Solution Design

1. Objective

Build a system to map varying product names to standardized names, using:

- Manual input for first-time entries.
- An automatic matching system powered by a mapping dictionary and string similarity logic.
- A mechanism to intelligently handle new variations.

---

2. Implementation Plan

Step 1: Data Input

- Input sources:
  - Manually input the first-time mappings.
  - Provide batch data for processing.

Step 2: Matching Mechanism

- Use a similarity-matching algorithm (e.g., Levenshtein Distance, cosine similarity, or fuzzy matching).
- Normalize the product names for:
  - Case insensitivity.
  - Removal of special characters, spaces, and abbreviations.

Step 3: Mapping Dictionary

- Maintain a key-value dictionary:
  - **Key:** Raw product names.
  - **Value:** Standardized product names.
- If no exact or close match is found, prompt for manual mapping.

Step 4: Handling New Variations

- Automatically flag names that do not match existing mappings.
- Allow easy updates to add new entries to the dictionary.

Step 5: Edge Case Handling

- Names with high similarity scores but incorrect matches.
- Edge cases like identical names with different attributes (e.g., color, size).

---

3. Expected Features

- **Interactive Manual Matching:**
  - Prompt users to map first-time entries to standardized names.
- **Automatic Matching:**
  - Match names with high confidence.
  - Suggest possible matches for manual confirmation when confidence is low.
- **Dictionary Maintenance:**
  - Provide CRUD (Create, Read, Update, Delete) operations for the mapping dictionary.
- **Feedback and Logging:**
  - Log unmatched names and reasons for failure.
  - Provide clear messages for invalid inputs.

---

4. Prototype Implementation

Here's a **Python-based implementation plan** using Flask for the API and fuzzywuzzy or rapidfuzz for similarity matching:

Modules to Use:

- Flask (backend).
- pandas (for dictionary storage in CSV/Excel).
- fuzzywuzzy or rapidfuzz (text matching).

Working Prototype:

```
from flask import Flask, request, jsonify
!pip install rapidfuzz
from rapidfuzz import process, fuzz
import pandas as pd

app = Flask(__name__)

# Load or initialize the mapping dictionary
try:
mappings =
pd.read_csv("product_mappings.csv").set_index("raw_name").to_dict()["standard_name
"]
except FileNotFoundError:
mappings = {}

def normalize_name(name):
return ' '.join(name.lower().strip().split())

@app.route('/match', methods=['POST'])
def match_product():
product_name = request.json.get('product_name')
normalized_name = normalize_name(product_name)

# Check for exact match
if normalized_name in mappings:
return jsonify({"status": "matched", "standard_name": mappings[normalized_name]})

# Find close matches
matches = process.extract(normalized_name, mappings.keys(),
scorer=fuzz.partial_ratio, limit=3)
suggestions = [match[0] for match in matches if match[1] > 80]  # Confidence threshold

if suggestions:
return jsonify({"status": "suggest", "suggestions": suggestions})
else:
return jsonify({"status": "not_matched", "message": "No close matches found. Please
provide a manual mapping."})
```

```
@app.route('/add_mapping', methods=['POST'])
def add_mapping():
raw_name = normalize_name(request.json.get('raw_name'))
standard_name = request.json.get('standard_name')

mappings[raw_name] = standard_name
pd.DataFrame(mappings.items(), columns=["raw_name",
"standard_name"]).to_csv("product_mappings.csv", index=False)
return jsonify({"status": "success", "message": f"Mapping for '{raw_name}' added."})

if __name__ == "__main__":
app.run(debug=True)
```

---

5. Assumptions Made

- Standardized names are predefined.
- Raw names can differ only in textual representation, not product attributes.

---

6. Identified Use Cases

Handled:

- Text variations (case, abbreviations, extra spaces).
- Incremental addition of new mappings.
- Suggestions for manual mapping.

Not Handled:

- Names with ambiguous context (e.g., different colors or sizes).
- Complex variations requiring deep NLP.

---

7. Future Improvements

- Enhance matching logic using machine learning (e.g., embeddings with transformers).
- Build a web interface for manual mapping.
- Add versioning for the mapping dictionary.
- Incorporate multilingual support.