AgentSociety Challenge
Building an Intelligent Recommendation Agent

Eliot Yoon [*1] **Ryan Wang** [*1]

Abstract

Large Language Model Agents have taken the world by storm for their ability to perform complex tasks by using natural language to plan and reason towards an intelligent solution. Especially when it comes to reasoning with language, LLMs excel at extracting the semantic value and drawing conclusions from the presented textual data. One area that stands to benefit is the task of recommendations based on textual user preference data. In this paper, we present our approach which employs a combination of feature engineering, task decomposition, reasoning, ensemble voting, and memory to make intelligent recommendations. These architectural choices allow our model to achieve an average hit rate of 56.94% in the AgentSociety Challenge Recommendation Track benchmark dataset (Yan et al., 2025), a 21% gain over the baseline model.

1. Introduction

In 2025, Yan et al. created the AgentSociety Challenge to explore the potential of Large Language Model (LLM) agents in modeling user behavior and enhancing recommender systems on web platforms (Yan et al., 2025). They posited that with LLMs demonstrating exceptional performance in reasoning and prediction tasks, extended by agents which further augment the capability of LLMs by allowing them to effectively simulate human reasoning and behavior in solving complex tasks, this could produce major improvements in how user behavior is modeled, simulated, and predicted. Although they produced a robust simulator for allowing LLMs to interact with web platforms, their insights into designing top agents for recommendation leave much to be explored on improving the agentic workflow architecture.

In this project we will be exploring how LLM agents can be used to generate personalized recommendations for users. By leveraging the latest insights in LLM agent research, we explore how changes in the workflow such as leveraging different blocks and feature engineering can improve recommendation performance.

1.1. Dataset

The primary dataset explored in this project is the Yelp dataset (Asghar, 2016). The dataset is split into 3 tables: user data, item (or business) data, and review data. The user data contains all the metadata about a user profile including but not limited to name, review count, and account creation date. The item data contains metadata about businesses, examples being business name, average stars, hours, etc. Review data contains all the data about a user review of a business. This includes the user's star rating of the business, the body text of the review, and other metadata about the review. It is worth noting that each review is connected to a user who created the review and a business that is the subject of the review. Being able to leverage the rich textual data of reviews combined with the numerical metadata will be key to this challenge.

1.2. Evaluation

To evaluate the performance of our agent, we will be using the simulator and evaluator provided by the AgentSociety Challenge Recommendation Track (Yan et al., 2025). The task presented to the LLM agent is that given a user and 20 candidate items, predict the preference order the user would assign to these candidate items from most preferred to least preferred. The metric we will be using to measure how successful our agent is at predicting preference order is HR@K. The formula for HR@K is as follows:

$$HR@K = \frac{1}{T}\sum_{t=1}^{T}\mathbb{I}(p_t \in P_t(K))$$

where T is the number of tasks, p_t is the ground truth item, and $P_t(K)$ is the set of the first K recommended items. We will be evaluating HR@K, where $K = 1, 3, 5$ and the

[*]Equal contribution [1]Department of Computer Science, University of California, Los Angeles, California, United States. Correspondence to: Ryan Wang <ryanwang25@g.ucla.edu>, Eliot Yoon <eyoon1131@cs.ucla.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

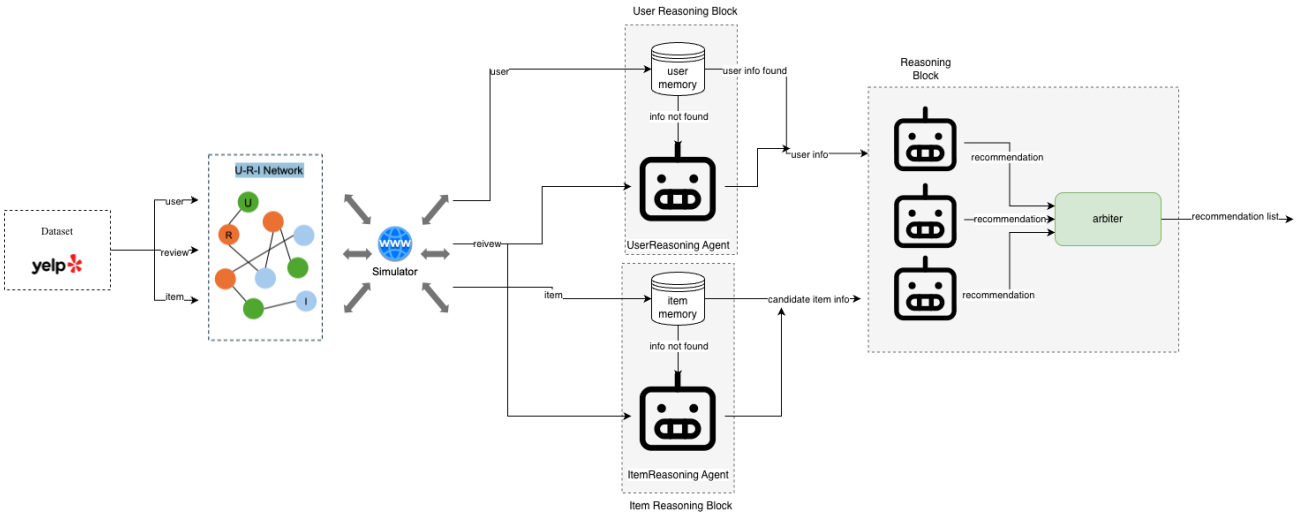

Figure 1. **Final Model Overview.** For each task, the simulator provides a user and a list of 20 candidate items along with info on all available users, items, and reviews. Based on a predetermined plan, our model first takes the User and goes to the user information block, which first checks the user memory store to see if we have info on the user. If there is already info on the user, the user memory store directly fetches that information. If not, we use our user reasoning agent to provide an overview of user preferences and store it for future use. Similarly, for each candidate item, we use our item information block, which utilizes a similar strategy of checking item memory first and referring to the item summary agent otherwise. Finally, we hand this information to the reasoning block, which utilizes 3 LLMs to make a recommendation decision based on the provided info. These 3 recommendations are then put through an arbitrator algorithm, which decides on the final recommendation list.

average hit rate is the average of these 3 metrics.

2. Method / System Design

Our model employs an LLM backbone, which we leverage as part of a greater agentic system. We chose to use Google Gemini-2.5-flash (Comanici et al., 2025) as our LLM of choice because of its cost-to-performance as well as Google Cloud Platform's offering of $300 in credits for new accounts. An overview of our model is provided in Figure 1.

2.1. Subtask Decomposition

Our first contribution concerns enriching the context and information available to the LLM through subtask decomposition and reasoning. In our pipeline, we have 3 main reasoning subtasks: User Reasoning, Item Reasoning, and Task Reasoning. This differs from the baseline, in which there is only one reasoning call to the LLM, which simply takes in all of the raw data and a simplistic prompt. We realized that dumping all of the reasoning responsibility in one API call to the LLM is detrimental to its accuracy; thus, we break the reasoning process down into concrete and distinct subtasks.

The first call to the LLM is in our User Reasoning module. This module simply takes in all of the raw user data and review history and is given one distinct goal: to provide a detailed, structured summary of the user's information, tastes, and sentiments that can be used to provide greater user context to the final ranking task. One of the key insights is that despite the abundance of user reviews, they are normally used no further than their numerical star ratings, which leaves a lot of rich semantic information unused. Prior research shows that textual feedback within reviews carry descriptive signals that quantitative features alone do not capture and that leveraging them properly improves recommender performance (Zhang et al., 2021). This is the main goal of this module: to provide greater context learned from each user's review history in a more compact and digestible format.

The second subtask is the Item Reasoning module. This module is very similar conceptually to the User Reasoning module: it is given the raw data of each candidate business and outputs a detailed summary of its attributes, categories, and review history. We found in our analysis of the baseline that only the raw item data for each candidate is passed to the LLM. This metadata mainly contains quantitative features such as star ratings and review counts. This is not incredibly useful to the LLM as it excludes qualitative data such as the general sentiment surrounding each business. Thus, we also pass in a subset of each candidate's reviews in this module. This allows the LLM to summarize the aggregated reviews for each item into concise natural language descriptors capturing qualitative aspects of atmosphere, service quality, or user sentiment. Such summaries carry additional semantic signals not expressed by the original structured attributes

and let the model form a more holistic understanding of each listing. The downside to this module is the performance impact: each candidate needs their own separate LLM call, which adds an overhead of 20 additional LLM calls per task. Thus, we implement system memory to cache these summaries, which we will discuss in detail shortly.

The final subtask is the Task Reasoning module, which is highly similar to the baseline. It simply passes the raw data, as well as our generated summaries from the previous sections, to the LLM with a prompt that gives instructions on how to rank these items. In the next section, we will discuss how we engineer this prompt to provide greater context and stricter ranking instructions to improve performance.

2.2. Feature/Prompt Engineering

Although we see an improvement in performance from the subtask modules, it frequently leads to issues with malformed LLM responses. These malformed responses are caused by exceeding the token budget of each LLM call, which result in either nonsense or nonexistent responses. Thus, our next goal is to greatly reduce the input tokens of our prompts by removing extraneous metadata.

Our insight is that LLMs can reason more effectively when the context is given in natural language rather than as raw structured data. Previous works support this notion that converting structured features into natural language enhances LLM decision making on downstream tasks (Brown et al., 2020; Zhou et al., 2023). As mentioned previously, we achieve this by first converting metadata into readable text summaries. Previously, we simply passed in the raw JSON data, which contains many irrelevant fields. Thus, we introduce filters to select the key attributes to pass to the LLM, maximizing signal density. The baseline also excludes important information, such as the business's categories field and information about the types of businesses that the user has reviewed in their history. We include this data to improve the context given to the LLM. This transformation step condenses the available information into a format that ultimately improves downstream recommendation quality and significantly reduces the input prompt length.

Additionally, we find that the specific instructions given to the LLM in the final ranking prompt are very important. The baseline simply tells the LLM to rank the items according to its preference. This does not give any direction or structure, leading to highly variable output. In our experiments, we tested variations of the prompt instructions, differing in the specific ranking instructions given to the LLM. We find that telling the LLM to specifically prioritize the general sentiment and ratings of each candidate over the specific tastes of the user leads to the highest performance.

2.3. Memory

Another feature that our model leverages is the memory module, which provides a way for us to bypass limitations set by the context window of the LLM while retaining long-range contextual information. Besides extending the system's reasoning horizon, memory also plays a dual role in caching results computed previously to enhance efficiency and reduce cost otherwise wasted through redundant LLM calls. Concretely, we use memory to store representations computed from user reviews as well as item summaries generated during the feature extraction stage, so these can be retrieved rather than regenerated when the same entities appear.

A key insight is that candidate items often repeat across evaluation batches in the AgentSociety benchmark; therefore, summarization would be wasteful to repeat. To this end, we utilize a persistent vector memory store implemented with langchain chroma: we index every item summary, sentiment representation, and feature vector for later retrieval. When an item is seen again, the system first checks memory; if there is already an embedding, the cached representation is loaded directly, bypassing the LLM inference pipeline. This reduces both time and API cost considerably during experimentation and allows for fast iteration across prompt ablations and architecture variants. We apply this strategy equivalently in the case of user preference representations by storing user profile embeddings derived from historical reviews, allowing user preference extraction to be computed once and reused throughout recommendation queries.

Our approach is thus in line with the increasing set of works in the realm of memory-augmented language models. In fact, previous work illustrates that external memory increases context retention and brings long-form reasoning to scale, well beyond fixed context windows (Wang et al., 2023). MemLong further demonstrates that dynamic memory retrieval reduces recomputation, making it more efficient for long-text generation (Liu et al., 2024). More recently, memory-centric architectures have emerged as the fulcrum of persistent and evolving agent behavior (Chhikara et al., 2025). By integrating memory within our recommendation pipeline, not only does our system become more scalable, but it also gains knowledge over time by accumulating structured user-item representations.

2.4. Ensembling

The final feature we implement in our best-performing agent is ensemble voting. One of the primary limitations of the baseline is that each LLM call returns a single ranking. This results in high variance of rankings during evaluation. Due to the nature of the dataset, there are often candidates that share very similar qualities, such as rating and sentiment. This makes it difficult for the LLM to decide the best can-

didate, ultimately coming down to a coin flip decision with weak reasoning. Additionally, occasional hallucinations in the LLM output can also lead to poor results.

Thus, ensemble voting is the obvious solution to this issue. We obtain multiple ranking outputs for each LLM call and use a ranked voting algorithm to decide the final ranking. We also significantly increase the temperature of the LLM call, so that the LLM utilizes slightly different ranking approaches for each output. This collective decision-making reduces variance and allows the agent to have more reproducible, deterministic results. Through experimentation, we find that the ideal ensemble size is $N = 3$.

Table 1. Comparative Study on key system components and engineered improvement measured in HR@K. Best results are bolded. Tested on complete dataset of $T = 384$ tasks.

Model Variant	Top 1	Top 3	Top 5	Average
Baseline (Fixed)	0.2604	0.5078	0.6458	0.4714
Base + **U**ser **R**easoning	0.2813	0.5391	0.6901	0.5035
Base + **I**tem **R**easoning	0.2656	0.5651	0.6953	0.5087
Base + **E**ngineered **P**rompt	0.2552	0.5286	0.7656	0.5165
Base + UR + IR + EP	0.3073	0.5964	0.7422	0.5486
Base + UR + IR + EP + Ensemble (Final)	**0.3099**	**0.6198**	**0.7786**	**0.5694**

3. Experiment Results

Table 1 summarizes the performance of the baseline compared to different iterations of our model tested on the full dataset of $T = 384$ tasks. One point to note is that this table confirms that each feature outlined in the previous section improves upon the baseline, even in isolation. Furthermore, our best model, which incorporates all the features outlined in the previous section outperforms the baseline in every metric with a relative improvement to the average hit rate of 21%, highlighting the dominance of our model in producing accurate recommendations over the baseline.

These results highlight the descriptive signal latent in the user review data and the item review data, as underscored by the 7% and 8% relative improvement over the baseline yielded by user reasoning and item reasoning, respectively.

They also show the combined performance improvement obtained from combining user reasoning, item reasoning, and prompt engineering, which yields an average hit rate 16% greater than the baseline.

These results also confirm that ensembling of multiple models does indeed produce a more robust output, as highlighted by the 0.5486 average hit rate yielded by the second-to-last row, which is the culmination of the previous features without ensembling, versus the 0.5694 average hit rate yielded by including emsembling, a 4% relative improvement.

Table 2. Evaluation Metrics of various prompt instructions. Tested on subset of $T = 100$ tasks.

Prompt Instructions	Top 1	Top 3	Top 5	Average
Baseline	0.24	0.56	0.7	0.5
User Preference > Item Rating	0.24	0.49	0.66	0.463
Item Rating > User Preference	0.29	0.68	0.78	0.583

Table 3. Evaluation Metrics of different ensemble sizes. Tested on subset of $T = 100$ tasks.

Ensemble Size	Top 1	Top 3	Top 5	Average
No Ensemble	0.3	0.7	0.82	0.607
$N = 3$	0.37	0.71	0.78	0.62
$N = 5$	0.34	0.71	0.76	0.603

4. Experiment Analysis

Throughout the process of our experimentation, we tested various combinations of features and different parameters for each feature, in order to ensure that our final model combines the highest performing variant of these features. The ablation analysis is described below.

4.1. Ablation Analysis

During prompt engineering, we experimented with various styles of instructions to the recommendation LLM. Table 2 shows the evaluation metrics across different prompt instructions, evaluated on a subset of the data with $T = 100$ tasks. A smaller subset was used to reduce experiment costs during testing. The subset results in metrics that are inflated compared to the metrics obtained from testing on the entire dataset; as these tests were done for ablation analysis, this is acceptable. The baseline simply uses the prompt that was given in the baseline. This prompt has no specific ranking instructions, simply leaving it up to the LLM. This results in an average hit rate of 0.5. The next variation is where the LLM is specifically instructed to value the user's preferences and tastes higher than the ratings and general sentiments of the candidates. With this instruction, the LLM will value a low-rated business that aligns with the user's preferences higher than a high-rated business that does not align with their preferences. This results in a significant performance degradation, with an average hit rate of 0.463. The last variant is where the candidate's ratings are prioritized over the user's preferences. This leads to the highest average hit rate by far, 0.583. This result shows that when it comes to recommendation systems, the overall quality of the business is much more important than whether that business specifically aligns with the user's interests, and this instruction is used in the final model.

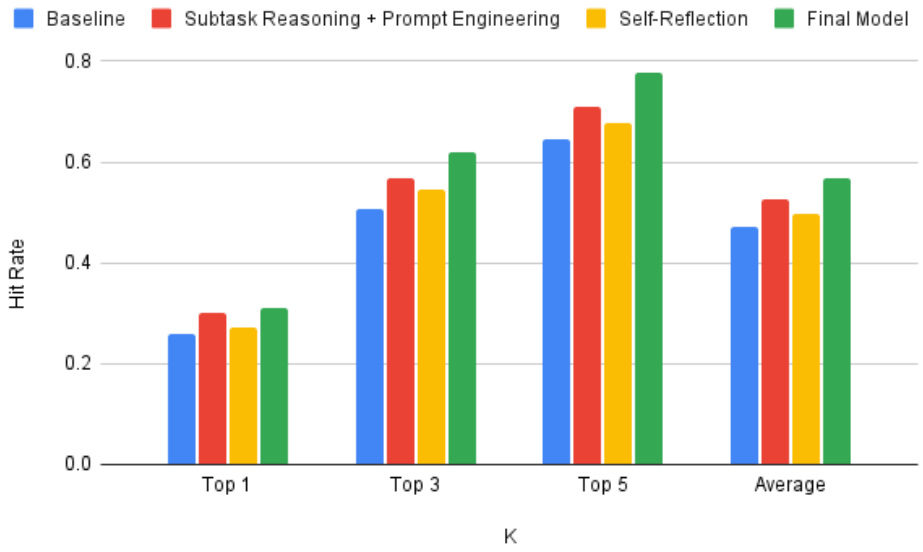

Figure 2. **HR@K for Various K over Different Model Iterations.** This graph shows the improvement of model performance over the stages of development. The major stages of development are broken down into Subtask Reasoning + Prompt Engineering (SR + PE), Self-Reflection, and Ensembling. The final model in green represents the combination of these features, with the exception of Self-Reflection. The significant performance degradation caused by adding Self-Reflection to the SR + PE model illustrates why we abandon this feature in the final model.

Another comparative study we performed was that of the ensemble size. Table 3 shows the evaluation metrics across ensemble sizes of $N = 1, 3, 5$, tested on the subset of $T = 100$ tasks. The results are not nearly as definitive as the prompt instructions, but still provide insight. An ensemble size of 3 outperforms both other options in average hit rate and also has the highest top-1 hit rate of 0.62. The larger ensemble size of 5 does not show any performance improvement and also increases output token size, thus it is not used in the final model. Having no ensemble yields slightly worse but similar results to $N = 3$, and even outperforms it in top-5 hit rate. The significantly lower top-1 hit rate of only 0.3 versus 0.37, however, as well as the higher variance in results due to the lack of voting, leads us to choose the ensemble size of 3 in the final model.

5. Limitations and Future Work

Throughout the development process, we explored some avenues of experimentation that did not end up in the final model due to poor performance. We initially planned on implementing a self-reflection strategy, where the agent is able to learn from previous tasks and adjust its approach, but abandoned this feature because it degraded the agent's performance when it was added to the subtask reasoning and prompt engineering model. This degradation is seen in Figure 2. We also plan on implementing further features in the future for further performance gains. These features are described further below.

5.1. Self-Reflection

One of the primary shortcomings of our model is its inability to learn from its previous mistakes. Before exploring ensembling, we attempted to create a self-reflection module on top of the already existing subtask reasoning and prompt engineering features. In the initial implementation, the output of the Task Reasoning module is augmented to include a confidence score from $0-1$. This score represents the LLM's confidence in the correctness of its ranking. If there were multiple strong candidates, the confidence score is lower. There is also a separate memory module in addition to the existing user and item memory modules. This memory module stores all completed task prompts and their outputs, with the prompts used as the keys. When a new task prompt enters the Task Reasoning module, it is passed to the self-reflection memory module, where the most semantically similar completed task is retrieved. If this task beats a similarity threshold, it is used to guide the reasoning process of the new task. The retrieved task is given to the LLM, along with its output. The LLM is instructed to analyze and modify the reasoning process according to its confidence score. If the confidence is low, the reasoning process is modified significantly, while it is left mostly unchanged if the confidence is high. This reasoning process is then returned by the LLM and used to guide the ranking process of the new task. This design is intended to allow for greater variation of ranking approaches for tasks that are difficult to solve, preventing the agent from using the same

Table 4. Average hit rate of various voting strategies for self-reflection over varying degrees of confidence generation. Tested on subset of $T = 100$ tasks.

CONFIDENCE GENERATION	NO VOTE	2-WAY VOTE	3-WAY VOTE
BASELINE	0.54	0.54	0.51
GREATER UNCERTAINTY	0.51	0.503	0.507

Table 5. Self-Reflection with varying keys and input/output structures. Tested on subset of $T = 100$ tasks.

SYSTEM MODIFICATION	TOP 1	TOP 3	TOP 5	AVERAGE
FULL TASK	0.25	0.64	0.73	0.54
USER SUMMARY	0.23	0.55	0.74	0.507
FIXED LLM FEEDBACK	0.29	0.60	0.74	0.543
STRUCTURED USER SUMMARIES	0.27	0.62	0.76	0.55

wrong approach every time.

On top of this design is a voting system, in which the original task is given to the ranking LLM along with the modified task with the new ranking approach. Then there is a vote among the output rankings, and the final result is returned by the Task Reasoning module. Table 4 shows the average hit rate over various voting strategies tested on the subset of 100 tasks. The baseline row compares the performance of the system outlined above with no vote (only the modified task is used), 2-way vote between the original and modified task, and 3-way vote where the 2 most similar tasks are retrieved from memory instead of just 1. The results show that all of the strategies reduce average hit rate compared to 0.583, the average hit rate of the best model from Table 2, which self-reflection is built on top of. The second row represents the same tests done on the model, with the only difference being the uncertainty of the confidence scores generated by the LLM. In this row, the LLM is specifically told to be harsher with the confidence scores. In the original, the LLM is very overconfident, with most confidence scores being around $0.8 - 0.9$. In the modified version, the LLM is told that the average top-1 hit rate is approximately $0.2 - 0.3$, which greatly reduces its confidence and leads to scores with a much larger range, from about $0.2 - 0.8$. Despite this resulting in greater variation of task modifications, it does not lead to any performance improvement, as shown in Table 4.

Due to the poor results, we modified the design to use the user summaries as the keys to be compared for semantic similarity instead of the task prompts. The logic is that every task prompt shares segments that are equivalent across all tasks, resulting in high semantic similarity for every prompt. Using user summaries as the keys allows for the self-reflection module to retrieve tasks that are truly similar to the current task, with the task category (Shopping, Dining, etc.) used as a filter. Since users in the same task category with similar profiles likely share preferences, in theory, using these previous results to modify the new task's reasoning instructions should allow the agent to learn and adapt its approach to difficult tasks. Comparisons of these results to the previous approach are shown in Table 5. The Full Task row represents the previous approach, which had an average hit rate of 0.54, while the User Summary row is the new approach, with an average of 0.507. While this seems to show that the new approach performs worse, a bug was found in which the modified reasoning instructions returned by the LLM contained information and instructions specific to the user from the previous task, restricting the LLM's ability to generalize these new instructions to a new user. Thus, the Fixed LLM Feedback row shows the results after the bug was fixed, which are very similar to the original approach. A further improvement was made, in which the User Reasoning module was modified to produce a more structured list, as opposed to the previous summaries, which were simply paragraphs of text. This modification results in a very slight improvement in performance, with an average hit rate of 0.55.

Ultimately, due to the poor results of self-reflection, we decided to abandon the approach and instead explored ensembling, which did lead to better performance. However, we learned important insights from this process, the most significant of which was the structured user summarization, which was integrated into the final model. Reflecting on our exploration of this strategy also reveals many shortcomings. The most significant is the LLM's inability to accurately assess the confidence of its own output. An LLM will almost always be highly confident in its output, due to its poor ability to admit when it's wrong or unsure (Yang & Jia, 2025). This issue can potentially be alleviated by introducing stronger signals of correctness/incorrectness to the agent. However, due to the limitations of the AgentSociety environment, we were unsure if utilizing the ground-truth data during evaluation was permitted, and did not explore this further.

5.2. Future Work

If our agent were deployed publicly, we could use user feedback on the rankings to adjust the model's instructions. We predict that this could lead to significant improvements in model accuracy as more user feedback is collected. In the future, we plan on testing the model on larger portions of the dataset. The AgentSociety environment only uses a small subset of the Yelp dataset, and training the model on more data while also using the ground truth data to modify and adjust the LLM's reasoning instructions could allow us to implement an improved version of self-reflection that

further boosts performance.

Another avenue of future work is to further refine representations of user profiles and item profiles to encapsulate the multimodal data available within the Yelp dataset. We have already demonstrated the semantic value of converting review text into an embedding space that can enhance downstream task performance. Therefore, if we are able to encapsulate both text and numerical features in the same embedding space, it could prove useful to further enhance downstream performance on reasoning and prediction tasks.

6. Conclusion

This work provides insight into the use of advanced LLM Agent-based techniques in the task of building intelligent recommender systems. We introduce a framework that efficiently leverages large amounts of user reviews as well as business reviews into a representation that improves downstream recommendation tasks. In doing so, we posit that as we become able to more accurately leverage textual data efficiently, we will be able to better model user behavior on web platforms, recommendation being a subset of a greater family of user modeling problems. We also demonstrate the value of ensembling in the context of agentic workflows. These results suggest that creating meaningful representations for LLMs is key to agent performance, and further research leveraging different modes of information is promising for further refinement of recommendation agents.

References

Asghar, N. Yelp dataset challenge: Review rating prediction, 2016. URL https://arxiv.org/abs/1605.05362.

Brown, T. B. et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

Chhikara, P. et al. Mem0: Building production-ready ai agents with memory-centric architecture. *arXiv preprint arXiv:2504.19413*, 2025.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Liu, W. et al. Memlong: Memory-augmented retrieval for long text generation. *arXiv preprint arXiv:2408.16967*, 2024.

Wang, W. et al. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 2023.

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

Yang, Y. and Jia, R. When do llms admit their mistakes? understanding the role of model belief in retraction, 2025. URL https://arxiv.org/abs/2505.16170.

Zhang, K., Qian, H., Liu, Q., Zhang, Z., Zhou, J., Ma, J., and Chen, E. Sifn: A sentiment-aware interactive fusion network for review-based item recommendation. In *Proceedings of the 30th ACM International Conference on Information amp; Knowledge Management*, CIKM '21, pp. 3627–3631. ACM, October 2021. doi: 10.1145/3459637.3482181. URL http://dx.doi.org/10.1145/3459637.3482181.

Zhou, Y., Sun, T., et al. We need structured output: Improving llm decision-making for knowledge tasks. In *Proceedings of the 2023 ACL Conference*, 2023.