عنوان پروژه:
Multimodal RAG

نام دانشجو: حسین آریان‌مهر
شماره دانشجویی: ۴۰۱۱۲۶۲۱۶۷
استاد: دکتر هادی صدوقی یزدی

۱ فهرست

فهرست مطالب

۲ مقدمه

در سال‌های اخیر، مدل‌های زبانی بزرگ (*LLMs*) توانسته‌اند پیشرفت چشم‌گیری در درک و تولید زبان طبیعی داشته باشند. با این حال، این مدل‌ها محدود به دانشی هستند که در زمان آموزش به آن‌ها داده شده است و توانایی دسترسی مستقیم به داده‌های جدید یا به‌روزشده را ندارند. برای حل این محدودیت، رویکردی به نام *RAG (Retrieval-Augmented Generation)* معرفی شده است. *RAG* ترکیبی از دو بخش اصلی است:

۱. بازیابی اطلاعات (*Retrieval*) از منابع بیرونی مانند پایگاه‌های داده، اسناد متنی یا بردارهای معنایی،

۲. تولید پاسخ (*Generation*) با استفاده از مدل زبانی که از اطلاعات بازیابی‌شده به‌صورت زمینه‌ای بهره می‌گیرد.

به بیان ساده، *RAG* به جای اتکا صرف به حافظه‌ی داخلی مدل، از داده‌های واقعی و به‌روز برای تولید پاسخ دقیق‌تر، مرتبط‌تر و قابل اعتمادتر استفاده می‌کند. این روش باعث می‌شود مدل در پاسخ‌گویی به سؤالات تخصصی، استناد به منابع و کاهش خطاهای توهم‌زا (*hallucinations*) عملکرد بهتری داشته باشد. با این حال، در بسیاری از کاربردهای واقعی، داده‌های موجود فقط متنی نیستند. امروزه حجم زیادی از اطلاعات به‌صورت تصویر، صوت، ویدیو و سایر حالت‌های غیرمتنی ذخیره می‌شوند. اینجاست که مفهوم *Multimodal RAG* مطرح می‌شود.

Multimodal RAG نسخه‌ی پیشرفته‌تری از *RAG* است که قادر است از چند نوع داده (یا «مودالیتی») به‌صورت هم‌زمان استفاده کند؛ *Multimodal RAG (Retrieval-Augmented Generation)* ترکیبی از دو مفهوم کلیدی در هوش مصنوعی است: بازیابی اطلاعات (*Retrieval*) و تولید متن (*Generation*)، که در آن مدل نه تنها از داده‌های متنی بلکه از چندین نوع داده (مانند تصویر، صدا، و ویدیو) برای پاسخ‌گویی استفاده می‌کند. در این معماری، ابتدا ورودی کاربر که می‌تواند شامل متن، تصویر یا صوت باشد، پردازش می‌شود. برای هر نوع داده، بردار ویژگی (*Embedding*) تولید می‌شود تا بتوان آن را در فضای مشترک مقایسه کرد. سپس این بردارها به یک پایگاه داده برداری (*Vector Database*) مانند *Weaviate*، *Pinecone* یا *FAISS* فرستاده می‌شوند تا نزدیک‌ترین داده‌ها (*retrieved items*) از میان داده‌های ذخیره‌شده پیدا شوند. این داده‌های بازیابی‌شده که ممکن است شامل چندین مدالیته (مثل توضیح متنی، تصویر مرتبط، یا توصیف صوتی) باشند، به‌صورت ساختاریافته به یک مدل زبانی بزرگ (*LLM*) فرستاده می‌شوند. مدل زبانی با ترکیب دانش خود و داده‌های بازیابی‌شده، پاسخ نهایی را تولید می‌کند. ویژگی اصلی *Multimodal RAG* این است که برخلاف *RAG* سنتی که فقط با متن کار می‌کند، می‌تواند اطلاعات متنوع را باهم ترکیب کند؛ مثلاً از روی تصویر و توضیحات صوتی درباره آن نتیجه‌گیری کند. این سیستم کاربردهای گسترده‌ای دارد، از جمله در چت‌بات‌های هوشمند چندوجهی، جست‌وجوی تصویری و صوتی، سیستم‌های آموزشی تعاملی، و تحلیل داده‌های چندرسانه‌ای. نیاز به *Multimodal RAG* از آن‌جا ناشی می‌شود که بسیاری از مسائل دنیای واقعی چندوجهی هستند. برای مثال، در یک سامانه‌ی جست‌وجوی هوشمند ممکن است کاربر تصویری از یک شیء ارسال کند، توضیح صوتی ارائه دهد، یا سؤالی متنی بپرسد، و سیستم باید بتواند از میان انواع داده‌های موجود، مرتبط‌ترین پاسخ را تولید کند. به طور خلاصه، *Multimodal RAG* با ترکیب قابلیت‌های درک چندنوع داده و تولید پاسخ زبانی، مشکلات زیر را برطرف می‌کند:

- محدودیت مدل‌های متنی در استفاده از داده‌های تصویری یا صوتی،
- ضعف مدل‌های زبانی در به‌روزرسانی دانش و استناد به داده‌های جدید،
- ناتوانی سیستم‌های تک‌مودال در پاسخ به پرسش‌های پیچیده و چندمنظوره.

بنابراین، *Multimodal RAG* گامی مهم در جهت ایجاد سامانه‌های هوشمند جامع‌تر، تعاملی‌تر و آگاه‌تر از دنیای واقعی محسوب می‌شود.

۱.۲ کاربردهای *Multimodal RAG*

- جست‌وجوی هوشمند چندرسانه‌ای
- سیستم‌های پاسخ‌گوی تصویری و صوتی
- دستیارهای مجازی با ورودی چندحالته
- تحلیل محتوای چندرسانه‌ای (متن، تصویر، صوت)
- بازیابی اطلاعات پزشکی از تصاویر و گزارش‌ها
- سیستم‌های آموزشی تعاملی
- تحلیل و مدیریت داده‌های امنیتی (تصویر و صدا)
- تولید توضیحات خودکار برای تصاویر یا ویدیوها
- موتورهای توصیه‌گر چندرسانه‌ای
- ربات‌های گفت‌گوگر با درک چندوجهی

۳ انواع روش‌های *Multimodal RAG*

۱.۳ رویکرد مبتنی بر توصیف متنی *(Textual Description-Based MRAG)*

در این روش، برای هر داده‌ی غیرمتنی مانند تصویر یا ویدئو، یک توصیف متنی (*Textual Description*) توسط یک مدل زبانی بزرگ (مانند *Gemini* یا *GPT*) تولید می‌شود. به این ترتیب، تمامی داده‌ها — صرف‌نظر از نوع *modality* — در قالب متن

بازنمایی می‌شوند. در مرحله‌ی بازیابی (*Retrieval*)، ورودی متنی کاربر با توصیف‌های متنی موجود در پایگاه داده‌ی برداری (*Vector Database*) مقایسه می‌شود و داده‌هایی که از لحاظ معنایی بیشترین شباهت را دارند (اعم از متن یا تصویر و...) بازیابی می‌گردند. در نهایت، داده‌های بازیابی‌شده به همراه ورودی کاربر به مدل زبانی ارسال می‌شوند تا خروجی نهایی تولید شود.

۱.۱.۳ مزایا

پیاده‌سازی ساده امکان پیاده‌سازی *MRAG* حتی در شرایطی که مدل زبانی چندوجهی (*MLLM*) در دسترس نباشد.

هزینه کمتر استفاده از مدل‌های تولید توضیح (*Captioning / Translation Models*) که سبک‌تر و ارزان‌تر از مدل‌های *MLLM* هستند.

۲.۱.۳ معایب

افت اطلاعات احتمال بروز افت اطلاعات (*Information Loss*) در فرآیند تبدیل داده‌های غیرمتنی به متن.

دقت پایین‌تر دقت پایین‌تر در مقایسه با روش‌های چندوجهی مستقیم، به‌ویژه در کاربردهایی که جزئیات بصری یا صوتی اهمیت بالایی دارند.

۳.۱.۳ کاربرد پیشنهادی

این روش زمانی مناسب است که مدل زبانی فقط از داده‌های متنی پشتیبانی کند (*LLM* غیرچندوجهی). با این حال، حتی در مدل‌های چندوجهی نیز می‌توان از این رویکرد برای انجام مرحله‌ی *Retrieval* بر اساس متن استفاده نمود، و سپس داده‌های اصلی را برای مرحله‌ی تولید نهایی (*Generation*) به *MLLM* ارسال کرد.

۲.۳ رویکرد مبتنی بر فضای تعبیه‌ی مشترک *(Unified Embedding Space MRAG)*

در این روش، تمامی داده‌ها — اعم از متن، تصویر، صوت و ویدئو — به یک فضای تعبیه‌ی مشترک (*Shared Embedding Space*) نگاشت می‌شوند. برای این منظور از مدل‌هایی مانند *CLIP* استفاده می‌شود که قادرند داده‌های متنی و تصویری را در یک فضا نمایش دهند. در این حالت، ورودی کاربر نیز به همان فضای تعبیه نگاشت می‌شود و داده‌هایی که بیشترین شباهت را از نظر برداری دارند، بازیابی شده و به مدل زبانی ارسال می‌گردند.

۱.۲.۳ مزایا

- دقت بالاتر نسبت به روش اول.
- کاهش چشمگیر افت اطلاعات (*Information Loss*) به دلیل عدم نیاز به توصیف متنی واسطه.
- امکان سنجش مستقیم شباهت بین *modality*های مختلف (برای مثال: تصویر و متن).

۲.۲.۳ چالش‌ها

- نیاز به مدل‌های بسیار انعطاف‌پذیر که بتوانند تمامی *modality*ها (متن، تصویر، صوت، ویدئو و...) را به یک فضای مشترک مپ کنند.
- محدود بودن مدل‌هایی که چنین قابلیتی دارند (برای مثال *CLIP* فقط برای تصویر و متن طراحی شده است).

۳.۳ رویکرد چندمدلی ترکیبی *(Hybrid Multimodal Embedding MRAG)*

در صورتی که مدلی وجود نداشته باشد که تمام *modality*ها را به یک فضای تعبیه‌ی واحد نگاشت کند، می‌توان از مجموعه‌ای از مدل‌ها استفاده کرد که هر یک تنها بخشی از *modality*ها را پوشش می‌دهند. برای مثال:

- مدلی مانند *CLIP* برای **متن و تصویر**
- مدلی دیگر برای **صوت و متن**
- و مدل‌های مشابه برای سایر ترکیبات

در زمان بازیابی، ورودی کاربر به همه‌ی این مدل‌ها داده می‌شود تا *embedding* متناظر در هر فضای تعبیه تولید شود. سپس، فرآیند بازیابی در هر فضای برداری انجام شده و مجموعه‌ای از داده‌های مشابه از هر مدل به‌دست می‌آید. در نهایت، از یک *Re-Ranker* برای انتخاب n مورد از نزدیک‌ترین داده‌ها به ورودی کاربر استفاده می‌شود و این داده‌ها به مدل زبانی چندوجهی (*MLLM*) ارسال می‌گردند.

۱.۳.۳ مزایا

انعطاف‌پذیری انعطاف‌پذیری بالا در شرایطی که مدل یکپارچه‌ی چندوجهی در دسترس نیست.

بهینه‌سازی امکان استفاده‌ی هم‌زمان از مدل‌های بهینه‌شده برای *modality*های خاص.

۲.۳.۳ معایب

پیچیدگی پیچیدگی پیاده‌سازی بالا به‌دلیل نیاز به هماهنگی بین چند فضای تعبیه‌ی متفاوت.

نیاز محاسباتی نیاز به زمان و توان محاسباتی بیشتر برای انجام فرآیندهای *retrieve* و *re-rank*.

۴ پایگاه داده برداری

پایگاه داده برداری سیستمی است که می‌تواند بردارهای عددی با ابعاد بالا (یعنی *embedding*ها) را ذخیره، ایندکس و جست‌وجو کند. این بردارها معمولاً نشان‌دهنده ویژگی‌های معنایی داده‌هایی مانند متن، تصویر، صوت یا ترکیبی از آن‌ها هستند. وقتی کاربر یک پرسش وارد می‌کند، آن پرسش نیز به یک بردار تبدیل می‌شود و سپس پایگاه داده، بردارهای نزدیک از نظر فاصله یا شباهت را بازمی‌گرداند، یعنی مشابه‌ترین داده‌های ذخیره‌شده را پیدا می‌کند. از آن‌جا که بردارها در ابعاد زیاد هستند، معمولاً از الگوریتم‌های میانگرا (*Approximate Nearest Neighbor*) مانند *HNSW* و روش‌های مشابه برای سرعت بخشیدن به جست‌وجو استفاده می‌شود.

۱.۴ دلایل نیاز به پایگاه دادهٔ برداری و ناکافی بودن پایگاه داده‌های سنتی

پایگاه دادهٔ برداری نوعی سامانهٔ ذخیره‌سازی و بازیابی داده است که برای مدیریت مؤثر داده‌های با ابعاد بالا طراحی شده است. این نوع پایگاه داده‌ها نقش مهمی در کاربردهای مبتنی بر هوش مصنوعی، به‌ویژه در جستجوی معنایی و سیستم‌های *RAG* دارند. در ادامه، دلایل اصلی نیاز به چنین پایگاه داده‌هایی بیان می‌شود:

۱. **جستجوی شباهت معنایی** (*Semantic Similarity Search*)
در بسیاری از کاربردها هدف یافتن محتواهای مرتبط از نظر معناست، نه صرفاً از نظر تطابق واژگانی. در این حالت، هر داده (مانند متن یا تصویر) به یک بردار عددی در فضای ویژگی‌ها تبدیل می‌شود و جستجو بر اساس معیارهایی مانند شباهت کسینوسی یا فاصلهٔ اقلیدسی انجام می‌گیرد.

۲. **کارایی و مقیاس‌پذیری در داده‌های بزرگ و غیرساختاریافته**
بخش زیادی از داده‌های دنیای واقعی (نظیر متن، تصویر و صوت) غیرساختاریافته‌اند. مدل‌های یادگیری عمیق این داده‌ها را به بردار تبدیل می‌کنند، و پایگاه دادهٔ برداری امکان ذخیره، ایندکس‌گذاری و بازیابی سریع این بردارها را در مقیاس وسیع فراهم می‌کند.

۳. **پشتیبانی از به‌روزرسانی پویا و داده‌های در حال تغییر**
در سیستم‌های تعاملی یا مبتنی بر یادگیری مداوم، داده‌ها به‌صورت پیوسته اضافه یا اصلاح می‌شوند. پایگاه دادهٔ برداری از درج، حذف و به‌روزرسانی بردارها پشتیبانی می‌کند و ایندکس‌های خود را به شکل بهینه به‌روزرسانی می‌نماید.

۴. **محدودیت پایگاه داده‌های سنتی در جستجوی معنایی**
پایگاه داده‌های رابطه‌ای یا *NoSQL* برای داده‌های ساختاریافته طراحی شده‌اند و قابلیت جستجو در فضای معنایی را ندارند. این پایگاه‌ها معمولاً تنها بر پایهٔ تطابق کلیدواژه عمل می‌کنند و نمی‌توانند مشابهت مفهومی میان داده‌ها را تشخیص دهند.

به طور خلاصه، پایگاه داده‌های سنتی در ذخیره و جستجوی داده‌های برداری با ابعاد بالا ناکارآمد هستند، در حالی که پایگاه دادهٔ برداری به‌طور اختصاصی برای جستجوی شباهت‌محور، پردازش داده‌های غیرساختاریافته و به‌روزرسانی پویا طراحی شده است.

۲.۴ چند نمونه از پایگاه داده‌های برداری *(Vector Databases)*

Pinecone یک سرویس ابری کاملاً مدیریت‌شده برای ذخیره و جستجوی بردارها است که کار با آن ساده بوده و برای پروژه‌های مقیاس‌پذیر و مبتنی بر هوش مصنوعی مناسب است.

Milvus پایگاه داده‌ای منبع‌باز و بسیار مقیاس‌پذیر است که برای جستجوی شباهت در داده‌های حجیم (متن، تصویر، صوت) طراحی شده و در کاربردهای صنعتی استفاده می‌شود.

Weaviate پایگاه دادهٔ منبع‌باز با قابلیت جستجوی ترکیبی است که هم برداری و هم کلیدواژه‌ای را پشتیبانی می‌کند و برای سیستم‌های *RAG* و چت‌بات‌های هوشمند بسیار مناسب است.

Chroma پایگاه داده‌ای سبک و ساده است که برای برنامه‌های مبتنی بر مدل‌های زبانی بزرگ (*LLM*) طراحی شده و اغلب در پروژه‌های تحقیقاتی یا نمونه‌سازی سریع استفاده می‌شود.

۵ Weaviate

Weaviate یک پایگاه دادهٔ برداری متن‌باز است که به‌طور ویژه برای توسعهٔ سریع برنامه‌های مبتنی بر هوش مصنوعی طراحی شده است. این سامانه قابلیت‌هایی مانند جستجوی برداری، جستجوی ترکیبی (*Hybrid Search*)، فیلترگذاری پیشرفته و ادغام مستقیم با مدل‌های یادگیری ماشین را در اختیار توسعه‌دهندگان قرار می‌دهد. یکی از ویژگی‌های برجستهٔ *Weaviate*، معماری *AI-Native* آن است؛ به این معنا که از ابتدا برای کار با داده‌های برداری (*Embeddings*) و عملیات مرتبط با هوش مصنوعی طراحی شده است، نه به عنوان افزونه‌ای بر یک پایگاه دادهٔ سنتی. از مهم‌ترین مزایای *Weaviate* می‌توان به موارد زیر اشاره کرد:

۱. **جستجوی هیبرید (*Hybrid Search*):** ترکیب جستجوی برداری و جستجوی متنی در یک چارچوب واحد برای دستیابی به نتایج دقیق‌تر.

۲. **انعطاف‌پذیری در استقرار:** قابلیت اجرا به صورت محلی، در سرورهای شخصی یا به شکل سرویس مدیریت‌شده در فضای ابری.

۳. **مدیریت داده‌های متادیتا و فیلترها:** امکان فیلتر کردن نتایج جستجو بر اساس ویژگی‌ها یا متادیتاهای دلخواه (مانند برچسب‌ها یا نوع داده).

۴. **پشتیبانی از مدل‌های یادگیری ماشین:** ادغام آسان با مدل‌های *embedding* و چارچوب‌های مختلف یادگیری ماشین.

۵. **متن‌باز بودن:** استفاده از نسخهٔ *self-hosted* بدون نیاز به پرداخت هزینهٔ لایسنس و با کنترل کامل بر زیرساخت و داده‌ها.

۱.۵ نسخه‌های :Weaviate لوکال و ابری

Weaviate در دو حالت اصلی قابل استفاده است: نسخهٔ لوکال (*Self-hosted / On-premises*) و نسخهٔ ابری (*Managed / Weaviate Cloud*). هر کدام از این نسخه‌ها مزایا و کاربردهای خاص خود را دارند.

۱.۱.۵ نسخهٔ لوکال (*Self-hosted / On-premises*)

از آن‌جا که *Weaviate* به‌صورت متن‌باز عرضه شده است، می‌توان آن را بر روی سرور یا سیستم محلی خود با استفاده از ابزارهایی مانند *Docker* یا *Kubernetes* راه‌اندازی کرد. در این حالت، نیازی به پرداخت هزینهٔ لایسنس وجود ندارد و تنها هزینه‌ها مربوط به زیرساخت (نظیر سرور، فضای ذخیره‌سازی و نگهداری) خواهد بود. در نسخهٔ لوکال، کنترل کامل بر داده‌ها، امنیت، پیکربندی، و بهینه‌سازی عملکرد بر عهدهٔ کاربر است. عملکرد و مقیاس‌پذیری نیز مستقیماً به منابع سخت‌افزاری موجود بستگی دارد.

۲.۱.۵ نسخهٔ ابری (*Weaviate Cloud / Managed Service*)

Weaviate نسخهٔ ابری خود را در قالب چند سطح خدمات ارائه می‌دهد:

Serverless Cloud این نسخه کاملاً مدیریت‌شده و دارای مقیاس خودکار است. برای پروژه‌های توسعه، آزمایشی یا کاربردهای متوسط مناسب بوده و معمولاً دارای دورهٔ آزمایشی رایگان (*Free Trial*) است.

Enterprise Cloud نسخه‌ای کاملاً مدیریت‌شده با منابع اختصاصی، پشتیبانی فنی و توافق‌نامهٔ سطح خدمات (*SLA*) بالا. مناسب برای پروژه‌های بزرگ و سازمان‌هایی است که به عملکرد و پایداری بالا نیاز دارند.

Sandbox نسخه‌ای کاملاً رایگان برای استفاده‌های کوتاه‌مدت و آزمایشی است. داده‌های ذخیره‌شده در این نسخه تنها تا ۱۴ روز نگهداری می‌شوند و پس از آن به‌صورت خودکار از فضای ابری حذف خواهند شد.

۲.۵ سرویس Weaviate و نحوهٔ راه‌اندازی نسخه‌های آن

سرویس *Weaviate* در دو نسخهٔ اصلی ارائه می‌شود: نسخهٔ لوکال (*Self-Hosted*) و نسخهٔ ابری (*Cloud*). نسخهٔ لوکال به‌صورت متن‌باز (*Open Source*) و رایگان در دسترس است و به کاربران این امکان را می‌دهد تا پایگاه دادهٔ خود را بر روی سرور یا سیستم محلی اجرا و مدیریت کنند. برای راه‌اندازی نسخهٔ لوکال، می‌توان از ابزار *Docker* استفاده کرد. در این حالت، لازم است ابتدا *Docker* روی سیستم نصب شود. سپس فایلی با نام *docker-compose.yml* ایجاد کرده و محتوای زیر در آن قرار گیرد:

```
version: '3.4'
services:
  weaviate:
    image: semitechnologies/weaviate:latest
    restart: always
    ports:
      - "8080:8080"
      - "50051:50051"
    environment:
      QUERY_DEFAULTS_LIMIT: 25
```

```
      AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: 'true'
      PERSISTENCE_DATA_PATH: "./data"
      DISABLE_MODULES: 'all'
```

با اجرای دستور زیر، سرویس *Weaviate* به صورت محلی بر روی پورت ۸۰۸۰ در دسترس خواهد بود:

```
docker-compose up -d
```

پس از راه‌اندازی، می‌توان با استفاده از کتابخانهٔ رسمی *Weaviate* در پایتون، به سرور متصل شد و یک *Collection* (مجموعه داده) ایجاد کرد. نمونه‌کد زیر نحوهٔ انجام این کار را نشان می‌دهد:

```
import weaviate
import weaviate.classes as wvc

# 1.      Weaviate
try:
    client = weaviate.connect_to_local()
    print("Successfully connected to Weaviate!")
except Exception as e:
    print(f"Failed to connect to Weaviate: {e}")
    exit()

collection_name = "Multimodal_Collection"

# 2.   Collection     )      (
if client.collections.exists(collection_name):
    print(f"Collection '{collection_name}' already exists. Deleting it.")
    client.collections.delete(collection_name)

# 3.    Collection
print(f"Creating collection '{collection_name}'...")
my_collection = client.collections.create(
    name=collection_name,
    properties=[
        wvc.config.Property(name="modality", data_type=wvc.config.DataType.TEXT),

  wvc.config.Property(name="content", data_type=wvc.config.DataType.TEXT),
        wvc.config.Property(name="contentId", data_type=wvc.config.DataType.TEXT),
        wvc.config.Property(name="filePath", data_type=wvc.config.DataType.TEXT),
    ]
)
print(f"Collection '{collection_name}' created successfully.")

client.close()
```

برای اتصال مجدد به پایگاه داده و استفاده از مجموعهٔ ایجادشده، می‌توان از کد زیر استفاده کرد:

```
import weaviate

try:
    weaviate_client = weaviate.connect_to_local()
    print (" Connected to Weaviate")
except Exception as e:
    print(f " Weaviate connection failed: {e}")
    weaviate_client = None
```

این روش امکان استقرار سریع، کنترل کامل بر داده‌ها، و توسعهٔ پروژه‌های مبتنی بر هوش مصنوعی را در محیط محلی فراهم می‌کند.

۳.۵ راه‌اندازی نسخهٔ Sandbox از Cloud Weaviate

نسخهٔ *Cloud* یا *Weaviate Cloud Service (WCS)* به‌صورت تجاری ارائه می‌شود و استفاده از آن به‌طور کامل رایگان نیست. با این حال، امکان استفادهٔ آزمایشی (*Sandbox*) از این سرویس برای مدت محدود وجود دارد. در نسخهٔ *Sandbox*، می‌توان هر *Collection* را به‌مدت ۱۴ روز به‌صورت رایگان در یک *Cluster* در فضای ابری *Weaviate* ایجاد و نگهداری کرد. پس از پایان این دوره، داده‌ها به‌صورت خودکار حذف می‌شوند. برای استفادهٔ بلندمدت یا در مقیاس بزرگ‌تر، لازم است از نسخه‌های پولی *Weaviate Cloud* استفاده شود. مراحل ایجاد یک نسخهٔ *Sandbox* به شرح زیر است:

۱. ورود به وب‌سایت رسمی *Weaviate Cloud* به آدرس *https://console.weaviate.cloud/*

۲. ایجاد حساب کاربری جدید (*Sign Up*)

۳. مراجعه به تب *Clusters*

۴. کلیک بر روی آیکون "+" برای ایجاد یک کلاستر جدید

۵. انتخاب گزینهٔ *Sandbox*

۶. وارد کردن نام دلخواه برای پایگاه داده و انتخاب دکمهٔ *Create Cluster*

پس از ایجاد کلاستر، لازم است یک *API Key* تولید کنید. این کلید از طریق صفحهٔ تنظیمات همان کلاستر قابل دسترسی است. سپس با استفاده از گزینهٔ "*How to Connect*" می‌توانید آدرس‌های اتصال (*URLs*) و دستورالعمل‌های مربوط به نحوهٔ ارتباط با پایگاه داده را مشاهده کرده و در کد خود مورد استفاده قرار دهید.

۶ مدل‌های تولید *Embedding*

مدل‌های تولید *Embedding*، مدل‌هایی هستند که داده‌های ورودی — مانند متن، تصویر، صوت یا حتی ویدیو — را به بردارهای عددی (*vector representations*) تبدیل می‌کنند. این بردارها به‌گونه‌ای طراحی می‌شوند که ویژگی‌های معنایی و محتوایی داده را در یک فضای ریاضی (*latent space*) نمایش دهند. در این فضا، داده‌های مشابه از نظر معنا یا محتوا به یکدیگر نزدیک‌تر هستند. هدف اصلی این مدل‌ها ایجاد یک فضای معنایی مشترک میان داده‌های مختلف است؛ به‌طوری که بتوان روابط میان مدالیته‌های گوناگون (مانند شباهت بین متن و تصویر) را محاسبه کرد. برای مثال، اگر مدل بداند «یک گربه روی میز» چه معنایی دارد، تصویر واقعی گربه روی میز نیز در همان فضای برداری نزدیک به این توصیف قرار می‌گیرد.

۱.۶ کاربردهای مدل‌های *Embedding*

- جست‌وجوی چندوجهی (*Multimodal Retrieval*): یافتن تصاویر، ویدیوها یا صداهایی که از نظر معنا با یک جمله یا پرسش متنی مشابه هستند.
- سیستم‌های *RAG* و *Chatbot*های چندوجهی: ترکیب متن، تصویر و صوت در یک فضای مشترک برای پاسخ‌گویی هوشمندتر.
- دسته‌بندی و خوشه‌بندی داده‌ها: بر اساس شباهت معنایی در فضای *embedding*.
- تشخیص شباهت تصویر یا متن (*Similarity Matching*): برای سیستم‌های پیشنهاددهنده یا کشف محتوای مشابه.
- درک زمینه‌ای در مدل‌های مولد: مثلاً برای تولید توضیح متن برای تصاویر (*Image Captioning*).

۲.۶ نمونه‌هایی از مدل‌های معروف تولید *Embedding*

۱.۲.۶ *CLIP (Contrastive Language–Image Pretraining)* — ساختهٔ *OpenAI*

قابلیت‌ها
- آموزش‌دیده بر میلیون‌ها جفت متن و تصویر.
- ایجاد فضای معنایی مشترک بین متن و تصویر.
- امکان انجام جست‌وجوی متنی در میان تصاویر و بالعکس.

مزایا دقت بالا در درک ارتباط بین متن و تصویر؛ قابل استفاده برای *zero-shot classification*.

محدودیت‌ها فقط از دو مدالیته (متن و تصویر) پشتیبانی می‌کند؛ عملکرد آن به کیفیت داده‌های آموزشی وابسته است.

۲.۲.۶ *ImageBind* — ساختهٔ *Meta AI*

قابلیت‌ها
- پشتیبانی از شش مدالیته: تصویر، متن، صوت، حرکات (*motion*)، عمق (*depth*)، و داده‌های حرارتی (*thermal*).
- تمام این مدالیته‌ها را در یک فضای برداری واحد قرار می‌دهد.
- امکان تطبیق داده‌های ناهمگون (مثلاً جست‌وجوی صوتی برای یافتن تصویر مرتبط).

مزایا ادغام چندنوع داده در یک فضای *embedding* مشترک؛ مناسب برای پروژه‌های چندرسانه‌ای پیشرفته.

محدودیت‌ها مدل بزرگ و سنگین است؛ برای اجرا نیاز به *GPU* قدرتمند دارد؛ داده‌های آموزشی آن عمومی نیستند.

۳.۲.۶ *OpenCLIP* — نسخهٔ متن‌باز *CLIP*

قابلیت‌ها عملکرد مشابه *CLIP* ولی با داده‌های عمومی و قابل دسترس.

مزایا قابل استفاده در پروژه‌های متن‌باز؛ قابل آموزش مجدد روی داده‌های اختصاصی.

محدودیت‌ها گاهی دقت پایین‌تر از *CLIP* اصلی دارد؛ نیازمند تنظیم دقیق داده‌ها و پارامترهاست.

۴.۲.۶ *SigLIP (Google Research)*

قابلیت‌ها مدل بهینه‌شده بر پایهٔ *CLIP* با بهبود در نحوهٔ آموزش تقابلی.

مزایا دقت بالاتر در ارزیابی شباهت تصویر—متن؛ پایداری بیشتر در داده‌های نویزی.

محدودیت‌ها فقط دو مدالیتهٔ متن و تصویر را پشتیبانی می‌کند.

۵.۲.۶ *Whisper Embeddings* **(برای صوت) — ساختهٔ** *OpenAI*

قابلیت‌ها تبدیل گفتار به بردار معنایی یا متن توصیفی.

کاربرد جست‌وجوی معنایی میان فایل‌های صوتی، تحلیل گفتار، یا ترکیب با مدل‌های چندوجهی.

محدودیت تنها برای داده‌های صوتی طراحی شده است؛ نیاز به مدل مکمل برای ترکیب با متن یا تصویر دارد.

۳.۶ جمع‌بندی

مدل‌های تولید *embedding* مانند *CLIP* و *ImageBind* ستون فقرات بسیاری از سیستم‌های *Multimodal AI* هستند. آن‌ها داده‌های متفاوت را در یک فضای عددی مشترک بازنمایی می‌کنند تا بتوان میان مدالیته‌های گوناگون ارتباط برقرار کرد. *CLIP* برای متن و تصویر بسیار کارآمد است، در حالی که *ImageBind* یک گام جلوتر رفته و چندین نوع داده را در یک چارچوب یکپارچه ترکیب می‌کند. انتخاب میان آن‌ها بسته به نیاز پروژه، نوع داده‌ها و منابع سخت‌افزاری در دسترس انجام می‌شود.

۷ مدل *OpenCLIP*

مدل *OpenCLIP* یکی از نسخه‌های متن‌باز و توسعه‌یافته مدل مشهور *CLIP (Contrastive Language–Image Pretraining)* است که در اصل توسط شرکت *OpenAI* معرفی شد. این مدل به‌طور خاص برای درک و تحلیل ارتباط بین تصویر و متن طراحی شده و به‌عنوان یکی از ابزارهای کلیدی در سیستم‌های چندرسانه‌ای (*Multimodal*) شناخته می‌شود. کاربردهای آن شامل پروژه‌های *RAG (Retrieval-Augmented Generation)*، جستجوی تصویری، تولید توضیحات خودکار برای تصاویر و سایر برنامه‌های هوش مصنوعی چندوجهی است. *OpenCLIP* به‌عنوان یک نسخهٔ متن‌باز (*open-source*) توسط تیم‌های *LAION* و *Hugging Face* توسعه یافته است. هدف از ایجاد این مدل، فراهم کردن نسخه‌ای آزاد و قابل استفاده برای عموم بود تا بتواند همان عملکرد *CLIP* را بدون محدودیت‌های لایسنس *OpenAI* ارائه دهد. این مدل به‌صورت عمومی در مخازن *GitHub* و از طریق کتابخانه‌هایی مانند *open_clip* و *transformers* در دسترس است. *OpenCLIP* از مدل‌های ازپیش‌آموزش‌دیده مختلف (*pretrained weights*) پشتیبانی می‌کند که بر روی مجموعه داده‌های بسیار بزرگ مانند *LAION-400M* و *LAION-2B* آموزش دیده‌اند. هدف اصلی *OpenCLIP* یادگیری ارتباط معنایی بین تصویر و متن است؛ به این معنا که مدل قادر است بفهمد کدام تصویر با کدام توضیح متنی بیشترین تطابق و نزدیکی معنایی را دارد. این قابلیت، *OpenCLIP* را به ابزار قدرتمندی برای تولید *embedding* مشترک برای داده‌های تصویری و متنی تبدیل کرده است. چنین *embedding*هایی پایه و اساس بسیاری از سیستم‌های *Multimodal RAG* و سایر برنامه‌های هوش مصنوعی پیشرفته هستند. از مهم‌ترین کاربردهای *OpenCLIP* می‌توان به موارد زیر اشاره کرد:

- جستجوی تصویر بر اساس متن (*Text-to-Image Retrieval*)
- جستجوی متن بر اساس تصویر (*Image-to-Text Retrieval*)
- ساخت *embedding* مشترک برای داده‌های تصویری و متنی در سیستم‌های چندرسانه‌ای
- استفاده به‌عنوان *Vision Encoder* در مدل‌های بزرگ چندحالته مانند *GPT-4V* یا *LLaVA*

با توجه به متن‌باز بودن، *OpenCLIP* امکان استفاده و آموزش مجدد روی داده‌های اختصاصی را فراهم می‌کند و به توسعه‌دهندگان اجازه می‌دهد از مزایای *CLIP* در پروژه‌های شخصی یا تحقیقاتی بهره‌برداری کنند بدون آنکه محدود به لایسنس *OpenAI* باشند. این ویژگی‌ها *OpenCLIP* را به یکی از گزینه‌های محبوب برای کاربردهای چندوجهی و هوش مصنوعی تعاملی تبدیل کرده است.

۱.۷ نحوه عملکرد و استفاده از مدل *CLIP / OpenCLIP*

مدل *OpenCLIP* از دو شبکهٔ عصبی اصلی تشکیل شده است: *Text Encoder* و *Image Encoder*. *Text Encoder* (رمزگذار متنی) معمولاً مبتنی بر مدل‌های *Transformer* مانند *BERT* یا نسخه‌های کوچک‌تر *GPT* است. وظیفهٔ این بخش، تبدیل متن ورودی — مانند جمله یا توضیح یک تصویر — به یک بردار عددی با ابعاد ثابت (*embedding*) می‌باشد. این بردار نمایانگر ویژگی‌های معنایی متن است و امکان مقایسه آن با سایر داده‌ها را فراهم می‌کند. *Image Encoder* (رمزگذار تصویری) معمولاً از مدل‌هایی مانند *Vision Transformer (ViT)* یا *ResNet* تشکیل شده است. این بخش تصویر ورودی را به برداری عددی با همان ابعاد *embedding* متنی تبدیل می‌کند تا بتوان آن را در همان فضای معنایی با متن مقایسه کرد. در مرحلهٔ آموزش، مدل تلاش

می‌کند که *embedding* تصاویر و متن‌های مرتبط با آن‌ها را به یکدیگر نزدیک کند و *embedding*های نامرتبط را از هم دور نگه دارد. این فرایند تحت عنوان *Contrastive Learning* (یادگیری تقابلی) شناخته می‌شود. هدف آموزش معمولاً بر اساس شباهت کسینوسی (*cosine similarity*) میان بردار تصویر و بردار متن تعریف می‌شود، به‌طوری که مدل سعی دارد تصاویر و متون مرتبط بیشترین شباهت کسینوسی را داشته باشند.

۱.۱.۷ نحوهٔ استفاده عملی از *OpenCLIP*

در کاربردهای عملی، *OpenCLIP* به‌طور عمده برای استخراج *embedding* از تصاویر و متون به کار می‌رود. به‌عنوان مثال، وقتی کاربر یک تصویر ارسال می‌کند، *Image Encoder* برداری ۵۱۲ یا ۱۰۲۴ بُعدی از تصویر تولید می‌کند. هنگامی که یک توضیح متنی ارسال شود، *Text Encoder* برداری با همان ابعاد ایجاد می‌کند. با مقایسهٔ این بردارها از طریق معیارهایی مانند *cosine similarity* می‌توان تعیین کرد که کدام تصویر با کدام متن بیشترین تطابق معنایی را دارد. در سیستم‌های *Multimodal RAG*، این *embedding*ها در پایگاه‌های داده برداری مانند *Weaviate* ذخیره می‌شوند تا در زمان پرس‌وجو، داده‌های مشابه و مرتبط سریع بازیابی شوند.

۲.۷ مزایای *OpenCLIP*

OpenCLIP مجموعه‌ای از ویژگی‌ها و مزایای عملی را ارائه می‌دهد:

- متن‌باز بودن: امکان استفاده در پروژه‌های تحقیقاتی و تجاری بدون محدودیت لایسنس.
- پشتیبانی از مدل‌های متنوع: شامل *ViT-B*، *ViT-L*، *ViT-H* و سایر نسخه‌ها که امکان انتخاب بین سرعت و دقت را فراهم می‌کنند.
- سازگاری با پلتفرم‌های مدرن: قابلیت استفاده با *PyTorch*، *Hugging Face* و ادغام با پایگاه‌های داده برداری مانند *Weaviate*.
- عملکرد بالا در مقیاس بزرگ: امکان *fine-tuning* روی داده‌های خاص دامنه برای بهبود عملکرد در کاربردهای تخصصی.
- دسترسی به مدل‌های *pretrained* گسترده: آموزش دیده بر روی مجموعه داده‌های عظیم و چندزبانه که دقت و انعطاف‌پذیری بالایی ارائه می‌دهند.

این ویژگی‌ها *OpenCLIP* را به ابزاری قدرتمند و قابل اعتماد برای کاربردهای چندوجهی و تحلیل معنایی تصاویر و متن در پروژه‌های پیشرفته هوش مصنوعی تبدیل کرده است.

۳.۷ استفاده از *OpenCLIP* و بارگذاری وزن‌های ازپیش‌آموزش‌دیده

برای بهره‌گیری از مدل *OpenCLIP*، ابتدا باید وزن‌های ازپیش‌آموزش‌دیده (*pretrained weights*) را از مخزن رسمی *OpenCLIP* در *GitHub* دانلود کنید. این وزن‌ها شامل پارامترهای آموزش‌دیدهٔ مدل هستند و به شما امکان می‌دهند بدون نیاز به آموزش مجدد، مستقیماً از مدل برای استخراج ویژگی‌ها (*feature extraction*) یا محاسبهٔ شباهت بین تصویر و متن استفاده کنید. وزن‌های مدل‌ها از طریق لینک زیر در دسترس هستند:

OpenCLIP GitHub Repository

پس از دانلود وزن‌ها، می‌توانید مدل را با استفاده از قطعه‌کد زیر بارگذاری کنید:

```
import open_clip

clip_model, _, preprocess = open_clip.create_model_and_transforms(
    CLIP_MODEL_NAME, pretrained=PRETRAINED_LOCAL_PATH
)
tokenizer = open_clip.get_tokenizer(CLIP_MODEL_NAME)
clip_model.to(DEVICE)
clip_model.eval()
```

۱.۳.۷ توضیحات کد

- *create_model_and_transforms* مدل را ایجاد می‌کند و توابع پیش‌پردازش مناسب برای تصاویر را برمی‌گرداند.
- *get_tokenizer* توکنایزر متن را بر اساس مدل انتخاب‌شده آماده می‌کند.
- *clip_model.to(DEVICE)* مدل را به *GPU* یا *CPU* منتقل می‌کند.
- *clip_model.eval()* مدل را در حالت ارزیابی (*evaluation mode*) قرار می‌دهد تا آمادهٔ استفاده برای *inference* شود.

پس از این مراحل، مدل آماده است تا ویژگی‌های متنی و تصویری را به یک فضای برداری مشترک تبدیل کند و در کاربردهایی مانند بازیابی چندوجهی (*Multimodal Retrieval*) یا محاسبهٔ شباهت تصویر–متن مورد استفاده قرار گیرد.

۸ مدل‌های تبدیل صوت به متن (*Speech-to-Text / Automatic Speech Recognition*)

مدل‌های *Speech-to-Text (ASR)* یا تبدیل صوت به متن، سیستم‌هایی هستند که صوت گفتاری انسان را به متن قابل پردازش توسط کامپیوتر تبدیل می‌کنند. این مدل‌ها معمولاً از شبکه‌های عصبی پیشرفته مانند *RNN*، *LSTM*، *Transformer* یا معماری‌های ترکیبی استفاده می‌کنند و با تحلیل ویژگی‌های صوتی، محتوا و معنای گفتار را استخراج می‌کنند. هدف اصلی این مدل‌ها، تبدیل سریع و دقیق گفتار به متن برای کاربردهای متنوع در تعامل انسان و ماشین، تحلیل گفتار و پردازش داده‌های صوتی است. مدل‌های تبدیل صوت به متن پایهٔ بسیاری از سیستم‌های هوش مصنوعی تعاملی، چندرسانه‌ای و پردازش گفتار هستند. با ترکیب این مدل‌ها با مدل‌های متن و تصویر، می‌توان سیستم‌های *Multimodal RAG* یا چت‌بات‌های هوشمند چندوجهی ایجاد کرد که توانایی پردازش همزمان متن، تصویر و صوت را دارند.

۱.۸ کاربردهای مدل‌های تبدیل صوت به متن

- دستیارهای صوتی و چت‌بات‌ها: مانند *Siri*، *Alexa* و *Google Assistant* که فرمان‌های صوتی را به متن تبدیل و پردازش می‌کنند.
- زیرنویس خودکار و کپشنینگ: تولید خودکار زیرنویس برای ویدیوها، کلاس‌های آنلاین یا محتوای رسانه‌ای.
- تحلیل و جستجوی صوتی: امکان جستجوی محتوا در میان فایل‌های صوتی و استخراج اطلاعات کلیدی.
- سیستم‌های چندوجهی (*Multimodal*): ترکیب با مدل‌های تصویر و متن در پروژه‌های *RAG* یا *Chatbot* برای درک چندرسانه‌ای.
- تبدیل گفتار به متن در محیط‌های صنعتی یا پزشکی: مستندسازی جلسات، یادداشت‌های پزشکی و گزارش‌های صوتی.

۲.۸ نمونه‌هایی از مدل‌های معروف

۱.۲.۸ *Whisper (OpenAI)*

توضیح یک مدل بزرگ و چندزبانه که قادر است گفتار را به متن تبدیل کند، از جمله ترجمهٔ همزمان به زبان‌های دیگر.

مزایا دقت بالا، پشتیبانی از چند زبان، قابلیت استفاده در پروژه‌های *Multimodal*.

محدودیت‌ها نیاز به منابع سخت‌افزاری قابل توجه برای *inference* در مدل‌های بزرگ.

۲.۲.۸ *wav2vec 2.0 (Facebook AI / Meta)*

توضیح مبتنی بر معماری *Transformer* و آموزش خودنظارتی (*self-supervised*) روی داده‌های صوتی بزرگ.

مزایا نیاز کمتر به داده‌های برچسب‌گذاری‌شده، دقت بالا.

محدودیت‌ها نیاز به آموزش یا *fine-tuning* روی زبان یا لهجهٔ هدف برای عملکرد بهینه.

۳.۲.۸ *DeepSpeech (Mozilla)*

توضیح مدل متن‌باز مبتنی بر *RNN* که گفتار را به متن تبدیل می‌کند.

مزایا متن‌باز، سبک و قابل استفاده در محیط‌های محدود منابع.

محدودیت‌ها دقت پایین‌تر نسبت به مدل‌های *Transformer* بزرگ، عملکرد محدود در محیط‌های نویزی.

۴.۲.۸ *Julius (Julius Speech Recognition Engine)*

توضیح موتور *ASR* سبک و متن‌باز با تمرکز روی پردازش سریع و قابل اجرا روی سخت‌افزار محدود.

مزایا سبک و سریع، مناسب برای کاربردهای تعبیه‌شده (*embedded*).

محدودیت‌ها قابلیت تشخیص لهجه‌ها و زبان‌های متنوع محدود است.

۹ Whisper

Whisper یک مدل پیشرفته و چندزبانه‌ی تبدیل گفتار به متن (*Speech-to-Text / ASR*) است که توسط شرکت *OpenAI* توسعه یافته است. این مدل برای دریافت ورودی‌های صوتی مانند فایل‌های *wav*. یا *mp3*. طراحی شده و خروجی آن متنی شامل محتوای گفتاری موجود در صدا است. *Whisper* با استفاده از شبکه‌های عصبی ترنسفورمر (*Transformer-based architecture*) آموزش دیده و روی مجموعه‌داده‌ای عظیم شامل حدود ۶۸۰ هزار ساعت گفتار چندزبانه آموزش دیده است. هدف این مدل فراهم کردن یک سیستم دقیق، مقیاس‌پذیر و قابل استفاده برای کاربردهای متنوع صوتی است.

۱.۹ ویژگی‌ها و قابلیت‌های اصلی *Whisper*

Whisper قادر است گفتار را به متن در بیش از ۹۰ تا ۱۰۰ زبان زنده دنیا تبدیل کند و قابلیت تشخیص خودکار زبان گفتاری را نیز دارد. این مدل علاوه بر تبدیل گفتار به متن، توانایی ترجمه خودکار گفتار به زبان‌های دیگر، از جمله انگلیسی، را نیز فراهم می‌کند. برخی از ویژگی‌های اصلی *Whisper* عبارت‌اند از:

- تبدیل گفتار به متن (*Speech-to-Text*): تبدیل مستقیم گفتار انسان به متن قابل پردازش.
- تشخیص زبان گفتار (*Language Detection*): شناسایی خودکار زبان ورودی صوتی.
- ترجمه خودکار گفتار (*Speech Translation*): تبدیل گفتار غیرانگلیسی به زبان انگلیسی یا سایر زبان‌ها.
- پشتیبانی چندزبانه: بیش از ۹۰ زبان، از جمله انگلیسی، فارسی، عربی، فرانسوی، آلمانی، اسپانیایی، روسی، چینی و ژاپنی.
- کارکرد آفلاین: نیازی به اینترنت برای پردازش صوت ندارد.
- معماری *Transformer*: امکان پردازش بهینه دنباله‌های طولانی صوتی و برخورداری از دقت بالا در محیط‌های نویزی.
- آموزش گسترده: توانایی پردازش گفتار طبیعی و لهجه‌های مختلف به دلیل آموزش روی مجموعه داده‌های عظیم و متنوع.

۲.۹ نسخه‌ها و مدل‌های *Whisper*

Whisper در نسخه‌های مختلف عرضه شده که هرکدام از نظر سرعت پردازش، دقت و حافظه موردنیاز متفاوت هستند:

جدول ۱: مقایسه نسخه‌های مختلف مدل Whisper

نام مدل	سرعت پردازش	دقت	حافظه موردنیاز	مناسب برای
tiny	بسیار سریع	پایین‌تر	*~75MB*	سیستم‌های سبک یا *real-time*
base	سریع	متوسط	*~142MB*	لپ‌تاپ‌ها یا *inference* سریع
small	نسبتاً سریع	خوب	*~462MB*	*GPU* ضعیف یا *CPU* قوی
medium	کندتر	بالا	*~1.5GB*	*GPU* متوسط (مثلاً *RTX 2060*)
large	سنگین	بسیار بالا	*~2.9GB*	*GPU* قدرتمند (مثل *RTX 3090*)

نکات مهم:

- تمام نسخه‌ها متن‌باز (*open source*) هستند و از طریق پکیج رسمی *whisper* قابل استفاده‌اند.
- برای پردازش سریع روی سیستم‌های ضعیف، مدل‌های *tiny* یا *base* مناسب‌اند.
- برای پروژه‌هایی که دقت بالا اهمیت دارد، مانند پژوهش‌های علمی یا آرشیو صوتی، مدل‌های *medium* یا *large* توصیه می‌شوند.

۳.۹ کاربردهای *Whisper*

- تولید زیرنویس خودکار و کپشنینگ ویدیوها
- استفاده در سیستم‌های چندرسانه‌ای و *Multimodal RAG* برای ترکیب صوت، تصویر و متن
- پیاده‌سازی دستیارهای صوتی و چت‌بات‌های هوشمند
- تحلیل و جستجوی صوتی در فایل‌ها و جلسات ضبط شده
- مستندسازی و ثبت اطلاعات پزشکی یا صنعتی از طریق گفتار

۴.۹ مزایا و محدودیت‌ها

۱.۴.۹ مزایا

- دقت بالا در تبدیل گفتار به متن حتی در محیط‌های نویزی
- پشتیبانی از زبان‌ها و لهجه‌های متنوع
- امکان استفاده مستقیم در پروژه‌های *Multimodal*
- مدل متن‌باز و در دسترس از طریق مخزن رسمی *OpenAI*

۲.۴.۹ محدودیت‌ها

- نیاز به منابع سخت‌افزاری قابل توجه برای *inference* مدل‌های بزرگ
- زمان پردازش طولانی‌تر نسبت به مدل‌های سبک‌تر در فایل‌های صوتی طولانی
- عملکرد در برخی لهجه‌ها یا محیط‌های بسیار نویزی ممکن است کاهش یابد

Whisper به‌عنوان یک ابزار چندمنظوره، متن‌باز و دقیق برای تبدیل گفتار به متن و ترجمه صوتی، پایه‌ای قوی برای توسعه سیستم‌های هوش مصنوعی تعاملی، چندرسانه‌ای و چندزبانه فراهم می‌کند و قابلیت ادغام آسان با سایر مدل‌ها و سیستم‌های برداری مانند *Weaviate* یا *OpenCLIP* را دارد.

۵.۹ نصب و استفاده از *Whisper*

برای نصب و استفاده از مدل *Whisper*، ابتدا باید بسته رسمی آن را از مخزن *GitHub* دانلود و نصب کنید. دستور نصب به صورت زیر است:

```
pip install git+https://github.com/openai/whisper.git
```

پس از نصب، می‌توانید مدل را در کد خود بارگذاری کرده و استفاده کنید:

```
import whisper
whisper_model = whisper.load_model(WHISPER_MODEL, device=DEVICE)
```

در این کد:

- *WHISPER_MODEL* می‌تواند یکی از نسخه‌های مدل باشد: *tiny*، *base*، *small*، *medium* یا *large*
- *DEVICE* مشخص می‌کند مدل روی *GPU (cuda)* یا *CPU* اجرا شود.

این روش امکان استفاده سریع از مدل برای تبدیل صوت به متن، تشخیص زبان گفتار و ترجمه صوتی را فراهم می‌کند.

۱۰ پشتیبانی از زبان فارسی

مدل‌های تولید بردارهای متنی (*Text Embeddings*)، از جمله *CLIP*، *OpenAI Embeddings*، *Sentence-BERT* و سایر مدل‌های مشابه، در مرحله آموزش اولیه خود عمدتاً روی داده‌های متنی به زبان انگلیسی آموزش دیده‌اند. این تمرکز بر زبان انگلیسی به دلیل دسترسی گسترده به داده‌های متنی و منابع آموزشی بزرگ صورت گرفته است. به‌دنبال این موضوع، کاربرد این مدل‌ها در زبان‌های دیگر، به‌ویژه زبان‌هایی با منابع کمتر مانند فارسی، با برخی چالش‌ها مواجه است. محدودیت‌های اصلی عبارتند از:

- کاهش دقت در درک معانی و روابط بین کلمات و جملات غیرانگلیسی
- کاهش کیفیت شباهت‌سنجی متنی در فضای *embedding*
- نیاز به تنظیم مجدد (*fine-tuning*) یا استفاده از مدل‌های چندزبانه برای جبران ضعف‌های زبانی

۱.۱۰ استفاده از ترجمه خودکار پیش از استخراج *Embedding*

یکی از راهکارهای رایج برای بهبود عملکرد مدل‌های *Embedding* محور انگلیسی‌محور روی زبان‌های دیگر، از جمله فارسی، استفاده از ترجمه خودکار متن قبل از استخراج *embedding* است. در این روش، متن غیرانگلیسی ابتدا با یک مدل ترجمه ماشینی مانند *MarianMT* یا سرویس‌های ابری مانند *Google Translate API* به زبان انگلیسی تبدیل می‌شود. سپس، متن ترجمه‌شده به مدل *embedding* داده شده و بردار متنی (*vector representation*) آن استخراج می‌گردد. این روش مزایای مشخصی دارد:

- امکان استفاده مستقیم از مدل‌های پیشرفته و با کیفیت بالا که تنها روی انگلیسی آموزش دیده‌اند، مانند *CLIP* یا *OpenAI Embeddings*.
- کاهش نیاز به آموزش مجدد یا *fine-tuning* مدل برای زبان‌های با منابع محدود.
- پیاده‌سازی ساده و سریع، بدون نیاز به تغییر ساختار مدل اصلی یا استفاده از مدل‌های چندزبانه پیچیده.

با این حال، محدودیت‌هایی نیز وجود دارد:

- کیفیت *embedding* به دقت ترجمه بستگی دارد. هرگونه خطا یا ابهام در ترجمه می‌تواند بر درک معنایی متن و دقت شباهت‌سنجی *embeddings* تأثیر منفی بگذارد.
- ترجمه خودکار ممکن است برخی ظرایف زبانی، اصطلاحات محلی یا مفاهیم خاص فرهنگی را به درستی منتقل نکند، که در نتیجه *embedding* نهایی ممکن است دقیق نباشد.

به‌طور کلی، این روش یک راه‌حل عملی و سریع برای بهره‌گیری از مدل‌های انگلیسی‌محور روی متون فارسی است و در پروژه‌هایی که کیفیت *embedding* انگلیسی مدل بسیار بالا است، می‌تواند به طور مؤثری عملکرد سیستم را بهبود دهد.

۲.۱۰ روش‌های ترجمه متن در پایتون قبل از استخراج *Embedding*

برای بهبود عملکرد مدل‌های انگلیسی‌محور روی متون غیرانگلیسی مانند فارسی، می‌توان از روش‌های مختلف ترجمه استفاده کرد. این روش‌ها شامل کتابخانه‌های سبک آفلاین، مدل‌های پیش‌آموزش‌دیده‌شده و مدل‌های بزرگ زبان (*LLM*) هستند.

۱.۲.۱۰ استفاده از کتابخانه‌های ترجمه آفلاین

کتابخانه‌های متداول این دسته عبارتند از *argos-translate* و *translate*.

مزایا
- قابلیت اجرا به‌صورت آفلاین و بدون نیاز به اینترنت
- رایگان و سبک، مناسب برای سیستم‌های محلی و پروژه‌های کوچک

معایب و محدودیت‌ها
- دقت نسبتاً پایین، به‌ویژه برای جملات پیچیده یا محاوره‌ای
- احتمال عدم ترجمه صحیح برخی اصطلاحات یا کلمات خاص زبان فارسی

نمونه کد با *argos-translate*:

```
import argostranslate.package
import argostranslate.translate

#
argostranslate.package.install_from_path('en_fa.argosmodel')

from argostranslate.translate import translate
translated_text = translate )"       ", "fa", "en")
print(translated_text)
```

۲.۲.۱۰ استفاده از مدل‌های *Translator* پیش‌آموزش‌دیده‌شده

مدل‌هایی مانند *Helsinki-NLP/opus-mt-fa-en* و *Helsinki-NLP/opus-mt-en-fa* از طریق *Hugging Face* در دسترس هستند.

مزایا
- دقت بالاتر نسبت به کتابخانه‌های سبک
- امکان استفاده آفلاین پس از دانلود مدل

محدودیت‌ها
- حجم مدل‌ها معمولاً بیشتر است و نیازمند منابع سخت‌افزاری مناسب (*GPU* یا *CPU* قوی) برای پردازش سریع حجم بالای داده‌ها

نمونه کد ساده با *Hugging Face*:

```
from transformers import MarianMTModel, MarianTokenizer

model_name = "Helsinki-NLP/opus-mt-fa-en"
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

text =  "       "
batch = tokenizer([text], return_tensors="pt", padding=True)
translated = model.generate(**batch)
translated_text = tokenizer.decode(translated[0], skip_special_tokens=True)
print(translated_text)
```

۳.۲.۱۰ استفاده از مدل‌های بزرگ زبان (*LLM*)

مدل‌های آزاد و رایگانی مانند *openai/gpt-oss-20b:free* نیز می‌توانند برای ترجمه متون فارسی به انگلیسی استفاده شوند.

مزایا
- دقت بسیار بالا و نزدیک به ترجمه انسانی
- توانایی درک جملات پیچیده، اصطلاحات محاوره‌ای و متون چندجمله‌ای
- امکان انجام ترجمه، خلاصه‌سازی و پردازش پیشرفته در یک مرحله

محدودیت‌ها
- محدودیت تعداد توکن‌ها و حجم داده‌ها
- نیاز به اتصال اینترنت

به‌طور خلاصه، انتخاب روش ترجمه بستگی به دقت مورد نیاز، منابع سخت‌افزاری و حجم داده‌ها دارد. روش‌های سبک آفلاین مناسب پروژه‌های کوچک و *real-time* هستند، مدل‌های پیش‌آموزش‌دیده‌شده تعادلی بین دقت و منابع ارائه می‌کنند و مدل‌های *LLM* بالاترین دقت و قابلیت‌های پردازشی پیشرفته را دارند.

۱۱ راه اندازی سیستم

برای آماده‌سازی محیط و ایجاد یک *Collection* در پایگاه داده برداری *Weaviate*، مراحل زیر به صورت رسمی و گام‌به‌گام انجام می‌شوند:

ابتدا داکر را نصب کرده و با اجرای فایل *docker-compose.yml* پایگاه داده *Weaviate* محلی راه‌اندازی می‌شود. این مرحله باعث ایجاد یک سرویس برداری محلی می‌شود که برای ذخیره و بازیابی *embedding*های متنی، تصویری و صوتی آماده است. سپس یک محیط مجازی (*Virtual Environment*) ایجاد کرده و تمام کتابخانه‌های مورد نیاز را با استفاده از فایل *requirements.txt* نصب کنید تا وابستگی‌های پروژه فراهم گردد. مرحله بعد اجرای فایل *create_database_collection.py* است که وظیفه ساخت *Collection* را بر عهده دارد. عملکرد این فایل به شرح زیر است:

- اتصال به *Weaviate* محلی برقرار می‌شود.
- بررسی می‌شود که آیا کالکشنی با نام "*Multimodal_Collection*" قبلاً وجود دارد یا خیر. در صورت وجود، کالکشن قدیمی حذف می‌گردد.
- یک *Collection* جدید با همان نام ایجاد می‌شود و شامل چهار ویژگی اصلی است:
  - *modality*: نوع داده، شامل متن (*text*)، تصویر (*image*) یا صوت (*audio*)
  - *content*: برای داده‌های متنی، خود متن؛ برای داده‌های صوتی، متن تبدیل شده (*transcription*)؛ برای تصاویر، رشته خالی ""
  - *contentId*: شناسه یکتا برای هر محتوا
  - *filePath*: مسیر نسبی فایل روی سیستم

در پایان، اتصال به *Weaviate* به‌صورت ایمن بسته می‌شود و محیط برای ورود داده‌های *Multimodal* آماده خواهد بود. این فرآیند تضمین می‌کند که یک *Collection* منظم و قابل استفاده برای ذخیره *embedding*ها و پردازش‌های بعدی در پروژه‌های *Multimodal RAG* آماده گردد.

۱۲ وارد کردن دیتا به دیتابیس

۱.۱۲ روند پردازش داده‌ها در پروژه

در این پروژه، فرایند پردازش داده‌ها به شکل زیر سازمان‌دهی شده است:
ابتدا برای تمام فایل‌های صوتی، از مدل *Whisper* استفاده می‌شود تا متن متناظر (*transcription*) هر فایل صوتی استخراج گردد. به این ترتیب، داده‌های صوتی به داده‌های متنی قابل پردازش تبدیل می‌شوند و یکپارچگی با سایر داده‌های متنی پروژه برقرار می‌گردد. پس از این مرحله، دو نوع داده اصلی در اختیار داریم:

- متن (*Text*)
- تصویر (*Image*)

سپس با استفاده از مدل *CLIP*، هر دو نوع داده متنی و تصویری به بردارهای عددی (*embeddings*) تبدیل می‌شوند. این بردارها نمایانگر معنای محتوای داده‌ها هستند و امکان اندازه‌گیری شباهت معنایی بین متن‌ها و تصاویر را فراهم می‌کنند. در نهایت، این *embeddings* همراه با اطلاعات مربوط به هر داده در پایگاه داده برداری *Weaviate* ذخیره می‌شوند. این کار امکان استفاده از داده‌ها در مراحل بعدی پروژه برای جست‌وجو، بازیابی و پاسخ‌گویی چندوجهی (*Multimodal Retrieval and Generation*) را فراهم می‌سازد. این فرایند، پایه‌ای قوی برای سیستم‌های چندرسانه‌ای و کاربردهای *RAG* فراهم می‌کند و تضمین می‌کند که داده‌ها به شکل منظم و استاندارد برای پردازش‌های معنایی آماده باشند.

۲.۱۲ روش ترجمه متن‌ها و فایل‌های صوتی در پروژه

در این پروژه، پیش از ذخیره‌سازی داده‌ها در پایگاه داده و انجام مراحل *retrieval*، متون فارسی با استفاده از مدل *LLM* رایگان *openai/gpt-oss-20b:free* ترجمه می‌شوند. دلیل انتخاب این مدل، دقت بالای ترجمه و همچنین رایگان بودن آن است که امکان پردازش حجم بالای داده‌ها را فراهم می‌کند. برای فایل‌های صوتی نیز روند مشابهی اعمال می‌شود. اگر فایل صوتی به زبان انگلیسی باشد، متن استخراج‌شده همان‌طور در دسترس قرار می‌گیرد. در غیر این صورت، ابتدا با استفاده از مدل *Whisper* متن اصلی استخراج شده و سپس در صورت نیاز به انگلیسی ترجمه می‌شود. نمونه کد استفاده‌شده در این پروژه به صورت زیر است:

```
def transcribe_to_english(path):
    """Transcribe audio to English using Whisper."""
    res = whisper_model.transcribe(path, task="transcribe")
    lang, text = res.get("language", ""), res.get("text", "").strip()

    #
    if lang.startswith("en"):
        return text

    #                    Whisper
    return (
        whisper_model.transcribe(path, task="translate", language="en")

       .get("text", "")
        .strip()
    )
```

در این روش:

- ابتدا *Transcription* انجام می‌شود تا متن اصلی فایل صوتی شناسایی گردد.
- اگر زبان فایل انگلیسی باشد، متن استخراج‌شده بازگردانده می‌شود.
- در صورت غیرانگلیسی بودن فایل، از قابلیت *Translate* مدل *Whisper* برای تبدیل متن به انگلیسی استفاده می‌شود.

به این ترتیب، تمامی فایل‌های صوتی به متن انگلیسی تبدیل شده و آماده پردازش با مدل‌هایی مانند *CLIP* و ذخیره در پایگاه داده برداری می‌شوند. همچنین، در مرحله‌ی پردازش کوئری‌های کاربران، ممکن است متن ورودی یا فایل صوتی به زبان فارسی باشد. برای امکان جست‌وجوی دقیق در دیتابیس برداری که مبتنی بر متن انگلیسی است، ابتدا کوئری‌ها به انگلیسی ترجمه شده و سپس مورد پردازش و بازیابی اطلاعات قرار می‌گیرند.

۳.۱۲ راهنمای رسمی آماده‌سازی و وارد کردن داده‌ها به پایگاه داده برداری (Weaviate)

برای آماده‌سازی داده‌ها و وارد کردن آن‌ها به پایگاه داده برداری در این پروژه، مراحل زیر به‌صورت رسمی و مرحله‌ای انجام می‌شوند:

۱.۳.۱۲ ساختار پوشه‌ها و قرار دادن داده‌ها

تمام داده‌های پروژه باید در مسیرهای مشخص زیر قرار گیرند:

- **تصاویر:**
  PROJECT_ROOT/content/image_dataset
- **فایل‌های صوتی:**
  PROJECT_ROOT/content/audio_dataset
- **متون:** ابتدا یک فایل *CSV* با نام *text_data.csv* ایجاد کرده و دو ستون داشته باشد:
  - *id* برای شناسه‌ی متن
  - *text* برای محتوای متن

  سپس این فایل در مسیر زیر قرار داده شود:
  PROJECT_ROOT/content/text_dataset

۲.۳.۱۲ مدل OpenCLIP و وزن‌های آن

در این پروژه برای تولید *embedding* از مدل *OpenCLIP* استفاده شده است. وزن‌های مدل باید پیش از اجرای اسکریپت‌ها دانلود و در مسیر زیر قرار گیرند:
PROJECT_ROOT/open_clip_weights/ViT-B-32-openai/open_clip_model.safetensors

- مدل مورد استفاده در این پروژه: *ViT-B-32*
- در صورت داشتن منابع سخت‌افزاری مناسب (*RAM* بالای ۸ گیگ و *GPU*)، امکان استفاده از مدل‌های قوی‌تر نیز وجود دارد.

همچنین لازم است متغیرهای زیر در فایل پیکربندی *config.py* تنظیم شوند:

```
OPENROUTER_API_KEY = os.getenv("LLM_API_KEY")
OPENROUTER_API_BASE = "https://openrouter.ai/api/v1"
LLM_MODEL_NAME = "openai/gpt-oss-20b:free"
```

- *API Key* در فایل *env*. با کلید *LLM_API_KEY* ذخیره می‌شود.
- مدل رایگان *openai/gpt-oss-20b:free* برای ترجمه متون فارسی به انگلیسی استفاده شده است.
- در صورت نیاز به دقت بالاتر، می‌توان از مدل‌های قوی‌تر مانند *Gemini 2.5* یا *ChatGPT-4* استفاده کرد، البته با توجه به حجم بالای داده‌ها باید مصرف توکن‌ها کنترل شود.

۳.۳.۱۲ وارد کردن داده‌ها به پایگاه داده

دو روش برای وارد کردن داده‌ها وجود دارد:

۱. **Import محلی:** داده‌های محلی روی سیستم شما به *embedding* تبدیل و مستقیماً در پایگاه داده *Weaviate* ذخیره می‌شوند.
مسیر اسکریپت‌ها:
PROJECT_ROOT/import_data_into_database/import_local_data

۲. **استفاده از سیستم دیگر / :Cloud** داده‌ها در محیط ابری (مثلاً *Google Colab*) به *embedding* تبدیل می‌شوند و ابتدا در پایگاه داده ابری ذخیره می‌شوند. سپس داده‌ها از آنجا *export* و به صورت *JSON* روی سیستم لوکال ذخیره شده و با اسکریپت وارد پایگاه داده محلی می‌شوند.
مسیر اسکریپت‌ها:
PROJECT_ROOT/import_data_into_database/
export_data_from_weaviate_cloud_into_json_file_and_import_it_into_local_database

این روش برای سیستم‌های ضعیف و فاقد *GPU* توصیه می‌شود.

۴.۳.۱۲ روش اول – Import داده‌های محلی

ایمپورت داده‌های متنی:

- فایل *CSV* متون (فارسی یا انگلیسی) را در مسیر *PROJECT_ROOT/content/text_dataset* قرار دهید.
- نام فایل را در اسکریپت *import_text_dataset_into_database.py* تنظیم کرده و اجرا کنید.
- این اسکریپت مراحل زیر را انجام می‌دهد:
  - خواندن داده‌ها از *CSV*
  - ترجمه متون غیرانگلیسی با استفاده از مدل *LLM* مشخص‌شده در *config.py*
  - تولید *embedding* هر متن با مدل *OpenCLIP*
  - ذخیره *embedding*ها در پایگاه داده *Weaviate*

ایمپورت داده‌های تصویری:

- تصاویر را در مسیر *PROJECT_ROOT/content/image_dataset* قرار دهید.
- اجرای اسکریپت: *import_image_dataset_into_database.py*
- این اسکریپت برای هر تصویر *embedding* تولید کرده و در دیتابیس ذخیره می‌کند.

ایمپورت داده‌های صوتی:

- فایل‌های صوتی را در مسیر *PROJECT_ROOT/content/audio_dataset* قرار دهید.
- اجرای اسکریپت: *import_audio_dataset_into_database.py*
- مراحل اجرا:
  - تشخیص زبان فایل صوتی با مدل *Whisper*
  - ترجمه به انگلیسی در صورت نیاز
  - ذخیره اطلاعات فایل و متن استخراج‌شده در *JSON*
  - تولید *embedding* متن صوتی با *OpenCLIP* و ذخیره در پایگاه داده

۵.۳.۱۲ روش دوم – استفاده از سیستم دیگر / Cloud

- داده‌ها را در مسیرهای مشخص شده در *Colab* یا سیستم ابری قرار دهید.
- تمام سلول‌های *Jupyter Notebook* در مسیر زیر به ترتیب اجرا شوند:
  PROJECT_ROOT/import_data_into_database/
  export_data_from_weaviate_cloud_into_json_file_and_import_it_into_local_database
- پس از پایان اجرا، فایل *JSON* یا *CSV* حاصل دانلود و در مسیر مشخص قرار داده شود.
- نام فایل در اسکریپت *import_data_from_json_into_local_weaviate.py* تنظیم شده و اسکریپت اجرا می‌شود.
- با این کار تمام داده‌ها به پایگاه داده محلی منتقل می‌شوند.
- دقت شود که داده‌ها باید در همان پوشه‌های اولیه وجود داشته باشند تا امکان بازیابی (*retrieve*) صحیح فراهم باشد.

این فرآیند تضمین می‌کند که داده‌های متنی، تصویری و صوتی به‌صورت منظم وارد پایگاه داده برداری شوند و برای پردازش‌های *Multimodal RAG* آماده باشند.

۱۳ پردازش کوئری‌های کاربر و بازیابی داده‌ها از سیستم

کاربران می‌توانند هر ترکیبی از داده‌های متن، تصویر یا صوت را به عنوان کوئری به سیستم وارد کنند. روند پردازش کوئری به صورت یکپارچه و رسمی به شرح زیر است:

۱.۱۳ پردازش اولیه کوئری

۱.۱.۱۳ کوئری متنی

در صورتی که متن ورودی غیرانگلیسی باشد، ابتدا با استفاده از تابع *normalize_text_to_english* (یا مشابه آن) به انگلیسی ترجمه می‌شود. برای این کار در این پروژه از کتابخانه‌ی *deep_translator* استفاده شده است:

```python
# server/language_utils.py
from deep_translator import GoogleTranslator

def normalize_text_to_english(text: str) -> str:
    """
    Convert any input text into English if it is not already English.
    Works offline for language detection + uses Google Translate API.
    """
    if not text or text.strip() == "":
        return text

    try:
        translated = GoogleTranslator(source='auto', target='en').translate(text)
        return translated

  except Exception as e:
        print(f"Translation error: {e}")
        return text
```

کتابخانه *deep_translator* یک ابزار سبک و سریع برای ترجمه متن است که با منابعی مانند *Google Translate*، *Deepl* و *Pons* کار می‌کند. مزیت اصلی آن سرعت بالا و مناسب بودن برای کاربردهای *real-time* است، اما دقت آن نسبت به مدل‌های *LLM* کمتر است.

۲.۱.۱۳ کوئری صوتی

اگر ورودی کاربر یک فایل صوتی باشد، ابتدا متن آن با استفاده از مدل *Whisper* استخراج می‌شود. سپس در صورت غیرانگلیسی بودن، متن به انگلیسی ترجمه می‌شود. این کار تضمین می‌کند که تمامی کوئری‌ها، چه متنی و چه صوتی، قبل از تبدیل به *embedding*، به زبان انگلیسی آماده پردازش شوند.

۲.۱۳ تبدیل کوئری به *embedding*

متن نهایی (انگلیسی) یا داده‌های تصویری با استفاده از مدل‌های *CLIP* یا *embedding text model* به بردارهای عددی (*embeddings*) تبدیل می‌شوند. این بردارها نمایانگر معنای محتوای داده هستند و امکان مقایسه معنایی بین داده‌ها را فراهم می‌کنند.

۳.۱۳ بازیابی داده‌های مشابه از پایگاه داده

- *embedding* کوئری با *embedding*های ذخیره‌شده در *Weaviate* مقایسه می‌شود.
- با استفاده از متد *collection.query.near_vector*، *k* مورد نزدیک‌ترین *embedding* انتخاب و داده‌های مرتبط به کاربر نمایش داده می‌شوند.
- در صورتی که کاربر بیش از یک ورودی داشته باشد، ابتدا *embedding* هر ورودی جداگانه محاسبه می‌شود و سپس با میانگین‌گیری (*average pooling*) ترکیب می‌شوند تا یک *embedding* واحد نماینده کل کوئری ایجاد گردد. این *embedding* واحد برای جست‌وجو در دیتابیس استفاده می‌شود.

۴.۱۳ پشتیبانی از کوئری‌های چندمدالیته و چندگانه

این مکانیزم باعث می‌شود سیستم بتواند:

- کوئری‌های متنی و صوتی چندگانه
- کوئری‌های ترکیبی متن، تصویر و صوت

را به شکل مؤثر پردازش کرده و نزدیک‌ترین نتایج معنایی را به کاربر ارائه دهد.

۵.۱۳ جمع‌بندی

با استفاده از روش‌های فوق، سیستم اطمینان حاصل می‌کند که تمام کوئری‌ها به انگلیسی تبدیل شده، *embedding* آن‌ها تولید شده و با پایگاه داده برداری مقایسه می‌شوند تا نتایج *retrieval* سریع و دقیق ارائه گردد. این رویکرد امکان پردازش *real-time* برای کاربران فارسی‌زبان و چندمدالیته را فراهم می‌آورد و هماهنگی کامل بین متن، تصویر و صوت را تضمین می‌کند.

۱۴ پردازش داده‌های بازیابی شده توسط *LLM*

پس از مرحله‌ی *Multimodal Retrieval*، داده‌های بازیابی شده برای تحلیل و تولید پاسخ توسط یک *LLM* چندوجهی آماده می‌شوند. در این پروژه، از مدل *openai/gpt-5-image-mini* استفاده شده و ارتباط با آن از طریق *OpenRouter API* برقرار می‌گردد:

```
client = OpenAI(
    base_url=LLM_API_BASE,  # "https://openrouter.ai/api/v1"
    api_key=LLM_API_KEY
)
```

۱.۱۴ داده‌های ارسال‌شده به *LLM*

داده‌هایی که به *LLM* فرستاده می‌شوند شامل موارد زیر هستند:

۱. **ورودی‌های کاربر**

- متن‌های کوئری
- تصاویر ارسال‌شده
- فایل‌های صوتی (فقط متن استخراج‌شده از آنها، بدون ارسال فایل خام)

۲. **موارد بازیابی‌شده** (*retrieved*)

- یک نمونه متن *retrieved*
- یک تصویر *retrieved*
- یک فایل صوتی *retrieved* به صورت متن *transcribe* شده

نکته: فایل‌های صوتی خام مستقیماً به مدل ارسال نمی‌شوند و تنها متن استخراج شده از آن‌ها پردازش می‌شود.

۲.۱۴ روند پردازش داخل تابع *feed_data_into_llm*

۱. داده‌های ورودی و *retrieved* به بخش‌های جداگانه (*sections*) تفکیک می‌شوند: متن، تصویر، صوت.

۲. تصاویر به ابعاد کوچک (*128x128*) تغییر اندازه داده شده و به *Base64* تبدیل می‌شوند تا امکان ارسال آن‌ها به *LLM* فراهم گردد.

۳. متن‌های صوت و *retrieved* نیز در قالب *Markdown* آماده‌سازی می‌شوند.

۴. تمام بخش‌ها با هم ترکیب شده و یک *context* کامل ایجاد می‌شود.

۵. این *context* به مدل *LLM* ارسال می‌گردد:

```
completion = client.chat.completions.create(
    model=LLM_MODEL_NAME,  # e.g., "openai/gpt-5-image-mini"
    messages=[
        {"role": "system", "content": "You are a multimodal reasoning
    assistant..."},
        {"role": "user", "content": content_list}
    ]
)
```

۶. پاسخ تولیدشده توسط *LLM* در متغیر *response_text* ذخیره می‌شود.

۳.۱۴ نکات کلیدی

- مدل *LLM* قادر است داده‌های چندوجهی (متن، تصویر، صوت) را تحلیل کرده و پاسخ یا توصیف جامع و دقیق تولید کند.
- تمامی داده‌های ارسالی و *context* مورد استفاده توسط مدل لاگ می‌شوند تا امکان بررسی و بازبینی فراهم گردد.
- این روش تضمین می‌کند که کاربر یک پاسخ کامل و متناسب با تمام داده‌ها دریافت کند، حتی اگر کوئری و داده‌های بازیابی شامل چندین نوع مودالیتی باشد.

این ساختار پردازش، امکان تحلیل دقیق و تولید پاسخ‌های چندوجهی و هماهنگ با داده‌های واقعی را فراهم می‌آورد.

۱۵ نحوه اجرای پروژه

در این سیستم، سرور با استفاده از *FastAPI* پیاده‌سازی شده و کلاینت (فرانت‌اند) با استفاده از *HTML*، *CSS* و *Vanilla JavaScript* پیاده‌سازی شده است. برای اجرای پروژه، پس از تکمیل مراحل اولیه راه‌اندازی، مراحل زیر را دنبال کنید:

۱. **فعال‌سازی محیط مجازی:** محیط مجازی (*virtual environment*) خود را فعال کنید.

۲. **ورود به مسیر پروژه:** در ترمینال به مسیر ریشه پروژه (*PROJECT_ROOT*) بروید.

۳. **اجرای سرور:** دستور زیر را اجرا کنید:

```
python -m server.main
```

۴. **دسترسی به رابط کاربری:** یک مرورگر باز کرده و به آدرس زیر مراجعه کنید:

```
http://localhost:8000
```

با انجام این مراحل، سرور راه‌اندازی شده و رابط کاربری پروژه در مرورگر قابل دسترسی خواهد بود.