Task 1 - Design Explanation

Objective

Design a reliable AI assistant that converts unstructured business text into strictly valid, schema-compliant JSON, while minimizing hallucinations and handling malformed or ambiguous inputs safely.

The system prioritizes: - Reliability over perfect extraction - Deterministic behavior - Guaranteed valid JSON output

---

Overall Design Philosophy

The system follows a zero-trust LLM architecture.

The LLM is treated as a semantic extractor, not a source of truth.
All correctness guarantees are enforced in deterministic Python code.

The workflow combines: - Constraint-first prompting - Safe JSON parsing - Post-generation validation - Graceful fallback handling

---

1. Input Handling and Auto-Detection

Before calling the LLM, the system inspects the input:

- Single logical request -> expect one JSON object
- Multiple requests (multiline input) -> expect a JSON array

Why this matters:

LLMs often return multiple top-level JSON objects for batch inputs, which is invalid JSON.
Auto-detection ensures the model is instructed to return either: - A single JSON object, or - A JSON array with one element per input line

This prevents downstream parsing failures.

---

2. Prompt Design (Constraint-First)

Each LLM call uses a tightly constrained prompt that includes:

- The exact JSON schema
- Explicit instructions:
  - Return ONLY valid JSON
  - No markdown, no explanations, no extra text
- Explicit null-handling rules:

    - Missing or uncertain information must be null
- Domain grounding:
    - Urgency inference rules
    - Date parsing expectations

The temperature is set to 0.1 to reduce randomness and ensure consistent outputs.

---

3. Hallucination Control

Hallucination is controlled using two layers.

Prompt-Level Controls

- Explicit do-not-invent or hallucinate instructions
- Clear urgency and deadline rules
- Null-over-guessing philosophy

Code-Level Controls

- Raw LLM output is never trusted directly
- Validation layer:
    - Drops extra fields
    - Normalizes null values
    - Enforces correct data types
    - Restricts urgency to low, medium, high
    - Rejects invalid or vague dates

---

4. Robust JSON Parsing (Critical Design Choice)

Single-Parse Rule

The system parses JSON exactly once using:

`json.JSONDecoder().raw_decode()`

Benefits

- Stops parsing exactly at the end of valid JSON
- Ignores trailing text produced by the LLM
- Eliminates Extra data JSON parsing errors permanently
- After parsing, the system works only with Python dictionaries or lists

This guarantees malformed or verbose LLM responses cannot break the pipeline.

---

5. Validation and Schema Enforcement

After JSON extraction:

- The output is rebuilt explicitly field-by-field
- Only allowed schema fields are included
- Invalid values are replaced with null
- Dates are normalized to ISO format (YYYY-MM-DD)
- Extra fields are silently dropped

This guarantees strict schema compliance regardless of model behavior.

---

6. Failure Handling and Graceful Degradation

If model output is malformed:

1. Markdown wrappers are stripped
2. Safe JSON extraction is attempted
3. Validation fixes minor issues
4. If parsing still fails:
    - A minimal valid fallback JSON object is returned

For batch inputs, each item is handled independently so partial failures do not break the entire response.

End-to-End Workflow

```
Input Text
-> Auto-detect single vs batch
-> LLM call (constrained prompt, low temperature)
-> Safe JSON extraction
-> Schema validation and normalization
-> Fallback handling if needed
-> Guaranteed valid JSON output
```

Task 3 - Edge Case Evaluation

1. Slang and Informal Language

Input:

```
get me 500 bags cement asap for highway project
```

- Quantity and urgency extracted correctly
- Missing fields set to null
- No hallucinated brand or location

Input:

```
Get me M-sand 10 truck ASAP for dlf project in gurgaon!!!
```

- Industry slang handled correctly
- Noise ignored
- Valid structured output produced

---

2. Incomplete Data

Input:

```
need rebar 10mm urgently
```

- Quantity, unit, project, location missing -> null
- Urgency inferred as high
- No invented values

---

3. Typos and Misspellings

Input:

```
tmr bars 16mm 500 pices projecct pheonix towrs urgent delvery in 1 wek
```

- Core meaning recovered
- Quantity and urgency extracted
- No fabricated fields

---

4. Conflicting Information

Input:

```
Get 25 bags of gypsum powder, quantity might be 30 also
```

- First explicit quantity chosen
- Limitation acknowledged
- Future improvement: conflict detection

---

5. Ambiguous Inputs

Input:

```
i need some concrete for my project
```

- Only material extracted
- All other fields set to null
- Prevents hallucination of typical quantities

Input:

```
Just send everything for Site 5
```

- No material specified
- Fallback response returned
- Valid JSON preserved

---

6. Multi-Material Requests (Known Limitation)

Input:

```
aggregates 20mm 10 tons, 10mm 5 tons for bridge construction ASAP
```

- Schema supports only one material
- LLM merges or prioritizes one item
- Requires schema redesign for full support

---

7. Batch and Multi-Line Inputs

Input:

```
Need 100 bags of cement
Order 5 truckloads of sand
Get steel rods urgently
```

- Auto-detected as batch
- JSON array returned
- Each item validated independently
- No Extra data parsing errors

Failure Analysis Summary

Where the LLM Hallucinates Most

- Quantities for vague inputs
- Deadlines inferred from urgency

- Project names inferred from locations

Controls That Worked Best

- Null-over-invention rule
- Low temperature (0.1)
- Schema-first prompting
- Post-generation validation
- Single-pass JSON parsing

Task 4 - Reflection

Hardest Part

Ensuring valid JSON under all conditions, especially:

- Multiline inputs
- Trailing LLM text
- Multiple JSON objects in one response

This required moving from regex-based parsing to a single-pass JSON decoder.

Best Controls

1. Single-pass JSON extraction
2. Auto-detection of batch inputs
3. Schema reconstruction during validation
4. Deterministic generation settings
5. Graceful fallback behavior

Improvements With More Time

- Pydantic schema enforcement
- Confidence score per extracted field
- Multi-material schema support
- Improved relative date parsing
- Human-in-the-loop review
- Contextual memory for follow-up requests

Summary

This solution demonstrates a robust, production-oriented approach to structured extraction using LLMs.

It minimizes hallucinations, guarantees valid JSON output, and handles real-world ambiguity gracefully while prioritizing reliability over speculative accuracy.

“‘