VAASTHU VISION AI

A Case Study on Building a Domain-Specific Expert Chatbot Project

1. EXECUTIVE SUMMARY:

1.1 Project Overview

Vaasthu Vision AI is a domain-specific conversational AI system designed to answer architectural and Vaasthu-related queries using principles of Vaasthu Shasthra. This chatbot is powered by a Retrieval-Augmented Generation (RAG) architecture built using LangChain, Qdrant vector database, HuggingFace embeddings, and Groq's LLaMA3 LLMs.

1.2 Final Outcomes

- A robust chatbot that answers Vaasthu questions in 2–4line expert recommendations.
- High accuracy through domain-restricted data and structured prompt engineering.
- Fast semantic retrieval using Qdrant with custom metadata tagging.
- UI development using Bolt AI with some custom modifications.
- FastAPI backend for API communication and future scalability.

1.3 Key Innovations

- Conversion of rigid JSON data into rich-text rules optimized for semantic search.
- Multi-prompt experiments including emotional and structured templates.
- Confidence thresholding and keyword routing for hybrid intent handling.
- Use of a finalised prompt which combines brevity and accuracy.

1.4 Timeline & Investment

- Duration: 8 days (June 27 – July 4, 2025)
- Iterations: 6 experimental versions
- Tools: LangChain, Groq, HuggingFace, Qdrant, Streamlit, FastAPI.

---

2. TECHNICAL ARCHITECTURE DEEP DIVE

2.1 Data Preparation

- Initial Format: JSON with 40 Vaasthu elements and 19 features per rule.
- Final Format: 350+ high-quality text-based rules separated by ---RULE_START:{id}--- and ---RULE_END---.
- Metadata: zone, rule_id, category.

Sample Converted Rule:

```
{
  "page_content": "Kitchen Placement According to Vaasthu...",
  "metadata": {
    "zone": "KITCHEN",
    "rule_id": "001",
    "category": "PLACEMENT"
  }
}
```

2.2 Vector Store Setup

- DB: Qdrant (Docker container)
- Embeddings: all-MiniLM-L6-v2 (via HuggingFace)
- Chunking Strategy: Each JSON rule converted into 8–10 granular rules for better granularity.

2.3 Retrieval-Augmented Generation (RAG)

- Framework: LangChain
- Retrieval: QdrantVectorStore (similarity search, k=3)
- LLM: LLaMA3-8B-8192 via Groq API
- Prompting: Custom directive-style prompt targeting concise expert answers

---

3. Development Journey & Problem-Solving

Day 1: June 27, 2025

- Explored RAG vs Fine-tuning.
- Rejected Fine-tuning due to format mismatch (JSON ≠ Q&A pairs) and insufficient volume (need 5K+, had 40).
- Decision: RAG with hybrid document-text rules + embeddings.

Day 2: June 28, 2025

- Attended an event at Draper Startup House where I met Gen AI developers
- Received suggestions on BAAI embeddings and chunk sizes.
- Converted JSON → 350+ text rules.

Day 3: June 29, 2025

- Cross-checked all rules with domain expert (grandfather).
- Created generate_qdrant_data.py, qdrant_setup.py, and test_qdrant_search.py.

- First functional similarity search using Qdrant.
- Implemented rag_pipeline.py using LLaMA3 with Prompt1.

Day 4: June 30, 2025

- Copied the old project as new one (vva) for clean structure.
- Docker setup and Qdrant initialization.
- Verified end-to-end pipeline (data → Qdrant → RAG → Streamlit).

Day 5: July 1, 2025

- Tried adding intent_classifier() and fallback logic.
- Added direction-based rules (e.g., northeast should contain and should not etc…).
- Caused hallucination + conflicting results → Reverted.
    - The hallucination is because of data quality and duplicated query matching.
- Decision: Remove fallback + directional rules for now.

Day 6: July 2, 2025

- Tested BAAI embeddings + LLaMA3-70B → still hallucinated.
    - Since the text rule content is around 3 to 4 lines in dataset, but with the prompt that supposed to generate a document kind of answer initially, along with the high parameter model made giving completely hallucinating answers.
- Compared Prompt1 and Prompt2 results using ChatGPT.
- Combined strengths → Prompt3 → Prompt4 (final).
- Created public-facing UI using Bolt AI.

Day 7: July 3, 2025

- Tested output determinism with (temperature = 0, top_p = 1) since I was faced a problem that every time I click "generate or Ask" button for a same query , it started giving different answers ever time which all were wrong most of the times.
- Finalized Prompt4 and deployed minimal 2–4line version.
- FastAPI backend connected successfully to website.
- Deployment to Render failed due to dependencies.

---

4. Experimental Analysis

4.1 Prompt Evolution

- **Prompt1:** Warm, poetic, and structured. Good for UX.
- **Prompt2:** Clear, emotionally supportive sections. Too long.
- **Prompt3:** Mixed tone. Balanced.
- **Prompt4:** Directive, concise, and production-ready. Final choice.

Final Prompt:

Prompt template for RAG (Vaasthu-specific)

template = """

You are VaasthuGPT™, an expert in Vaasthu Shastra. Answer the user's home-related question clearly and briefly — in 2 to 4 lines. Do not include background explanation, emotional tone, remedies, or follow-up prompts. Just give the direct Vaasthu answer to the question.

📌 CLIENT QUESTION:

{question}

📖 CONTEXT:

{context}

🪔 FINAL ANSWER:

Give a short and clear Vaasthu-based recommendation in 2 to 4 lines only.

"""

4.2 Models & Embeddings

Component	**Choice**	**Alternatives Tried**	**Verdict**
LLM	LLaMA3-8B (Groq)	LLaMA3-70B	8B was stable & fast
Embeddings	MiniLM-L6-v2	BAAI and other.	MiniLM had fewer hallucinations
VectorDB	Qdrant	Chroma	Qdrant's hybrid search performed well

4.3 Performance Observations

- Directional rules degraded accuracy due to ambiguity.
- Emotionally rich prompts generated fluff or hallucinations.
- 2–4line strict directive format performed best among the all the variations I tested.

---

5. Implementation Insights

Best Practices

- Keep vector DB input as clean, rich-text chunks with minimal overlap.
- Use structured metadata for better retrieval filtering.
- Always validate hallucinations manually with domain expert.

Pitfalls to Avoid

- Avoid over-complicating fallback logic unless intent detection is robust.
- Don't mix multiple styles of rules (e.g., directional vs structural) until those are having same format as previous rules had and we are sure about the data quality.

Optimization Strategies

- Use minimal prompt with strict output constraints.
- Ensure embedding model aligns with text tone and complexity.

---

6. RESOURCE APPENDIX:

Tools & Libraries

- LangChain, qdrant-client, HuggingFaceEmbeddings, ChatGroq
- Docker for Qdrant
- Streamlit for UI, FastAPI for backend

Key Scripts

- generate_qdrant_data.py → Converts RULE_START/END text files into JSON with metadata
- qdrant_setup.py → Uploads to Qdrant DB
- rag_pipeline.py → Final RAG with LLaMA3-8B-8192

Troubleshooting

- ❌ Docker not running → Use docker ps to check
- ❌ Embedding mismatch → Ensure consistent model across upload + retrieval
- ❌ Output inconsistency → Set temperature=0, top_p=1 (but testing the output is mandatory with the different parameters)

Testing & evaluation results

- **Vaasthu Queries:** Handled standard and complex Vaasthu questions accurately with short, relevant answers.
- **Partially Related Queries:** Managed edge cases (e.g., pooja room near bathroom) with context-aware, balanced responses.
- **Irrelevant Inputs:** Responded sensibly to casual queries (e.g., “What is 2 + 2?” → “4”) using fallback logic.
- **RAG Function Verified:** Compared responses from LLM and chatbot to confirm vector DB retrieval was working as intended.
- **Every Fix Tested:** After each change (prompt update, fallback removal, data cleanup), performance was revalidated with real queries.

INTELLIGENT FALLBACK HANDLING FOR IRRELEVANT QUERIES:

One of the major breakthroughs in **Vaasthu Vision AI** was implementing a smart fallback mechanism to handle non-Vaasthu queries effectively — a feature that significantly improved the system's reliability and user experience.

The Problem

Initially, the system’s knowledge base only contained Vaasthu-related rules stored in a vector database. This led to issues when users asked unrelated or casual questions. Since vector similarity search attempts to find the closest match, even non-Vaasthu queries like “What is your name?” or “2 + 2?” would retrieve unrelated Vaasthu rules, resulting in confusing or incorrect answers saying, “Your kitchen should locate at the southwest corner” which is untrue.

THE SOLUTION:

To solve this, I designed a custom **fallback routing system** that intelligently decides how to handle each query. The core logic consists of three elements:

1. Critical Keyword Override:

A list of essential Vaasthu-related keywords was created. If any of these appear in the user query, the system forcibly routes the query to the **RAG chain**, ensuring a relevant answer from the vector database.

2. Similarity Score Thresholds

Every incoming query undergoes similarity scoring using Qdrant:

- If the top match score ≥ 0.78, it's considered a high-confidence query and sent to the RAG system.
- If score is between 0.60 and 0.78, the system politely declines with a message:

  ❌ "Sorry, I have no idea about the query you asked."

- If score < 0.60, it's treated as unrelated or casual.

3. LLM-Based Fallback Chain

For queries that are not Vaasthu-related and have low similarity scores, a separate fallback LLM chain is triggered. It uses a **friendly prompt** to respond naturally to casual inputs like greetings, jokes, or random messages — without attempting to provide Vaasthu guidance.

Example fallback prompt:

fallback_prompt = PromptTemplate.from_template("""

You are a friendly assistant. Reply naturally to casual or random messages like greetings, small talk, or gibberish. Do NOT answer any Vaasthu or architecture related question. Just say: "Sorry, I can answer Vaasthu-related questions."

User: {query}

AI:""")

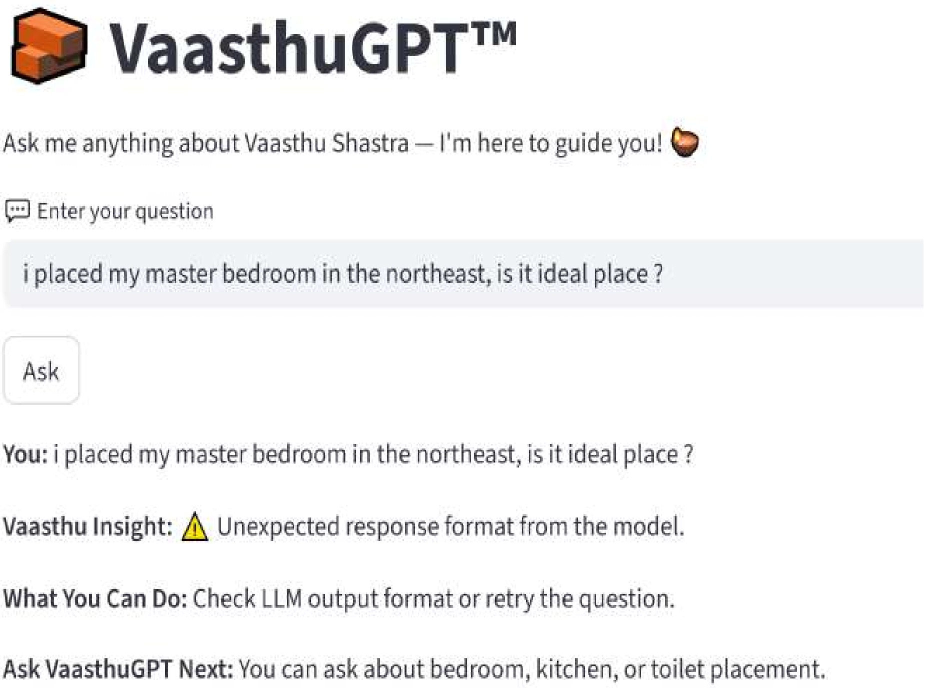
Real-World Impact

- ✅ Avoided **hallucinations** on unrelated input
- ✅ Ensured domain **consistency** by rejecting non-Vaasthu queries cleanly
- ✅ Enhanced user **trust** through predictable and contextual behaviour
- ✅ Maintained **accuracy** even when handling irrelevant, casual, or confusing queries

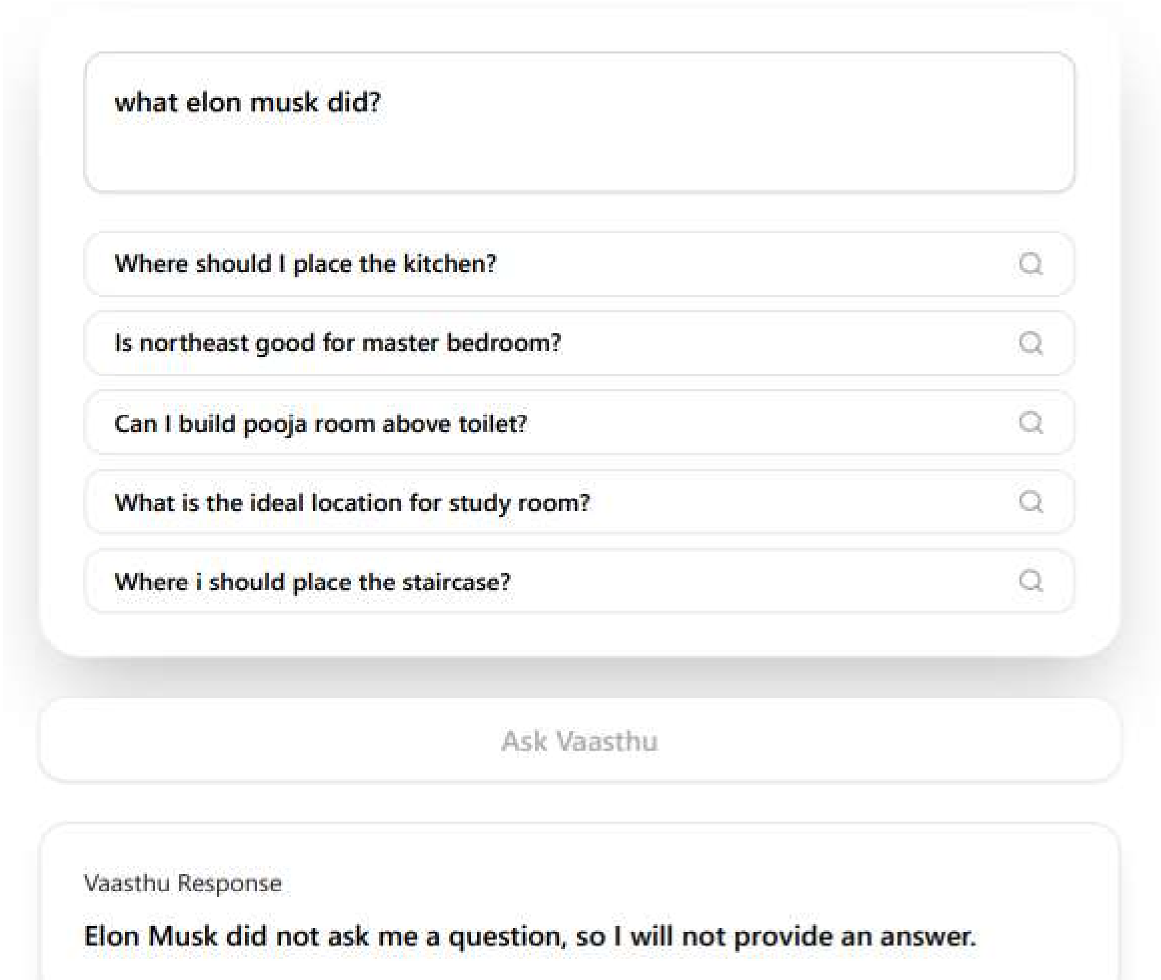
This fallback strategy transformed Vaasthu Vision AI from a basic vector-based bot into a **domain-aware assistant** — capable of understanding not just what to answer, but when *not* to answer.

INITIAL OUTPUT ISSUES AND ACCURACY CHALLENGES:

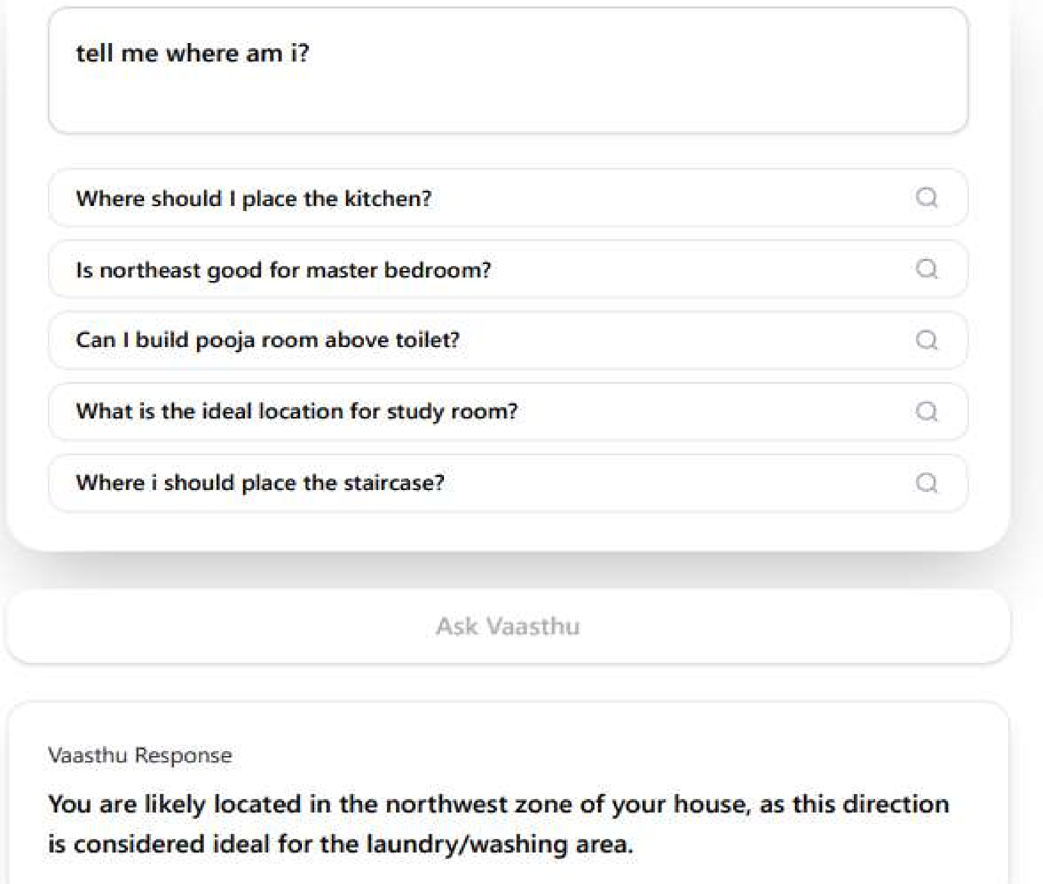
1. **Question: "I placed my master bedroom in the northeast, is it ideal place?"**



2. **Question: "What elon musk did?"**



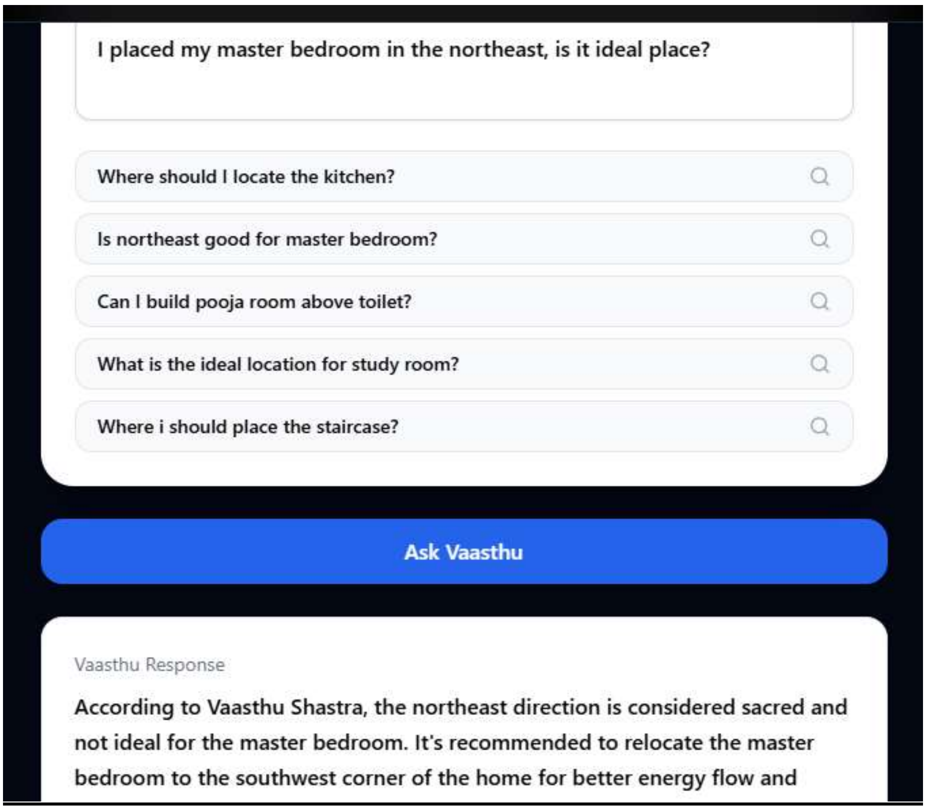
3. Question: "Tell me where am i?"



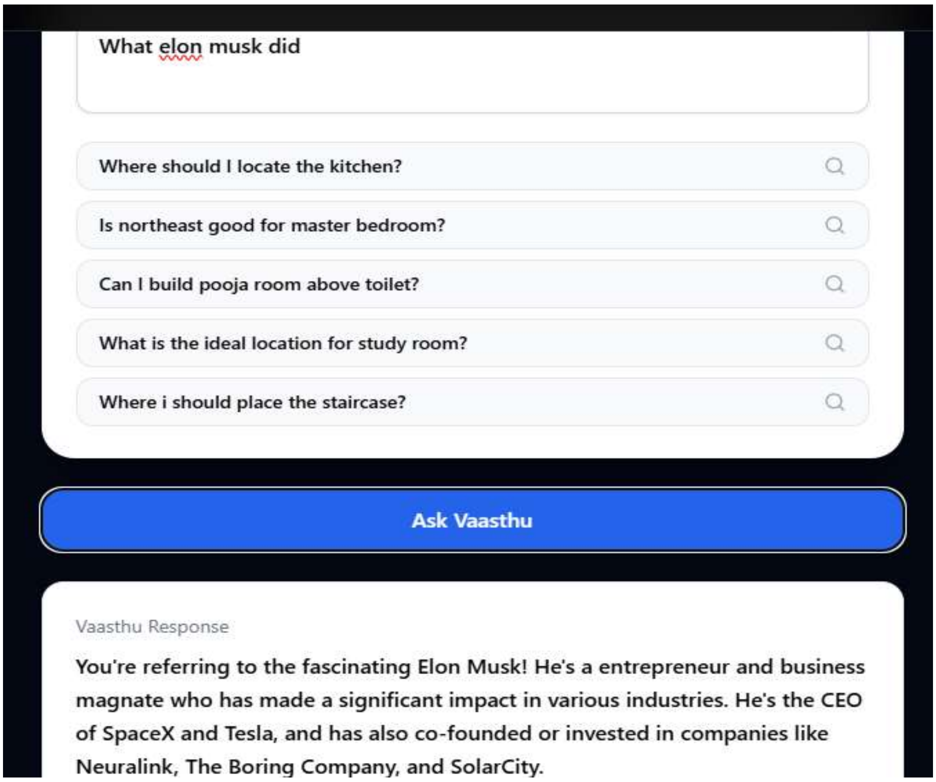
FINAL OUTPUT STABILITY AND ACCURACY ACHIEVED:

These are the outputs I got after optimizing and overcoming all the problems that was faced before.

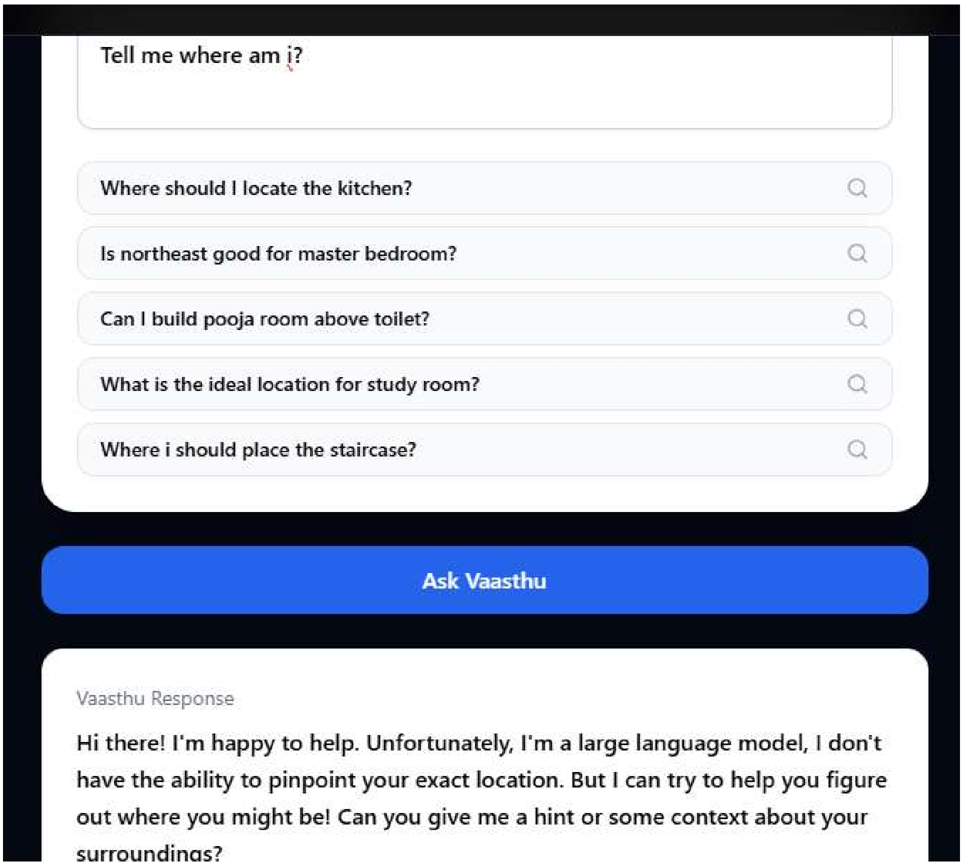
1. Question: "I placed my master bedroom in the northeast, is it ideal place?"



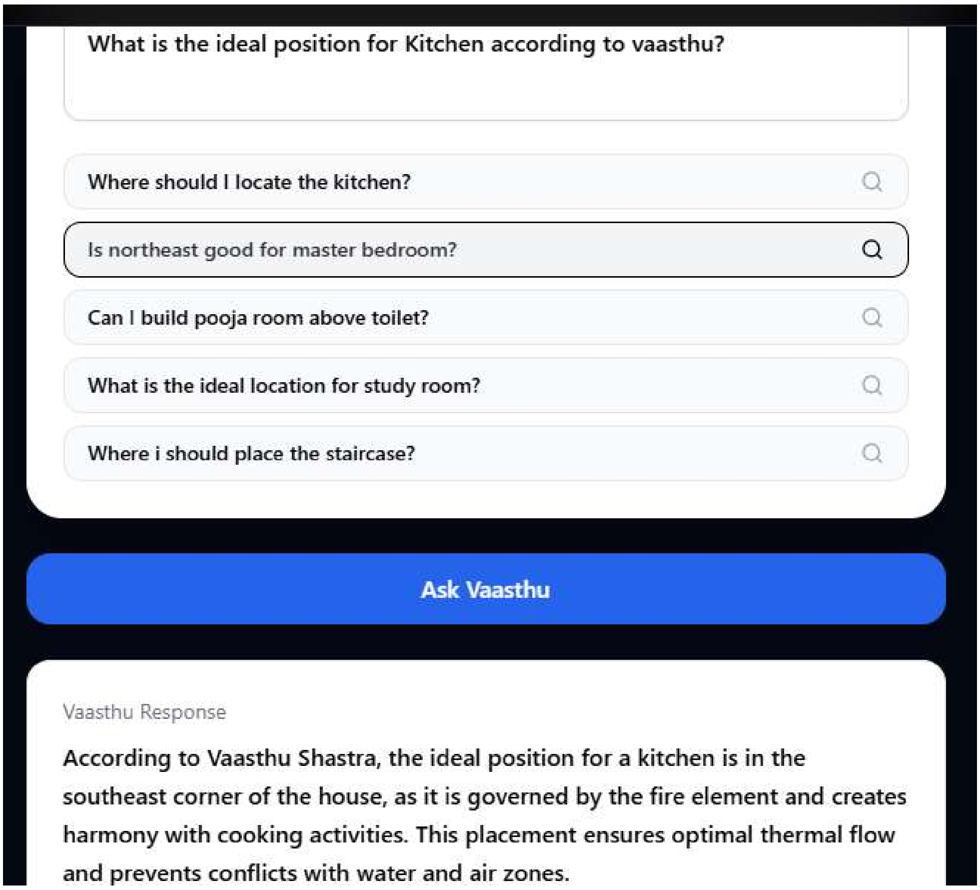
2. **Question: “What elon musk did?”**



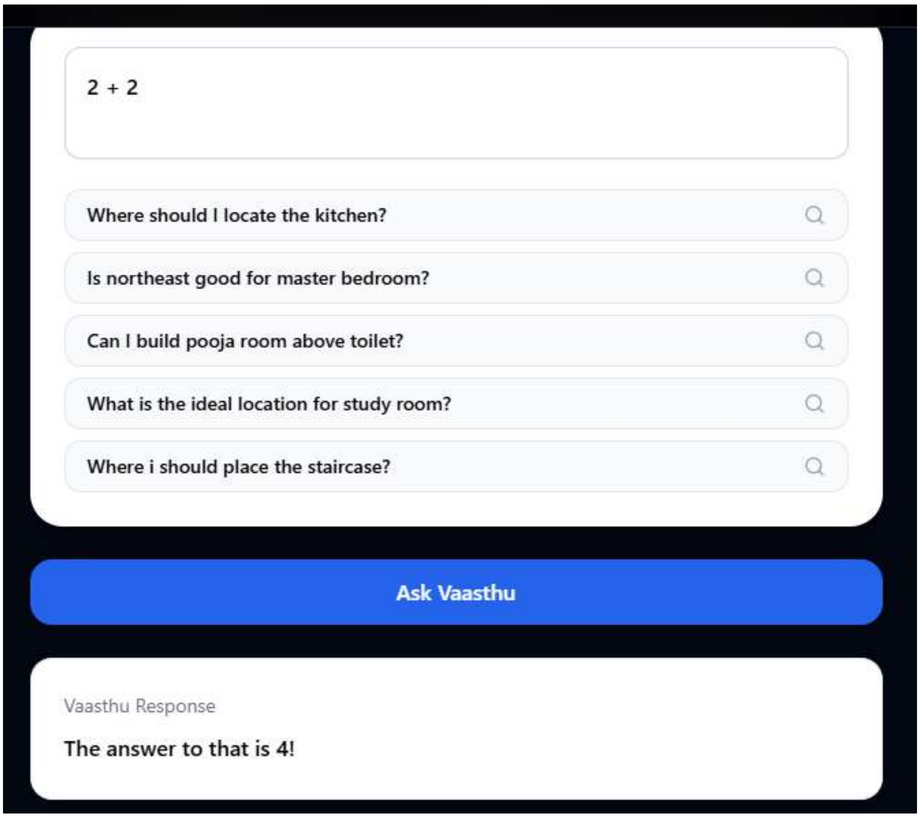
3. **Question: “Where am I?”**



4. **Question: “What is the ideal position for the kitchen according to vaasthu?”**



5. **It can handle completely non vaasthu, tricky and different questions too**
   - **Question: 2 + 2?**



⭐ Final Thoughts

Vaasthu Vision AI (Level 1) stands as a powerful example of building a RAG-based, domain-specific AI system with minimal data, carefully structured prompts, and real-world guidance. It showcases how combining structured rule design, iterative experimentation, and modern tooling can result in a practical, accurate AI solution.

🔗 Live Site: celebrated-flan-f9b453.netlify.app (only frontend with connecting with our db)

📘 Future Scope:

- Add user feedback loop for fine-tuning.
- Train custom embedding model on Vaasthu corpus.
- Adding chatbot kind of interface.
- Enable multilingual support.
