FinMatch: Multimodal Embedding and Vector Search Pipeline

Amine Rachid Nour Zouari Karim Dami Eya Azouzi

January 26, 2026

Contents

1 Overview

FinMatch is a **multimodal recommendation system** that combines:

- Text
- Images
- Audio

All inputs are converted into a **single semantic vector** using pretrained models. This vector can be stored in a vector database such as **Qdrant** for similarity search.

Main technologies used:

- CLIP — for text and image embeddings
- Whisper — for audio transcription
- PyTorch — for tensor operations
- Qdrant — for vector storage and retrieval

2 Pipeline Description

The system follows these steps:

1. Transcribe audio into text using Whisper
2. Generate embeddings for text using CLIP
3. Generate embeddings for images using CLIP
4. Normalize all embeddings
5. Fuse embeddings using weighted combination
6. Store vectors in Qdrant

3 Embedding Functions

All embeddings have dimension **512** and are L2-normalized.

3.1 Audio Embedding

Audio is first converted into text, then embedded using CLIP.

```python
def audioVector(audio_path):
    result = whisper_model.transcribe(audio_path)
    text = result["text"]

    text_inputs = processor(
        text=[text],
        return_tensors="pt",
        padding=True
    )

    text_emb = model.get_text_features(**text_inputs)

    # Normalize embedding
    text_emb = text_emb / text_emb.norm(dim=1, keepdim=True)

    return text_emb
```

Explanation:

- Whisper converts speech to text
- CLIP embeds the resulting text
- Normalization ensures cosine similarity works correctly

3.2 Text Embedding

```
def txtVector(text):
    text_inputs = processor(
        text=[text],
        return_tensors="pt",
        padding=True
    )

    text_emb = model.get_text_features(**text_inputs)
    text_emb = text_emb / text_emb.norm(dim=1, keepdim=True)

    return text_emb
```

This function directly embeds textual input into a semantic vector.

3.3 Image Embedding

```
def imVector(image):
    image_inputs = processor(
        images=image,
        return_tensors="pt"
    )

    image_emb = model.get_image_features(**image_inputs)
    image_emb = image_emb / image_emb.norm(dim=1, keepdim=True)

    return image_emb
```

Images are encoded using CLIP's image encoder.

4 Multimodal Fusion Logic

Each modality is assigned a weight:

$$x_{\text{image}} = 0.4, \quad x_{\text{text}} = 0.4, \quad x_{\text{audio}} = 0.2$$

If a modality is missing, its weight is redistributed to the remaining ones.

```
q = v_image * x1 + v_text * x2 + v_audio * x3
q = q / q.norm(dim=1, keepdim=True)
```

Result: A single normalized vector representing all available modalities.

5 Main Script

The main script:

- Loads available inputs
- Computes embeddings
- Applies fusion logic

```python
if __name__ == "__main__":
    x1, x2, x3 = 0.4, 0.4, 0.2
    v0 = torch.zeros(1, 512)
    v1=v0.clone()
    v2=v0.clone()
    v3=v0.clone()

    if torch.all(v1==v0):
        x2=x2+x1/2
        x3=x3+x1/2
        x1=0

    if torch.all(v2==v0):
        x1=x1+x2/2
        x3=x3+x2/2
        x2=0

    if torch.all(v3==v0):
        x1=x1+x3/2
        x3=0

    v_image = v_text = v_audio = v0.clone()

    if txt_path.exists():
        v_text = txtVector(text)

    if img_path.exists():
        v_image = imVector(image)

    if audio_path.exists():
        v_audio = audioVector(audio_path)

    q = v_image*x1 + v_text*x2 + v_audio*x3
    q = q / q.norm(dim=1, keepdim=True)

    print(q)
```

6 CSV Product Embedding Preparation

Products are read from a CSV file and embedded using text only.

6.1 Reading Product Data

```python
def read_products(path):
    with open(path, newline="", encoding="utf-8") as f:
        reader = csv.reader(f)
        next(reader)
        return list(reader)
```

6.2 Text Embedding Preparation

```python
def prepareVector(product):
    text = product[2] + "␣" + product[5]
    v = txtVector(text)
    return v.squeeze(0)
```

7 Vector Storage with Qdrant

7.1 Connection

```python
client = QdrantClient(
    url="https://<QDRANT_URL>",
    api_key="<API_KEY>"
)
```

7.2 Point Construction and Insertion

```python
points.append(
    PointStruct(
        id=int(p[0]),
        vector=v,
        payload={
            "title": p[2],
            "price": p[3],
            "ram": p[8],
            "os": p[16]
        }
    )
)

client.upsert(
    collection_name="finmatch",
    points=points
)
```

8 Conclusion

This system:

- Combines multiple modalities into one vector
- Enables semantic similarity search
- Scales efficiently using Qdrant

It forms a solid foundation for intelligent product recommendation systems.