Enhancing LLM-based Recommendation Agent for the AgentSociety Challenge

Jake Engelberg
Samueli School of Engineering
University of California, Los Angeles
United States
jengelberg@ucla.edu

Jaelyn Fan
Samueli School of Engineering
University of California, Los Angeles
United States
jfan981@ucla.edu

Adam Thieme
Samueli School of Engineering
University of California, Los Angeles
United States
adamthieme@ucla.edu

Mingqi Zhao
Samueli School of Engineering
University of California, Los Angeles
United States
mizhao34@ucla.edu

Abstract: With the rise of Large Language Models, "Agents" or "Agentic AI" have been proposed as use-cases or wrappers for the LLMs in order to perform specific tasks for the user. AgentSociety Challenge is a competition consisting of either User Modeling or Recommendation tracks, where participants use Yelp, Amazon, and Goodreads datasets to either model user behaviors or predict users' preferences. In this paper, we discuss our attempt at creating a recommendation agent for this challenge. We optimize specifically for the Yelp dataset towards predicting the correct "Top K" business recommendations for a given user, evaluating the model for $K \in 1, 3, 5$. Building on the model given in the AgentSociety Git repository, we improve it with bug fixes, updates in the underlying LLM, context retrieval, prompt engineering, robustness guarantees, and Chain of Thought reasoning. Through our evaluation, we improved the baseline model by 1126%, 279%, and 110% for $K = 1, 3$, and 5, respectively.

1 Introduction

AgentSociety Challenge [1] is a framework for building, simulating, and evaluating realistic multi-agent systems towards an intelligent recommendation or user simulation agent. It was first proposed and used in a competition during ACM's WWW'25 conference to promote the potential of "LLM agents in online review platforms and recommendation systems." [2] Participants can choose from the given Amazon, Goodreads, and Yelp datasets for their models.

In this paper, we chose to focus on building a recommendation agent that adapts to diverse user preferences to optimize user engagement. We also chose the Yelp dataset, as it was easier to obtain and seemed more intuitive.

The Agent is evaluated on a per-task basis, where, for each task, it is given 20 businesses and a user and ranks the businesses based on how likely the user will prefer them. To evaluate our agent, we used Hit Rate at K (HR@K), specifically looking at HR@1, HR@3, and HR@5, which determines how often the "correct" business appears in the top 1, 3, or 5 positions of the list.

The AgentSociety GitHub repository [3] contains an example agent we used as a baseline for our testing. It reads the user profile, their past reviews, and data for each business, before giving a ranked list of candidate IDs. When we ran the full evaluation set on the baseline model, it performed extremely poorly, only getting a 3.1% HR@1, 18% HR@3, and 40% HR@5, which is essentially

unusable, and barely exceeds the performance of an otherwise completely random algorithm for the first two.

To address this, we made a number of sequential improvements to the model and ran experiments along the way to show their effectiveness at improving the agent's recommendation accuracy. The first experiment we ran addressed a series of bugs (specifically in the `item_list` loop) and tweaked the prompting strategy to better utilize user information and get better outputs. The second experiment was transitioning from ChatGPT 3.5 (a non-reasoning model) to GPT-4o-mini (a reasoning model) to evaluate performance between different models. Third, we added a category-based context retrieval strategy, which allowed the model to use category classifications to prioritize businesses that were in similar categories to their past reviews. The last experiment was to implement advanced reasoning and reliability strategies, specifically a chain of thought, a self-correction loop (reflection), robust parsing, and a safety fallback, ensuring the agent never receives a 0.0 for syntax errors and guaranteeing a baseline hit rate.

Along the way, we evaluated these different strategies using 50-task ablation studies to compare the effect of the different modifications, and then ran a full evaluation by comparing three agents: the original baseline, our improved agent using GPT-4o-mini, and the same improved agent using a Deepseek model.

2 Incremental Improvements

To evaluate model performance, we used Hit Rate at K, a common metric used for ranking tasks in a recommendation system. The formula is:

$$\text{HR@K} = \frac{\text{Number of tasks where the "correct" item appears in the Top-K}}{\text{Total number of tasks}}.$$

The baseline model performs the following steps:

1. **Data Gathering:** The process fetches user information, followed by item information by going through all candidate item IDs and extracting relevant features, and then fetching the review history of the user.
2. **Prompt construction:** All gathered data is assembled into a comprehensive prompt. This instructs the LLM to rank 20 candidates to match the user's preference.
3. **LLM Reasoning:** The finished prompt is passed to the LLM, which processes the information and outputs a ranked list of the candidates.

However, as mentioned above, there were a handful of issues with the baseline model that resulted in the poor performance:

1. **Data Aggregation Bug:** The `item_list` variable was overwritten in each iteration rather than appended to. Consequently, the agent only received information for the single last item in the candidate list, causing it to hallucinate the other 19 rankings.
2. **Unused Features:** Although `user_info` (tenure, average rating) and `candidate_category` were fetched from the database, they were never injected into the final prompt, depriving the model of essential context.
3. **Weak Prompting:** The instructions lacked structure, output constraints, or examples, leading to inconsistent formatting.

2.1 Critical Bug Fixes & Prompt Engineering

Our first optimization phase focused on architectural repair. We corrected the aggregation logic to initialize an empty list outside the loop and append each dictionary inside, ensuring all 20 candidates were visible to the model. We also modified the prompt construction to explicitly include the user's metadata and the specific target category (e.g., "Rank these *Japanese* restaurants").

Simultaneously, we implemented structured prompt engineering. We organized the input using Markdown headers (e.g., `### User History`, `### Candidate Details`) to clearly delineate data sections. To enforce parsing reliability, we defined a strict Python list output format and provided few-shot examples illustrating correct versus incorrect outputs, such as warning against including business names alongside IDs.

2.2 Model Selection

We observed that the baseline model, `gpt-3.5-turbo`, frequently failed to adhere to negative constraints, often outputting conversational filler ("Here is your list...") or hallucinating duplicate IDs. To address this, we upgraded the underlying LLM to a lighter-weight reasoning model, `gpt-4o-mini`. This model was selected as the optimal balance between performance and resource efficiency; it demonstrated significantly stronger instruction-following capabilities compared to GPT-3.5 while remaining cost-effective for high-volume iterative testing compared to GPT-4o.

2.3 Context Retrieval

To improve ranking precision, we implemented a Domain-Specific Context Retrieval (CR) system. The objective was to filter the user's massive review history to prioritize experiences relevant to the current task (e.g., prioritizing past "Sushi" reviews when ranking a Japanese restaurant).

We developed a keyword mapping system covering four primary domains:

- **Shopping:** 30+ keywords (e.g., outlet, boutique, hardware, clothing, shoes, books, etc.).
- **Beverages & Bars:** 15+ keywords (e.g., pub, brewery, boba, tea, cafe, etc.).
- **Beauty & Wellness:** 20+ keywords (e.g., spa, barber, waxing, yoga, massage, gym, etc.).
- **Food & Dining:** 40+ keywords (e.g., bistro, grill, sushi, pizza, burger, Thai, Mexican, etc.).

Iteration 1 (Strict Filtering): Initially, we employed a strict filter that only passed reviews matching the target category keywords to the LLM. While this maximized context relevance, it introduced a failure mode: if the keyword mapping failed to categorize a review correctly (e.g., a "Gastropub" not matching "Food"), the model received zero context (an empty history). This resulted in a high Top-1 precision but a significant drop in Top-5 recall as the model lost general user signals.

Iteration 2 (Soft Filtering): To address this trade-off, we shifted to a "Soft Filtering" strategy. Instead of discarding irrelevant reviews, we sorted the user's history to place relevant reviews at the top, followed by the most recent reviews from other categories as backup. This ensures the model receives high-signal data without losing general user preference context, effectively solving the recall degradation.

2.4 Advanced Reasoning & Robustness

In our final iteration, we incorporated advanced cognitive architectures and engineering safety nets.

Chain of Thought (CoT): We modified the system prompt to require an "Analysis" section before the final ranking. By forcing the model to generate reasoning tokens—explicitly comparing the user's profile against item attributes—we reduced logical inconsistencies where the model would acknowledge a preference but fail to apply it to the ranking.

Reflection (Self-Correction Loop): We implemented a programmatic feedback loop with `max_retries=2`. If the model's output fails validation (e.g., invalid syntax, missing brackets, or hallucinated IDs not present in the candidate list), the system captures the specific error message and re-prompts the model: "*Error: [Exception]. Please output a valid Python list.*" This allows the model to self-correct formatting errors that would otherwise result in a low score.

Engineering Safety Nets: To further ensure reliability, we added robust parsing using Regex to extract lists even if they are wrapped in Markdown code blocks. As a final fail-safe, if all retries fail,

the system returns the original candidate list instead of an empty list, guaranteeing a baseline hit rate is always achieved.

3 Results

We evaluated our agent using a 50-task ablation set to see the impact of each incremental improvement, and a 384-task full evaluation was used for the final performance. The evaluation determined the Hit Rate at K for 1, 3, and 5, determining how often, for each experiment, the correct business is in the top-K.

3.1 50-Task Model Evaluation – OpenAI API

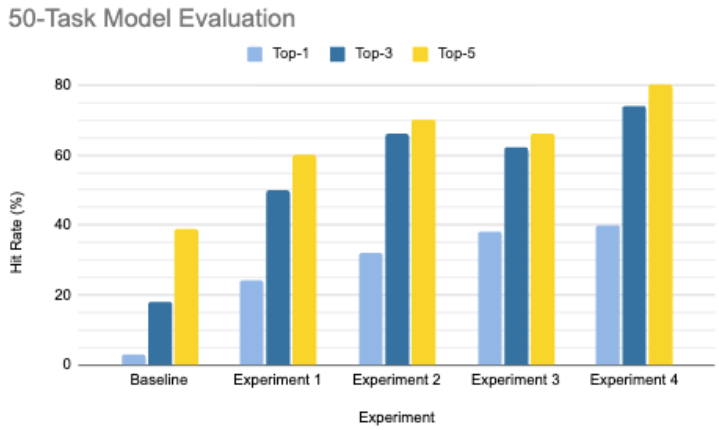

Figure 1: 50-Task Ablation Evaluation.

Table 1: 50-task model evaluation (OpenAI API).

Top-K	Baseline	Exp 1	Exp 2	Exp 3	Exp 4
Top-1	3%	24%	32%	38%	40%
Top-3	18%	50%	66%	62%	74%
Top-5	39%	60%	70%	66%	80%

Exp 1 The baseline agent performed incredibly poorly, having a 3% HR@1, 18% HR@3, and 39% HR@5. After fixing the major errors and implementing structure to the prompts, performance improved dramatically. HR@1 increases to 24%, HR@3 increases to 50% and HR@5 increases to 60%.

Exp 2 Upgrading from GPT-3.5 to GPT-4o-mini also improved our results as it was better at instruction following and avoided some of the formatting mistakes that GPT-3.5 made. There was an increase for all hit rates of K (33%, 32%, and 17% for HR- 1, 3, and 5).

Exp 3 Implementing context retrieval, specifically around the business category, resulted in some interesting results. By selecting the reviews that are relevant to the current task's target category, we had a 19% increase in Top-1 hit rate, but saw 6% decreases in both the Top-3 and Top-5 hit rates. This likely means that the model became more confident in its top prediction, but less successful at ranking the alternate options.

Exp 4 Focusing the model to generate reasoning tokens (chain of thought), implementing a self-correction loop to catch model hallucinations, and adding different safety nets greatly increased Top-3 (by 19%) and Top-5 (by 21%) performance (making the model more successful at identifying the right set of candidates). This made up for the decreases in performance of the Top-3 and Top-5 during **Exp 3**. It also caused a slight increase of 5% for the Top-1 hit rate.

3.2 Full 384-Task Evaluation

Once we were confident in our results from the ablation testing, we compared the baseline and our final agent on the full set of 384 tasks.

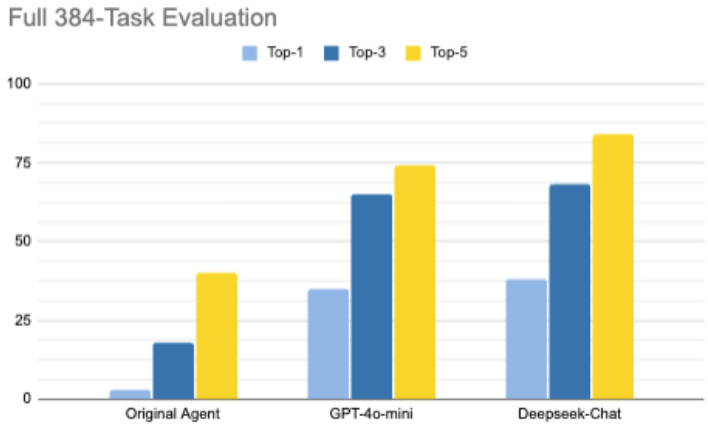

Figure 2: 384-Task Evaluation.

Table 2: Full 384-task evaluation.

Top-K	Original	4o-mini	Deepseek
Top-1	3.1%	35.42%	38.02%
Top-3	18%	65.26%	68.23%
Top-5	40%	74.48%	84.11%

The original agent performed almost identically to the ablation set. Interestingly, while our updated model still far outperformed the original agent, with 1/3/5 hit rates of 35%, 65%, and 74%, respectively, these results were lower than the ablation set. This could simply be a result of more variance for the 50-task set, making the 384-task evaluation a more accurate picture of the agent's performance. We also tested the Deepseek model using the exact same structure and retrieval strategies. Deepseek outperformed GPT-4o-mini across all hit rates at K, resulting in 38% HR@1, 68% HR@3, and 84% HR@5.

4 Experiment Analysis and Lessons Learned

During our testing phase, we discovered a handful of key insights that influenced our improvements.

1. Architectural correctness precedes algorithmic complexity. The bug fixes provided the largest gains. The baseline agent had a critical bug in the retrieval loop, which prevented the model from seeing all 20 businesses. After we corrected the bugs and introduced the structured prompts with the necessary context to the LLM, the model's Top-1 accuracy increased by approximately 7x. The primary takeaway is that before you even begin to implement learning techniques, you must get the agent working properly and have a clear set of instructions for it to follow.

2. Model quality drives baseline performance. Even when everything else is the same, the model quality can shift the recommendation agent's performance.

Upgrading from GPT-3.5 to GPT-4o-mini significantly improved Top-K accuracy. Even with the same prompts and identical architecture, GPT-4o-mini was better for all values of K. This is because GPT-4o-mini is better at following instructions and produces more consistent outputs, reducing the number of unformatted lists or hallucinations. We believe that this has a lot to do with the "reasoning" nature of GPT-4o in general compared to the non-reasoning GPT-3.5 In other words, the updated model may be implementing many of the improvements we made during **Exp 3** and **Exp 4**.

3. Context Retrieval trade-off and Soft Filtering. Context retrieval introduced a trade-off. By looking through a user's history to identify reviews that were relevant to the target candidate's category, the agent was more effective at identifying the number one business. However, this actually lowered the Top-3 and Top-5 accuracy since by focusing on the "ideal" category it limits the amount of information that the model sees. This underfits the data and results in a model that is not able to make broad recommendations.

4. Simplicity reduces cognitive load. Simple, well-structured prompts worked better than over-engineered formats.

The LLMs were more effective when they had a clear structure but did not overly describe the process. Giving too much detail actually may have hurt the model, and explicit review labeling resulted in worse performance. LLMs can infer patterns without over-engineering, and sometimes it can overload the model, as it has to waste focus deciphering the instructions, so it is not able to concentrate on the ranking task it was assigned. Even though it is important to give clear instructions, you need to give guidance on how to work, not micromanage.

5. Reliability mechanisms and Hybrid Memory. Adding chain-of-thought, a reflection step, and safety nets for messy outputs resulted in improved Top-K results for Top-1, Top-3, and Top-5 on the 50-task set. The chain of thought forced the model to explain its reasoning before generating a ranking, and the reflection step allowed the agent to fix formatting mistakes before they entered the model. The safety nets (regex extraction and safety fallback) help eliminate 0.0 scores for syntax errors. This allows the hit rate scores to reach their highest levels, with big boosts to Top-3 and Top-5, meaning that it allows the model to be more successful at picking an ideal set of candidate businesses, even if it does not significantly impact its ability to pick the top choice.

6. Statistical rigor is required. Results on the 50-task ablation set potentially overstated how effective some of the experiments were, likely signaling that a more thorough evaluation should be carried out. With only 50 tasks, each correct or incorrect prediction shifts HR@1 by two percentage points. In addition, the variance between tasks themselves is higher with 50, since outliers would have a larger effect. Therefore, a smaller sample size results in more variance in the results.

The full evaluation gives a more stable and accurate estimate of the performance while taking more resources and time to run. This makes the ablation set valuable for identifying trends in experiment success, while leaving the more accurate evaluation to the full set.

7. Model-internal biases persist. Finally, even with an identical pipeline, the Deepseek model outperformed GPT-4o-mini on the full 384-task evaluation, especially on Top-5 accuracy. This could mean that within each model, there are certain biases present, patterns in the training data, or other ranking strategies that are different between the two LLMs. Since both agents have the same inputs, prompts, and architecture, the difference in results likely comes from something inside the LLMs. Therefore, it is important to be consistent about the model that you are using, so that you do not accidentally blend results from multiple LLMs (ChatGPT, Google Gemini, Claude.ai, etc.).

5 Limitations and Future Work

Despite our model significantly improving in performance as we implemented our different experiments, there were still some limitations that prevented us from improving it even more. The first is API cost. Since we need to use LLMs to make recommendations based on hundreds of experiments, it quickly becomes expensive for both money and time. This meant that we could only run our tests on smaller ablation sets (which resulted in more variance). Additionally, we primarily used Hit Rate@K to evaluate, which is an effective metric to see the precision/accuracy of the first few selections, but does not inform about the entire list, and there are other aspects of a recommendation that it does not reveal, like the confidence in the recommendation. Adding additional metrics to better understand the agent could help us decide which adjustments to make to make it provide the strongest recommendations possible.

For future work to continue to improve our recommendation agent, we could test it on additional datasets. We currently just use the Yelp business reviews, but we could test it on the Amazon and Goodreads datasets, which are the other datasets in the AgentSociety Challenge framework. Additionally, we could evaluate our model using other metrics, including Recall@K, Precision@K, F1@K, Mean Average Precision@K (MAP@K), Mean Reciprocal Rank (MRR), or Normalized Discounted Cumulative Gain (NDCG), all common metrics that can be used to evaluate recommendation systems. Finally, we would continue to look at alternative retrieval or decision-making strategies, such as advanced planning strategies. Also, given that we received slightly different results using Deepseek vs. OpenAI, we could continue to test with other LLM APIs to determine how model biases can impact our recommendation results.

6 Conclusion

For this project, we were tasked with creating a recommendation agent for the AgentSociety Challenge focused on a provided Yelp Reviews dataset. We had to rank a list of 20 business candidates for a user based on their Yelp review history. We evaluated the model using Hit Rate at K to see how often the "correct" businesses were included in the top K spots in the ranking. This metric was used since we valued top-end accuracy, getting the best list for the initial K slots that we could.

The baseline agent performed quite poorly (3.1% HR@1), so we identified and corrected a critical bug in the retrieval loop and adjusted the prompts so that there was a clear structure. This actually resulted in the largest jump of any design change that we implemented. Then we compared different LLM models, evaluating how the results change when we test on an advanced reasoning model. Finally, we added a context retrieval strategy using the review categories, and then an advanced reasoning strategy that added chain-of-thought, a reflection step, and safety nets to improve flawed prompts/outputs. Finally, we tested on different LLMs to determine if there were differences in the models that impacted our agent's performance.

These experiments showed significant improvement over the baseline model. On the ablation set of 50 tasks, performance increased from 3% Top-1, 18% Top-3, and 39% Top-5 all the way to 40% Top-1 (+37pts), 74% Top-3 (+56pts), and 80% Top-5 (+41pts). On the full task set (384 tasks), our updated GPT-4o-mini agent reached 35% Top-1, 65% Top-3, and 74% Top-5, while the Deepseek version reached 38.0%, 68%, and 84%, respectively.

Our results show that each of the four experiments we implemented provided a positive impact on our model, and revealed the following takeaways:

- Structured prompts and clean code are crucial, as they provide the largest performance boost of any experiment.
- Model choice is critical. Stronger models are more likely to follow instructions and less likely to hallucinate.
- Context retrieval resulted in a trade-off, increasing Top-1, making the model more precise at predicting the top choice, but decreasing Top-3 and Top-5 hit rates, meaning it actually decreased overall agent accuracy.
- Adding the advanced reasoning and robustness strategies helped prevent errors or poorly structured prompts, simplifying the process for the agent, which was then able to recommend more accurately.
- Different LLMs will result in different outcomes, even when provided with identical architecture and prompts, likely due to inherent biases and differences in training data.

These takeaways will help future work continue to build stronger, more accurate recommendation systems across a variety of domains and datasets.

References

[1] Y. Yan, Y. Shang, Q. Zeng, Y. Li, K. Zhao, Z. Zheng, X. Ning, T. Wu, S. Yan, Y. Wang, et al. Agentsociety challenge: Designing llm agents for user modeling and recommendation on web platforms. *arXiv preprint arXiv:2502.18754*, 2025.

[2] Y. Yan et al. Agentsociety challenge workshop. https://tsinghua-fib-lab.github.io/AgentSocietyChallenge/workshop/index.html, 2025.

[3] Y. Yan et al. Agentsociety challenge repository. https://github.com/tsinghua-fib-lab/AgentSocietyChallenge, 2025.