Mercari Japan AI Shopper

Table of Contents

Overview

This project is an AI-powered shopping assistant designed to navigate Mercari Japan, a popular Japanese e-commerce platform. It leverages a Large Language Model (LLM) configured as a ReAct (Reasoning and Acting) agent to provide users with intelligent, context-aware product recommendations. Given a natural language query (e.g., "a good phone for gaming under $500"), the agent can perform multi-step reasoning, use a suite of tools to search for products, analyze market prices, evaluate their relevance, and present the top 3 choices with detailed, user-centric justifications.

Setup Instructions

Prerequisites

- Docker
- Docker Compose

Setup

1. Create a `.env` file, or you can copy the `.env.example` file and rename it to `.env`.

```
cp .env.example .env
```

2. Fill in the `.env` file with the correct values.

- `ANTHROPIC_API_KEY`: The API key for the Anthropic API.
- `MODEL_NAME`: The name of the model to use. Defaults to `claude-3-5-sonnet-latest`.
- `REDIS_HOST`: The host of the Redis server. Defaults to `redis`.

- `REDIS_PORT`: The port of the Redis server. Defaults to `6379`.

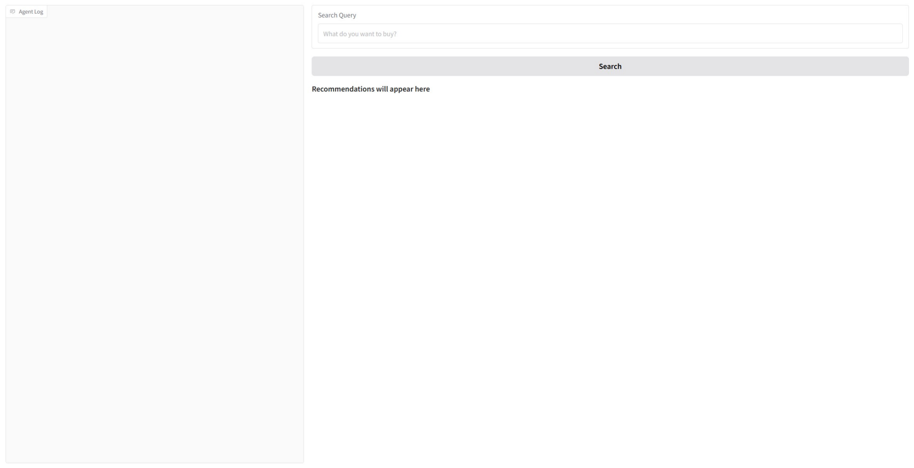
3. Build and run the Docker container:

```
docker-compose up
```

your application should be running on http://localhost:7860.

Usage Instructions

1. Open your browser and navigate to http://localhost:7860. It should look like this:



2. Enter your query in the input field and click the "Search" button.

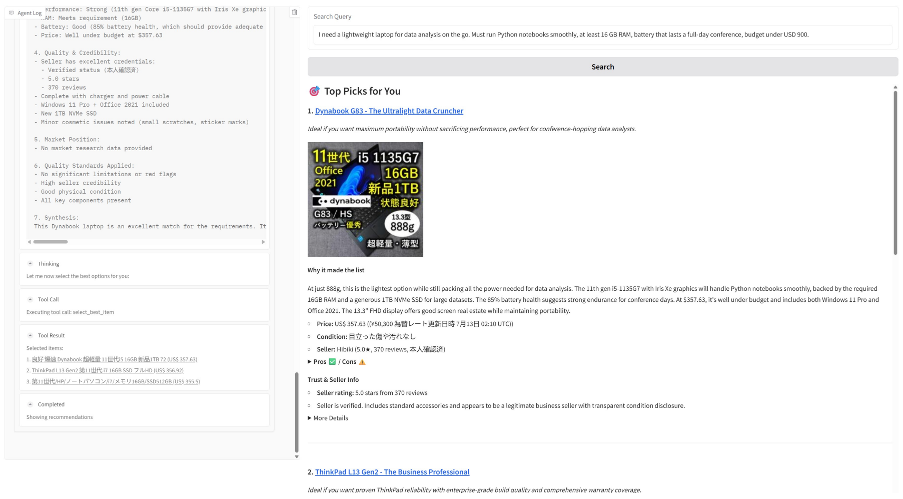
For example, type in the following query:

```
I need a lightweight laptop for data analysis on the go. Must run Python notebooks smoothly
```

and press enter or click the "Search" button.

3. Wait the process to complete. It took me about 5 minutes to complete the process.

4. You can see the recommended items and the reasoning process.



```
erformance: Strong (11th gen Core i5-1135G7 with Iris Xe graphic
 .AM: Meets requirement (16GB)
- Battery: Good (85% battery health, which should provide adequate
- Price: Well under budget at $357.63

4. Quality & Credibility:
- Seller has excellent credentials:
  - Verified status (本人確認済)
  - 5.0 stars
  - 370 reviews
- Complete with charger and power cable
- Windows 11 Pro + Office 2021 included
- New 1TB NVMe SSD
- Minor cosmetic issues noted (small scratches, sticker marks)

5. Market Position:
- No market research data provided

6. Quality Standards Applied:
- No significant limitations or red flags
- High seller credibility
- Good physical condition
- All key components present

7. Synthesis:
This Dynabook laptop is an excellent match for the requirements. It
```

Thinking

Let me now select the best options for you:

Tool Call

Executing tool call: select_best_item

Tool Result

Selected items:

1. 良好 爆速 Dynabook 超軽量 11世代i5 16GB 新品1TB 72 (US$ 357.63)
2. ThinkPad L13 Gen2 第11世代 i7 16GB SSD フルHD (US$ 356.92)
3. 第11世代/HP/ノートパソコン/i7/メモリ16GB/SSD512GB (US$ 355.5)

Completed

Showing recommendations

Search Query

I need a lightweight laptop for data analysis on the go. Must run Python notebooks smoothly, at least 16 GB RAM, battery that lasts a full-day conference, budget under USD 900.

Search

🎯 Top Picks for You

1. Dynabook G83 - The Ultralight Data Cruncher

Ideal if you want maximum portability without sacrificing performance, perfect for conference-hopping data analysts.

Why it made the list

At just 888g, this is the lightest option while still packing all the power needed for data analysis. The 11th gen i5-1135G7 with Iris Xe graphics will handle Python notebooks smoothly, backed by the required 16GB RAM and a generous 1TB NVMe SSD for large datasets. The 85% battery health suggests strong endurance for conference days. At $357.63, it's well under budget and includes both Windows 11 Pro and Office 2021. The 13.3" FHD display offers good screen real estate while maintaining portability.

- **Price:** US$ 357.63 ((¥50,300 為替レート更新日時 7月13日 02:10 UTC))
- **Condition:** 目立った傷や汚れなし
- **Seller:** Hibiki (5.0★, 370 reviews, 本人確認済)

▶ Pros ✅ / Cons ⚠️

Trust & Seller Info

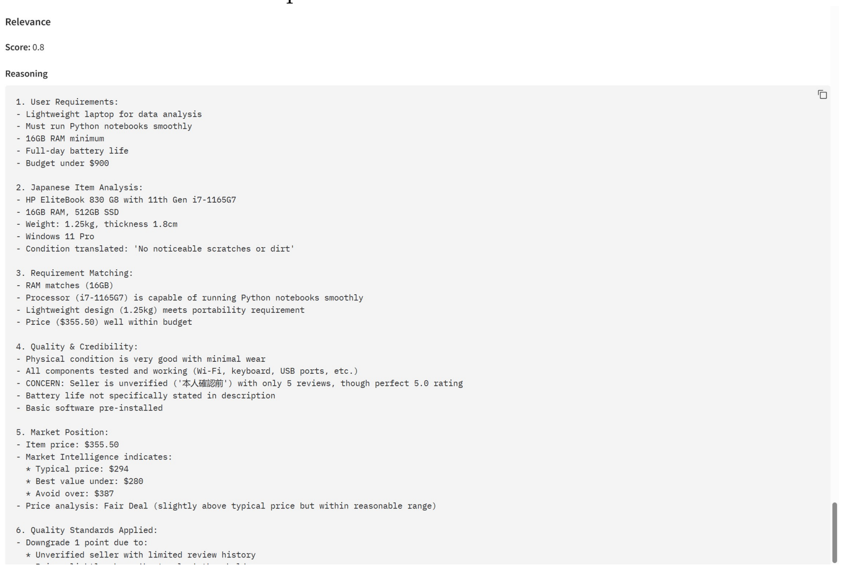
- **Seller rating:** 5.0 stars from 370 reviews
- Seller is verified. Includes standard accessories and appears to be a legitimate business seller with transparent condition disclosure.

▶ More Details

2. ThinkPad L13 Gen2 - The Business Professional

Ideal if you want proven ThinkPad reliability with enterprise-grade build quality and comprehensive warranty coverage.

5. You also can see the detailed report for each item in the `More Details` section.



Relevance

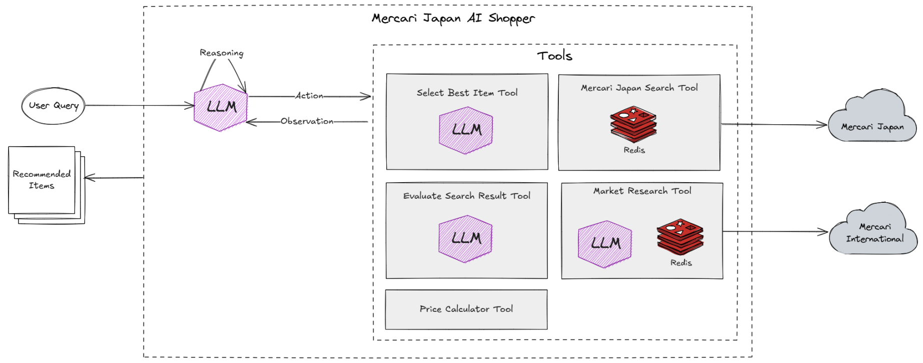
Score: 0.8

Reasoning

```
1. User Requirements:
- Lightweight laptop for data analysis
- Must run Python notebooks smoothly
- 16GB RAM minimum
- Full-day battery life
- Budget under $900

2. Japanese Item Analysis:
- HP EliteBook 830 G8 with 11th Gen i7-1165G7
- 16GB RAM, 512GB SSD
- Weight: 1.25kg, thickness 1.8cm
- Windows 11 Pro
- Condition translated: 'No noticeable scratches or dirt'

3. Requirement Matching:
- RAM matches (16GB)
- Processor (i7-1165G7) is capable of running Python notebooks smoothly
- Lightweight design (1.25kg) meets portability requirement
- Price ($355.50) well within budget

4. Quality & Credibility:
- Physical condition is very good with minimal wear
- All components tested and working (Wi-Fi, keyboard, USB ports, etc.)
- CONCERN: Seller is unverified ('本人確認前') with only 5 reviews, though perfect 5.0 rating
- Battery life not specifically stated in description
- Basic software pre-installed

5. Market Position:
- Item price: $355.50
- Market Intelligence indicates:
  * Typical price: $294
  * Best value under: $280
  * Avoid over: $387
- Price analysis: Fair Deal (slightly above typical price but within reasonable range)

6. Quality Standards Applied:
- Downgrade 1 point due to:
  * Unverified seller with limited review history
```

Design Choices

Architecture



In this project, I decided to use ReAct agent architecture proposed by Yao et al., 2022. This architecture is a combination of reasoning and acting in language models.

Generating reasoning traces allow the model to induce, track, and update action plans, and even handle exceptions. The action step allows to interface with and gather information from external sources such as knowledge bases or environments.

The ReAct framework can allow LLMs to interact with external tools to retrieve additional information that leads to more reliable and factual responses.

Results show that ReAct can outperform several state-of-the-art baselines on language and decision-making tasks. ReAct also leads to improved human interpretability and trustworthiness of LLMs. Overall, the authors found that best approach uses ReAct combined with chain-of-thought (CoT) that allows use of both internal knowledge and external information obtained during reasoning.

For this shopping agent, the ReAct paradigm is particularly effective. The process of finding the right product is not a single task; it's a sequence of searching, filtering, analyzing, and comparing. ReAct allows the agent to break down this complex process into a series of thoughts and actions. It can formulate a search strategy, execute a search tool, observe the results, and then reason about the next best action—whether that's refining the search, researching a specific product's market value, or evaluating its relevance. This iterative process, combining the LLM's reasoning with factual information from external tools, makes the agent's recommendations more reliable and transparent.

State

The agent maintains two primary forms of state during a single user session: the conversational chat history and a structured `State` object. The chat history provides sequential context for the LLM, allowing it to follow the flow of reasoning and tool usage.

The `State` object, in parallel, acts as a structured "scratchpad" for the current job. It's created at the beginning of a run and passed between the agent and its tools. This allows for the structured accumulation and modification of data, like search results and candidate items, separate from the conversational flow.

The `State` object contains the following fields:

Field	Type	Description
`user_query`	string	The original, unmodified query provided by the user. This is used as the "source of truth" for all evaluation steps.
`search_results`	list[Item]	A list of all items found by the `MercariJPSearchTool`. This list can grow as the agent performs multiple searches.
`recommended_candidates`	list[Item]	A subset of items from `search_results` that have been evaluated with a relevance score. This list acts as the pool of candidates for the final selection.
`recommended_items`	list[ItemRecommendation]	The final list of top 3 items selected by the `SelectBestItemTool`, complete with detailed reasoning. When this list contains 3 items, the agent's job is done.

Tools

The agent's capabilities are extended through a set of specialized tools. Each tool is a function that the agent can choose to call to gather information or perform a specific action. This separation of concerns allows the agent to interact with external services and data sources in a structured way.

Mercari Japan Search Tool This is the agent's primary tool for discovery. It takes a search query (which the agent translates to Japanese) and optional price filters (in JPY) to search for products on the Mercari Japan website. It returns a list of candidate items that serve as the starting point for the agent's analysis.

Parameter	Type	Description
`query`	string	The search term, preferably in Japanese.
`min_price`	integer	(Optional) The minimum price in JPY.
`max_price`	integer	(Optional) The maximum price in JPY.
`max_items`	integer	(Optional) The maximum number of items to return. Defaults to 10.
`sort_by`	string	(Optional) Field to sort by. Can be `num_likes`, `score`, `created_time`, or `price`. Defaults to `score`.
`order`	string	(Optional) Sort order. Can be `asc` or `desc`. Defaults to `desc`.

To perform the search, I implemented scraping using Playwright, since the website contain anti-bot detection, so by using requests and beautifulsoup, is not enough, we need to execute javascript code to get the data. The search process follow this steps: 1. Search the query in `https://jp.mercari.com/search?keyword={query}` with optional parameters like `min_price`, `max_price`, `sort_by`, `order`, `max_items`. 2. Wait for the page to load. If not found page that contains text found, the search is failed and stopped. 3. Get all items until we reach the `max_items` limit. In this step we only get basic information like `item_id`, `item_url`, `item_name`, `item_price`, `item_currency`, and `item_image_url`. 4. To get more detailed information, we need to visit the item page. In this step, I scraped the item details in parallel using `asyncio.gather` and `asyncio.Semaphore(5)` to speed up and maintain memory usage. 5. At this point, each item detail will be cached in Redis, so if the same item is searched again, the result will be returned from the cache. 6. All the item details are then gathered and combined to the inital item list. Below is the data structure of the item:

```python
class Item(BaseModel):
    """A structured representation of a product listing from Mercari."""

    id: str
    """The ID of the item."""

    name: str
    """The name of the item."""

    price: float
    """The price of the item."""

    currency: str
    """The currency of the item."""

    brand: str | None = None
    """The brand of the item."""

    condition_grade: str | None = None
    """The condition grade of the item."""

    availability: str | None = None
    """The availability of the item."""

    image_url: str
    """The URL of the item's image."""

    item_url: str
    """The URL of the item."""

    item_detail: ItemDetail | None = None
    """The detail of the item."""

    relevance_score: ItemRelevanceScore | None = None
    """The relevance score of the item."""

    market_research_result: MarketIntelligenceResult | None = None
    """The market research result of the item."""

class ItemDetail(BaseModel):
    """A structured representation of a product listing from Mercari."""

    converted_price: str | None = None
    """The converted price of the item."""

    description: str
    """The description of the item."""

    condition_type: str
    """The condition type of the item."""
    posted_date: str | None = None
    """The date the item was posted."""

    delivery_from: str
    """The country the item is being delivered from."""

    shipping_fee: str | None = None
    """The shipping fee of the item."""

    seller_name: str
    """The name of the seller."""

    seller_username: str | None = None
    """The username of the seller."""

    seller_review: int
    """The number of reviews the seller has."""

    seller_review_stars: float
    """The average rating of the seller."""

    categories: list[str]
    """The categories the item belongs to."""

    seller_verification_status: str | None = None
    """The credibility of the seller."""

    num_likes: int | None = None
    """The number of likes the item has."""
```

Price Calculator Tool A simple but essential utility tool. When a user provides a budget in a currency like USD, the agent uses this tool to convert it to Japanese Yen (JPY). This ensures that the price filters used in the `MercariJPSearchTool` are accurate for the target marketplace.

Parameter	Type	Description
`source_currency`	string	The currency to convert from (Only “USD” and “JPY” are supported).
`target_currency`	string	The currency to convert to (Only “USD” and “JPY” are supported).
`source_price`	float	The amount of money in the source currency.

The conversion implementation is actually hardcoded for simplicity, here is the conversion rate:

```
JPY to USD: 0.0068
USD to JPY: 147.42
```

Market Research Tool This tool provides crucial financial context. Given a specific item, it uses an LLM to analyze the product's details and search for comparable listings to determine its approximate market value. It returns a summary of typical market prices, helping the agent to assess whether a listing is a bargain, fairly priced, or overpriced.

Parameter	Type	Description
`item_ids`	list[string]	A list of item IDs (from a previous search) to research.

This tool follow this steps: 1. Use LLM to generate a specific query in English based on the item name, description and categories. 2. Use that query to search in Mercari International. 3. Retrieve all item prices from the search results. 4. Next we will analyze the statistics of the prices like `min`, `max`, `average`, `median`, `budget_range_max`, `mid_range_min`, `mid_range_max`, `premium_range_min`, `excellent_deal_max`, `good_deal_max`, `overpriced_min`. 5. Then we will calculate the price volatility using `Interquartile Range (IQR)` to assess price volatility and generate `Quartile Coefficient of Dispersion (QCD)` score and follow this rules: - If QCD <= 0.10, the price is `stable`. - If QCD <= 0.20, the price is `moderate`. - If QCD > 0.20, the price is `volatile`. 6. Then we will generate a natural language summary of the market research result.

Here is the example:

```
Market Intelligence Report:
Summary: Market intelligence for HP EliteBook 830 G8 11th Gen Intel Core i7-1165G7 16GB 512
Analyzed 13 items.
Typical price: USD294 (¥43,346) (range: USD89 (¥13,120)-USD549 (¥80,934)).
Price volatility: volatile.
Recommendation: Look for items under USD280 (¥41,219) for best value. Avoid items over USD3
Price Strategy: High price volatility. Wait for significant discounts.
Value Strategy: Best value: items under USD280 (¥41,219). Avoid: items over USD387 (¥57,052
Expected Price: Expect to pay around USD294 (¥43,346) for typical quality
Price Volatility: volatile
```

The market research result is cached in Redis, so if the same item is researched again, the result will be returned from the cache.

Evaluate Search Result Tool After finding items and analyzing their market price, this tool evaluates how well each item matches the user's original query. It uses an evaluation technique called G-Eval to compare the item's specifications and description against the user's needs, producing a `relevance_score` from 1 to 5. This score is critical for filtering out irrelevant items and identifying the most promising candidates. Basically G-Eval utilizing chain-of-thought (CoT) and form-filling paradigms to evaluate the relevance of the item to the user's query. In their paper, they use the probability-weighted summation of the output scores as the final score, but in this project, I just use the value generated from the LLM as the final score. To make the score easy to understand, I convert it to 0-1 range.

I run the evaluation in parallel using `asyncio.gather` to speed up the process. All item that has been evaluated will be stored in `recommendation_candidates` key in the State object.

Parameter	Type	Description
`item_ids`	list[string]	A list of item IDs to evaluate for relevance.

Select Best Item Tool This is the final tool. Once the agent has gathered and evaluated a set of strong candidate items (with a relevance score >= 0.8), it calls this tool. It uses a specialized prompt to instruct an LLM to perform a final, holistic comparison of the candidates. It selects the top 3 items and generates the detailed, user-friendly reasoning for each, including a title, persona fit, pros, and cons.

This tool takes no arguments from the agent; it automatically operates on all candidate items in `recommendation_candidates` key in the State object that have a relevance score of 0.8 or higher.

Caching

To improve performance and reduce redundant computations, caching is implemented for the following tools: - `MercariJPSearchTool` - `MarketResearchTool`

The cache is powered by Redis and managed using the aiocache library.

Large Language Model (LLM)

This project uses the `claude-3-5-sonnet-latest` model from Anthropic. Several strategies are employed to optimize its use.

Prompt Caching To reduce costs and improve latency on repeated, identical prompts, this project utilizes Anthropic's prompt caching feature.

Context Size Management To manage the LLM's context window, a hybrid approach is used: 1. The agent maintains a full chat history until the context size approaches its limit (`70,000` tokens). 2. If the limit is reached, the history is condensed. The last `3` messages are preserved, and a summary of the `recommendation_candidates` is injected into a new user prompt to maintain the most critical context for the next step.

Structured Output To ensure reliable JSON parsing from the LLM's output, this project uses the json-repair library. This robustly handles common formatting errors in the model's raw text, providing a reliable mechanism for working with structured data.

Stop Condition

The agent's primary goal is to provide three high-quality recommendations. Therefore, the main stop condition is met when the `recommended_items` list in the State object contains three items. The agent will also stop if it reaches the maximum number of iterations (15) to prevent infinite loops.

User Interface

The user interface is built with Gradio. Gradio was chosen for its simplicity and its ability to quickly create interactive web UIs for machine learning models. Its native support for rendering Markdown and images makes it well-suited for displaying the agent's detailed, multi-faceted recommendations.

Error Handling and Retries

To handle transient network issues and API errors from Anthropic (such as `529 overloaded_error` or `429 rate_limit_error`), the agent employs a retry mechanism. This is implemented using the `aioretry` library, which automatically re-attempts failed API calls with an exponential backoff strategy (up to 5 retries). For other unexpected errors during tool execution, the system logs the issue and is designed to continue processing where possible, ensuring greater resilience.

Logging

The project uses Loguru for logging. It provides clear, color-coded logs of the agent's thoughts, actions, and tool calls, which is invaluable for debugging and tracing the agent's reasoning process.

Potential Improvements

- **Enhanced Search Strategy:** Implement more sophisticated search techniques, such as browsing by category or using filters for brand, condition, and seller, to better mimic human browsing patterns and yield more targeted results.

- **Interactive User Feedback:** Allow the agent to ask clarifying questions or for the user to provide feedback on initial results (e.g., “Show me more like this,” “I don’t like this brand”). This would create a more dynamic and conversational experience.
- **Visual Analysis:** Integrate a multi-modal model to analyze product images. This could be used to automatically detect cosmetic defects, verify included accessories, or identify details not mentioned in the text description.
- **Long-Term Monitoring:** Implement a feature where the agent can save a user’s search and monitor Mercari over time, notifying the user when a new listing that matches their criteria and is a good value appears.
- **Improve Market Research:** Improve the market research by searching the similar item review to avoid recommending poor item quality.
- **Async tool calls:** Tools like market research can be run in background to avoid blocking.
- **Use Reflexion:**: We can use Reflexion to give feedback for each action to the LLM to improve the agent’s performance.