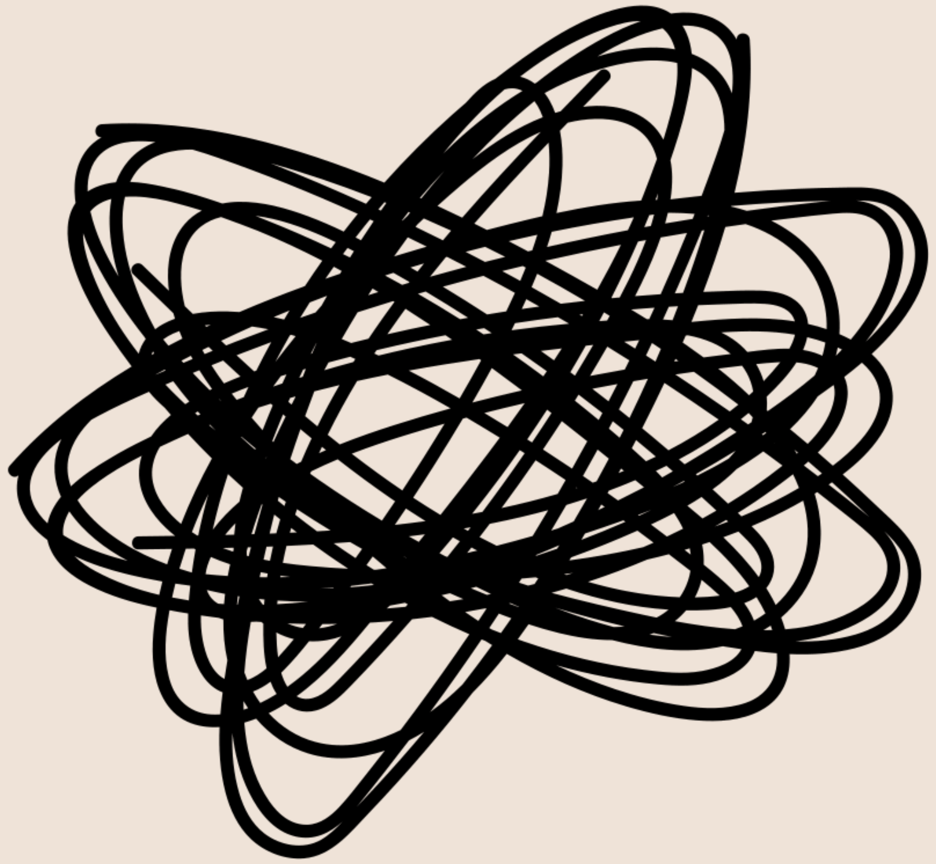


We use systems engineering principles: modularity to interchange parts, separation for functional clarity, and standardization of interfaces to integrate everything.

Strategies against sensitivity

Prompt engineering to avoid ambiguities and improve the understanding of the LLM from the start.

Chaos Mitigation



We implement progressive learning and detect error patterns and use them to fine-tune the model and reduce randomness in SVG outputs.

Technologies

Language: Python

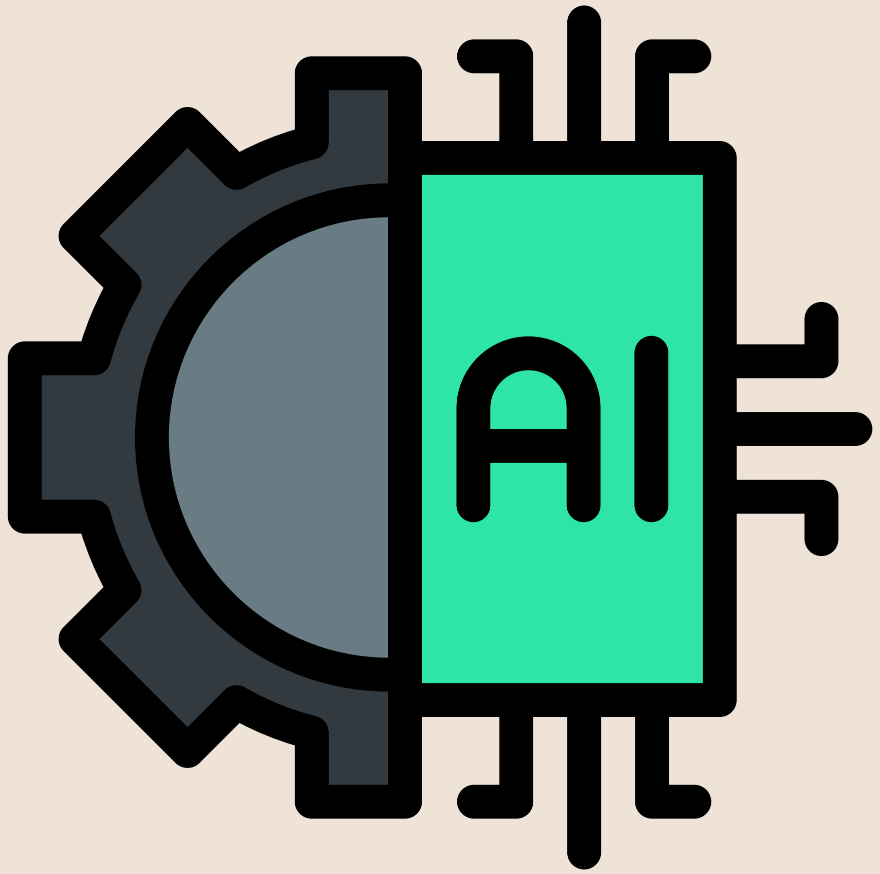
Hugging Face for LLMs

NumPy for vector operations

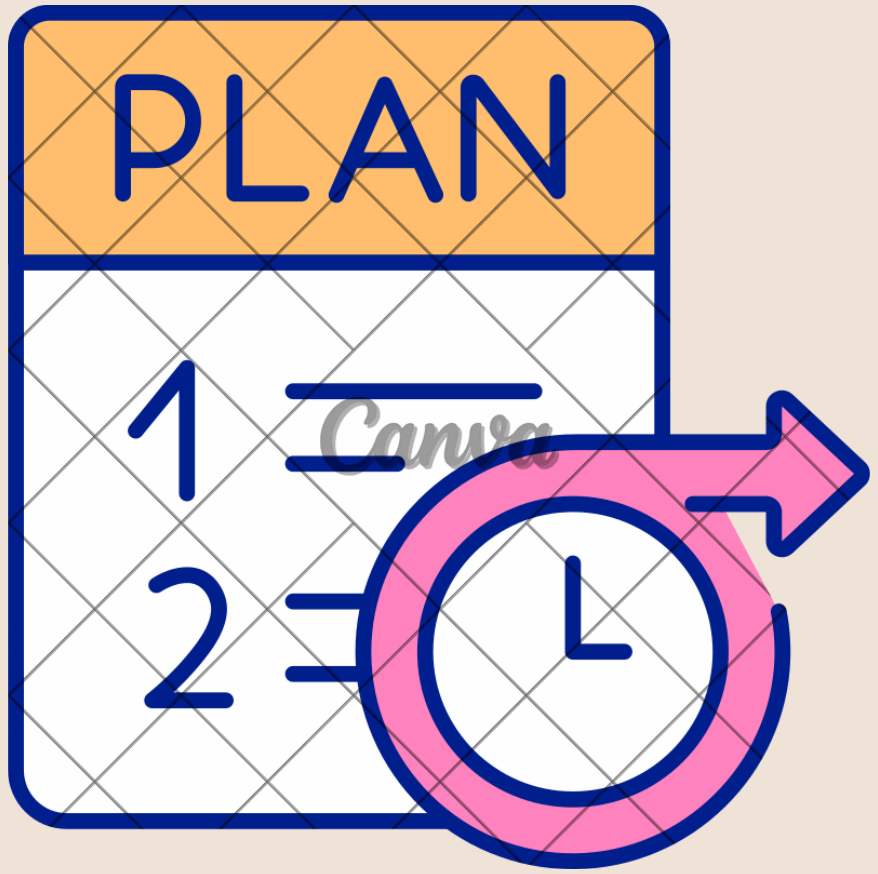
Jupyter as the development environment.

LLM Models

Open source,
lightweight, and
compatible with
environments without
internet access

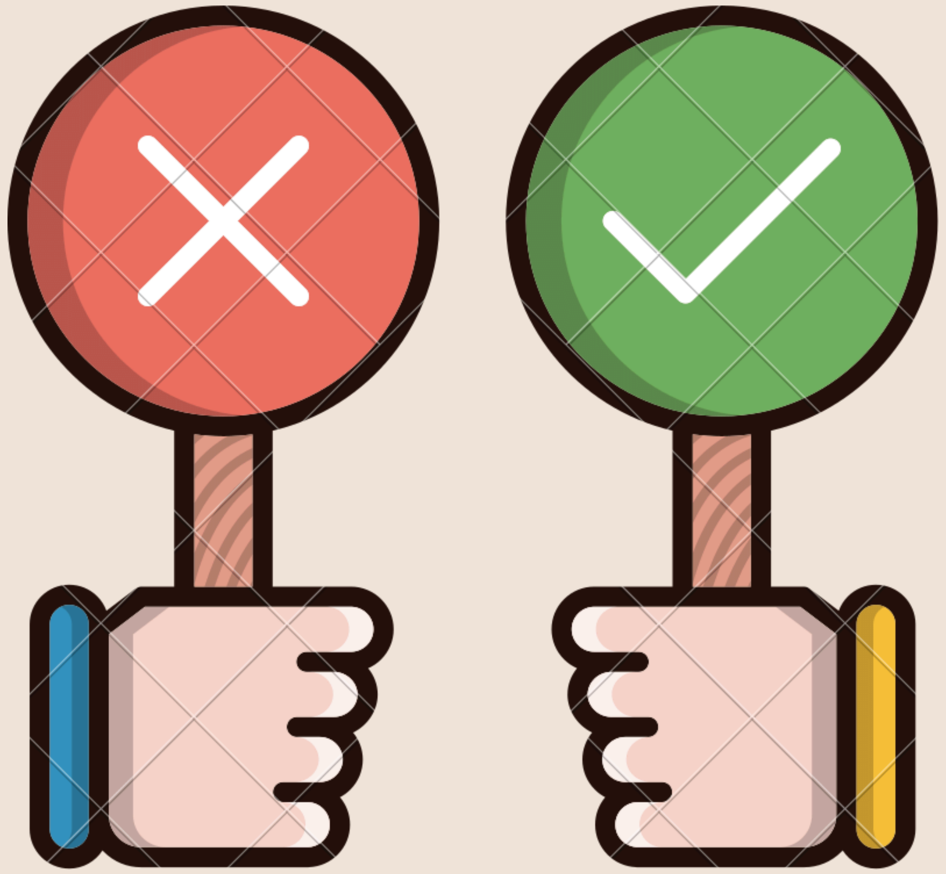


Plan development



1. Set up the basic infrastructure.
2. Adjust and optimize prompts.
3. Test and improve performance.

Testing and Feedback



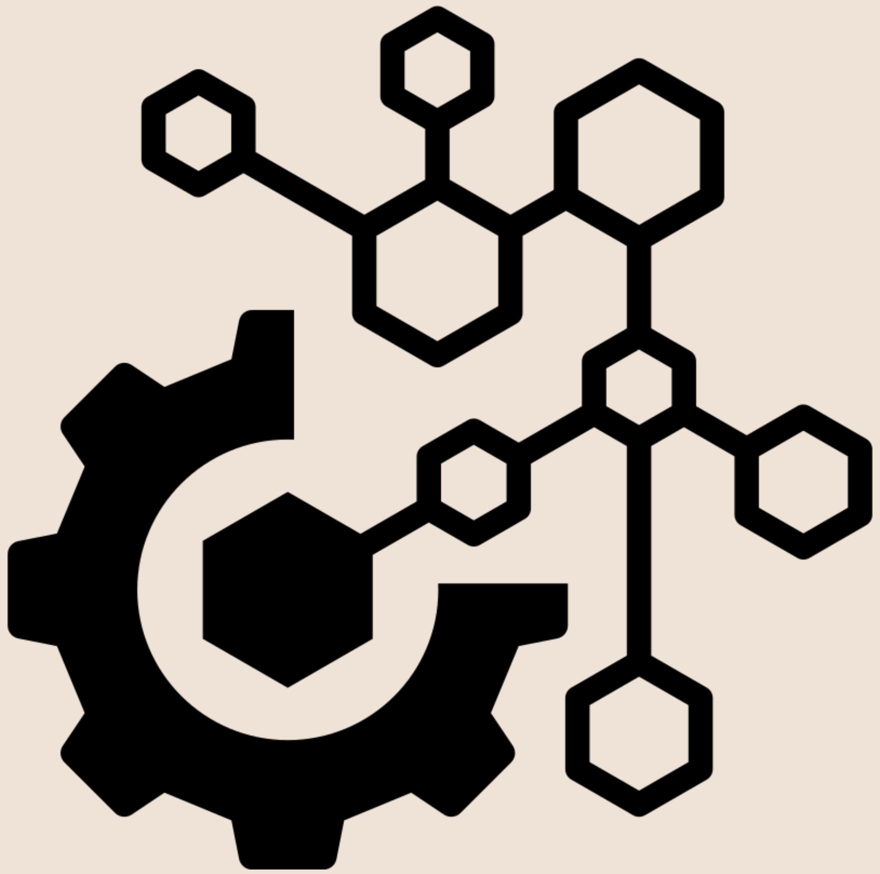
Performance testing before assessment and feedback during assessment

System adaptability

Change components easily

Incorporate human or machine feedback to retrain the model.

Possible Results



A well design LLM system

Underfing

Overfitting

Conclusion

System capable of transforming text into SVG images

Modular architecture and feedback

The power of LLMs

Thanks for your attention

Credits

Team: Daniel, Carlos y José.

Greatings to Carlos Andres Sierra.

Information used based on Kaggle and Hugging Face sources.