Vibe-Fusion: Architecture Design Document

Stakeholders and Development Team

- **Developers:** NLP/ML engineers, backend developers, and UX designers implement and maintain the pipeline modules (NLP Analyzer, Similarity Matcher, etc.) and the **Vibe Knowledge Base**.
- **Stakeholders:** Fashion domain experts, product managers, and e-commerce owners guide system requirements (e.g. accuracy, explainability, brand voice).
- **End Users:** Shoppers interacting via the conversational interface – their feedback can drive continuous refinement.

System Overview

This system translates user "vibe" descriptions into outfit recommendations. It uses a **hybrid architecture** combining rule-based logic with AI fallback. The high-level pipeline is:

- **User Query:** Natural-language style request (e.g. "something elegant for a dinner date").
- **NLP Analyzer:** Parses the query into fashion attributes (category, style, fit, etc.).
- **Similarity Matcher:** Uses the parsed terms to look up standardized attributes via word similarity and the curated **Vibe Knowledge Base**. If confidence is high, it proceeds.
- **GPT Inference:** If key attributes are missing or the match is low-confidence, GPT-4 is called to infer a complete attribute set.
- **Catalog Filter:** Filters the product catalog by the confirmed attributes (e.g. size, color, occasion).
- **NLG Generator:** Converts the filtered results into a natural-language response.

Each stage is modular, so that common cases use fast, deterministic logic while complex or ambiguous queries use the AI fallback. Context is preserved across turns to handle follow-up questions or corrections.

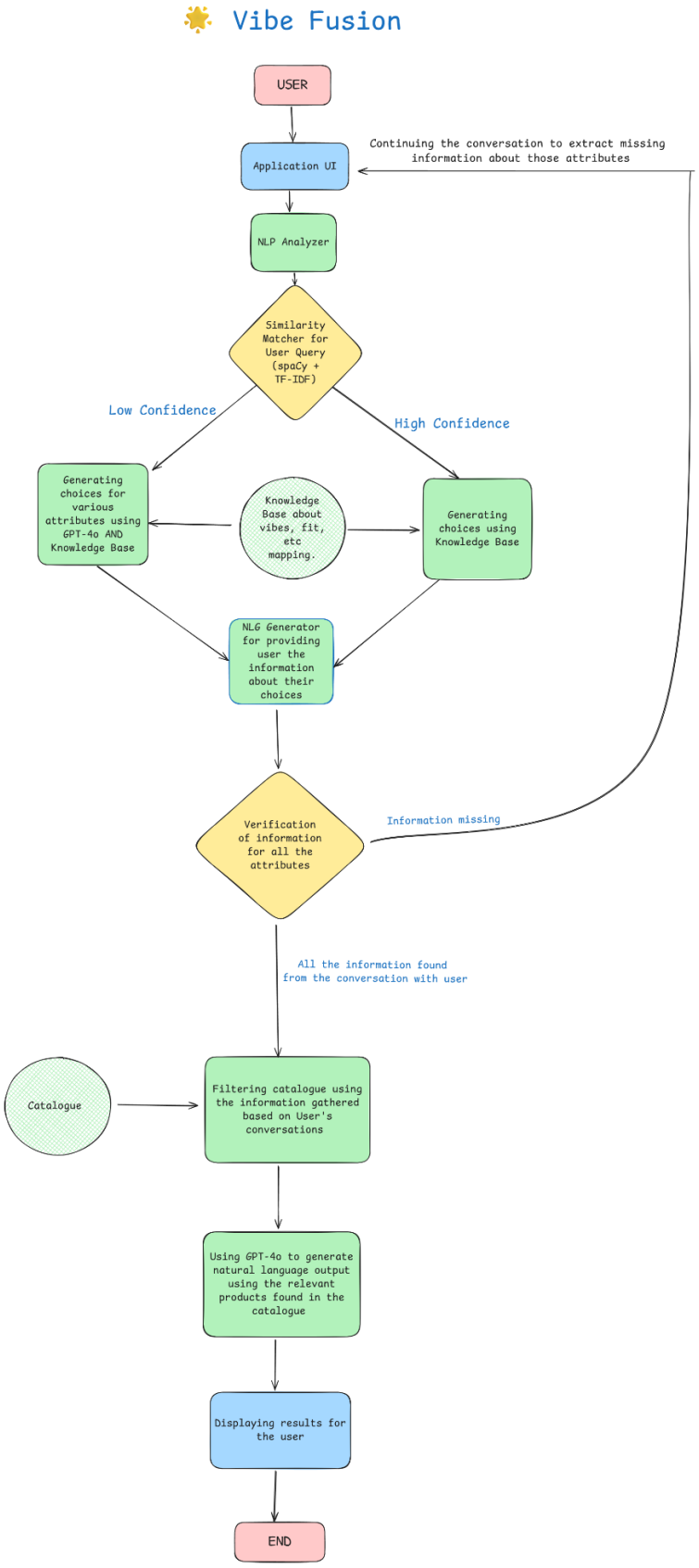
Data Flow and Component Diagram

The data flows through the system as follows:

1. **User Input (UI):** The user submits a text query via the chat interface.
2. **NLP Analyzer:** The query is tokenized and tagged. Fashion-specific entities (like "brunch" or "bodycon") are extracted with patterns and spaCy's vectors.
3. **Similarity Matcher:** Extracted terms are matched against the **Vibe Knowledge Base**. SpaCy cosine similarity and TF-IDF compare user phrases to standard attributes (e.g. "form-fitting" → "bodycon"). If the match score is high, the attribute is accepted.
4. **GPT Fallback:** If matches are weak or missing, the system invokes GPT-4 with a prompt containing the query and any partial attributes. GPT returns a structured JSON of inferred attributes. (This fallback is slow and used selectively.)
5. **Catalog Filter:** The final attribute set (category, size, color, fabric, etc.) is used to filter products from the catalog (using Pandas or a database query). Category-specific filters apply (e.g. neckline only for dresses).

6. **NLG Response:** Templates generate a human-friendly response describing the recommended items.

Below is a conceptual flow diagram of the component:



Core Components

NLP Analyzer

- **Design:** Combines spaCy (for tokenization, part-of-speech, dependency parsing, and word vectors) with NLTK stopword filtering. We use custom entity/pattern lists for fashion (occasions, styles, fits, colors).
- **Why Chosen:** spaCy is fast and lightweight compared to large transformer models. Its medium model (en_core_web_md) includes word vectors for semantic similarity. NLTK aids in cleaning text. This hybrid approach is more accurate than simple regex and much quicker than calling a full transformer for every query.
- **Pros:** Millisecond-level performance and reliable linguistic features. Domain-specific patterns improve accuracy for known fashion terms. The pipeline is deterministic and easy to debug.
- **Cons:** May miss nuance in very creative language or new slang. Less contextual understanding than a tuned transformer model. Requires manual updates to add new patterns or vocabulary.

Similarity Matcher

- **Design:** Uses a dual approach: spaCy cosine similarity (contextual embeddings) and TF-IDF string matching against our **Vibe Knowledge Base** (JSON mappings of slang to attributes). For example, "form-fitting" is semantically matched to "Bodycon" fit; "pastels" expands to a set of light colors.
- **Why Chosen:** This covers both semantic synonyms and exact/fuzzy matches. spaCy handles meaning ("relaxed" $\approx$ "comfy"), while TF-IDF catches literal hits. The JSON mappings (fits, colors, occasions, fabrics) store curated knowledge.
- **Pros:** High precision for known terms, and good recall for synonyms. Easy to maintain: updating a JSON mapping (say, adding a new color slang) requires no model retraining. Successful matches produce attribute values with a confidence score.
- **Cons:** Matching thresholds must be tuned (we use ~0.8) to avoid false positives. Rare or very novel phrases may fall below threshold. TF-IDF ignores word order and context nuances. The system can mis-rank if the user phrasing is ambiguous.

GPT Inference (Fallback)

- **Design:** Invoked only when rule-based confidence is low. The system builds a structured prompt including the user's query, any partially matched attributes, and a strict output schema. We use GPT-4 via OpenAI's API with low temperature (0.3) for consistency.
- **Why Chosen:** GPT-4 can infer implicit or complex requirements (e.g. "boho chic" or multi-faceted queries) that the fixed rules might miss. It broadens the system's understanding without extensive manual rule-writing.
- **Pros:** Great for edge cases and creative language. For example, it might infer that "elegant" implies silk fabrics and tailored fits. We enforce a JSON schema in the prompt to minimize gibberish.
- **Cons:** Significantly slower (1–3 seconds per call) and requires internet/API access. There's also a risk of **hallucinations** (confident but incorrect output). We mitigate this by validating GPT's answers against known attribute values. If GPT fails, the system gracefully degrades (it will simply make do with whatever partial data it has).

Catalog Filter

- **Design:** Applies the confirmed attributes to the product database (initially a Pandas DataFrame from apparel.xlsx). We filter by category, price range, size availability, color family (expanding "earth tones" to multiple colors), fit, etc. Category-specific filters are enforced (e.g.

only dresses have a “neckline” filter).

- **Why Chosen:** Pandas filtering is fast and straightforward for a catalog of typical size (hundreds or thousands of items). The logic is transparent and easy to extend.
- **Pros:** Millisecond response for lookups. We can incorporate complex business rules (e.g. exclude sold-out items, respect user budget). Color and size flexibility (numerical vs S/M/L) is handled here.
- **Cons:** Scaling to millions of products would require a proper database or search engine. Currently, all products load in memory. Data quality is crucial: inconsistent size/color labels or missing attributes can break filtering

NLG Generator

- Uses template-based generation. We store response templates for different situations (single product vs. list, follow-up question vs final answer). Placeholders (e.g. {color}, {fit}, {price}) are filled with actual data.
- **Why Chosen:** Templates ensure consistency and factual accuracy. We avoid relying on a generative model to describe products, which can hallucinate or change details. This also lets us control tone and brand voice.
- **Pros:** Always grammatically correct and on-topic. Easy to maintain and adapt (just add or adjust templates). Fast and no API calls.
- **Cons:** Less variety and creativity. Responses can feel formulaic if not enough templates exist. Extending to new scenarios (e.g. multi-product comparisons) requires adding new templates.

Knowledge Base (Vibe Mappings)

- **Design:** A set of JSON files (fit_mapping.json, color_mapping.json, etc.) map casual terms to structured attributes. E.g., { "comfy": {"fit":"Relaxed"}, "bodycon": {"fit":"Body hugging"} }. Complex scenarios (like “romantic dinner date”) map to several attributes at once.
- **Why Chosen:** This encodes fashion expertise explicitly. Non-technical editors can update it (just JSON) to refine mappings. It augments the statistical NLP with human judgment.
- **Pros:** Highly explainable matches. Quick wins for known “vibes” (just add them to the JSON). Reduces ambiguity (we decide what “elegant” really entails in our catalog context).
- **Cons:** Maintenance burden: the base of knowledge must be grown over time. If a new trendy phrase appears, it must be added manually. Unmapped phrases fall back to GPT or partial matching. It’s not a substitute for real-world data learning.

Product Catalog and Data

- **Design:** The catalog (an Excel/CSV) has columns for category-specific attributes (e.g. neckline, sleeve_length) plus common ones (size, price, color). We standardize color names and unify size systems.
- **Pros:** Structured data enables precise filtering. We can easily support multiple size conventions, price sorting, and inventory checks. Catalog stats (e.g. how many dresses) inform system health.
- **Cons:** As an Excel file, this is only a prototype. Large inventories should move to SQL/NoSQL. Data consistency (spellings, missing values) is critical and requires cleaning.

Conversational Interface (Streamlit UI)

- **Design:** A chat-like UI built with Streamlit, handling multi-turn conversation. It tracks session context

(previous answers, pending questions). The UI can prompt the user if critical information is missing (e.g. "What size do you need?").

- **Why Chosen:** Conversational interfaces let users describe their needs naturally. They're more engaging than rigid forms and can clarify ambiguous requests on the fly. According to UX studies, chat interfaces provide a more "frictionless" experience than traditional web forms.

Deployment (Streamlit Cloud)

- The app is hosted on **Streamlit Cloud** for simplicity. A run.py script automates setup: installs dependencies, downloads the spaCy model, generates a sample catalog, and launches the Streamlit app.

Design Rationale and Trade-Offs

- **Design:** The catalog (an Excel/CSV) has columns for category-specific attributes (e.g. neckline, sleeve_length) plus common ones (size, price, color). We standardize color names and unify size systems.
- **Pros:** Structured data enables precise filtering. We can easily support multiple size conventions, price sorting, and inventory checks. Catalog stats (e.g. how many dresses) inform system health.
- **Cons:** As an Excel file, this is only a prototype. Large inventories should move to SQL/NoSQL. Data consistency (spellings, missing values) is critical and requires cleaning.
- **Hybrid AI + Rules vs Pure Approaches:** Pure rule-based systems are predictable but can't cover every phrasing. Pure AI (end-to-end LLMs) can generalize but are black boxes and costly to run continuously. Our hybrid gives the **best of both**: fast, transparent rule-based answers for standard cases, with GPT-4 stepping in for the rest. Many companies employ this strategy, starting with hard-coded rules and then adding AI as data grows.
- **Knowledge Base vs End-to-End Learning:** Training a large model on all possible fashion queries would be expensive and opaque. Instead, we use a curated knowledge base for clear mappings. This is more maintainable and explainable (experts can edit JSON), though it requires manual upkeep.
- **Modular Pipeline vs Monolithic Design:** Each component (NLP, matching, GPT, filtering, NLG) is separate. This helps testing and allows individual upgrades (e.g. swap in a new NLP tool). The trade-off is managing interfaces between modules. In practice, the clarity and testability gained outweigh the slight overhead.
- **Conversational UI vs Static Forms:** We chose a free-form chat to let users express themselves naturally. Conversational interfaces have been shown to reduce friction, as users need not navigate a complex form. The downside is that conversations are unpredictable – we must carefully handle incomplete or contradictory inputs. A hybrid UI (chat plus optional buttons) could combine the best of both worlds.

Conclusion

This architecture provides a robust, explainable fashion recommender. By combining **rule-based matching** with **GPT-4 fallback**, it handles both routine and unexpected queries efficiently. The modular design means we can update vocabulary, swap models, or expand templates without reworking the whole system. Instead of relying solely on opaque neural methods, our hybrid approach ensures speed and predictability while still leveraging AI "smartness" when needed. In summary, the system delivers personalized outfit suggestions with a balance of human expertise and intelligent automation, meeting the needs of developers and stakeholders alike.