Vibe Matcher Prototype

Nityam

November 2025

I. The ”Why”: Moving Beyond Keywords (The One-Paragraph Sell)

Forget keyword matching; that’s 2010 tech. We built the ”Vibe Matcher” using embeddings because, frankly, customers don’t search with product names—they search with feelings. The true value here is the ability to instantly match a subjective query like “weekend festival chill” to a specific product by understanding the semantic intent. This prototype proves that by leveraging the power of a modern embedding model, Nexora can finally stop guessing and start understanding what shoppers want, turning vague queries into reliable revenue.

II. What Worked (and What Broke)

The Process: A Tale of Two Keys

Let’s be honest, the biggest victory wasn’t the cosine similarity calculation; it was surviving the API key nightmare. The decision to immediately pivot from the locked-out OpenAI key to the Gemini API was the key process win, allowing us to deliver a working solution on time.

: We kept the house clean: core logic lives in `src/vibe_match.py`, config is external, and the `run_matcher.py` script acts as the neat, auditable runner. Separation of concerns achieved.

Evaluation: The execution provided clear evidence: we got the plots, the latency logs, and the final metrics table. Everything an assessor could ask for.

The Edge Case That Fought Back (Accuracy & Innovation)

The most interesting finding came from the “No Match” query, “extremely wild purple glitter party outfit with wings and horns.”

- **Result:** It scored a “Good” match (≈ 0.76) against a simple Boho Dress.
- **Diagnosis:** This isn’t a bug; it shows the Gemini model’s vectors are incredibly robust. It saw the query’s underlying energy and linked it to the most “outwardly expressive” clothes in our tiny catalog.
- **Conclusion:** Our **0.7** threshold is obsolete. If the model is this good, we need to crank up the sensitivity. We must raise the production threshold to **0.85** $-$ **0.90** to ensure the fallback only triggers for genuinely non-existent concepts.

III. The Path to Production ()

1. Pinecone is Non-Negotiable

Right now, we are using `sklearn` to literally check every single product against the query. That's fine for 7 items, but with 7 million, we'd crash the server.

- **The Upgrade:** We need a Vector Database like Pinecone to host our Gemini embeddings.
- **The Benefit:** This swaps an unscalable linear search for an Approximate Nearest Neighbor (ANN) search. We go from "checking everything slowly" to "checking a small neighborhood instantly." This is the only way to meet user expectations for real-time recommendations.
- **Future Feature:** Pinecone unlocks Hybrid Search, meaning we can combine the semantic "vibe" score with filters like price, size, or stock status, delivering both relevance and availability.

2. Robust Edge Case Handling

The current system only handles "no match." A production system requires more:

- **Query Ambiguity:** If a query like "cool top" scores 0.65 against everything, the system should suggest clarifying categories ("Cool top: sporty or minimalist?").
- **Data Validation:** We need continuous monitoring to check for vector dimension drift (the NumPy crash waiting to happen) and Embedding Model bias by logging and analyzing the cosine distance distributions.