

HOMEWORK 2

Due date: 03/10/2026 11:59 PM EST

11-766 Spring
2026

In this homework, you will (1) design, implement, and analyze memory and retrieval mechanisms for long-context language model systems; (2) build and extend an LLM-driven autonomous agent, incorporating memory or multi-agent strategies to improve long-horizon decision making; and (3) train and evaluate reward models and preference-based optimization methods (e.g., RLHF and DPO), while (4) critically examining the limitations of reward modeling and LLM-based evaluation.

For Problems 1.1 and 2, we have provided starter code in the form of IPython notebooks. These IPython notebooks are designed to be run in Google Colab and may need modifications to run in local environments. Note that you will need a GPU machine to complete this homework—the TAs were able to complete all questions using a Colab Pro instance. For Problem 1.2-1.3, we have also provided starter code in a zip file. Please download, run, and extend following the instructions in README.md and this homework. If you have any trouble running the starter code or accessing a GPU machine, please post on Piazza or ask for help in office hours.

You must submit:

- Your revised versions of the two IPython notebooks (Problems 1.1 and 2) and your updated `simple_llm_agent.py` file (Problems 1.2-1.3). Though we are asking for your code, we do not plan to directly assess it (except to check for cheating). You may feel free to make any changes that support you in answering the questions in this document.
- Your written report as `report.pdf`, containing answers to the questions in the blue “Deliverables” boxes. When answering questions, please strive to be precise but succinct with your language.

As a reminder, you are permitted to use AI for the homework, so long as (1) all words that appear in your report are written by you unless you are explicitly quoting from an LLM’s output; and (2) you include an “AI Attestation” section in your report that describes how you used AI to help you with the homework.

Problem 1: CMU Food Truck Bench

In this problem, you will explore how agents can construct and query long-term memory...

[Question 1.1] (*Discuss and implement four kinds of memory units*) In this problem, you will explore how agents can construct and query long-term memory. You will implement and compare four retrieval-based memory systems—BM25 [18], dense embeddings, KV cache attention [21], and agentic memory control [15]—and evaluate them on both a synthetic needle-in-a-haystack benchmark and a real long-document QA benchmark (QuALITY [16]). Starter code is provided in the IPython notebook `11766_hw2_p1_1.ipynb`; it is designed to run on a Colab instance with an L4 GPU.

All four methods share the same chunking front-end (recursive character splitting, 200-char chunks with 50-char overlap) and the same top- $k = 3$ retrieval interface, so performance differences are attributable solely to the retrieval mechanism:

1. **BM25** [18]: Sparse lexical retrieval via `rank_bm25`.
2. **Dense retrieval** [5]: Cosine similarity over embeddings from a small encoder. In this question, we use MiniCPM-Embedding-Light [12].
3. **KV cache retrieval** [21]: Memorizing-Transformers-style attention scoring. Pre-RoPE key projections from the last decoder layer of `Qwen2.5-0.5B-Instruct` are stored per chunk; at query time, query projections are extracted and relevance is computed via QK^\top/\sqrt{d} with GQA head expansion.
4. **Agentic memory control** [15]: A sliding-window buffer over chunks scored by the LM itself. `Qwen2.5-0.5B` rates each chunk’s relevance to the query on a 1–10 scale; the top-scoring chunks are returned.

DELIVERABLES FOR Q1.1

Please answer the following questions.

- A. Implement the four retrieval methods above (BM25, dense embedding, KV cache, agentic). Pick one method and describe how you could potentially improve its design. (*TODO 1.1 in notebook*)
- B. Run the needle-in-a-haystack benchmark (5 needles \times 5 depth positions). Compare the performance of the four methods and explain the fundamental reason for any speed differences you observe. (*TODO 1.2a*)
- C. The agentic memory controller uses a 0.5B-parameter model to make relevance decisions. What challenges do you expect with such a small model? Modify the scoring prompt, observe how it affects behavior, and report whether you can improve performance. (*TODO 1.2b*)
- D. Evaluate all four methods on the QuALITY long-document QA benchmark using token-level F1 between the top- k retrieved chunks and the gold answer. For each method, find one example where it works well, present the example, and discuss one advantage and one disadvantage. (*TODO 1.3*)
- E. Consider an autonomous agent operating over a long horizon (e.g., managing a vending machine business over weeks of interactions as in VendingBench^a). The agent must remember past customer interactions, inventory decisions, and market trends. For each of the four memory methods, describe an agent scenario where it would be the most appropriate choice, and identify a failure mode where it would break down. (*TODO 1.4*)

^a<https://arxiv.org/pdf/2502.15840>

[Question 1.2] (*Implementing a Simple LLM Agent for the CMU Food Truck Environment*)

In this problem, you will implement a minimal LLM-driven agent that simulates a food truck owner at CMU to earn profit and glory. We have provided you with a starter codebase in `11766_hw2_p2.zip` containing the modules that define the environment. The food truck consists of a fridge, a pantry, and a menu billboard. Every day begins with a prep stage, where the agent needs to purchase ingredients from a market with daily floating prices, and build a menu by selecting dishes from a recipe book and setting a reasonable price. After a maximum number of actions or by the agent decision, the prep stage ends, the food truck opens for business, and customers will start ordering from your menu. You will fill the orders by cooking with the ingredients you purchased, or

leave the order unfilled if there aren't enough left. At the end of the day, you will pick up the earnings, pay a small rent, and move on to the prep stage of the next day. You can find more details and instructions in `README.md`. To get you started with the environment, we also provide a reference script `simple_llm_agent.py` which contains the skeleton code for a `SimpleLLMAgent` class and a routine for initializing the environment and running the agent until finish.

DELIVERABLES FOR Q1.2

Please answer the following questions and include the required code.

- A. Complete the `SimpleLLMAgent` class by implementing the methods `__init__()` and `act(observation: str) -> str`. Your `act()` method must return exactly one valid command line (no extra prose). You may refer to the codebase for available commands. For the prompt, you must maintain a rolling transcript of interaction, formatted as a sequence of entries containing interleaving observation blocks and action lines. To manage the context length, you should implement truncation logic so that before each LLM call, you keep the transcript within a maximum context budget of 32,000 tokens by removing the oldest transcript entries.

Please report the following:

- Describe the prompt you use to query the model.
 - Explain how you design and format the interaction transcript and why.
 - Explain your truncation method and how it optimizes for ease of processing by the LLM (e.g., avoiding cutting mid-line).
- B. Run your agent for at least 3 different environment seeds (e.g., 0, 1, and 2) for the default number of days (20 in this code base) and report the following:
- What were the final funds at the end of each run?
 - Briefly describe the strategy the agent appears to follow qualitatively.
- C. Provide two examples where your agent makes a poor decision (e.g., buys the wrong ingredient, sets an unprofitable price, fails to re-check information, etc.). For each, explain the likely cause (e.g., prompt design, memory, action format).

[Question 1.3] (*Advanced Agent Systems: Memory and Multi-Agent Collaboration*) Recent work on autonomous agents has explored mechanisms beyond simple context management to enable long-horizon coherent behavior. In this subproblem, you will extend your CMU Food Truck agent by incorporating either **memory augmentation** or **multi-agent collaboration** strategies, drawing inspiration from recent research in these areas.

Background 1: Vending-Bench and Memory Tools. Vending-Bench [1] is a benchmark that tests LLM agents' ability to manage a vending machine business over extended periods (>20M tokens per run). The benchmark provides agents with three types of external memory tools:

1. **Scratchpad:** A basic text storage system for writing and retrieving free-form notes (e.g., daily summaries, plans).
2. **Key-Value Store:** A structured database for storing and accessing data pairs (e.g., `inventory_target: 50`, `best_selling_item: chips`).
3. **Vector Database:** Semantic retrieval using embeddings (e.g., OpenAI's `text-embedding-3-small`) with cosine similarity search.

A key finding from Vending-Bench is that *memory capacity alone is insufficient*—even with access to all three memory tools, top models like Claude 3.5 Sonnet rarely retrieve their own stored notes, suggesting that the challenge lies in **strategic memory usage** rather than storage limits. Counterintuitively, agents with larger context windows (60k tokens) performed worse than those with smaller windows (10k–30k tokens), indicating that more memory does not automatically translate to better long-term coherence.

The benchmark also employs a **hierarchical multi-agent structure**¹ where the primary agent delegates physical-world tasks to sub-agents via tools like `run_sub_agent` (for delegation) and `chat_with_sub_agent`

¹<https://github.com/AndonLabs/multiagent-inspect>

(for querying completed actions). This decomposition mirrors how complex real-world tasks often benefit from specialized sub-agents.

Background 2: Multi-Agent Collaboration and Debate. Multi-agent collaboration [8] has emerged as a promising paradigm for improving LLM performance. Du et al. [2] demonstrate that having multiple LLM instances propose and debate their responses over multiple rounds significantly improves mathematical reasoning and reduces factual hallucinations. Liang et al. [9] extend this with structured “tit-for-tat” debate frameworks managed by a judge agent. For a comprehensive survey on LLM-based multi-agent systems, see [3, 6].

Background 3: Agent Memory Systems. Beyond simple retrieval methods we introduced in Question 1.1, recent work has explored more sophisticated memory architectures for LLM agents. MemGPT [15] treats the LLM as an operating system kernel that manages its own memory hierarchy, with explicit operations for moving information between working memory and long-term storage. A-MEM [22] proposes agentic memory that dynamically organizes memories following the Zettelkasten method, creating interconnected knowledge networks through dynamic indexing and linking. For an overview of memory mechanisms in LLM agents, see Liu et al. [11].

DELIVERABLES FOR Q1.3

Please answer the following questions.

A. Write a detailed plan (2–3 paragraphs) for improving your agent’s performance using **either** (a) a memory system (e.g., scratchpad, key-value store, vector database, or anything else you think of) **or** (b) a multi-agent framework (e.g., debate, delegation, specialized sub-agents, or anything else you think of). Your plan should include:

- A clear description of the architecture and how it integrates with your existing agent. Including references is recommended.
- Justification for why this approach should help with the specific challenges you observed in Question 1.2 (e.g., forgetting past decisions, inefficient planning, repeated mistakes).
- Implementation details: what information will be stored/shared, when will memory be written/read or when will agents communicate, and how will retrieved information or agent consensus be incorporated into actions.

Hint: You are free to incorporate the memory modules from Question 1.1.

B. Implement your plan and evaluate on the same seeds used in Question 1.2 (at least seeds 0, 1, 2). Please report the following:

- Final funds for each run, compared to the baseline from Question 1.2.
- Any additional metrics relevant to your approach (e.g., memory utilization, number of agent communications, retrieval accuracy).
- A qualitative description of how the agent’s behavior differs from the baseline.

C. Please provide a comparative analysis of your improved system versus the baseline including the following:

- One specific scenario (with transcript excerpts) where your memory/multi-agent system leads to a better decision than the baseline. Explain what information was retained/shared and how it contributed to the improved outcome.
- One scenario where your system still fails or performs worse than expected. Analyze the root cause—is it a limitation of the memory/collaboration mechanism, the underlying LLM, or the integration design?
- Based on your analysis, briefly describe (1 paragraph) how you might further improve the implementation. Consider insights from the Vending-Bench findings (e.g., strategic memory retrieval) or multi-agent literature (e.g., debate convergence, role specialization).

Problem 2: Reward Modeling and RLHF

A central component of the modern LLM alignment pipeline is *Reinforcement Learning from Human Feedback* (RLHF) [14]. The standard RLHF pipeline consists of three stages: (1) supervised fine-tuning (SFT) on high-quality demonstrations, (2) training a *reward model* r_ϕ on human preference data, and (3) optimizing the language model policy π_θ against r_ϕ using a reinforcement learning algorithm such as Proximal Policy Optimization (PPO) [19]. At stage (3), the objective is:

$$\max_{\theta} \mathbb{E}_{c \sim \mathcal{D}, x \sim \pi_\theta(\cdot | c)} [r_\phi(c, x)] - \beta \text{KL}[\pi_\theta(\cdot | c) \parallel \pi_{\text{ref}}(\cdot | c)],$$

where π_{ref} is the SFT model used as a reference and β controls the strength of the KL penalty that prevents the policy from deviating too far from π_{ref} .

The reward model r_ϕ is trained on a dataset of preference pairs $\{(c_i, x_i^+, x_i^-)\}$ where x^+ is preferred over x^- . A common formulation uses the Bradley–Terry model, maximizing:

$$\mathcal{L}_{\text{RM}}(\phi) = \mathbb{E}[\log \sigma(r_\phi(c, x^+) - r_\phi(c, x^-))],$$

where σ is the sigmoid function.

Direct Preference Optimization (DPO) [17] simplifies this pipeline by eliminating the explicit reward model. DPO shows that the optimal policy under the RLHF objective has a closed-form relationship with the reward:

$$r(c, x) = \beta \log \frac{\pi_\theta(x | c)}{\pi_{\text{ref}}(x | c)} + \beta \log Z(c),$$

which allows reparameterizing the preference loss directly in terms of the policy:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(x^+ | c)}{\pi_{\text{ref}}(x^+ | c)} - \beta \log \frac{\pi_\theta(x^- | c)}{\pi_{\text{ref}}(x^- | c)} \right) \right].$$

In this problem, you will (1) fine-tune a reward model to capture a specific preference (safety), (2) explore DPO training and compare it with PPO-based RLHF, and (3) analyze limitations of reward models. Starter code is provided in the IPython notebook `11766_hw2_p2.ipynb`.

[Question 2.1] (*Fine-tune a Reward Model*) You will train a reward model on top of Qwen/Qwen3-0.6B using the PKU-SafeRLHF dataset [4], which provides human-annotated preference pairs where responses are ranked by safety. The dataset separates annotations of helpfulness and harmlessness, providing $\sim 166\text{k}$ preference pairs across 19 harm categories with three severity levels. The starter code loads 1000 training and 200 evaluation preference pairs, where each pair consists of a chosen (safer) and rejected (less safe) response to the same prompt.

You will also evaluate your trained model on RewardBench [7], a benchmark for evaluating reward models across chat, reasoning, and safety categories.

DELIVERABLES FOR Q2.1

Please answer the following questions.

- Examine 3 examples from the training data. For each, identify what makes the chosen response “safer” and what pattern the reward model might learn.
- Evaluate the untrained (baseline) reward model on the evaluation set. Why is the baseline accuracy close to 50%? What does this tell us about the untrained model’s ability to judge safety?
- Adjust the training configuration so that your trained reward model achieves higher accuracy than the baseline. Report the accuracy and average reward margin (chosen – rejected) for both the baseline and trained models.
- Evaluate your trained reward model on RewardBench safety samples. Pick one example where the model is correct and one where it is wrong, and analyze why the reward scores make sense or fail. Provide one reason why your reward model training works and one reason why it does not.

[Question 2.2] (*DPO Training*) You will use DPO [17] to fine-tune Qwen/Qwen2.5-0.5B on the same safety preference data. Recall that unlike PPO-based RLHF [14, 19], DPO does not require an explicit reward model or online generation during training—it directly optimizes the policy from offline preference pairs. You will compare the model’s behavior before and after DPO training, and use your trained reward model from Problem 4.1 to score the DPO model’s outputs.

DELIVERABLES FOR Q2.2

Please answer the following questions.

- A. Characterize the base model’s responses to harmful prompts before DPO training. Does it comply with harmful requests, refuse them, or do something else?
- B. Run DPO training and report the training and validation loss for each epoch.
- C. Compare sample responses before and after DPO. Does the training work? Pick one good example and one bad example and explain.
- D. Check the reward scores given by your trained reward model on the DPO model’s outputs. Do the scores look reasonable? Pick one good example and one bad example and explain.
- E. Compare DPO and PPO-based RLHF. Discuss:
 - What are the key differences in their training pipelines?
 - What is one advantage of DPO over PPO?
 - What is one advantage of PPO over DPO?
 - In what scenarios might you prefer one over the other?
- F. Explain how the reward model you trained in Problem 4.1 could be used in a PPO-based RLHF pipeline. Specifically:
 - How would the reward model provide training signal during PPO?
 - What could go wrong if the reward model has biases or limitations ?

[Question 2.3] (*Limitations of Reward Models*) A known limitation of reward models is that they can assign significantly different scores to equally valid responses, penalizing diversity in model outputs. Recent work has shown that current LLMs already generate text that is less diverse than human-authored text [13], and reward-model-driven optimization can exacerbate this by collapsing outputs toward a narrow set of high-reward patterns.

Using prompts from NoveltyBench [23]—a benchmark designed to measure whether language models can produce humanlike diversity in their outputs—you will generate multiple responses per prompt and analyze how the reward model scores them.

DELIVERABLES FOR Q2.3

Please answer the following questions.

- A. Identify 4 pairs of responses to the same prompt where both responses are valid but receive significantly different reward scores. For each pair, describe: (1) what the prompt asks, (2) why both responses seem equally good, and (3) what might cause the reward model to prefer one over the other.
- B. Design and run an experiment to demonstrate a specific limitation of the reward model (e.g., length bias, tone bias, keyword sensitivity). Describe your experiment, show results, and explain the limitation with at least 2 supporting examples.
- C. Based on all your findings, what are the implications for using reward models in RLHF? Suggest at least two potential mitigations.

Problem 3: LLM as Judge

[Question 3.1] (Naïve LLM-as-a-judge) An LLM is a distribution $\Pr(\cdot|c; \theta)$ parameterized by θ over responses given a context (e.g., prompt). We would like to compare two LLMs parameterized by θ and θ' by comparing the outputs x (from θ) and x' (from θ') for a fixed input c .

One way to compare these outputs is to present a (human) judge with the input and the two outputs and to request which they prefer. Define this preference as,

$$\Delta(x, x' | c) = \begin{cases} 1 & \text{user prefers } x \text{ to } x' \\ -1 & \text{user prefers } x' \text{ to } x \\ 0 & \text{user is indifferent} \end{cases}$$

Given a dataset of inputs \mathcal{C} and user judgments, we can compute the expected preference as the pairwise evaluation metric.

In recent years, it has become popular to evaluate model outputs by asking an LLM parameterized by ϕ to provide a preference [20, 14]. Define the naïve LLM-as-a-judge as,

$$\tilde{\Delta}(x, x' | c, \phi) = \text{sgn} [\Pr(x | c; \phi) - \Pr(x' | c; \phi)]$$

As with human-based preferences, we can compute the preference based on LLM-as-a-judge using a dataset of inputs.

Given \mathcal{C} , we can measure the consistency between Δ and $\tilde{\Delta}$ with respect to θ and θ' as,

$$\mu(\Delta, \tilde{\Delta}, \theta, \theta', \mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathbb{I} [\Delta(x, x' | c) = \tilde{\Delta}(x, x' | c; \phi)]$$

where x and x' are drawn from language models θ and θ' . In all situations, we randomize the order of x and x' as arguments, although they are consistent for a given $c \in \mathcal{C}$.

DELIVERABLES FOR Q3.1

Please answer the following questions.

- A. Assume ϕ is completely untrained (neither pretrained nor fine-tuned). What is the expected consistency μ between Δ and $\tilde{\Delta}$?
- B. Assume that ϕ is an extremely large and well-trained language model. When will the expected consistency μ between Δ and $\tilde{\Delta}$ be less than 1?
- C. Assume that $\mathbb{E}_{c \sim \mathcal{C}} [\Delta(x, x' | c)] \approx 0$ but $\mathbb{E}_{c \sim \mathcal{C}} [\tilde{\Delta}(x, x' | c; \phi)] \approx 1$. Provide at least two reasons why this might happen.

[Question 3.2] (*Rubric-based LLM-as-Judge*) Consider an evaluation setting where performance can be decomposed into a collection of explicit *criteria*. For example, in a question answering system, we may assess whether specific required facts are present; in an agentic system, we may evaluate whether particular sub-goals are achieved. We refer to evaluating along multiple explicit dimensions rather than assigning a single holistic score as *rubric-based evaluation*.

Formally, suppose a judge is provided with a rubric consisting of m textual rules $\{f_i\}_{i=1}^m$. For each rule f_i , the judge compares two responses x and x' under context c . Let $\delta(x, x' \mid c, f_i) \in [-1, 1]$ denote the degree to which x is preferred to x' with respect to rule f_i . For example, this quantity may reflect whether a required fact is present in one response but not the other. A global pairwise preference can then be obtained by aggregating these per-criterion comparisons using a uniform combination,

$$\Delta(x, x' \mid c) \propto \sum_{i=1}^m w_i \delta(x, x' \mid c, f_i)$$

where $w_i = \frac{1}{m}$.

Now suppose we adopt an LLM-based judge that is substantially stronger than the models being evaluated (e.g., it has been trained on significantly more data or has greater capacity). We use this LLM judge to evaluate each criterion independently, producing $\tilde{\delta}(x, x' \mid c, f_i)$ via the following prompt:

```
Please evaluate these two system responses subject to the following criteria,
CRITERIA: <f_i>
subject to the following input,
INPUT: <c>
Generate "+1" if OUTPUT 1 better satisfies CRITERIA than OUTPUT 2.
Generate "-1" if OUTPUT 2 better satisfies CRITERIA than OUTPUT 1.
Generate "0" if both OUTPUT 1 and OUTPUT 2 satisfy or do not satisfy CRITERIA.
The two system responses follow.
OUTPUT 1: <x>
OUTPUT 2: <x'>
```

Assume that, on a validation set, we observe that $\tilde{\delta}(x, x' \mid c, f_i) \approx \delta(x, x' \mid c, f_i)$ for all i . That is, for each individual criterion, the LLM judge's decisions are statistically indistinguishable from those of a human annotator. Under this assumption, we aggregate the LLM's per-criterion judgments using the same weighting scheme applied to human evaluations.

In order to more efficiently evaluate, you use the following *composite LLM-as-a-judge prompt*,

```
Please evaluate these two system responses subject to the following criteria,
CRITERIA 1: <f_1>
...
CRITERIA m: <f_m>
subject to the following input,
INPUT: <c>
For each CRITERIA i, do the following.
Generate "f_i: +1" if OUTPUT 1 better satisfies CRITERIA i than OUTPUT 2.
Generate "f_i: -1" if OUTPUT 2 better satisfies CRITERIA i than OUTPUT 1.
Generate "f_i: 0" if both OUTPUT 1 and OUTPUT 2 satisfy or do not satisfy CRITERIA i.
The two system responses follow.
OUTPUT 1: <x>
OUTPUT 2: <x'>
```

and aggregate the criteria scores with a uniform weight.

DELIVERABLES FOR Q3.2

Please answer the following questions.

- A. You notice that your composite prompt is not correlated with human ratings.
- (a) Explain why this might be happening (**Hint:** [10]).
 - (b) Propose a solution.
 - (c) Discuss any limitations of your solution.
- B. Assume you tested your approach to (A) and the correlation with human preferences is good for short-answer question answering. You set your approach on long-answer question answering and the correlation again degrades.
- (a) Explain what might be happening.
 - (b) Propose a solution.
 - (c) Discuss any limitations of your solution.

References

- [1] Axel Backlund and Lukas Petersson. Vending-Bench: A benchmark for long-term coherence of autonomous agents. *arXiv preprint arXiv:2502.15840*, 2025.
- [2] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning (ICML)*, 2024.
- [3] Taicheng Guo, Xiying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xi-angliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *Proceedings of IJCAI*, 2024.
- [4] Jiaming Ji, Donghai Hong, Borong Zhang, Boyuan Chen, Juntao Dai, Boren Zheng, Tianyi Qiu, Jiayi Zhou, Kaile Wang, Boxuan Li, Sirui Han, Yike Guo, and Yaodong Yang. PKU-SafeRLHF: Towards multi-level safety alignment for LLMs with human preference. *arXiv preprint arXiv:2406.15513*, 2024.
- [5] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics.
- [6] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Mark Malhotra, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.
- [7] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*, 2024.
- [8] Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *arXiv preprint arXiv:2402.05120*, 2024.
- [9] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *Proceedings of EMNLP*, 2024.
- [10] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.
- [11] Shichun Liu et al. Memory in the age of AI agents: A survey. *arXiv preprint*, 2025.

- [12] Team MiniCPM. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025.
- [13] Vaishnavh Nagarajan, Chen Henry Wu, Charles Ding, and Aditi Raghunathan. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. In *International Conference on Machine Learning (ICML)*, 2025.
- [14] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [15] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.
- [16] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R Bowman. QuALITY: Question answering with long input texts, yes! *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2022.
- [17] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2023.
- [18] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. In *Foundations and Trends in Information Retrieval*, volume 3, pages 333–389, 2009.
- [19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [20] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.
- [21] Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations (ICLR)*, 2022.
- [22] Wujiang Zhang, Zekun Zhou, Yufan Li, and Fei Tao. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.
- [23] Yiming Zhang, Harshita Diddee, Susan Holm, Hanchen Liu, Xinyue Liu, Vinay Samuel, Barry Wang, and Daphne Ippolito. NoveltyBench: Evaluating language models for humanlike diversity. *arXiv preprint arXiv:2504.05228*, 2025.