Multi-Modal Prompt Refinement System – Design Explanation

Author: Krishil Thakkar
Assignment: Internship Task – Dignifiedme Technologies
Date: 18th January 2025

1. **Complete Thought Process**

1.1 Understanding the Problem

The goal of this assignment is not simply to "use an LLM," but to design a system that can take heterogeneous inputs (text, images, documents, or combinations) and transform them into a consistent, structured prompt suitable for downstream AI usage.

Early on, I identified several core challenges:

- Inputs may be incomplete, vague, or contradictory
- Different modalities may imply different product intents
- The system must avoid hallucination while still being useful
- Validation should be informative, not overly strict
- "Irrelevant input" is ambiguous and needs a clear definition

The key difficulty is not extraction itself, but deciding what to do when information is missing or ambiguous.

1.2 Architectural Decisions

I considered two main architectural approaches.

Option A: Single-Stage Multi-Modal Refinement

```
Inputs (text + images + documents)
        ↓
Single multi-modal LLM call
        ↓
Structured template output
        ↓
Explicit validation
```

Option B: Multi-Stage Modality-Specific Pipeline

```
Input classification
 → image pipeline
 → document pipeline
 → text pipeline
 → merge outputs
 → validation
```

While Option B offers more granular control, I chose Option A for the following reasons:

- Modern models (GPT-4o) natively support multi-modal reasoning
- Multiple stages increase failure points without clear benefit for this task
- A single call preserves relationships between modalities
- The assignment prioritizes design reasoning, not pipeline complexity
- Simpler systems are easier to explain, validate, and maintain

This decision reflects a bias toward robustness and clarity over over-engineering.

1.3 Core Design Principle

Through iteration and testing, I arrived at the guiding principle:

Transparency over assumptions

This means:

- Missing information is never silently filled
- Ambiguity is surfaced, not resolved automatically
- Conflicts are documented with evidence
- Assumptions are explicit and paired with risk
- Output reflects confidence, not just content

This principle directly influenced the template structure, validation logic, and conflict handling.

2. Template and Data Structure Design

2.1 Overall Structure

The refined prompt is organized around semantic intent, not input modality:

```
{
  "intent": {...},
  "requirements": [...],
  "constraints": [...],
  "deliverables": [...],
  "conflicts_and_ambiguities": [...],
  "assumptions": [...]
}
```

This structure answers:

- What is being built
- Why it is needed
- What it must do
- What limits exist
- What is unclear or conflicting

2.2 Intent

```
"intent": {
  "purpose": "",
  "problem_being_solved": "",
  "domain": "",
  "confidence": "high | medium | low"
}
```

Rationale:

- These fields define the foundation of the product
- Confidence explicitly signals uncertainty
- Grouping keeps "what" and "why" tightly coupled

Intent clarity is weighted heavily in validation because all downstream decisions depend on it.

2.3 Requirements

```
"requirements": [
  {
    "text": "",
    "status": "confirmed | inferred",
    "source": "text | image | document"
  }
]
```

Design choices:

- Requirements represent functional behavior, not gaps
- Only confirmed or inferred items appear here
- Source attribution enables multi-modal traceability

Important rule:
Missing information is not treated as a requirement. Instead, it is surfaced via constraints or assumptions.

This prevents semantic confusion between what exists and what is absent.

2.4 Constraints

```
"constraints": [
  {
    "text": "",
    "status": "confirmed | inferred",
    "impact": ""
  }
```

```
]
```

Constraints are separated from requirements because they limit implementation choices rather than define functionality.

The impact field explains why a constraint matters, which helps downstream decision-making.

2.5 Conflicts and Ambiguities

```
"conflicts_and_ambiguities": [
  {
    "issue": "",
    "evidence": {},
    "impact": ""
  }
]
```

This section exists because multi-modal inputs can legitimately disagree.

Examples:

- Text describes a mobile app, image shows a desktop dashboard
- Text implies movie booking, image resembles food delivery
- Document contradicts both

Key decision:
Conflicts are never auto-resolved.

Reasoning:

- Users may intentionally combine ideas
- Automatic resolution introduces hidden assumptions
- This system is batch-oriented, not conversational

Instead, conflicts are clearly documented with evidence.

2.6 Assumptions

```
"assumptions": [
  {
    "assumption": "",
    "risk_if_wrong": ""
  }
]
```

Assumptions make inference visible.

Each assumption includes a risk description so users can prioritize clarification.

3. Information Extraction and Prioritization

3.1 Completeness Weighting

The system uses a weighted completeness score rather than binary validity.

```
COMPLETENESS_WEIGHTS = {
  "requirements": 0.40,
  "intent": 0.30,
  "constraints": 0.15,
  "deliverables": 0.10,
```

```
  "no_conflicts": 0.05
}
```

Why this distribution:

- Requirements define what must be built
- Intent provides foundational context
- Constraints guide feasibility
- Deliverables are often implied
- Conflicts reduce clarity but do not invalidate

This approach mirrors real product discovery workflows.

3.2 Essential vs Optional Information

Essential for high-confidence validation:

- Clear intent (purpose)
- At least one functional requirement

Optional but valuable:

- Constraints
- Deliverables
- Assumptions

Missing optional data lowers completeness but does not cause rejection.

4. Alternative Approaches Considered

4.1 Interactive Clarification

Asking users follow-up questions when ambiguity is detected was considered.

Rejected because:

- Changes system into a chatbot
- Breaks batch processing
- Adds control-flow complexity
- Not required by assignment

4.2 Domain-Specific Templates

Separate schemas for e-commerce, fintech, healthcare, etc.

Rejected because:

- Requires domain classification
- Reduces generality
- Increases maintenance cost

4.3 Strict Validation

Rejecting prompts unless all fields are present.

Rejected because:

- Real inputs are often vague
- Encourages premature rejection
- Completeness scoring provides better feedback

5. AI-Assisted vs Original Contributions

5.1 AI-Assisted Work

AI tools were used for:

- Code scaffolding and implementation
- Prompt iteration
- Test case generation
- Boilerplate UI components

AI acted as an implementation accelerator, not a design authority.

5.2 Original Contributions

The following were my independent design decisions:

1. Single-stage multi-modal architecture
2. Transparency-over-assumptions principle
3. Conflict flagging instead of resolution

4. Completeness scoring model
5. Essential vs optional field definitions
6. Source-aware requirement extraction
7. Explicit assumption risk modeling
8. Broad interpretation of "irrelevant input"

These decisions directly shaped the system's behavior.

7. **Lessons Learned**

Technical lessons:

- Multi-modal reasoning benefits from explicit structure
- Validation logic is as important as extraction
- Scoring communicates quality better than pass/fail

Design lessons:

- Ambiguity should be surfaced, not hidden
- Over-engineering is easy to justify but hard to maintain
- Documentation clarifies thinking as much as code

Conclusion

This system demonstrates my ability to:

- Reason under ambiguity
- Design explainable data structures
- Make and justify architectural tradeoffs
- Use AI tools responsibly
- Translate vague inputs into actionable structure

The result is a working, transparent prompt refinement system that prioritizes correctness, clarity, and trust over aggressive automation.

Krishil Thakkar
Internship Assignment – Dignifiedme Technologies