این گزارش جامع پروژه بر اساس اطلاعات استخراج شده از منابع و مکالمات، به طور مفصل به اهداف، معماری فنی، گردش کار، چالش‌ها و نتایج پروژه **سامانه هوشمند تگ‌گذاری محصولات ترب (Torob Image Tagging)** می‌پردازد.

گزارش جامع پروژه: سامانه هوشمند تگ‌گذاری محصولات ترب (Torob Image Tagging)

۱. هدف و مقدمه پروژه

هدف اصلی این پروژه توسعه یک **سیستم هوشمند** است که با استفاده از هوش مصنوعی، بتواند ویژگی‌های محصول (تگ‌ها) مانند **رنگ، دسته‌بندی، جنسیت و برند** را مستقیماً از تصاویر محصولات استخراج کند. خروجی نهایی این سیستم باید به صورت **ساختاریافته (JSON)** باشد. داده‌های استخراج شده برای بهبود موتور جستجو و فیلتر محصولات در وب‌سایت ترب مورد استفاده قرار می‌گیرند تا کاربران بتوانند جستجوی دقیق‌تری داشته باشند.

ماهیت مسئله: این پروژه نیازمند استخراج **بیش از ۱۰۰۰ نوع تگ مختلف** است، که برخی از این تگ‌ها هزاران مقدار گوناگون دارند.

رویکرد حل مسئله: تیم تصمیم گرفت به جای رویکردهای کلاسیک مانند **Multi-label Classification**، از **مدل‌های مولد (Generative Models)** نظیر **VLM/LLM** استفاده کند.

۲. معماری فنی و تکنولوژی‌ها

معماری پروژه به دو بخش اصلی تقسیم می‌شود که با استفاده از API با یکدیگر در ارتباط هستند.

۲.۱. ساختار کلی

1. **بک‌اند (Backend):** مغز سیستم، که وظیفه پردازش تصاویر و تولید تگ‌ها را بر عهده دارد.
    - **زبان و فریمورک: Python 3.10+** و **FastAPI** (برای ساخت API).
    - **نقطه شروع:** فایل src/main.py برنامه API را در پورت ۸۰۰۰ راه‌اندازی می‌کند.
2. **فرانت‌اند (Frontend):** رابط کاربری که وظیفه نمایش صفحه وب برای کاربر و تعامل با سیستم را دارد.

- **زبان و فریمورک: JavaScript، React**، و **Next.js**. یک رابط کاربری وب (UI) با React/Next.js توسط مهدی ساخته و دیپلوی شد.

۲.۲. تکنولوژی‌های کلیدی

بخش	تکنولوژی	وظیفه اصلی
گردش کار (Workflow)	**LangGraph**	ساخت و اجرای "گردش کار" هوشمند چندمرحله‌ای.
دسترسی به AI	**OpenRouter**	دسترسی به مدل‌های هوش مصنوعی مختلف (Vision و Translate).
کشینگ (Caching)	**Redis**	ذخیره موقت نتایج (Key: image_tags:{hash}) برای سرعت بیشتر. طول عمر (TTL) نتایج کش شده ۱ ساعت است.
ذخیره‌سازی دائمی	**MongoDB**	پایگاه داده برای ذخیره دائمی اطلاعات.
ذخیره فایل	**MinIO**	ذخیره تصاویر محصولات آپلود شده.
استقرار	**Docker Compose**	اجرای آسان و کانتینری همه سرویس‌ها (Backend, Frontend, DBs, MinIO).
مانیتورینگ	**Prometheus/Grafana**	جمع‌آوری و نمایش معیارهای عملکرد (مثل زمان پاسخ و خطاها).

۳. گردش کار و پردازش سیستم

فایل src/controller/api_controller.py مغز API است و شامل توابع /generate-tags و /upload-and-tag می‌شود.

۳.۱. مراحل کامل گردش کار (LangGraph Workflow)

گردش کار (Workflow) با استفاده از **LangGraph** ساخته شده که در آن **Nodes** (گره‌ها) مراحل مختلف پردازش و **Edges** (یال‌ها) مسیرها را تعریف می‌کنند. **State** نیز مانند یک دفترچه یادداشت، اطلاعات را بین مراحل مختلف نگه می‌دارد.

1. **دریافت و چک‌های اولیه (API Controller):**
   - **Rate Limiting:** برای هر IP آدرس، تعداد درخواست‌ها شمرده می‌شود و اگر بیش از **۱۰ درخواست در ۶۰ ثانیه** باشد، خطا می‌دهد.
   - **تولید Hash و چک کش:** یک Hash (اثر انگشت) منحصر به فرد برای تصویر تولید می‌شود. کلید {image_tags:{hash در Redis چک می‌شود. اگر نتیجه قبلاً پردازش شده باشد، از کش برگردانده شده و پاسخ **فوری** است (سرعت کمتر از ۱۰۰ میلی‌ثانیه).
2. **ذخیره تصویر (MinIO):** تصویر در سرویس MinIO ذخیره می‌شود.
3. **پردازش LangGraph (اگر در کش نبود):**
   - **Node: image_to_tags:** این مرحله یک دستورالعمل (Prompt) می‌سازد. تصویر و دستورالعمل به **مدل Vision** (مانند Qwen2.5-VL یا LLaVA) ارسال می‌شود. مدل تصویر را تحلیل کرده و تگ‌های انگلیسی را به صورت JSON برمی‌گرداند.
   - **Node: serpapi_search:** این مرحله **فقط در حالت Advanced** اجرا می‌شود. با استفاده از Serper API، اطلاعات اضافی از اینترنت جستجو می‌شود (رویکرد RAG) که به ترجمه دقیق‌تر و **همسویی فرهنگی (Alignment)** کمک می‌کند.
   - **Node: translate_tags:** یک مدل ترجمه (مانند Longcat Flash Chat) تگ‌های انگلیسی را به فارسی روان و طبیعی ترجمه می‌کند و از نتایج SerpAPI برای اطمینان از **ترجمه طبیعی و بومی** (مثل تشخیص "پولوشرت") استفاده می‌کند.
   - **Node: merge_results:** نتایج نهایی (شامل تگ‌های فارسی و انگلیسی) آماده می‌شود.
4. **ذخیره‌سازی و بازگشت نتیجه:**
   - نتیجه نهایی در **Redis Cache** با TTL یک ساعته ذخیره می‌شود.
   - اطلاعات نهایی در **MongoDB** ذخیره می‌شود. این کار به صورت **وظیفه پس‌زمینه (Background Task)** انجام می‌شود تا API مجبور نباشد منتظر اتمام عملیات ذخیره بماند و پاسخ را سریع‌تر برگرداند.
   - نتیجه نهایی به فرانت‌اند برگردانده می‌شود.

۳.۲. حالت‌های پردازش (Modes)

سیستم سه حالت پردازش مختلف ارائه می‌دهد که تعادلی بین سرعت و دقت ایجاد می‌کنند:

حالت	Vision Model	Translate Model	SerpAPI	سرعت	دقت
⚡ **Fast**	nemotron-nano	longcat-flash	❌	⚡⚡⚡	⭐⭐
🧠 **Reasoning**	nemotron-nano	deepseek-r1	❌	⚡⚡	⭐⭐⭐
🚀 **Advanced**	nemotron-nano	deepseek-r1	✅	⚡	⭐⭐⭐⭐

۴. چالش‌های فنی و مدیریت خطا

پروژه با چالش‌های مهمی در استفاده از مدل‌های مولد مواجه شد که راه‌حل‌های زیر برای آن‌ها پیاده‌سازی گردید:

1. **محدودیت نرخ (Rate Limit) در APIهای خارجی:** استفاده از مدل‌های آزاد OpenRouter منجر به خطاهای مکرر Rate Limit شد.
    - **راه‌حل:** پیاده‌سازی منطق **تلاش مجدد (Retry)** با **تأخیر تصاعدی (Exponential Backoff)** درون LangGraph و استفاده از کلیدهای API متعدد.
2. **عدم نرمال‌سازی خروجی VLM:** مدل‌های VLM گاهی خروجی JSON را ناقص یا نامنظم برمی‌گرداندند.
    - **راه‌حل:** پیاده‌سازی قابلیت **JSON Extraction** از متن و مدیریت خطا (Error Handling) در فایل model_client.py.
3. **عدم همسویی فرهنگی (Cultural Alignment):** مدل‌ها در ابتدا در تشخیص واژگان بومی (مانند "**شلوار کردی**" یا "**مانتو دانشجویی**") مشکل داشتند.

- **راه‌حل:** استفاده از رویکرد **RAG (بازیابی اطلاعات)** از طریق Serper API (جستجوی عنوان‌های مشابه در گوگل) و تزریق نتایج جستجو به مدل اصلی، که باعث شد مدل بتواند تگ‌های بومی را با موفقیت تولید کند.

۵. ارزیابی و نتایج

۵.۱. داده و Ground Truth

- **تحلیل داده‌های اکتشافی (EDA):** تحلیل داده‌ها (توسط پیمان) نشان داد که در دیتاسِت ترب، بیش از ۱۱۵۳ کلید (Entity Name) منحصر به فرد وجود دارد و به طور متوسط برای هر محصول ۲.۶ تگ ثبت شده است.
- **کیفیت داده خام:** تحلیل‌ها نشان داد که داده‌های خام موجود از کیفیت پایینی برخوردارند.
- **داده مرجع (Ground Truth):** به دلیل پایین بودن کیفیت داده‌های خام، یک **نمونه مرجع ۳۰۰ تایی (Toy Sample)** تمیز شده (با حداقل ۸ تگ برای هر محصول) ساخته شد تا برای ارزیابی کمی و کیفی استفاده شود.

۵.۲. معیارها و عملکرد

- **معیارهای کمی:** برای سنجش دقت مدل، از متریک‌هایی نظیر **Semantic Similarity** (که در حضور مترادف‌هایی مانند "سیاه" و "مشکی" ضروری است)، **Exact Match، F1، Dice و ROUGE-1** استفاده شد.
- **نتایج کلیدی:** پایپ‌لاین پیاده‌سازی شده (VLM + Translator) توانست نرخ موفقیت **بیش از ۹۰ درصد** را در تولید خروجی JSON (فقط ۲۷ مورد ناموفق در نمونه ۳۰۰ تایی) نشان دهد.
- **مقایسه مدل:** نتایج ارزیابی نشان داد که رویکرد ترکیبی (VLM + Captioning + LLM Refinement) با استفاده از مدل‌هایی مانند **GPT-4o-mini** می‌تواند دقتی معادل مدل‌های مقرون‌به‌صرفه‌تری مانند **Claude Sonnet 4.5** به دست آورد.
- **سرعت:** سیستم با فعال بودن کش، سرعت پاسخگویی **کمتر از ۱۰۰ میلی‌ثانیه** را دارد.

۶. مسیر توسعه و کارهای آتی

بر اساس چالش‌ها و توصیه‌های منتور (آقای همایون)، مسیر آتی پروژه شامل موارد زیر است:

1. **Fine-Tuning مدل‌های کوچک:** استفاده از سیستم فعلی برای گرفتن خروجی VLM برای تمام تصاویر ترب، ساخت یک دیتاسِت تمیز و بزرگ و فاین تیون کردن یک مدل محلی و کوچک‌تر برای کاهش هزینه‌ها و وابستگی به APIهای خارجی.
2. **پیاده‌سازی VLM به صورت سرویس لوکال:** برای حل نهایی مشکل سرعت و هزینه، پیاده‌سازی مدل‌های VLM (مانند **Gemma-3-4b-it**) به صورت لوکال بر روی سرورهای داخلی ضروری است.
3. **تگ‌گذاری سلسله مراتبی (Hierarchical Tagging):** بهبود پرامپت‌ها برای تولید تگ‌ها به صورت ساختاریافته‌تر (شامل سکشن و ساب‌سکشن) که به افزایش فهم و دقت مدل کمک شایانی می‌کند.
4. **مدیریت ناهماهنگی لغوی:** ساخت یک **گراف دانش (Knowledge Graph)** برای اتصال تگ‌های مشابه (مانند "سیاه" و "مشکی") برای مدیریت بهتر مترادف‌ها پیشنهاد شد.