دانشگاه فردوسی مشهد

دانشکده علوم ریاضی

MultiModal Agentic RAG

درس: پروژه کارشناسی
استاد درس: دکتر جلال‌الدّین نصیری

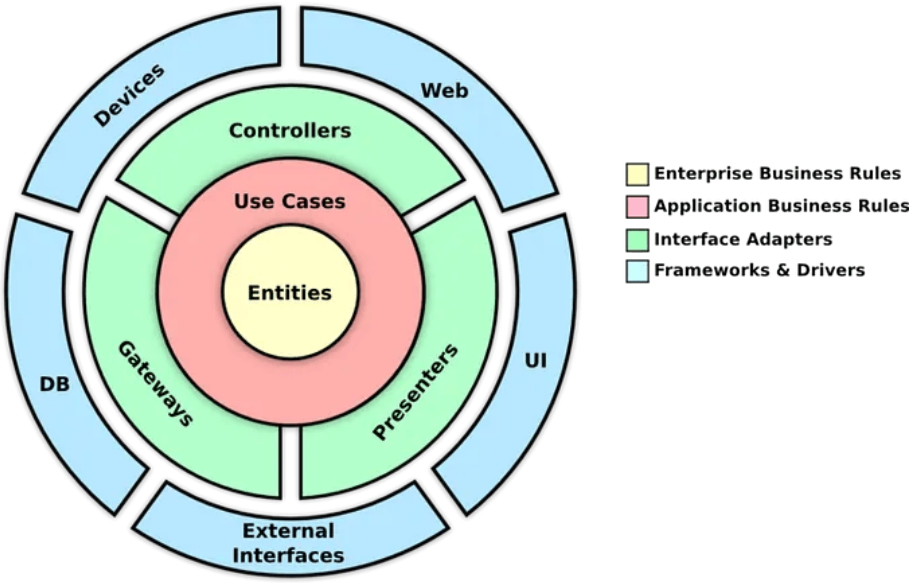
دانشجو: علی محمودجانلو

سال ۱۴۰۴

فهرست مطالب

۱ معماری سیستم

این پروژه بر اساس الگوی معماری Clean Architecture پیاده‌سازی شده است که امکان جداسازی مناسب نگرانی‌ها، مقیاس‌پذیری و نگهداری‌پذیری بالا را فراهم می‌آورد. انتخاب این معماری برای یک سیستم مبتنی بر مدل‌های زبانی بزرگ (LLM) که نیازمند انعطاف‌پذیری بالا و قابلیت تست‌پذیری است، از اهمیت ویژه‌ای برخوردار است.



شکل ۱: نمودار معماری Clean Architecture و لایه‌های آن

شکل ۱ نمودار کلی معماری Clean را نشان می‌دهد که در آن جهت وابستگی‌ها از بیرون به درون است و لایه‌های مرکزی از جزئیات پیاده‌سازی مستقل هستند.

۱.۱ دلایل انتخاب معماری Clean

انتخاب معماری Clean برای این پروژه بر اساس چندین معیار کلیدی صورت گرفته است:

- **قابلیت تست‌پذیری:** با جداسازی لایه‌های مختلف و استفاده از تزریق وابستگی (Dependency Injection)، امکان نوشتن تست‌های واحد و یکپارچه با پوشش بالا میسر می‌شود.
- **انعطاف‌پذیری در تعویض ارائه‌دهندگان LLM:** سیستم طراحی شده به گونه‌ای است که تغییر یا افزودن ارائه‌دهندگان مختلف LLM (مانند OpenAI، Cohere، یا مدل‌های متن‌باز) بدون تغییر در منطق اصلی امکان‌پذیر است.
- **مقیاس‌پذیری:** ساختار لایه‌بندی شده امکان توسعه و گسترش سیستم را در آینده تسهیل می‌کند.

۲.۱ لایه‌های معماری

معماری این سیستم شامل چهار لایه اصلی است که هر کدام مسئولیت‌های مشخصی دارند و قانون وابستگی (Dependency Rule) در آن‌ها رعایت می‌شود.

۱.۲.۱ لایه موجودیت‌ها (Entities)

این لایه، قلب سیستم را تشکیل می‌دهد و شامل enterprise business rules است که مستقل از هرگونه جزئیات پیاده‌سازی خارجی هستند.
مسئولیت‌های این لایه:

- تعریف core data models با استفاده از Pydantic
- Data validation و اعمال business rules
- تضمین type-safety و automatic serialization داده‌ها

به عنوان مثال، موجودیت `DocChunk` در سیستم ما شامل قوانین کسب‌وکار زیر است:

- **تولید شناسه یکتا:** هر chunk باید دارای شناسه‌ای یکتا باشد که بر اساس `document_id` و index تولید می‌شود: `{document_id}_chunk_{index}`
- **اعتبارسنجی بردار embedding:** اگر بردار embedding موجود باشد، باید از نوع `List[float]` و غیر خالی باشد
- **حفظ metadata اصلی:** تمام اطلاعات meta شامل schema_name، version و doc_items باید حفظ شوند
- **تبدیل فرمت:** موجودیت باید قابلیت تبدیل دوطرفه به فرمت Elasticsearch و Docling را داشته باشد

این قوانین مستقل از جزئیات پیاده‌سازی database یا search engine تعریف شده‌اند و در صورت تغییر فناوری زیرساخت، دست‌نخورده باقی می‌مانند.

۲.۲.۱ لایه (Use Cases)

این لایه application business logic را در بر می‌گیرد و workflow بین entities و interface adapters را هماهنگ می‌کند.
مسئولیت‌های کلیدی:

- مدیریت جریان‌های کاری پیچیده با استفاده از LangGraph
- پیاده‌سازی RAG (Retrieval-Augmented Generation) pipelines
- تولید و مدیریت embeddings
- هماهنگی با سرویس‌های خارجی از طریق interfaces

۳.۲.۱ لایه آداپتورهای رابط (Interface Adapters)

این لایه وظیفه تبدیل داده‌ها بین موارد استفاده و دنیای خارج را بر عهده دارد.
اجزای این لایه شامل:

- **Controllers:** مدیریت HTTP requests و responses
- **Presenters:** قالب‌بندی داده‌ها برای رابط کاربری یا مصرف‌کنندگان API
- **Gateways:** رابط با سرویس‌های خارجی مانند databases، vector stores و third-party APIs

۴.۲.۱ لایه فریم‌ورک‌ها و درایورها (Frameworks and Drivers)

بیرونی‌ترین لایه سیستم که شامل جزئیات پیاده‌سازی ابزارها و فریم‌ورک‌های خارجی است.
مثال‌های موجود در این لایه:

- Web frameworks (FastAPI، Flask)
- LLM providers (OpenAI، Cohere، open-source models)
- Databases و vector stores (Elasticsearch، Redis، Pinecone)

۳.۱ اصول طراحی

۱.۳.۱ قانون وابستگی (Dependency Rule)

در این معماری، وابستگی‌ها همواره از لایه‌های بیرونی به سمت لایه‌های درونی هستند. این بدان معناست که:

$$\text{Frameworks/Drivers} \rightarrow \text{Interface Adapters} \rightarrow \text{Use Cases} \rightarrow \text{Entities} \quad (۱)$$

لایه‌های درونی هیچ‌گونه اطلاعی از لایه‌های بیرونی ندارند و این استقلال امکان تغییر پیاده‌سازی‌های خارجی را بدون تأثیر بر هسته سیستم فراهم می‌آورد.

۲.۳.۱ اصل وارونگی وابستگی (Dependency Inversion Principle)

برای حفظ قانون وابستگی، از interfaces و کلاس‌های انتزاعی استفاده می‌شود. به این ترتیب:

- Interfaces در لایه‌های هسته (Use Cases) تعریف می‌شوند
- پیاده‌سازی‌های واقعی در لایه‌های بیرونی (Interface Adapters یا Frameworks) قرار می‌گیرند
- لایه‌های سطح بالا به انتزاع وابسته‌اند، نه به پیاده‌سازی‌های خاص

مثال کاربردی: در سیستم ما، یuse caseای به نام agentic_rag ماژولی از نوع EmbeddingServiceInterface دریافت می‌کند. این use case توابعی مانند embed_single را فراخوانی می‌کند، بدون آنکه بداند پیاده‌سازی واقعی از چه ارائه‌دهنده‌ای استفاده می‌کند (مانند OpenAI، Google GenAI، Azure، یا مدل‌های متن‌باز). این امر به این دلیل امکان‌پذیر است که:

- Use case تنها به contract تعریف شده در interface وابسته است
- تعویض ارائه‌دهنده embedding بدون تغییر در منطق کسب‌وکار به راحتی امکان‌پذیر است

۴.۱ سیستم Dependency Injection

برای مدیریت وابستگی‌ها و پیکربندی سیستم، از کتابخانه `dependency-injector` استفاده شده است. این سیستم مزایای زیر را فراهم می‌آورد:

۱.۴.۱ مزایای Dependency Injection

- **Centralized configuration:** تمام تنظیمات و وابستگی‌ها در یک container مرکزی مدیریت می‌شوند
- **تسهیل تست:** امکان جایگزینی وابستگی‌های واقعی با mocks برای تست‌نویسی
- **مدیریت singleton:** کنترل lifecycle منابع گران‌قیمت مانند database connections
- **انعطاف‌پذیری:** تغییر configuration در runtime بدون تغییر کد

انواع providers در کتابخانه dependency-injector:

- **Factory:** برای ایجاد نمونه جدید در هر بار استفاده. این نوع provider هر بار که درخواست می‌شود، یک instance جدید از کلاس مورد نظر ایجاد می‌کند. برای اشیائی مناسب است که stateless هستند یا هر بار نیاز به تنظیمات تازه دارند.
- **Singleton:** برای نگهداری یک نمونه واحد در طول عمر برنامه. این provider فقط یک بار شیء را می‌سازد و در تمام درخواست‌های بعدی همان نمونه را بازمی‌گرداند. برای منابع گران‌قیمت مانند اتصالات database، HTTP clients یا configuration objects مفید است.
- **Resource:** برای مدیریت چرخه حیات اشیاء پیچیده که نیاز به راه‌اندازی و پاکسازی دارند. این provider قابلیت‌های lifecycle management مانند initialization، cleanup و مدیریت منابع سیستمی را فراهم می‌آورد. برای سرویس‌هایی که نیاز به graceful shutdown دارند، ایده‌آل است.
- **Configuration:** برای مدیریت و تزریق مقادیر پیکربندی مانند environment variables، تنظیمات API، یا پارامترهای سیستم. این provider امکان خواندن مقادیر از فایل‌های config، متغیرهای محیطی یا منابع خارجی را فراهم می‌آورد و آن‌ها را در سراسر سیستم در دسترس قرار می‌دهد.

۵.۱ یکپارچگی با فریم‌ورک‌های LLM

یکی از چالش‌های اصلی این پروژه، یکپارچگی با فریم‌ورک‌های مختلف LLM بوده است. معماری طراحی شده این امکان را فراهم می‌آورد که:

- از چندین ارائه‌دهنده LLM به صورت همزمان استفاده شود
- جریان‌های کاری پیچیده با LangGraph مدیریت شوند
- عملیات RAG به صورت ماژولار پیاده‌سازی شوند
- تغییر یا ارتقای فریم‌ورک‌ها بدون تغییر منطق اصلی امکان‌پذیر باشد

۶.۱ استفاده از Pydantic

در لایه موجودیت‌ها، از کتابخانه Pydantic برای تعریف data models استفاده شده است. این انتخاب به دلایل زیر صورت گرفته است:

- **Automatic validation:** داده‌ها به صورت خودکار در زمان ایجاد object اعتبارسنجی می‌شوند

- **Type Safety:** تضمین صحت data types در زمان توسعه
- **Serialization:** تبدیل خودکار به JSON و سایر فرمت ها
- **Automatic documentation:** تولید schema برای APIها

۷.۱ مدیریت Error و Logging

سیستم دارای یک لایه یکپارچه برای error handling و logging است که:

- از multi-level logging پشتیبانی می‌کند
- خطاها را به صورت ساختاریافته ثبت می‌کند
- امکان request tracing را فراهم می‌آورد
- با سیستم‌های monitoring خارجی قابل یکپارچگی است

به عنوان مثال، میتوان از Prometheus برای جمع‌آوری متریک ها و از Grafana برای visualization استفاده کرد تا به اهداف زیر رسید:

- **Metrics Collection:** Response times، تعداد درخواست‌ها، نرخ خطا، و استفاده از منابع
- **Health Checks:** بررسی وضعیت سرویس‌های LLM، database connections، و vector stores
- **Alerting:** اطلاع‌رسانی خودکار در صورت بروز مشکل از طریق Slack یا Email

سیستم logging پیاده‌سازی شده در این پروژه، از قابلیت automatic rotation و مدیریت log levels مختلف برخوردار است. شکل ۲ نمونه‌ای از خروجی سیستم logging را نشان می‌دهد که شامل timestamp، نام module، log level، و message مربوطه است.

```
2025-10-13 21:02:09 - multimodal_rag.frameworks.google_genai_base_service - INFO - google_genai_base_
service.py:79 - _switch_to_next_api_key() - Switched to API key index 0
2025-10-13 21:02:09 - multimodal_rag.frameworks.google_genai_base_service - WARNING - google_genai_ba
se_service.py:83 - _switch_to_next_api_key() - Cycled back to token 0, waiting 60 seconds before cont
inuing
2025-10-13 21:03:09 - multimodal_rag.frameworks.google_genai_base_service - INFO - google_genai_base_
service.py:115 - _execute_with_retry_and_token_switching() - Trying embeddings_for_content with API k
ey index 0
2025-10-13 21:03:11 - multimodal_rag.frameworks.google_genai_base_service - INFO - google_genai_base_
service.py:122 - _execute_with_retry_and_token_switching() - Successfully executed embeddings_for_con
tent with API key index 0
2025-10-13 21:03:11 - multimodal_rag.frameworks.google_genai_base_service - INFO - google_genai_base_
service.py:115 - _execute_with_retry_and_token_switching() - Trying embeddings_for_content with API k
ey index 0
2025-10-13 21:03:13 - multimodal_rag.frameworks.google_genai_base_service - WARNING - google_genai_ba
se_service.py:134 - _execute_with_retry_and_token_switching() - Rate limit error with API key index 0
, switching immediately
2025-10-13 21:03:13 - multimodal_rag.frameworks.google_genai_base_service - INFO - google_genai_base_
service.py:79 - _switch_to_next_api_key() - Switched to API key index 1
2025-10-13 21:03:13 - multimodal_rag.frameworks.google_genai_base_service - INFO - google_genai_base_
service.py:115 - _execute_with_retry_and_token_switching() - Trying embeddings_for_content with API k
ey index 1
```

شکل ۲: نمونه output سیستم logging

۸.۱ نتیجه‌گیری معماری

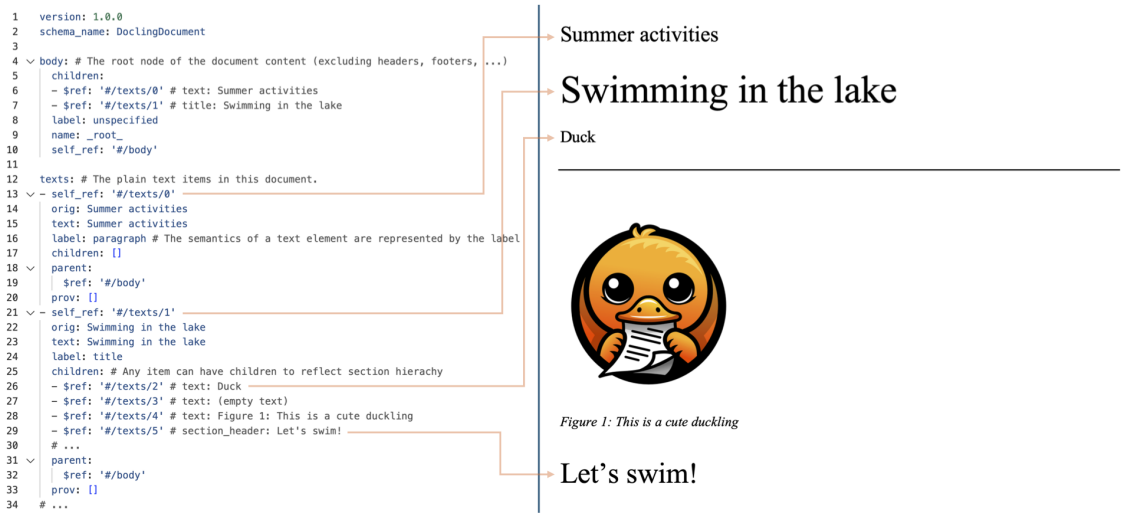
معماری Clean برای این پروژه MultiModal RAG، بستری مناسب برای توسعه یک سیستم scalable، maintainable و flexible فراهم آورده است. جداسازی واضح لایه‌ها، استفاده از dependency injection، و رعایت اصول SOLID، امکان توسعه پایدار و با کیفیت این سیستم را در بلندمدت تضمین می‌کند. این رویکرد معماری به‌ویژه برای سیستم‌های مبتنی بر LLM که نیازمند flexibility بالا در تعویض providers و models هستند، مناسب است.

۲ پردازش و نمایه‌سازی اسناد با Docling

۱.۲ معرفی Docling

Docling یک کتابخانه منبع‌باز پیشرفته برای پردازش و تبدیل اسناد ساختارنیافته به فرمت‌های قابل استفاده در سیستم‌های بازیابی اطلاعات است. این ابزار توسط تیم تحقیقاتی IBM توسعه یافته و به‌صورت رایگان در دسترس جامعه علمی قرار گرفته است. Docling با استفاده از مدل‌های یادگیری عمیق پیشرفته، قادر به تشخیص ساختار سلسله‌مراتبی اسناد، استخراج جداول پیچیده، و پردازش محتوای چندوجهی شامل متن، تصاویر و نمودارها می‌باشد.
ویژگی‌های کلیدی Docling عبارتند از:

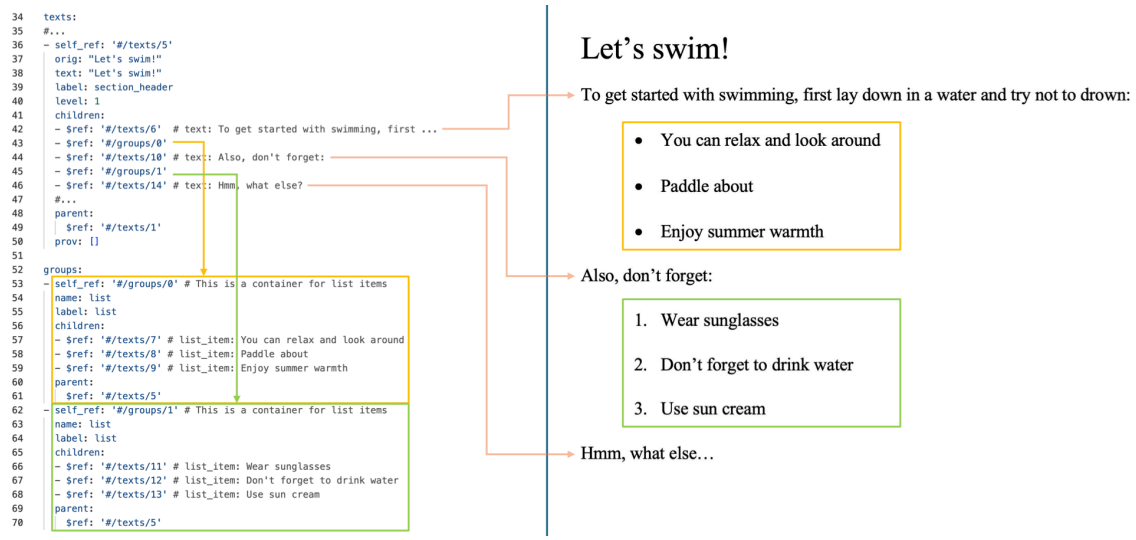
- **تشخیص ساختار سلسله‌مراتبی:** شناسایی خودکار عناوین، زیرعناوین، پاراگراف‌ها، و بخش‌های مختلف سند
- **استخراج دقیق جداول:** تبدیل جداول پیچیده با سلول‌های ادغام‌شده به فرمت ساختاریافته
- **پشتیبانی از محتوای چندوجهی:** پردازش همزمان متن، تصاویر، نمودارها و فرمول‌های ریاضی
- **خروجی استاندارد:** تولید فایل‌های JSON ، YAML و Markdown با حفظ روابط والد-فرزندی بین عناصر
- **تکه‌سازی هوشمند:** ترکیب عناصر مرتبط در تکه‌های معنادار با رعایت محدودیت‌های طول توکن



شکل ۳: نمونه‌ای از ساختار DoclingDocument. سمت راست صفحه‌ای با عناوین، تصویر، زیرنویس و پاراگراف‌ها را نشان می‌دهد. سمت چپ درخت سند را نمایش می‌دهد که در آن هر عنصر با برچسب و اشاره‌گر خود در ساختار سلسله‌مراتبی قرار گرفته است.

۲.۲ دلایل انتخاب Docling

انتخاب Docling به‌عنوان ابزار اصلی نمایه‌سازی کتاب پزشکی در این پروژه بر اساس ارزیابی دقیق نیازمندی‌های سیستم و مقایسه با ابزارهای موجود انجام شده است. در ادامه مهم‌ترین دلایل این انتخاب شرح داده می‌شود.



شکل ۴: فهرست‌های تو در تو و گروه‌ها در DoclingDocument. سمت راست دو فهرست (یکی نقطه‌ای و دیگری شماره‌دار) را نشان می‌دهد. سمت چپ نشان می‌دهد که هر فهرست در یک کانتینر گروهی قرار می‌گیرد و عناصر آن به‌عنوان فرزند اضافه می‌شوند.

۱.۲.۲ پردازش اسناد پزشکی پیچیده

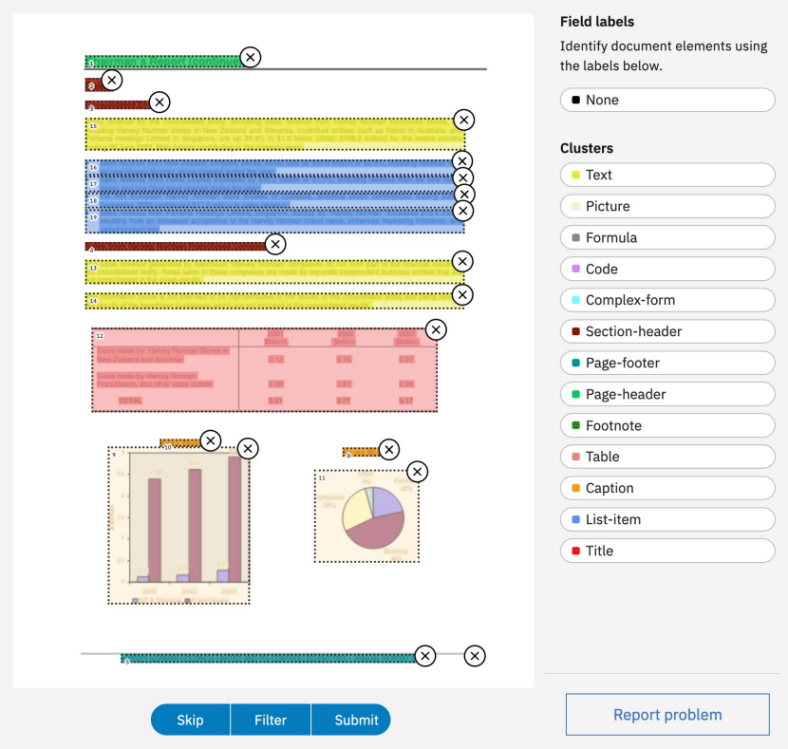
اسناد پزشکی از پیچیده‌ترین انواع محتوای علمی هستند که شامل ترکیبی از متن تخصصی، جداول آماری، تصاویر تشخیصی، نمودارهای آناتومیکی و فرمول‌های دارویی می‌باشند. Docling با معماری پردازشی چندمرحله‌ای خود، قادر به تشخیص و استخراج دقیق این عناصر متنوع است. برخلاف ابزارهای سنتی که صفحات را به‌صورت یک‌پارچه پردازش می‌کنند، Docling از یک خط‌لوله مرحله‌ای استفاده می‌کند که در آن هر نوع محتوا با مدل تخصصی خود پردازش می‌شود.

۲.۲.۲ معماری مبتنی بر مدل‌های تخصصی

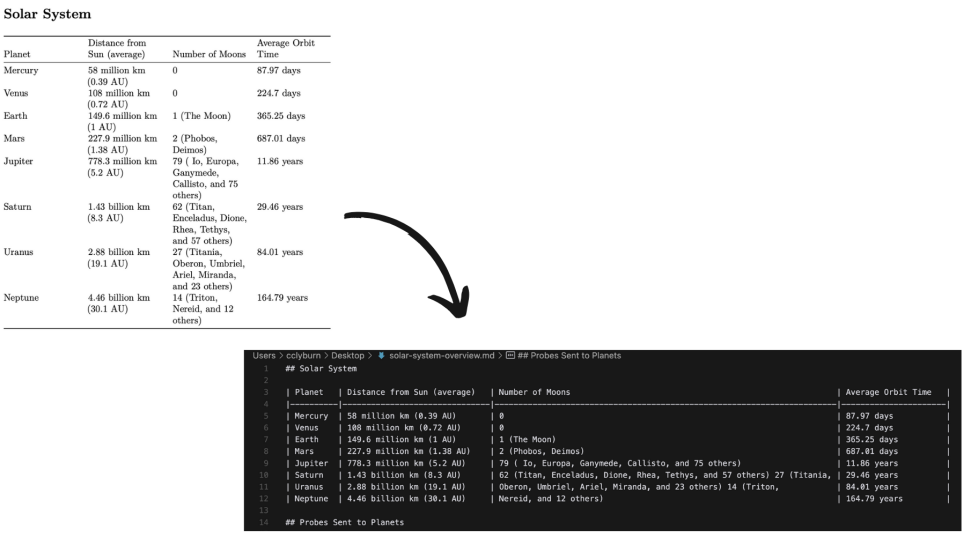
یکی از برتری‌های اصلی Docling نسبت به روش‌های مبتنی بر مدل‌های زبانی بزرگ چندوجهی (Vision Language Models) این است که به‌جای استفاده از یک مدل عمومی برای تمام وظایف، از مدل‌های تخصصی برای هر بخش استفاده می‌کند:

- **مدل تحلیل چیدمان (Layout Analysis):** Docling از مدل Heron مبتنی بر معماری RT-DETR با پشتوانه ResNet-50 استفاده می‌کند که روی مجموعه داده DocLayNet شامل بیش از ۸۰ هزار صفحه برچسب‌گذاری شده آموزش دیده است. این مدل قادر به تشخیص ۱۷ کلاس مختلف شامل عنوان، متن، فهرست، جدول، تصویر، عنوان بخش، فرمول، زیرنویس و سایر عناصر است. دقت این مدل بر روی مجموعه داده استاندارد DocLayNet به ۷۸ درصد mAP می‌رسد (شکل ۵).

- **مدل استخراج ساختار جداول (TableFormer):** برای پردازش جداول پیچیده، Docling از مدل TableFormer استفاده می‌کند که یک معماری ترنسفورمر مبتنی بر بینایی است. این مدل روی مجموعه داده‌های بزرگ شامل PubTabNet (بیش از ۲۲۷ هزار جدول) آموزش دیده و قادر به تشخیص سطرها، ستون‌ها، سلول‌های ادغام‌شده، و ساختارهای پیچیده جداول است. خروجی این مدل شامل توالی توکن‌های HTML و مختصات مرزی هر سلول است که سپس با متن اصلی PDF تطبیق داده می‌شود (شکل ۶).



شکل ۵: نمونه‌ای از برچسب‌گذاری عناصر مختلف سند در مجموعه داده DocLayNet. این تصویر نشان می‌دهد که چگونه هر عنصر صفحه (متن، عنوان، جدول، تصویر، فهرست و غیره) با کلاس‌های مختلف شناسایی و برچسب‌گذاری می‌شود. این برچسب‌های دقیق انسانی پایه آموزش مدل Heron را تشکیل می‌دهند.



Solar System

Planet	Distance from Sun (average)	Number of Moons	Average Orbit Time
Mercury	58 million km (0.39 AU)	0	87.97 days
Venus	108 million km (0.72 AU)	0	224.7 days
Earth	149.6 million km (1 AU)	1 (The Moon)	365.25 days
Mars	227.9 million km (1.38 AU)	2 (Phobos, Deimos)	687.01 days
Jupiter	778.3 million km (5.2 AU)	79 (Io, Europa, Ganymede, Callisto, and 75 others)	11.86 years
Saturn	1.43 billion km (8.3 AU)	62 (Titan, Enceladus, Dione, Rhea, Tethys, and 57 others)	29.46 years
Uranus	2.88 billion km (19.1 AU)	27 (Titania, Oberon, Umbriel, Ariel, Miranda, and 23 others)	84.01 years
Neptune	4.46 billion km (30.1 AU)	14 (Triton, Nereid, and 12 others)	164.79 years

شکل ۶: مثالی از استخراج دقیق جدول توسط TableFormer. بالا: جدول اصلی منظم شامل اطلاعات سیارات منظومه شمسی. پایین: جدول استخراج شده در فرمت Markdown که ساختار دقیق ستون‌ها و سطرها حفظ شده است.

- **مدل توصیف تصاویر (Picture Description Model):** یکی از چالش‌های مهم در پردازش کتاب‌های پزشکی، تولید توصیفات دقیق و جامع از تصاویر پیچیده پزشکی است. مدل پیش‌فرض Docling برای این منظور SmolVLM با حدود ۲۵۶ میلیون پارامتر است. با وجود سبک بودن این مدل، عملکرد آن در توصیف تصاویر پیچیده و نموداری پزشکی مطلوب نبود. در مرحله بعد، از مدل Granite-3.1 Vision شرکت IBM با ۲ میلیارد پارامتر استفاده شد که نتایج بهتری ارائه داد، اما به دلیل مشکل نشت حافظه (Memory Leak) در Docling، اجرای پایدار آن در محیط Google Colab با GPU T4 امکان‌پذیر نبود. در نهایت، برای دستیابی به بهترین کیفیت توصیف، از API رایگان مدل Gemini 2.5-Flash شرکت Google استفاده شد. برای مدیریت محدودیت‌های نرخ درخواست، یک سیستم تعویض خودکار بین توکن‌های مختلف پیاده‌سازی شد تا در صورت برخورد به خطای محدودیت نرخ با یک توکن، به‌طور خودکار از توکن دیگری استفاده شود و پردازش بدون وقفه ادامه یابد (شکل ۷).

- **موتور تشخیص نوشتار (OCR):** برای پردازش محتوای اسکن‌شده یا تصویری، Docling از EasyOCR استفاده می‌کند که یک کتابخانه یادگیری عمیق مبتنی بر PyTorch است. این موتور از مدل CRAFT برای تشخیص مناطق متنی و مدل CRNN برای خواندن کاراکترها استفاده می‌کند. Docling صفحات را با وضوح ۲۱۶ DPI رندر کرده و برای دستیابی به دقت بالا در تشخیص فونت‌های کوچک، از این موتور بهره می‌برد.

این معماری متخصص‌محور باعث می‌شود دقت پردازش در مقایسه با مدل‌های عمومی چندوجهی به‌طور قابل توجهی افزایش یابد، زیرا هر مدل فقط بر روی یک وظیفه خاص تمرکز دارد و برای آن بهینه‌سازی شده است.

۳.۲.۲ حفظ ساختار سلسله‌مراتبی و روابط معنایی

یکی از چالش‌های اساسی در پردازش اسناد پزشکی، حفظ روابط سلسله‌مراتبی بین عناصر مختلف است. کتاب‌های پزشکی دارای ساختار چندسطحی هستند که شامل فصل‌ها، بخش‌ها، زیربخش‌ها، پاراگراف‌ها، فهرست‌ها و عناصر وابسته مانند تصاویر و جداول می‌باشند. Docling با استفاده از مدل ترتیب خواندن (Reading Order Model) و سیستم مرجع‌دهی مبتنی بر JSON Pointer، این روابط را به‌طور کامل حفظ می‌کند.
در مدل داده DoclingDocument، تمام عناصر در یک ساختار درختی سازماندهی می‌شوند که در آن:

- هر عنصر دارای یک شناسه یکتا است
- روابط والد-فرزندی از طریق اشاره‌گرهای JSON تعریف می‌شوند
- عناصر گروهی مانند فهرست‌ها در کانتینرهای مجزا نگهداری می‌شوند
- ترتیب طبیعی خواندن عناصر حفظ می‌شود

این قابلیت برای سیستم بازیابی اطلاعات بسیار حیاتی است، زیرا امکان بازیابی محتوا با در نظر گرفتن بافت سلسله‌مراتبی آن را فراهم می‌کند. به‌عنوان مثال، هنگام بازیابی یک پاراگراف، سیستم می‌تواند به‌طور خودکار عنوان بخش و زیربخش مربوطه را نیز در نتایج قرار دهد.

۴.۲.۲ تکه‌سازی هوشمند با حفظ زمینه

Docling دارای یک سیستم تکه‌سازی هوشمند (Intelligent Chunker) است که یکی از نقاط قوت اصلی این ابزار محسوب می‌شود. برخلاف روش‌های سنتی تکه‌سازی که صرفاً بر اساس تعداد کاراکتر یا پاراگراف عمل می‌کنند، تکه‌ساز Docling از اطلاعات ساختاری سند برای ترکیب هوشمند عناصر مرتبط استفاده می‌کند.
ویژگی‌های تکه‌ساز Docling:

شکل ۷: نمونه‌ای از خروجی مدل توصیف تصویر Gemini 2.5-Flash برای یک تصویر پزشکی پیچیده. تصویر بالا یک ضایعه پوستی ناشی از تماس با موی بید برزیلی را نشان می‌دهد که شامل تاول‌ها و پاپول‌های قرمز روی پوست انسان است. مدل Gemini توانسته است جزئیات کامل تصویر شامل نوع ضایعه، موقعیت آناتومیکی، و ویژگی‌های بالینی را به‌طور دقیق توصیف کند، که نشان‌دهنده برتری آن نسبت به مدل‌های سبک‌تر مانند SmolVLM در پردازش تصاویر پزشکی پیچیده است.

- **ترکیب مبتنی بر معنا:** عناصر مرتبط مانند عنوان، پاراگراف‌های توضیحی، جداول و تصاویر مربوطه در یک تکه قرار می‌گیرند
- **رعایت محدودیت توکن:** تکه‌ها با توجه به حداکثر طول توکن مجاز ساخته می‌شوند
- **حفظ اطلاعات زمینه:** هر تکه شامل اطلاعات سلسله‌مراتبی مانند عنوان بخش و فصل است
- **جلوگیری از شکستگی محتوا:** جداول و فهرست‌ها به‌صورت یک‌پارچه در تکه‌ها قرار می‌گیرند

این رویکرد باعث می‌شود که تکه‌های تولید شده از نظر معنایی منسجم و برای سیستم بازیابی قابل استفاده باشند.

۵.۲.۲ خروجی استاندارد و قابل پردازش

Docling دو نوع خروجی اصلی ارائه می‌دهد که هر کدام برای کاربردهای خاصی مناسب هستند:

- **فرمت Markdown:** نسخه تمیز و خوانا از سند که در آن ساختار سلسله‌مراتبی با استفاده از سرفصل‌های Markdown (مثل #، ##، ###) نمایش داده می‌شود، جداول به‌صورت جداول Markdown فرمت می‌شوند، و محتوای متنی به‌صورت پاراگراف‌های منظم سازماندهی می‌شود.
- **فرمت JSON:** نمایش کامل و ساختاریافته سند که شامل تمام متادیتا، روابط سلسله‌مراتبی، مختصات عناصر، و اطلاعات تکمیلی است. این فرمت برای پردازش خودکار و نمایه‌سازی در پایگاه‌های داده بسیار مناسب است.

در این پروژه، از فرمت JSON برای استخراج دقیق اطلاعات و نمایه‌سازی در Elasticsearch استفاده می‌شود، در حالی که فرمت Markdown برای بازبینی انسانی و کنترل کیفیت مورد استفاده قرار می‌گیرد.

۶.۲.۲ منبع‌باز و قابل توسعه

Docling به‌صورت کامل منبع‌باز و تحت مجوز MIT منتشر شده است که مزایای زیر را به همراه دارد:

- امکان بررسی و درک کامل نحوه عملکرد داخلی سیستم
- قابلیت سفارشی‌سازی و توسعه مدل‌ها برای نیازهای خاص
- کاهش هزینه‌های عملیاتی با حذف نیاز به API پولی
- امکان اجرا در محیط‌های محلی و امن

این ویژگی برای پروژه‌های پزشکی که با داده‌های حساس سر و کار دارند، بسیار حیاتی است.

۷.۲.۲ کارایی و سرعت پردازش

Docling با استفاده از بهینه‌سازی‌های مختلف، سرعت پردازش قابل قبولی را ارائه می‌دهد:

- استفاده از ONNX Runtime برای استنتاج سریع‌تر مدل‌ها
- پردازش موازی صفحات در صورت وجود منابع کافی
- استفاده از رزولوشن مناسب برای هر مرحله (۷۲ DPI برای تحلیل چیدمان و ۲۱۶ DPI برای OCR)
- کش کردن نتایج میانی برای جلوگیری از پردازش مجدد

در آزمایش‌های انجام شده بر روی کتاب پزشکی مورد استفاده در این پروژه، Docling توانست هر صفحه را در زمان متوسط ۱۰ تا ۱۵ ثانیه (با GPU T4 در محیط Google Colab) پردازش کند که برای کاربردهای دسته‌ای قابل قبول است.

۸.۲.۲ یکپارچگی با ابزارهای پردازش زبان طبیعی

Docling به‌راحتی با کتابخانه‌های پردازش زبان طبیعی و سیستم‌های بازیابی یکپارچه می‌شود. در این پروژه، خروجی Docling به‌طور مستقیم به Elasticsearch منتقل می‌شود و با استفاده از LangChain برای ایجاد زنجیره‌های پردازشی پیچیده‌تر استفاده می‌شود. این یکپارچگی آسان امکان ساخت خط‌لوله‌های پردازشی پیچیده را با حداقل کد اضافی فراهم می‌کند.

۳.۲ مقایسه با روش‌های جایگزین

برای ارزیابی جامع‌تر انتخاب Docling، مقایسه‌ای با سایر روش‌های موجود انجام شده است:

۱.۳.۲ مدل‌های زبانی بزرگ چندوجهی (Vision Language Models)

مدل‌هایی مانند OpenAI GPT، Claude 4، و Gemini قادر به پردازش تصاویر صفحات هستند، اما محدودیت‌های زیر را دارند:

- **هزینه بالا:** هر صفحه نیاز به فراخوانی API دارد که برای کتاب‌های بزرگ هزینه‌بر است
- **عدم تضمین ساختار:** خروجی این مدل‌ها ممکن است فاقد ساختار ثابت باشد. مقایسه‌ای از عملکرد پایپلاین آماده شده با داکلینگ و GPT-۵ در این آدرس تهیه شده است.
- **محدودیت در جداول پیچیده:** دقت پایین در استخراج جداول با ساختار پیچیده

۲.۳.۲ کتابخانه‌های سنتی PDF مانند PyPDF2 و PDFMiner

این ابزارها فقط متن برنامه‌نویسی‌شده PDF را استخراج می‌کنند و محدودیت‌های اساسی دارند:

- عدم تشخیص ساختار سلسله‌مراتبی
- ناتوانی در پردازش جداول پیچیده
- عدم پشتیبانی از محتوای اسکن‌شده
- خروجی بدون ساختار و فاقد متادیتا

۳.۳.۲ مقایسه با Unstructured.io

یکی از جدیدترین و قدرتمندترین جایگزین‌های تجاری برای Docling، پلتفرم Unstructured.io است که به‌صورت ترکیبی از نسخه منبع‌باز و سرویس ابری ارائه می‌شود. این پلتفرم نیز هدف مشابهی با Docling دارد: تبدیل اسناد ساختارنیافته به داده‌های قابل استفاده برای سیستم‌های هوش مصنوعی. ویژگی‌های کلیدی Unstructured.io:

- **تجزیه و تقسیم‌بندی پیشرفته:** قابلیت شناسایی و جداسازی عناصر مختلف سند شامل پاراگراف‌ها، جداول، عناوین و تصاویر
- **تمیزسازی و بهنجارسازی:** حذف نویزها و نرمال‌سازی محتوا برای کیفیت بهتر پردازش
- **تکه‌سازی انطباقی:** تقسیم محتوا به تکه‌های مناسب برای ورودی مدل‌های زبانی
- **کانکتورهای یکپارچه‌سازی:** اتصال مستقیم به پایگاه‌های داده، دریاچه‌های داده و سیستم‌های ذخیره‌سازی

Unstructured.io برای پروژه‌هایی که بودجه کافی برای سرویس‌های ابری موجود است، و اولویت بالایی برای سرعت پیاده‌سازی نسبت به کنترل کامل فرآیند قائل هستند، گزینه مناسبی محسوب می‌شود.
با توجه به این مقایسه، Docling ترکیبی بهینه از دقت، کارایی، انعطاف‌پذیری و هزینه را ارائه می‌دهد که برای پردازش کتاب‌های پزشکی ایده‌آل است.

۴.۲ یکپارچگی با سیستم نمایه‌سازی

در این پروژه، خروجی Docling به‌صورت زیر با سیستم نمایه‌سازی یکپارچه می‌شود:

۱. **دریافت DoclingDocument:** پس از پردازش PDF، شی DoclingDocument حاوی تمام اطلاعات ساختاریافته دریافت می‌شود

۲. **تکه‌سازی هوشمند:** با استفاده از HybridChunker داخلی Docling، سند به تکه‌های معنادار با حداکثر طول توکن مشخص تقسیم می‌شود

۳. **غنی‌سازی با متادیتا:** هر تکه با اطلاعات اضافی شامل عنوان بخش، شماره صفحه، نوع محتوا، و روابط سلسله‌مراتبی غنی می‌شود

۴. **پردازش تصاویر:** تصاویر استخراج‌شده به مدل Gemini 2.5-Flash ارسال می‌شوند تا توصیف دقیق و جامعی از آن‌ها تولید شود

۵. **نمایه‌سازی در Elasticsearch:** تکه‌های نهایی به همراه توصیف تصاویر و متادیتای کامل در Elasticsearch نمایه‌سازی می‌شوند

۵.۲ نتیجه‌گیری

با توجه به ویژگی‌های منحصر به فرد Docling از جمله معماری مبتنی بر مدل‌های تخصصی، حفظ ساختار سلسله‌مراتبی، تکه‌سازی هوشمند، و ماهیت منبع‌باز آن، این ابزار به‌عنوان بهترین انتخاب برای نمایه‌سازی کتاب پزشکی در این پروژه شناسایی شد. ترکیب Docling با مدل چندوجهی Gemini برای توصیف تصاویر و Elasticsearch برای ذخیره‌سازی و بازیابی، یک سیستم جامع و کارآمد برای بازیابی اطلاعات پزشکی ایجاد می‌کند که قادر به پاسخگویی به پرسش‌های پیچیده با استفاده از اطلاعات متنی، جدولی و تصویری است.

۳ References

References

[۱] C. Ribeiro، “Sports scheduling: Problems and applications،” International Transactions in Operational Research، vol.۱۹، pp.۲۲۶–۲۰۱، Jan. .۲۰۱۲