1. Intent Extraction Strategy

Upon researching about the scenario, we are dealing with here I could come up with two ways to do it. First using the intent extraction pipeline and vector databases with hybrid search setup.

Intent extraction pipeline

In this approach we have 4 stages

1. Preprocessing

- Normalize text (lowercase, remove punctuation, replace ₹ → INR)
- Tokenize (spaCy)
- Extract raw numbers/currency (under ₹1000, below 1k)
- Convert short forms (1k → 1000)
- Use precompiled regex + lookup libraries (RE regular expression)

2. Category Detection

We have two ways to detect categories from the query and map them to the database

- **FastText**:
  - Use a small fine-tuned classification model (e.g., FastText) to predict the category from the query.
  - Map the predicted category to the existing categories in the database.
- **Lookup Tables**
  - Leverage the tokens generated during preprocessing.
  - Match them against predefined lookup tables to identify the category.

3. Filters Detection

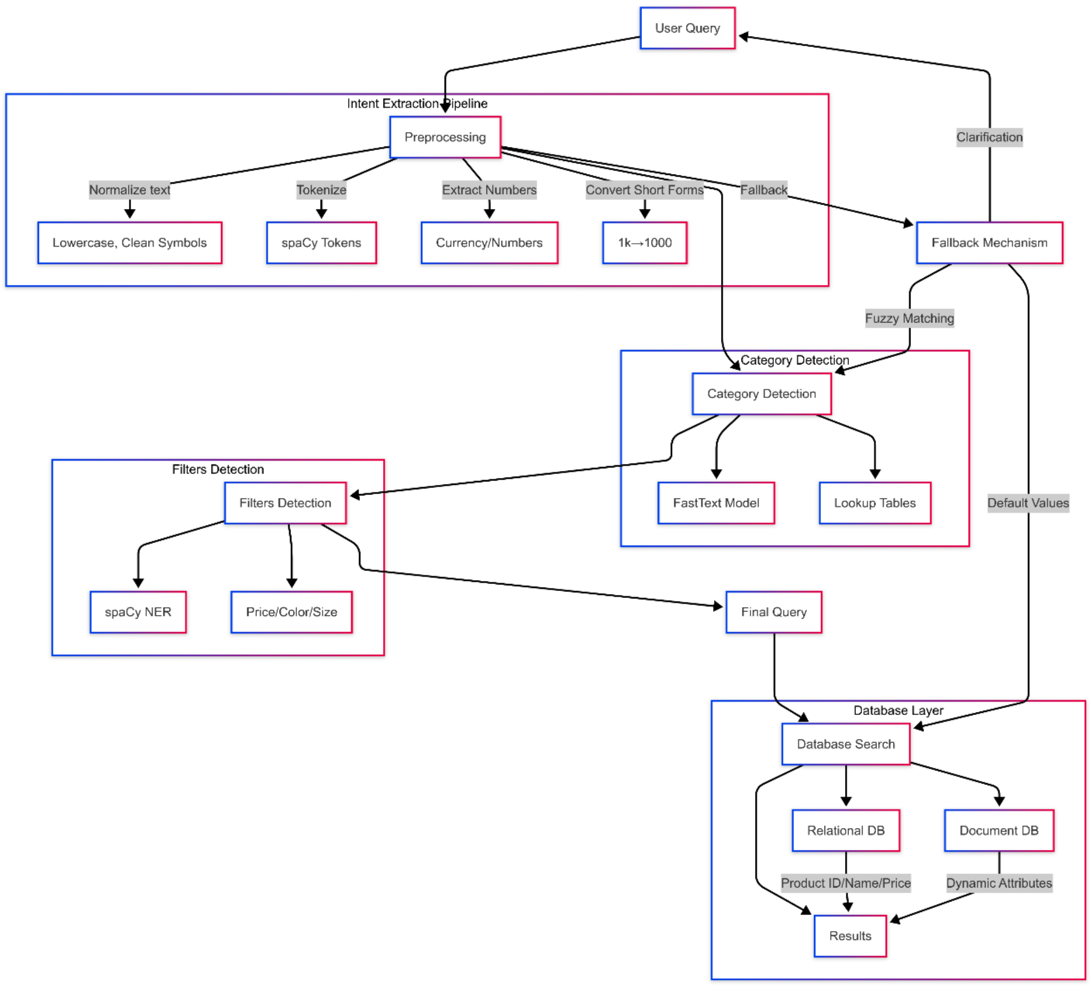
- Named Entity Recognition (NER) Utilize pre-trained models from libraries like spaCy or Hugging Face's Transformers to identify entities such as product types, sizes, colours, prices, and locations.

4. Final Query:

- Use all the data generated from the query to create a database search query for the desired ORM.

```
 {
    "tokens": ["running", "shoes", "size", "9", "red", "under", "inr", "2000", "in", "mumbai"],
    "numbers": ["9", "2000"],
    "currency_string": "inr 2000",
    "category": "shoes",
    "attributes": {
        "product": {"value": "running shoes", "confidence": 0.95},
        "size": {"value": "9", "confidence": 0.90},
        "color": {"value": "red", "confidence": 0.85},
        "price": {"value": {"lte": 2000}, "confidence": 0.88},
        "location": {"value": "mumbai", "confidence": 0.92}
    }
}
```

Flow Chart



Fallback Mechanism

- Ask the user for clarification on ambiguous parts of the query. For example, if the system is unsure about the colour, it can ask, "Did you mean red or another color?"
- Use default values for filters that are not confidently classified. For example, if the price range is unclear, default to a common price range for the product type.
- Implement fuzzy matching to handle variations in terminology. For example, "sneakers" and "running shoes" can be treated as synonyms.

2. Flexible Schema for Category-Specific Attributes

Data Model Choice

For a flexible schema that allows merchandisers to add new attributes without large-scale migrations, a hybrid data model is recommended. This model combines the strengths of relational and document databases:

Relational Database: Use a relational database for structured data that requires strong consistency and complex queries. This includes core product information like product ID, name, and price.

Document Database: Use a document database for category-specific attributes. Each product can have a document that stores attributes as key-value pairs. This allows for easy addition of new attributes without schema changes.

Supporting Fast Multi-Attribute Filtering

The hybrid model supports fast multi-attribute filtering and attribute existence queries through:

1. Indexing: Create indexes on frequently queried attributes in both the relational and document databases to speed up search operations.
2. Denormalization: Denormalize data where necessary to reduce the need for joins, which can slow down queries. For example, store commonly accessed attributes directly in the product document.
3. Caching: Implement caching mechanisms to store frequently accessed data in memory, reducing the need for repeated database queries.