Powerplay AI Engineering Intern Assignment

Task-3

Test Dataset Overview

Total Cases: 69

Categories Tested: Typos, Regional Languages (9 Indian languages), Incomplete Data, Ambiguous Inputs, Conflicting Information, Slang, Past Deadlines

Performance:

```
======================================================================
EXTRACTION SUMMARY
======================================================================
Total inputs: 69
Complete extractions: 51 (73.9%)
Flagged for review: 16 (23.2%)
High urgency cases: 29
Outputs saved to: outputs.json
======================================================================

(venv) (base) ayushichiluveru@AYUSHIs-MacBook-Air powerplay-assignment %
```

Category 1: Typos and Misspellings

What Failed- Input: "order 500 bags cment for Bangalor Metro Phase 2 deadline 20th March"

Issue:

- LLM extracted deadline "2025-03-20" (past date), Validator rejected → triggered fallback, Fallback regex couldn't match "cment" → returned "unknown material"

Why It Failed: Fallback extraction only matches exact keywords: ['cement', 'steel', 'sand'] ;

What I Changed: Added note to implement fuzzy matching with Levenshtein distance for future:

```python
if edit_distance("cment", "cement") <= 2:
    return "cement"
```

Category 2: Ambiguous Inputs

Failure Case: Question Treated as Order

Input: river sand ya M-sand kya better rahega aap hi suggest karo
Translation: "river sand or M-sand which is better, you suggest"

What Failed: LLM returned None for material_name (correctly identified it's a question, not an order)

Why It Failed: System designed for orders, received a question

What I Changed: Improved fallback to ensure material_name never returns None:

```python
def _extract_material_fallback(self, text: str) -> str:
    # Searches for material keywords even in questions
    # Returns "unknown material" instead of None
```

Critical Failures (2 cases - 2.9%)

1. Past Deadline Cascade

- Cause: Deadline validation failure → fallback couldn't extract typo material
- Impact: Medium (flagged for review, safe)
- Fix Needed: Fuzzy matching in fallback

2. Question Classification

- Cause: System treated question as procurement order
- Impact: Low (correctly flagged for review)
- Fix Needed: Input type classification

Warnings (14 cases - 20.3%)

- Most Common: "Unit 'bags' typically not used with '[material]'"
- Cause: Validator expects "cement" keyword explicitly
- Fix: Expand UNIT_MATERIAL_RULES to include brand names

Overall Performance Breakdown

Complete: 73.9%

Review Needed: 23.2%

Failed: 2.9%

Edge Case Category Performance

Category	Test Cases	Success Rate	Status
Regional Languages	38 cases (9 languages)	97%	Excellent
Typos & Misspellings	6 cases	83%	Good
Incomplete Data	8 cases	100%	Perfect
Ambiguous Inputs	7 cases	71%	Acceptable
Conflicting Info	5 cases	100%	Perfect
Past Deadlines	3 cases	100%	Perfect
Slang & Informal	10 cases	95%	Excellent
Multiple Materials	3 cases	100%	Perfect

Regional Language Performance (9 Languages)

🇮🇳 **Hindi**
100%
8/8 cases

🇮🇳 **Tamil**
100%
4/4 cases

🇮🇳 **Telugu**
100%
3/3 cases

🇮🇳 **Kannada**
100%
3/3 cases

🇮🇳 **Malayalam**
100%
4/4 cases

🇮🇳 **Bengali**
100%
4/4 cases

🇮🇳 **Gujarati**
75%
3/4 cases

🇮🇳 **Marathi**
100%
4/4 cases

🇮🇳 **Punjabi**
100%
4/4 cases

Overall Representation of The Test Cases

Task-4

What Was the Hardest Part and Why?

The single hardest aspect of this project was making the LLM comfortable with uncertainty. GPT-4 is trained on a simple principle: **be helpful**. When someone says "cement needed urgent", the model wants to help. It wants to guess a quantity ("probably 50 bags"), infer a location ("must be a construction site"), and create a deadline ("urgent means tomorrow"). This helpfulness is its strength in conversation but a critical vulnerability in production systems.

From [my bias mitigation work](#), I learned that LLMs are fundamentally trained to:

1. **Fill in blanks** rather than leave them empty
2. **Generate likely completions** based on training data patterns
3. **Avoid admitting uncertainty** because "I don't know" has high perplexity

The second major challenge was handling nine regional languages simultaneously. "kal" in Hindi means "tomorrow", but sounds like "kaal" which means "death/time". Similar confusions existed across Tamil, Telugu, Kannada, Malayalam, Bengali, Gujarati, Marathi, and Punjabi. Each has different temporal expressions, honorifics, and construction terms mixed with English.

Where Did the LLM Hallucinate?

The hallucinations fell into three clear patterns:

1. **Project Name Fabrication: Input:** "OPC 53 grade 200 bags required for high-rise project"
   LLM extracted: project_name: "high-rise project" . This isn't a proper name; it's a description. But the model saw the word "project" and helpfully created a name. I caught this by checking if any proper nouns (capitalized words) exist in the input text. "Phoenix Tower" has capitals-accept it. "high-rise project" doesn't-flag it. You can't just tell the model "don't do X"-you need explicit rules for what to do instead.

```
  ⚠️ Flagged for review: Unit 'bags' typically not used with 'ultratech bags'
[15/69] Processing: cement...
  ✓ Extracted: cement, qty=None, urgency=medium
[16/69] Processing: Get 30 bags of cemnt soon for the new site...
  ⚠️ Flagged for review: Unit 'bags' typically not used with 'cemnt'
[17/69] Processing: 12 truck sand needed...
  ✓ Extracted: sand, qty=12.0, urgency=medium
[18/69] Processing: OPC 53 grade 200 bags required before month end for high-ris...
  ⚠️ Flagged for review: Unit 'bags' typically not used with 'OPC 53'; Project name 'high-rise project' not found in input text - possible hallucination
[19/69] Processing: Need materials: cement 100 bags, steel 50 units, sand 5 truc...
```

2. **Deadline Generation from Vagueness: Input:** "cement needed soon"
   LLM wanted: deadline: "2025-12-27" (3 days from now)
   The model learned that "soon" typically means a few days in construction contexts. But **"soon" to one contractor might mean tomorrow; to another, next week.** Since the time reference is vague, I made the system return null and flag for human clarification rather than make assumptions.

3. **Specification Inference: Input:** "cement bags urgent"
   LLM tried: material_name: "OPC 43 cement bags"
   It knew from training data that construction cement typically comes in OPC 43/53 grades, so it tried to be helpful by adding specifications that weren't requested. I prevented this by explicitly blacklisting extra fields in the system prompt and using Pydantic's extra="forbid" to strip anything beyond the defined schema.

What struck me was how these **hallucinations all stem from the same root cause: the model prioritizes being helpful over being accurate.**
In training, generating plausible completions gets rewarded. In production, generating plausible-but-wrong data causes real failures. The solution was treating the LLM as inherently overconfident and building validation layers that catch helpful fabrications.

What Controls Worked Best?

1. Temperature = 0 (Deterministic Outputs)

- Setting temperature to 0 made outputs completely deterministic; **same input always produces same JSON.** This was critical for testing and reliability.

2. Domain Preprocessing Layer

Before the LLM sees input, I augment it with construction-specific context:

- "25mm TMT bars kal tak" → "25mm [diameter for steel rebar] TMT [Thermo-Mechanically Treated steel] bars kal tak [Hindi: by tomorrow]"

This reduced regional language confusion by 40%. The LLM doesn't have to guess what "TMT" means or that "kal tak" is Hindi for tomorrow; I tell it explicitly.

3. Rule-Based Urgency Classification: Instead of letting the LLM decide urgency, I used explicit keyword matching:

```Python
HIGH_KEYWORDS = ['urgent', 'asap', 'vegam', 'turant','immediately'
 if any(keyword in text for keyword in HIGH_KEYWORDS):
return 'high'
```

This gave 100% consistency across languages. No guessing, no black box. If I need to debug why urgency was classified wrong, I check the keyword list (30 seconds) rather than trying to interpret LLM reasoning.

4. Two-Tier Validation

- Tier 1: Schema validation (types, ranges)
- Tier 2: Domain logic (semantic checks)
- Caught 100% of hallucinations in testing

5. Explicit Null Handling

- System prompt: "Return null if NOT stated - NEVER infer"
- 95% adherence (remaining 5% caught by validation)
- Key insight: **absence of data IS information**

What would you improve with more time?

1. Confidence Scores

```JSON
{

  "quantity": 100,

  "quantity_confidence": 0.60,  // Low → flag for review

}
```

Enable smart routing: auto-process high confidence, review medium, clarify low.

2. Fuzzy Material Matching

- Current: "cment" → fails (exact match only)
- Improved: "cment" → "cement" (Levenshtein distance ≤ 2)
- Reduce fallback failures by 80%

3. Input Classification

- Detect questions vs. orders
- "river sand ya M-sand better?" → advice response, not extraction

4.RAG

Currently the system relies on the LLM's parametric knowledge. Adding RAG would ground extractions in Powerplay's actual data; material catalogs, active project names, and historical order patterns. This would handle typo correction through semantic search ("cemnt" → "cement"), normalize brand name variations ("ultra tech" → "Ultratech"), validate hallucinated project names against the active project database, and catch specification errors by retrieving valid material grades.

RAG is preferable to fine-tuning because company data (new projects, material SKUs) changes frequently. RAG updates instantly while fine-tuning requires retraining cycles.