Cooperative Agentic Framework for Enhanced Function Calling

Wei Jiang[1*], Junjie Ma[1], Yong Zhang[1], Yuwei Wang[1], Weiyu Ji[1], and Zichao Zhang[1]

[1]Information Science Academy of China Electronics Technology Group Corporation, Beijing, China
loftyjiang@163.com
junjiema7237@gmail.com
lucky111107@163.com
wangyuwei092233@163.com
jiwy1111@126.com
zhangzc1@yeah.net

[1]

Abstract. A cooperative agentic framework is presented in this paper, which enhances function calling capabilities in Large Language Model (LLM) powered agents while optimizing resource utilization. Weaker and stronger agents are combined through a consensus-based method in this approach, addressing the challenge of balancing performance with computational costs. The framework is evaluated across four diverse test suites: Open Weather, The Cat API, Home Search, and Booking, with the number of calls to weaker agents, consensus thresholds, and fallback strategies being manipulated. It is shown by the results that success rates are significantly improved by increasing calls to weaker agents and implementing strong fallback mechanisms, particularly for initially lower-performing agents. Notably, high success rates comparable to GPT-4 at significantly lower costs were achieved by models like Claude-3-Haiku and DeepSeek-Coder. Particular efficacy is demonstrated by the framework in complex scenarios, where performance was substantially boosted by strategic fallbacks. A practical solution for deploying LLM-powered agents in resource-constrained environments without compromising performance is offered by this approach, potentially increasing the accessibility and adoption of powerful agents in real-world applications ranging from API interactions to complex systems.

Keywords: Large Language Model · Function Calling · Consensus-Based Approach · Multi-Agent Systems.

1 Introduction

LLM powered AI agents [42, 7, 23, 17, 16, 20, 36, 11, 12, 35, 15] have revolutionized various tasks, from creative writing to code completion, necessitating a

[1] *corresponding author

deeper understanding of how to optimally utilize these powerful yet resource-intensive tools. The ability to perform function calls is of paramount importance for these models. As LLMs and AI agents continue to evolve, their ability to interact with external systems through function calls becomes increasingly crucial. As these agents grow in size and capability, they present a significant trade-off between performance and cost. Stronger models generally offer superior results but require substantial computational resources, while weaker models are more efficient but may lag in performance [41, 31]. This trade-off poses a critical challenge for researchers and practitioners with limited budgets, time constraints, or environmental concerns. Deciding which model to use for a given task is not always straightforward, especially when considering the diverse range of query difficulties encountered in real-world applications.

Recent works have attempted to optimize the use of agents by employing various strategies. Two main approaches have emerged: routing [8, 40], where queries are directed to either a weak or strong model based on a decision criterion, and cascading [29, 13, 4], where queries always start with the cheaper model and escalate to the more expensive one if necessary. However, these methods often rely on auxiliary models or repeated calls to weaker agents, introducing additional complexity and potentially undermining the cost-saving objectives they aim to achieve.

To address the above problems, this paper proposes a novel cooperative framework for enhancing the function calling capabilities of agents while optimizing resource utilization. The approach is inspired by the observation that many tasks contain a subset of "easy" queries that can be effectively handled by weaker, less expensive models. This insight is leveraged to develop a system that combines the efficiency of weaker models with the power of stronger ones, employing a consensus-based approach inspired by self-consensus techniques.

Our framework focuses specifically on function calling tasks, which are crucial for integrating agents with external systems and APIs. The approach is evaluated using four diverse test suites: Open Weather, The Cat API, Home Search, and Booking, all of which are introduced in [37]. These test suites cover a range of scenarios from real-world API interactions to fictional, complex booking systems, providing a comprehensive assessment of function calling capabilities.

The key contributions of this work are as follows:

1. A Cooperative Agentic Framework: A cooperative framework is introduced, combining both weaker and stronger agents to optimize function calling performance while minimizing costs.
2. Consensus-Based Query Routing: This framework employs a consensus - based approach to determine when to route queries to stronger agents, effectively avoiding the need for auxiliary models or excessive repeated calls.
3. Evaluation and Strategies for Inconsistent Responses: A comprehensive evaluation of the framework is presented across four diverse test suites, demonstrating its effectiveness in balancing performance and cost. Additionally, strategies for handling inconsistent responses from weaker models are ex-

plored, either by selecting from the weaker agent's outputs or by falling back to stronger agents.

Our findings suggest that understanding and leveraging the inherent signals from agents can lead to more efficient and cost-effective deployment strategies, potentially increasing the accessibility and adoption of these powerful tools across various domains and applications.

2 Background and Related Work

Recent advancements in large language models have shown impressive results across various tasks, including function calling [6]. These models, often referred to as foundation models [5], have demonstrated remarkable capabilities in complex reasoning [34] and synergizing reasoning with action [39]. However, the performance gap between weaker and stronger agents remains significant, especially in complex scenarios [27]. This disparity has motivated research into methods for leveraging stronger models to improve the performance of weaker ones [32], as well as exploring the fundamental pattern recognition capabilities of these models [24]. Despite these advancements, efficiently balancing performance and computational cost remains a critical challenge in the field.

At the core of this cooperative agentic framework is a decision-making mechanism that determines whether to route a query to the stronger agent based on the output of the weaker agent. The ideal decision maker should only invoke the stronger agent when the weaker agent's answer is likely to be incorrect, thereby minimizing total cost while maintaining overall task performance. To achieve this, a novel approach based on the "answer consensus" of the weaker agents is proposed.

Answer Consensus and Sampling Strategies Our approach is inspired by the work of [32], which demonstrated that answer consistency and consensus can enhance performance in reasoning tasks. Rather than relying on a single answer, multiple answers are sampled for each query, and their consensus is assessed. The hypothesis is that a high level of consensus among sampled answers reflects the weaker agent's confidence in solving the query, suggesting that the most consistent answer is likely correct.

To implement this idea, the work of [32] is expanded upon, considering two sources of sampling consensus:

1. In-Distribution Sampling. Multiple answers are generated using the same instructions for the weaker agents, achieved by setting a non-zero temperature during inference.
2. Diverse In-Context Demonstrations. Answers are sampled using two distinct sets of task demonstrations to ensure diversity in the generated responses.

Consensus-Based Decision Making In this method, the consensus of the weaker agent's answer samples is calculated through a voting mechanism. For a given query Q, the set of answers is denoted as $(A_{w1}, A_{w2}, \ldots, A_{wK})$, where K is the number of samples. The most consistent answer is selected as the one with the highest agreement among the samples. Consensus is measured using an agreement score s:

$$s = \frac{\sum_{i=1}^{K} \mathbf{1}[A_{wi} = A_w]}{K} \tag{1}$$

where A_w is the answer from the weaker agent and $\mathbf{1}[\cdot]$ is an indicator function. If s exceeds a predefined threshold, the weaker agent's answer is accepted if the consensus score meets a predefined threshold; otherwise, the query is routed to the stronger agent for resolution.

Tool-Augmented Language Models Recent research has demonstrated that LLMs can significantly enhance their capabilities by utilizing external tools. Work [30] introduced Toolformer, a model trained to decide which APIs to call, when to call them, and how to incorporate the results into future token predictions. This self-supervised approach achieved improved zero-shot performance across various tasks without sacrificing core language modeling abilities.

Building on this concept, work [28] proposed ToolLLM, a comprehensive framework for tool use in LLMs. ToolLLM encompasses data construction, model training, and evaluation, addressing the limitations of open-source LLMs in tool-use capabilities. Their approach demonstrated remarkable ability in executing complex instructions and generalizing to unseen APIs.

Evaluation of Tool Utilization Capabilities To assess the effectiveness of tool-augmented LLMs, several benchmarks and evaluation frameworks have been developed. [19] introduced API-Bank, a benchmark specifically designed for tool-augmented LLMs. This comprehensive evaluation system consists of 73 API tools and assesses LLMs' capabilities in planning, retrieving, and calling APIs.

The study [10] proposed T-Eval, a framework that evaluates the tool utilization capability of LLMs step by step. T-Eval decomposes tool utilization into multiple sub-processes, providing a more fine-grained analysis of LLMs' competencies in this domain.

Enhancing Open-Source LLMs While closed-source LLMs have shown impressive tool manipulation capabilities, there has been a growing interest in enhancing open-source models to achieve similar performance. The research [37] investigated methods to improve the tool manipulation capabilities of open-source LLMs with practical amounts of human supervision. Their work demonstrated that techniques such as programmatic data generation, system prompts, and in-context demonstration retrievers could significantly boost the performance of open-source LLMs in tool manipulation tasks.

Multi-Agent Frameworks Recent researches have explored the potential of multi-agent systems to enhance LLM performance. Work [18] proposed a sampling-and-voting method that scales LLM performance with the number of agents instantiated. This approach showed improvements across various LLM benchmarks, particularly for more difficult tasks.

In a related vein, the work [21] introduced CMAT, a multi-agent collaboration tuning framework designed to enhance weak language models. CMAT fosters collaborative learning and real-time adaptation among multiple intelligent agents, improving context-awareness and long-term memory.

Reasoning Techniques in LLMs Several techniques have been developed to improve the reasoning capabilities of LLMs, which are crucial for effective tool utilization. The research [33] demonstrated that chain-of-thought prompting significantly enhances the ability of large language models to perform complex reasoning. This approach involves generating a series of intermediate reasoning steps, leading to improved performance on various tasks.

Building on this concept, the study [32] introduced self-consistency, a decoding strategy that samples multiple reasoning paths and selects the most consensus answer. This method has shown significant improvements in arithmetic and commonsense reasoning benchmarks.

Authors in [9] proposed the Program of Thoughts (PoT) prompting method, which disentangles computation from reasoning in numerical reasoning tasks. PoT uses language models to express the reasoning process as a program, relegating computation to an external computer.

Cost-Effective LLM Deployment As LLMs grow in size and capability, there is an increasing focus on cost-effective deployment strategies. The research [8] introduced FrugalGPT, a framework that explores various strategies to reduce inference costs associated with using LLMs. These strategies include prompt adaptation, LLM approximation, and LLM cascade.

The research [29] proposed a framework called "Fly-swat or cannon" (FORC) for cost-effective language model choice. FORC uses a meta-model to assign inputs to appropriate LLMs, aiming to achieve high overall performance at low cost.

In conclusion, while significant progress has been made in enhancing LLMs' tool utilization capabilities and developing cost-effective deployment strategies, there remains ample room for improvement in creating cooperative frameworks that can further enhance function calling capabilities in language models.

3 Methodology And Experiments Setting

This methodology aims to enhance the performance of function calling tasks through a collaborative framework while maintaining efficiency in resource-constrained environments. The core of this approach leverages multiple outputs from weaker agents to reach a consensus, with fallback to stronger agents when necessary.

1. Baseline Performance: This work begins by establishing baseline performance metrics for multiple Language Models (LLMs). This initial step involves executing each model on an identical set of function calling tasks to determine initial accuracy, consensus, and resource utilization. A diverse range of LLMs, including both weaker and stronger models, is selected. To ensure a comprehensive evaluation, a test suite encompassing function calling tasks of varying complexity and domains is created. Each model undergoes multiple runs through this test suite to account for potential output variations. Key performance indicators, including accuracy, consistency, and resource utilization, are recorded, laying the groundwork for subsequent comparisons.
2. Consensus-Based Approach: The next phase implements a consensus-based approach. For each query, the weaker agent is prompted multiple times to generate function calling code snippets. This repetitive prompting captures a range of possible responses and identifies patterns in the generated outputs. An optimal number of prompts is determined based on the balance between accuracy and computational cost. To encourage diverse outputs, a prompting strategy is implemented that introduces slight variations in the input. All generated code snippets are stored in a structured format for easy analysis.
3. Consensus Check: During the consensus check phase, regular expressions are utilized to analyze the generated code snippets for consensus. A response is considered to have reached consensus if a majority of the snippets (quorum) are identical. This majority rule filters out inconsistencies and identifies the most reliable response from the weaker agent. Robust regular expressions are developed to handle formatting variations while still identifying functionally equivalent code snippets. Additionally, a scoring system is implemented to determine the degree of similarity between non-identical snippets. Based on empirical testing and specific use case requirements, a threshold for consensus is set.
4. Response Selection Strategies: When no clear majority (less than half) is achieved, two alternative strategies are implemented.
   - Weak Agent Selection. The first is a weak agent selection strategy, where the plurality response (the one with the most occurrences, even if not a majority) is chosen, or a random response is selected from the generated set. A weighted random selection method is implemented, favoring responses with higher occurrence frequencies, and a confidence score based on the similarity of non-identical responses is considered for incorporation.
   - Strong Agent Fallback. The second strategy is a strong agent fallback, routing the query to a more powerful agent (e.g., GPT-4) to generate the function call code. Criteria are developed for determining when to fallback to the stronger agent, and a caching mechanism is implemented to store strong agent responses for similar queries, reducing computational overhead in future runs.
5. Performance Evaluation: Performance evaluation is a crucial component of the method. The results of the collaborative framework, including both response selection strategies, are compared against baseline performances, fo-

cusing on metrics such as accuracy, consensus, and resource utilization. A comprehensive evaluation framework is developed, considering the success rate of function calls, the variability of responses across multiple runs, and computational costs, including processing time for weak agent consensus generation, the frequency and impact of strong agent fallbacks, and overall system latency and throughput. Automated testing pipelines are implemented to facilitate large-scale evaluation across diverse scenarios, and ablation studies are conducted to isolate the impact of individual components of the framework.

Based on the evaluation results, optimization is performed. Consensus thresholds and prompting strategies are adjusted to optimize the trade-off between accuracy and resource utilization. Experiments are also conducted with different weak agent models and ensemble techniques to enhance baseline performance. Furthermore, adaptive strategies are developed to dynamically adjust the framework's parameters based on task characteristics and real-time performance metrics.

Scalability and robustness testing are conducted to assess the framework's performance under various load conditions and edge cases. Stress tests are performed to determine system capacity and identify bottlenecks, while network latency and service disruptions are simulated to evaluate the framework's resilience. A wide range of input types, including edge cases and potentially malformed queries, are tested to ensure robust error handling.

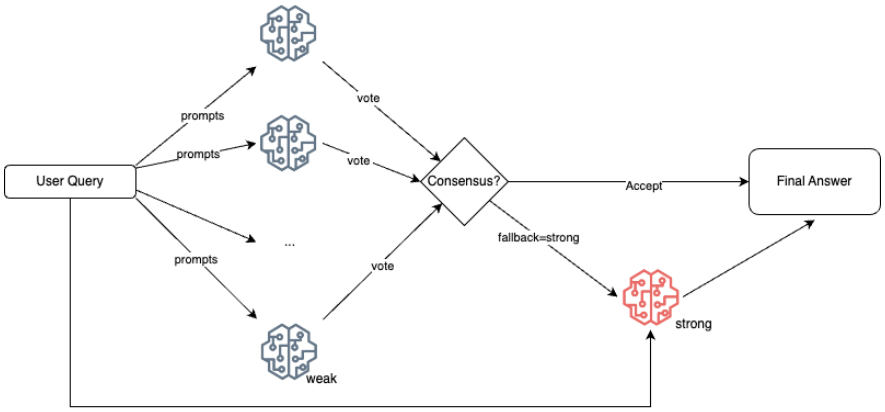

Fig. 1: Illustration of the proposed method.

Through this comprehensive approach, a practical framework is provided to researchers and practitioners for deploying agents in resource-constrained environments without compromising performance. The detailed methodology facilitates experiment replication while offering insights into potential areas for further optimization and research.

3.1 Test Suites

Open Weather This suite tests the agents' ability to make API calls for weather data retrieval[3]. It includes tasks such as querying current weather conditions, forecasts, and historical weather data for various locations, simulating real-world scenarios where accurate interpretation and API usage are crucial.

The Cat API Focused on interactions with a REST API for cat-related operations this suite evaluates the models' capability to handle HTTP methods and parse JSON responses effectivelycitethecatapi. Tasks involve retrieving information about different cat breeds, searching for images of cats, and performing CRUD operations (Create, Read, Update, Delete) on user-generated cat-related data.

Home Search Utilizing a fictional API for home searching, this suite evaluates the models' performance on domain-specific tasks. It tests the ability to understand and generate queries for property search parameters, such as location, price range, number of bedrooms, and property type, assessing how well models handle detailed and structured data requests.

Booking Designed to test complex dependencies in trip and hotel booking scenarios, this suite challenges the agents with multi-step, interdependent API calls. Tasks include booking flights, hotels, and car rentals with specific requirements and constraints, managing reservations, and handling cancellations and modifications, simulating real-world travel booking systems.

To systematically evaluate the performance and adaptability of the cooperative framework, three main control variables were established in the experimental setup:

- Number of Calls (num): Represents the number of times an agent is called for each query. Increasing the number of calls is hypothesized to improve the consensus and accuracy of the responses due to the diversity in generated outputs.
- Consensus Threshold (vote): This threshold determines when a majority consensus has been achieved among the responses generated by the agent. A response is considered as the majority consensus if it reaches or exceeds this threshold.
- Fallback Strategy (fallback): Determines the course of action when the majority consensus is not reached:
  1. fallback=no: Only one call is made, and the response from this single call is used.
  2. fallback=weak: The most frequent response among the generated ones is chosen.
  3. fallback=strong: Engages a stronger agent (such as GPT-4 using Greedy Decoding) to handle the query.

These control settings lead to the following experimental conditions:

- Condition A (num=1; vote=1; fallback=no): Basic condition with a single call and no fallback.
- Condition B (num=3; vote=0.5; fallback=weak): Multiple calls with selection of the most frequent response if no clear majority is found.
- Condition C (num=5; vote=0.5; fallback=weak): Increased number of calls, still selecting the most frequent response under weak fallback.
- Condition D (num=3; vote=0.5; fallback=strong): Same as B but with a strong fallback to a more capable agent if needed.
- Condition E (num=5; vote=0.5; fallback=strong): Increased calls with strong fallback, providing a robust test of fallback efficacy.

Diversity Additionally, to introduce diversity and mitigate potential bias in response generation, the temperature parameter was set to 0.5. This setting helps in diversifying the outputs, making them less deterministic and more representative of different potential responses. During the retrieval of examples for generating prompts, the order of examples is shuffled to further ensure that the responses are not influenced by any fixed pattern in the data presentation [22, 25].

3.2 Overview of Models Used by Agents

To assess the effectiveness of the framework across a spectrum of model capabilities, the following language models were included in the experiments, with a brief overview of their characteristics, performance, and costs provided.

Baichuan-13B and Baichuan2-Turbo [38] from Baichuan-AI. These models are trained on a whopping 2.6 trillion high-quality tokens, making them excellent performers in both Chinese and English tasks. Baichuan-13B is hosted on a local A100 service for the experiments, but the pricing is calculated according to vendor rates.

GPT-3.5 and GPT-4 [26] from OpenAI. GPT-3.5 is already well-regarded for handling a wide range of natural language tasks, but GPT-4 takes it a step further with even better accuracy and understanding.

DeepSeek-Coder [14] from DeepSeek. This model is specifically tailored for coding, trained on 2 trillion tokens with 87% of them being code and 13% natural language, making it highly specialized for programming-related tasks.

Claude-3-Haiku [1] from Anthropic is designed for enterprise applications. It's fast, affordable, and performs well on industry benchmarks. It can process up to 21K tokens per second for prompts under 32K tokens, which is impressive.

Qwen1.5-72B-Chat-GPTQ-Int4 [2] from Alibaba. This model offers substantial performance improvements and supports multiple languages. It can handle up to 32K context length, which is quite versatile for different applications. Although this model is hosted on a local A100 service for the experiments, its pricing is calculated based on vendor rates.

The following table summarizes the costs and hosting details for these models:

Each of these models brings its own strengths and cost factors, making them suitable for different use cases. GPT-4, though high-performing, come

Model	Price (RMB per million tokens)	Hosted by
GPT-4	217.5	Azure
GPT-3.5	10.88	Azure
DeepSeek-Coder	1	DeepSeek
Claude-3-Haiku	1.81	Anthropic
Qwen1.5-72B-Chat	3.5	Local (A100)
Baichuan2-turbo	8	Baichuan-AI
Baichuan-13B	6	Local (A100)

Table 1: Model Pricing and Hosting Information. The price data was retrieved on 2024/07/04 from the respective vendors' websites.

with higher costs. On the other hand, models like DeepSeek-Coder, Claude-3-Haiku, Qwen1.5-72B-Chat-GPTQ-Int4, and Baichuan-13B provide more budget-friendly options with specialized capabilities, particularly in coding and function contexts.

3.3 Evaluation Metrics

Our evaluation focuses on four key aspects:

Success Rate Measuring the correctness of the generated function calls and their outputs. This includes verifying if the function calls produce the expected results when interfacing with various APIs.

Strong Fallback Rate This metric measures the frequency at which the framework resorts to using a stronger language model, such as GPT-4, to generate function call code when the weaker models fail to produce consensus or accurate results.

Relative Cost-Efficiency It represents the inverse of the cost relative to GPT-4. It indicates how many times more expensive GPT-4 is compared to the model being evaluated. For GPT-4, this value is set to 1.0. A higher number means the model is cheaper. The formula is:

$$\text{Relative Cost-Efficiency} = \frac{\text{Cost of GPT-4}}{\text{Cost of Model}} \tag{2}$$

Relative Success Rate It represents the success rate of the model relative to GPT-4's success rate. For GPT-4, this value is set to 1.0. A higher number indicates higher accuracy. The formula is:

$$\text{Relative Success Rate} = \frac{\text{Success Rate of Model}}{\text{Success Rate of GPT-4}} \tag{3}$$

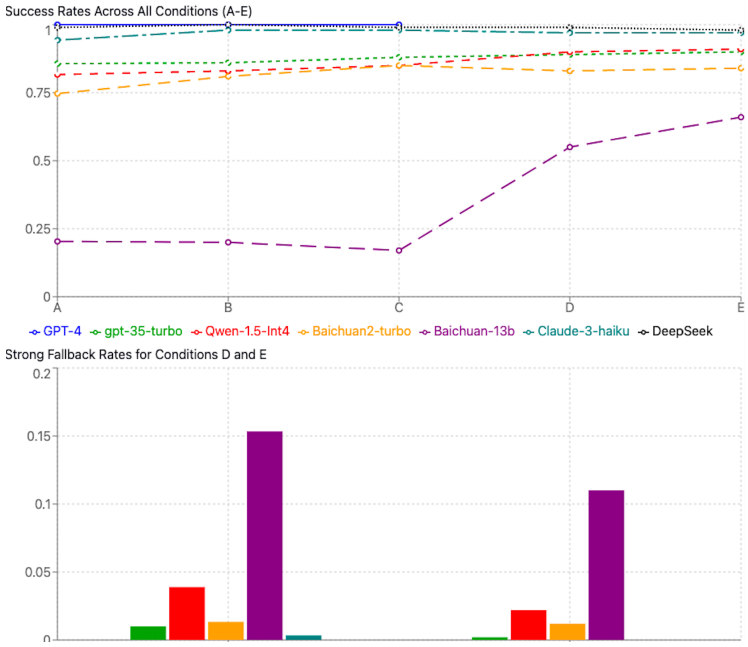

Fig. 2: Relative Success Rate and Strong Fallback Rate for the Open Weather test suite.

3.4 Open Weather

The OpenWeather test suite evaluates the agents' capability to make API calls for weather data retrieval. Throughout the Fig. 2,6-10, it was observed that increasing the number of calls to the weaker agent (num) generally led to improved success rates across nearly all models. This enhancement was most notable in scenarios where a strong fallback strategy was employed. Specifically, in conditions D and E, the fallback to a stronger agent like GPT-4 significantly boosted success rates. For example, Baichuan-13b's success rate increased markedly from 0.203 in a single-call setup (Condition A) to 0.66 with multiple calls and a strong fallback (Condition E).

The implementation of a strong fallback mechanism was particularly effective, enhancing performance notably for models that initially showed lower success rates, such as Qwen-1.5-Int4, which saw an increase from 0.83 to 0.91 when moving from weak to strong fallback scenarios. However, while the strong fallback strategy enhanced performance, it also led to increased costs due to the higher number of calls involved, including those to the more expensive agents. Despite this, models like Claude-3-Haiku and DeepSeek-Coder demonstrated exceptional cost-efficiency, achieving high success rates comparable to GPT-4 but at significantly lower operational costs. These models managed to deliver near-optimal

performance without frequent reliance on stronger agent fallbacks, highlighting their utility in cost-sensitive environments.

For detailed experimental data and additional results, please refer to the Tables 2-6 in Appendix B.

3.5 The Cat API

The Cat API test suite assesses the models' capability to interact with a REST API for cat-related operations, including handling various HTTP methods and parsing JSON responses effectively. The results are shown in Fig. 3,11-15, and details can be found in Tables 7-11 in Appendix C.

Impact of Increasing Calls As the number of calls to the weaker agents increases, there is a general trend of improvement in success rates across almost all models. This enhancement is apparent when comparing conditions with weak fallback (A, B, C) and those with strong fallback (A, D, E).

Effectiveness of Strong LLM Fallback Implementing a strong fallback significantly boosts success rates. This is particularly evident when transitioning from a weak fallback to a strong fallback setup, as seen in the comparative analysis of conditions B and D, as well as C and E.

Cost-Efficiency Models like Claude-3-Haiku and DeepSeek showcase exceptional cost-efficiency, achieving high success rates with minimal additional cost, making them ideal for budget-sensitive applications.

The Cat API test results underline the beneficial impact of increasing the number of interactions (`num`) and the strategic use of strong fallback mechanisms. Particularly, models like Qwen-1.5-Int4 and Baichuan2-turbo show remarkable improvements, highlighting the framework's capability to adaptively leverage stronger agents when necessary. Moreover, the high performance combined with cost-efficiency of models like Claude-3-Haiku and DeepSeek in handling API interactions demonstrates the practicality of the cooperative framework in efficiently managing API calls while maintaining high success rates. These results confirm the robustness of this approach in optimizing LLMs for real-world application scenarios.

3.6 HomeSearch Test Suite

The HomeSearch test suite critically evaluates the capability of various language models to interact with a fictional API for property searching. This suite poses complex, domain-specific challenges that involve understanding and generating appropriate queries for property search parameters. As this test suite is fictional, it is less likely to be included in the LLMs' pretraining data.

The performance and cost-efficiency of the models are analyzed under various experimental conditions, as shown in Fig. 4, 16–20. The details are provided in Tables 12–16 in Appendix D.

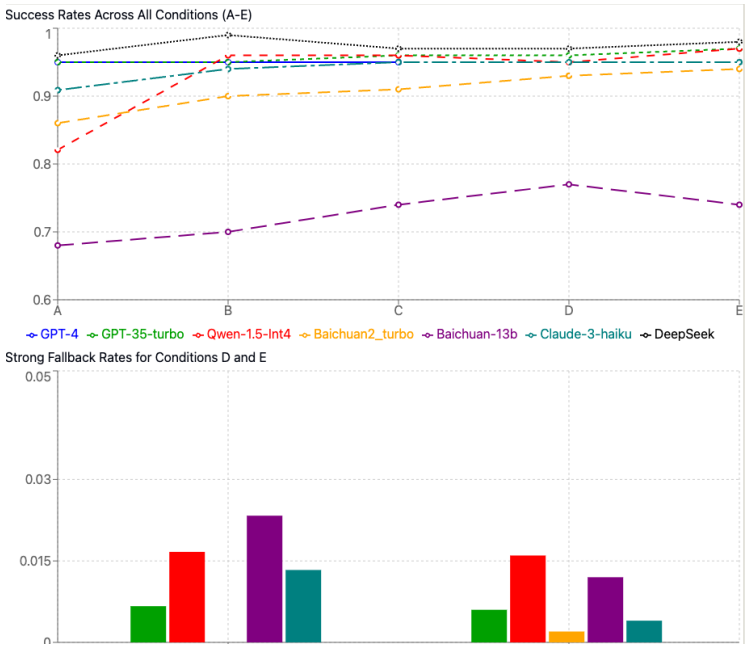

Fig. 3: Relative Success Rate and Strong Fallback Rate for the CAT API test suite.

High Performance and Cost Efficiency Claude-3-Haiku demonstrated outstanding performance across all conditions, consistently achieving high success rates. Notably, it reached a success rate of 0.98 in Condition D, showcasing its robustness and cost-efficiency as one of the top performers. Qwen-1.5-Int4 also exhibited strong performance, particularly excelling in Condition E with a success rate of 0.98. This model maintained excellent cost-efficiency throughout the tests, marking it as a highly effective option.

GPT-4 showed reliable high success rates ranging from 0.93 to 0.96 across different test conditions. However, it was less cost-efficient compared to the more economical models like Claude-3-Haiku and Qwen-1.5-Int4, due to its higher operational costs. GPT-3.5-Turbo and Baichuan2-Turbo provided balanced performance with success rates between 0.89 and 0.91. These models proved particularly cost-effective, especially beneficial in scenarios requiring increased calls and flexible fallback strategies.

DeepSeek consistently displayed high success rates from 0.86 to 0.91, paired with excellent cost-efficiency, making it a viable choice for budget-constrained environments. The performance of DeepSeek notably improved in scenarios with increased calls and strategic fallback applications, as seen in Conditions B and D.

Baichuan-13b lagged behind in performance across all test conditions, with success rates only ranging from 0.05 to 0.20. Despite some improvements when

fallback strategies were employed, it remained the least competitive model in the suite.

Impact of Increased Calls and Strong Fallback Strategy The increase in the number of LLM calls generally led to improved success rates across most models, with significant enhancements for initially lower-performing models like Baichuan-13b and Qwen-1.5-Int4. Implementing a strong LLM fallback, as seen in Conditions D and E, significantly boosted the success rates for nearly all models. For instance, Claude-3-Haiku and Qwen-1.5-Int4 reached peak success rates of 0.98 and 0.97, respectively. However, this approach did increase the total number of calls, which could impact cost-efficiency for models requiring frequent fallback interventions.

Models like Claude-3-Haiku and Qwen-1.5-Int4 stood out as top performers, offering an optimal balance of high success rates and cost efficiency. Conversely, while GPT-4 maintained stable and reliable outputs, its higher cost could pose limitations in budget-sensitive scenarios. On the other hand, models like GPT-3.5-Turbo, Baichuan2-Turbo, and DeepSeek offered a good blend of performance and cost-effectiveness, suitable for varied application needs.

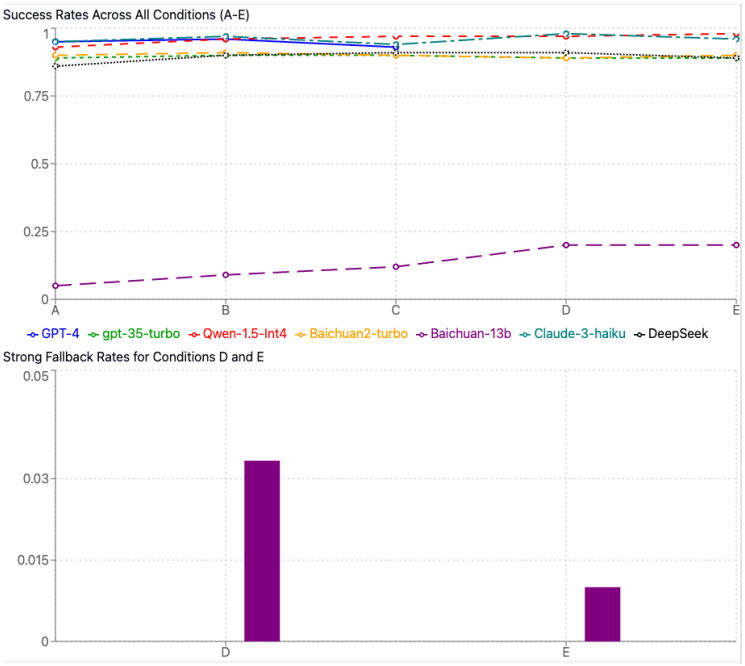

Fig. 4: Relative Success Rate and Strong Fallback Rate for the Home Search test suite.

3.7 Booking

The Booking test suite evaluates the models' ability to handle complex dependencies in trip and hotel booking scenarios. This suite is more challenging than the previous ones, which is reflected in the generally lower success rates across all conditions. The test involves multi-step, interdependent API calls, simulating real-world travel booking systems.

Additionally, an example of the prompts used in this test suite is provided in Appendix A of the prompts used in this test suite, together with the result Fig. 5,21-25. Detailed data can be found in Tables 17-21 in Appendix E.

Impact of Increasing Calls (**num**): The experiments confirm that increasing the number of calls generally enhances performance, particularly when combined with strong fallback mechanisms. This approach significantly improves accuracy, especially for agents that initially perform poorly.

The strong fallback strategy significantly boosts accuracy. This combination typically results in the most substantial performance gains, supporting both proposed viewpoints robustly. Agents respond differently to these strategies. Some agents, like Baichuan-13b, benefit immensely from increased calls and strong fallback mechanisms. In contrast, others, such as Claude-3-Haiku, show less dramatic improvements. This variability underscores the need to tailor strategies based on individual agent' capabilities and task complexities.

These findings strongly support the findings that optimizing model performance in complex tasks can benefit significantly from increased engagement and strategic fallbacks.

4 Conclusion

A novel cooperative framework is introduced by this research, which significantly enhances the function calling capabilities of language models. The strengths of both weaker and stronger agents are leveraged in this approach. Consensus-based methods are combined with strategic response selection, demonstrating a promising path towards optimizing performance while computational resources are effectively managed.

Key findings from extensive experiments across diverse test suites (Open Weather, The Cat API, Home Search, and Booking) are revealed:

1. Efficacy of Increasing Calls to Weaker Agents. Increasing the number of calls to weaker agents generally improved success rates across most models and scenarios, highlighting the benefits of greater engagement with lower-cost models.
2. Performance Boost with Strong Fallback Mechanisms. Implementing strong fallback mechanisms provided significant performance improvements, particularly for initially lower-performing models, ensuring high accuracy and reliability.
3. Cost-Efficiency of Certain Models. Models like Claude-3-Haiku and DeepSeek-Coder achieved high success rates comparable to more expensive models at

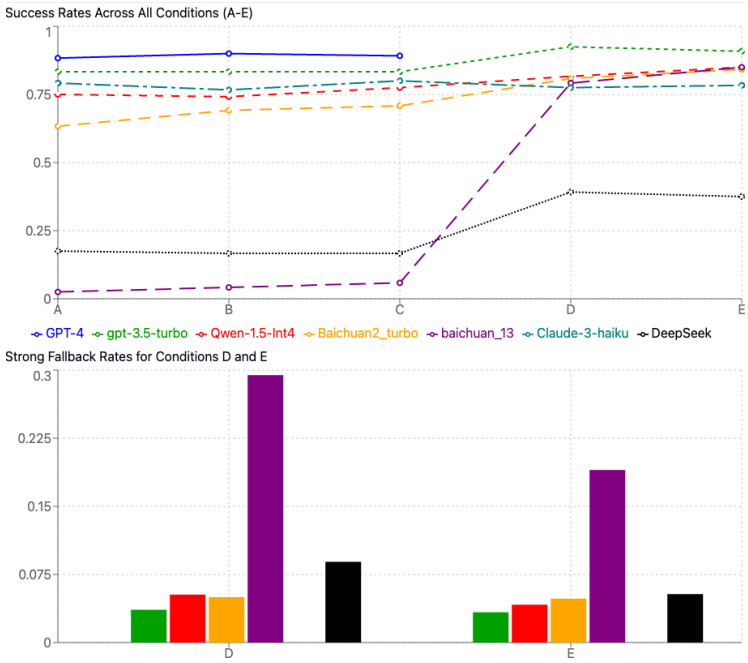

Fig. 5: Relative Success Rate and Strong Fallback Rate for the Booking test suite.

significantly lower operational costs, demonstrating their potential for cost-effective deployment.

It is suggested by these findings that more efficient and cost-effective deployment strategies can be developed through understanding and leveraging the inherent capabilities of both weaker and stronger agents. The potential to increase the accessibility and adoption of powerful AI agents across various domains and applications is created by this approach, particularly in resource-constrained environments.

References

1. Claude 3 haiku: our fastest model yet - anthropic, https://www.anthropic.com/news/claude-3-haiku
2. Qwen/qwen1.5-72b-chat-GPTQ-int4 · hugging face, https://huggingface.co/Qwen/Qwen1.5-72B-Chat-GPTQ-Int4
3. Weather api - openweathermap, https://openweathermap.org/api
4. Aggarwal, P., Madaan, A., Anand, A., Potharaju, S.P., Mishra, S., Zhou, P., Gupta, A., Rajagopal, D., Kappaganthu, K., Yang, Y., Upadhyay, S., Faruqui, M., Mausam: AutoMix: Automatically Mixing Language Models. Tech. rep. (Jun 2024). https://doi.org/10.48550/arXiv.2310.12963, http://arxiv.org/abs/2310.12963, arXiv:2310.12963 [cs] type: article
5. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M.S., Bohg, J., Bosselut, A., Brunskill, E., et al.: On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258 (2021)
6. Brown, T.B.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)
7. Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P.S., Yang, Q., Xie, X.: A survey on evaluation of large language models. ACM Trans. Intell. Syst. Technol. **15**(3) (mar 2024), https://doi.org/10.1145/3641289
8. Chen, L., Zaharia, M., Zou, J.: Frugalgpt: How to use large language models while reducing cost and improving performance. arXiv preprint arXiv:2305.05176 (2023)
9. Chen, W., Ma, X., Wang, X., Cohen, W.W.: Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. arXiv preprint arXiv:2211.12588 (2023)
10. Chen, Z., Du, W., Zhang, W., Liu, K., Liu, J., Zheng, M., Zhuo, J., Zhang, S., Lin, D., Chen, K., Zhao, F.: T-eval: Evaluating the tool utilization capability of large language models step by step. arXiv preprint arXiv:2312.14033 (2024)
11. Cheng, Y., Zhang, C., Zhang, Z., Meng, X., Hong, S., Li, W., Wang, Z., Wang, Z., Yin, F., Zhao, J., He, X.: Exploring large language model based intelligent agents: Definitions, methods, and prospects (2024), https://arxiv.org/abs/2401.03428
12. Dam, S.K., Hong, C.S., Qiao, Y., Zhang, C.: A complete survey on llm-based ai chatbots (2024), https://arxiv.org/abs/2406.16937
13. Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Ruhle, V., Lakshmanan, L.V.S., Awadallah, A.H.: Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. Tech. rep. (Apr 2024). https://doi.org/10.48550/arXiv.2404.14618, http://arxiv.org/abs/2404.14618, arXiv:2404.14618 [cs] type: article

14. Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y.K., Luo, F., Xiong, Y., Liang, W.: DeepSeek-coder: When the large language model meets programming – the rise of code intelligence, http://arxiv.org/abs/2401.14196
15. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges (2024), https://arxiv.org/abs/2402.01680
16. Hadi, M.U., Qureshi, R., Shah, A., Irfan, M., Zafar, A., Shaikh, M.B., Akhtar, N., Wu, J., Mirjalili, S., et al.: A survey on large language models: Applications, challenges, limitations, and practical usage. Authorea Preprints (2023)
17. Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., Chen, E.: Understanding the planning of llm agents: A survey (2024), https://arxiv.org/abs/2402.02716
18. Li, J., Zhang, Q., Yu, Y., Fu, Q., Ye, D.: More agents is all you need. arXiv preprint arXiv:2402.05120 (2024)
19. Li, M., Zhao, Y., Yu, B., Song, F., Li, H., Yu, H., Li, Z., Huang, F., Li, Y.: Api-bank: A comprehensive benchmark for tool-augmented llms. arXiv preprint arXiv:2304.08244 (2023)
20. Li, Y., Wen, H., Wang, W., Li, X., Yuan, Y., Liu, G., Liu, J., Xu, W., Wang, X., Sun, Y., Kong, R., Wang, Y., Geng, H., Luan, J., Jin, X., Ye, Z., Xiong, G., Zhang, F., Li, X., Xu, M., Li, Z., Li, P., Liu, Y., Zhang, Y.Q., Liu, Y.: Personal llm agents: Insights and survey about the capability, efficiency and security (2024), https://arxiv.org/abs/2401.05459
21. Liang, X., Tao, M., Shi, T., Xie, Y.: Cmat: A multi-agent collaboration tuning framework for enhancing small language models. arXiv preprint arXiv:2404.01663 (2024)
22. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts (2023), https://arxiv.org/abs/2307.03172
23. Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., Gao, J.: Large language models: A survey (2024), https://arxiv.org/abs/2402.06196
24. Mirchandani, S., Xia, F., Florence, P., Ichter, B., Driess, D., Arenas, M.G., Rao, K., Sadigh, D., Zeng, A.: Large language models as general pattern machines. arXiv preprint arXiv:2307.04721 (2023)
25. Nori, H., Lee, Y.T., Zhang, S., Carignan, D., Edgar, R., Fusi, N., King, N., Larson, J., Li, Y., Liu, W., Luo, R., McKinney, S.M., Ness, R.O., Poon, H., Qin, T., Usuyama, N., White, C., Horvitz, E.: Can generalist foundation models outcompete special-purpose tuning? case study in medicine (2023), https://arxiv.org/abs/2311.16452
26. OpenAI, Achiam, J., Adler: GPT-4 technical report, http://arxiv.org/abs/2303.08774
27. Peng, B., Li, C., He, P., Galley, M., Gao, J.: Instruction tuning with gpt-4 (2023), https://arxiv.org/abs/2304.03277
28. Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., et al.: Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789 (2023)
29. Šakota, M., Peyrard, M., West, R.: Fly-swat or cannon? cost-effective language model choice via meta-modeling. In: Proceedings of the 17th ACM International Conference on Web Search and Data Mining. pp. 606–615 (2024)

30. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761 (2023)
31. Wang, C., Zhang, B., Sui, D., Tu, Z., Liu, X., Kang, J.: A survey on effective invocation methods of massive llm services (2024), https://arxiv.org/abs/2402.03408
32. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171 (2023)
33. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903 (2023)
34. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems **35**, 24824–24837 (2022)
35. Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., Zheng, R., Fan, X., Wang, X., Xiong, L., Zhou, Y., Wang, W., Jiang, C., Zou, Y., Liu, X., Yin, Z., Dou, S., Weng, R., Cheng, W., Zhang, Q., Qin, W., Zheng, Y., Qiu, X., Huang, X., Gui, T.: The rise and potential of large language model based agents: A survey (2023), https://arxiv.org/abs/2309.07864
36. Xie, J., Chen, Z., Zhang, R., Wan, X., Li, G.: Large multimodal agents: A survey (2024), https://arxiv.org/abs/2402.15116
37. Xu, Q., Hong, F., Li, B., Hu, C., Chen, Z., Zhang, J.: On the tool manipulation capability of open-source large language models. arXiv preprint arXiv:2305.16504 (2023)
38. Yang, A., Xiao: Baichuan 2: Open Large-scale Language Models (Sep 2023). https://doi.org/10.48550/arXiv.2309.10305, http://arxiv.org/abs/2309.10305, arXiv:2309.10305 [cs]
39. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2022)
40. Yue, M., Zhao, J., Zhang, M., Du, L., Yao, Z.: Large Language Model Cascades with Mixture of Thoughts Representations for Cost-efficient Reasoning. Tech. rep. (Feb 2024), http://arxiv.org/abs/2310.03094, arXiv:2310.03094 [cs] type: article
41. Zhang, J., Krishna, R., Awadallah, A.H., Wang, C.: Ecoassistant: Using llm assistant more affordably and accurately (2023), https://arxiv.org/abs/2310.03046
42. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.Y., Wen, J.R.: A survey of large language models (2023), https://arxiv.org/abs/2303.18223

A Prompts Example of Booking

```
I have the following set of API:

# To set the departure date of the trip, given a Date object.
# This function must be called if booking type is 'trip tickets'.
# If booking type is 'both' and this function is not called explicitly,
# 'departure_date' will be set to 'hotel_checkin_date' implicitly.
```

```
API.set_departure_date(Date)

# To define a date.
date = Date(month, day, year)

# To set minimum ticket price.
API.set_min_ticket_price(value)

# < other apis ...
# The order of functions and tasks' examples has been shuffled
# to introduce more diveristy. >

-------------

I have the following set of examples:

Task: I'm interested in booking 2 Queen Bed room in Garden Grove for 5 night(s),
from Dec 17 2022 to Dec 22 2022. Find me something within my budget of 800 USD.
Action:
API.select_booking_type("hotels")
API.set_num_rooms(2)
API.select_room_type("Queen Bed")
location = Loc("Garden Grove")
API.set_hotel_location(location)
checkin_date = Date(12, 17, 2022)
API.set_checkin_date(checkin_date)
checkout_date = Date(12, 22, 2022)
API.set_checkout_date(checkout_date)
API.set_max_room_price(800)
API.search()

Task: < other task examples ... order shuffled >

-------------

Task: Find round trip flight tickets from Ontario to Kansas City for 4 adults,
leaving on Dec 19 2022 and returning on Dec 25 2022.
Action: [\n]
```

B Model Performance for OpenWeather

Model (Greedy Decoding)	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.857	100	0
GPT-4	1	100	0
Qwen-1.5-Int4	0.817	100	0
Baichuan2-turbo	0.747	100	0
Baichuan-13b	0.203	100	0
Claude-3-haiku	0.943	100	0
DeepSeek	0.99	100	0

Table 2: OpenWeather: Condition A (num=1; vote=1; fallback=no)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.86	300	0
Qwen-1.5-Int4	0.83	300	0
Baichuan2-turbo	0.81	300	0
Baichuan-13b	0.2	300	0
Claude-3-haiku	0.98	300	0
DeepSeek	1	300	0
GPT-4	1	300	0

Table 3: OpenWeather: Condition B (num=3; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.88	500	0
Qwen-1.5-Int4	0.85	500	0
Baichuan2-turbo	0.85	500	0
Baichuan-13b	0.17	500	0
Claude-3-haiku	0.98	500	0
DeepSeek-Coder	0.99	500	0
GPT-4	1	500	0

Table 4: OpenWeather: Condition C (num=5; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.89	300	3
Qwen-1.5-GPTQ	0.9	360	14
Baichuan2-turbo	0.83	300	4
Baichuan-13b	0.55	300	46
Claude-3-haiku	0.97	300	1
DeepSeek-Coder	0.99	300	0
GPT-4 (Greedy Decoding)	1	100	0

Table 5: OpenWeather: Condition D (num=3; vote=0.5; fallback=strong)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.9	500	1
Qwen-1.5-Int4	0.91	500	11
Baichuan2-turbo	0.84	500	6
Baichuan-13b	0.66	500	55
Claude-3-haiku	0.97	500	0
DeepSeek-Coder	0.98	500	0
GPT-4 (Greedy Decoding)	1	100	0

Table 6: OpenWeather: Condition E (num=5; vote=0.5; fallback=strong)

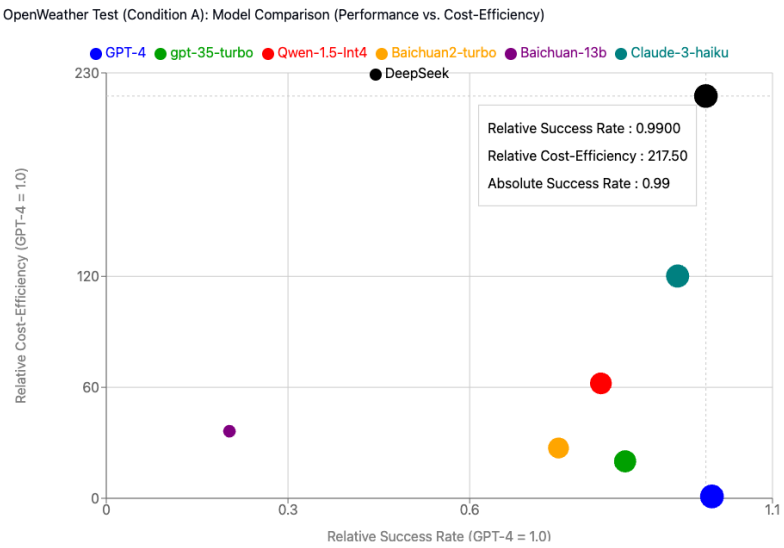

Fig. 6: OpenWeather A. The top-right corner indicates better performance for relative cost and relative success rate. The size of the bubble represents absolute accuracy. GPT-4 is shown as the blue bubble in the bottom-right corner as baseline.

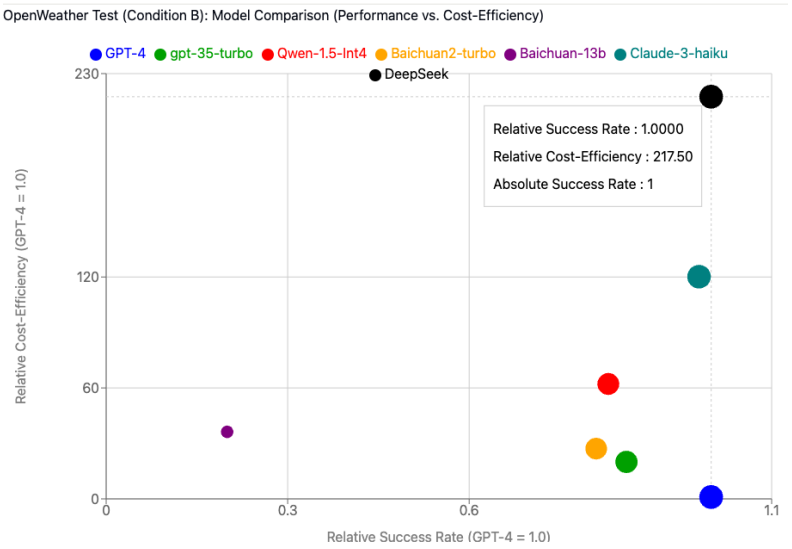

Fig. 7: OpenWeather B

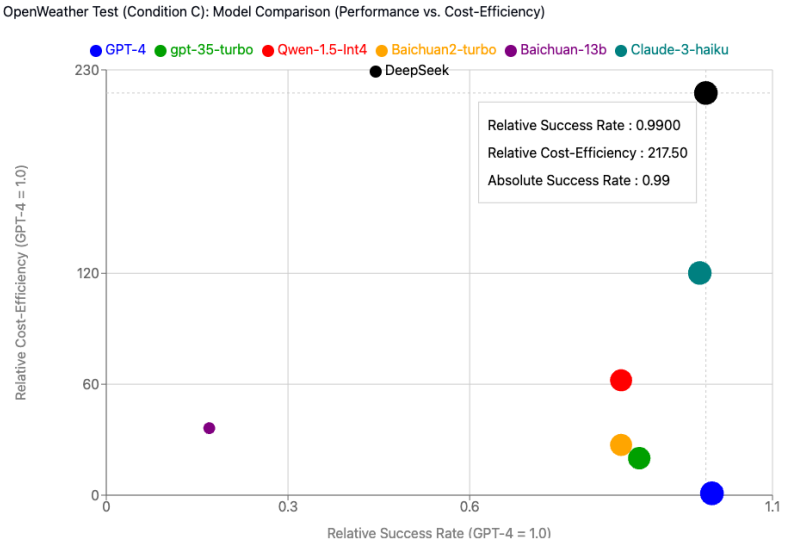

Fig. 8: OpenWeather C

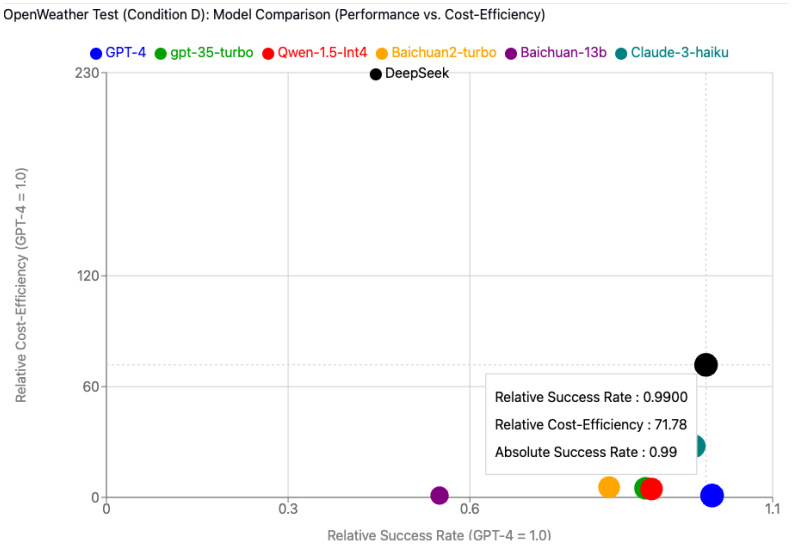

Fig. 9: OpenWeather D

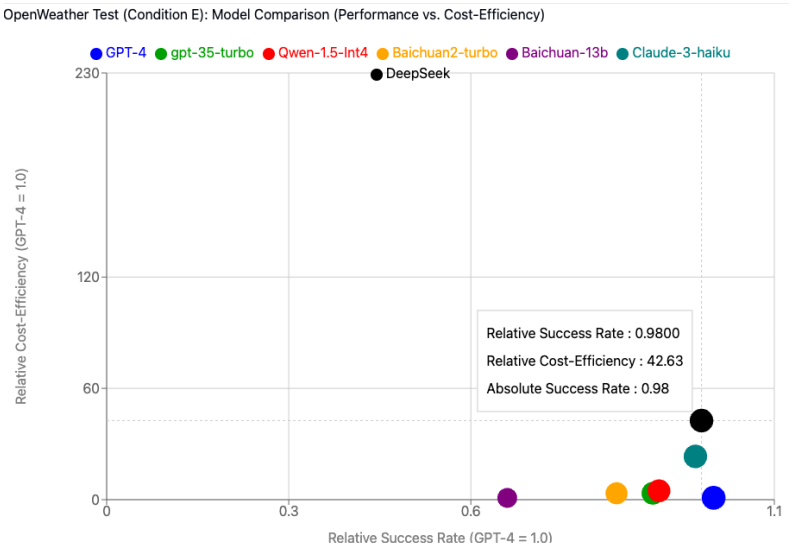

Fig. 10: OpenWeather E

C Model Performance for The Cat API

Model (Greedy Decoding)	Success Rate	Weak LLM Calls	Strong LLM Calls
Baichuan-13b	0.68	100	0
Baichuan2-turbo	0.860	100	0
GPT-35-turbo	0.95	100	0
GPT-4	0.95	100	0
Claude-3-haiku	0.909	100	0
Qwen-1.5-Int4	0.821	100	0
DeepSeek-Coder	0.96	100	0

Table 7: The Cat API: Condition A (num=1; vote=1; fallback=no)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
Baichuan-13b	0.7	300	0
Baichuan2-turbo	0.9	300	0
GPT-35-turbo	0.95	300	0
GPT-4	0.95	300	0
Claude-3-haiku	0.94	300	0
Qwen-1.5-Int4	0.96	300	0
DeepSeek-Coder	0.99	300	0

Table 8: The Cat API: Condition B (num=3; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
Baichuan-13b	0.74	500	0
Baichuan2-turbo	0.91	500	0
GPT-35-turbo	0.96	500	0
GPT-4	0.95	500	0
Claude-3-haiku	0.95	500	0
Qwen-1.5-Int4	0.96	500	0
DeepSeek-Coder	0.97	500	0

Table 9: The Cat API: Condition C (num=5; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
baichuan-13	0.77	300	7
Baichuan-turbo	0.93	300	0
GPT35	0.96	300	2
Haiku	0.95	300	4
Qwen	0.95	300	5
DeepSeek-Coder	0.97	300	0
GPT-4 (Greedy Decoding)	0.95	100	0

Table 10: The Cat API: Condition D (num=3; vote=0.5; fallback=strong)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
Baichuan-13b	0.74	500	6
Baichuan2-turbo	0.94	500	1
GPT-35-turbo	0.97	500	3
Claude-3-haiku	0.95	500	2
Qwen-1.5-Int4	0.97	500	8
DeepSeek-Coder	0.98	500	0
GPT-4 (Greedy Decoding)	0.95	100	0

Table 11: The Cat API: Condition E (num=5; vote=0.5; fallback=strong)

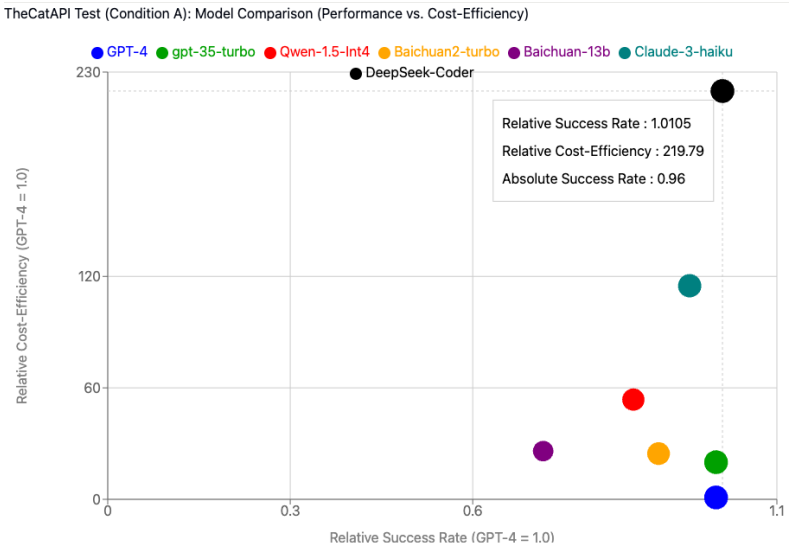

Fig. 11: The top-right corner indicates better performance with lower cost. For example, the dark bubble represents DeepSeek-Coder, which demonstrates more (1.0105) the accuracy of GPT-4 at only 1/219.79 of the cost compared to GPT-4.

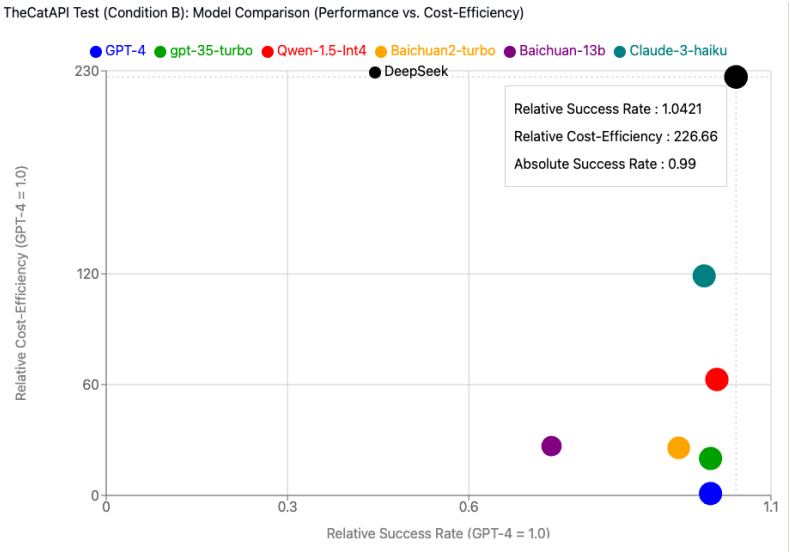

Fig. 12: The Cat API B

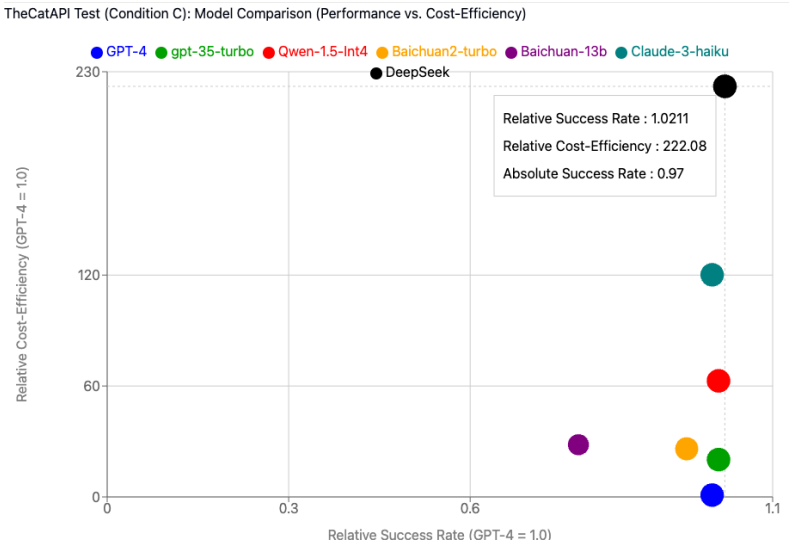

Fig. 13: The Cat API C

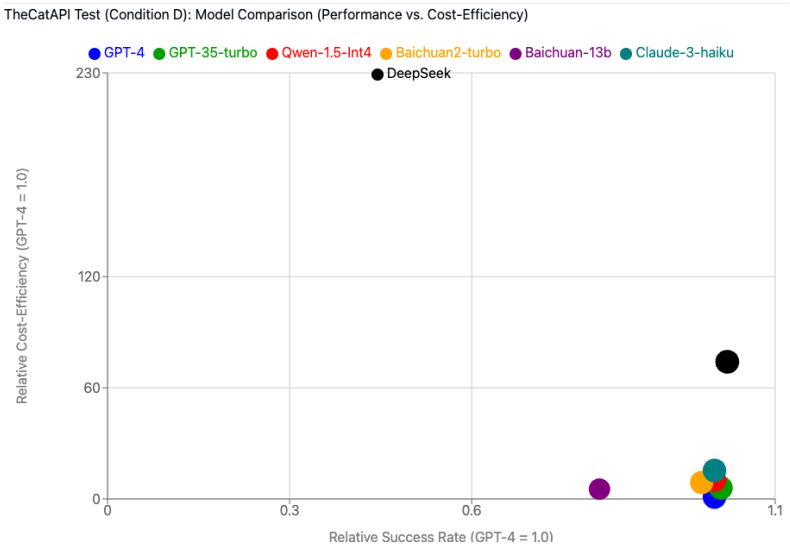

Fig. 14: The Cat API D

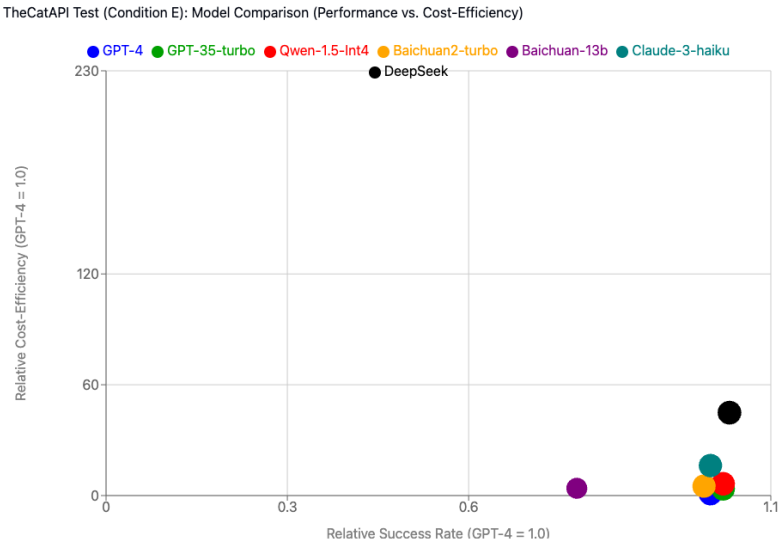

Fig. 15: The Cat API E

D Model Performance for HomeSearch

Model (Greedy Decoding)	Success Rate	Weak LLM Calls	Strong LLM Calls
GPT-4	0.95	100	0
gpt-35-turbo	0.89	100	0
Baichuan2-turbo	0.90	100	0
Qwen-1.5-Int4	0.93	100	0
Claude-3-haiku	0.95	100	0
Baichuan-13b	0.05	100	0
DeepSeek-Coder	0.86	100	0

Table 12: HomeSearch: Condition A (num=1; vote=1; fallback=no)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
GPT-4	0.96	300	0
gpt-35-turbo	0.90	300	0
Baichuan2-turbo	0.91	300	0
Qwen-1.5-Int4	0.96	300	0
Claude-3-haiku	0.97	300	0
Baichuan-13b	0.09	300	0
DeepSeek-Coder	0.90	300	0

Table 13: HomeSearch: Condition B (num=3; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
GPT-4	0.93	500	0
gpt-35-turbo	0.90	500	0
Baichuan2-turbo	0.90	500	0
Qwen-1.5-Int4	0.97	500	0
Claude-3-haiku	0.94	500	0
Baichuan-13b	0.12	500	0
DeepSeek-Coder	0.91	500	0

Table 14: HomeSearch: Condition C (num=5; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.89	300	0
Baichuan2-turbo	0.89	300	0
Qwen-1.5-Int4	0.97	300	0
Claude-3-haiku	0.98	300	0
Baichuan-13b	0.20	300	10
DeepSeek-Coder	0.91	300	0
GPT-4 (Greedy Decoding)	0.95	100	0

Table 15: HomeSearch: Condition D (num=3; vote=0.5; fallback=strong)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-35-turbo	0.89	500	0
Baichuan2-turbo	0.90	500	0
Qwen-1.5-Int4	0.98	500	0
Claude-3-haiku	0.96	500	0
Baichuan-13b	0.20	500	5
DeepSeek-Coder	0.89	500	0
GPT-4 (Greedy Decoding)	0.95	100	0

Table 16: HomeSearch: Condition E (num=5; vote=0.5; fallback=strong)

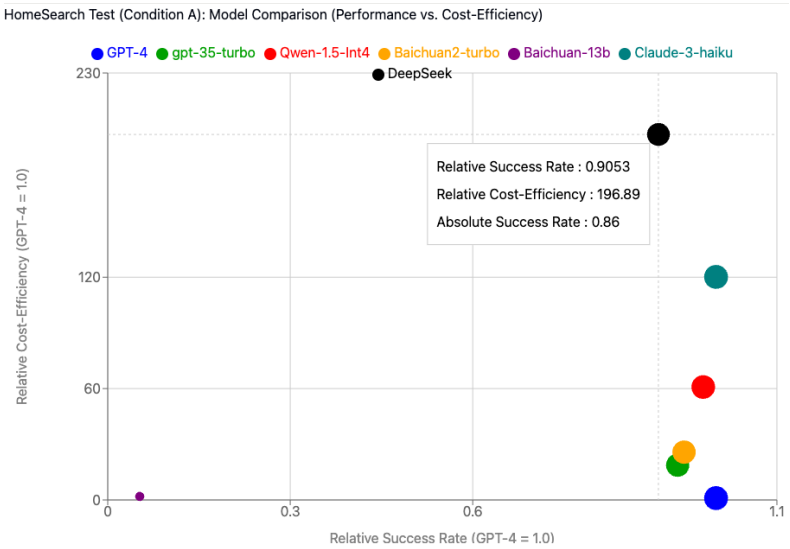

Fig. 16: HomeSearch A

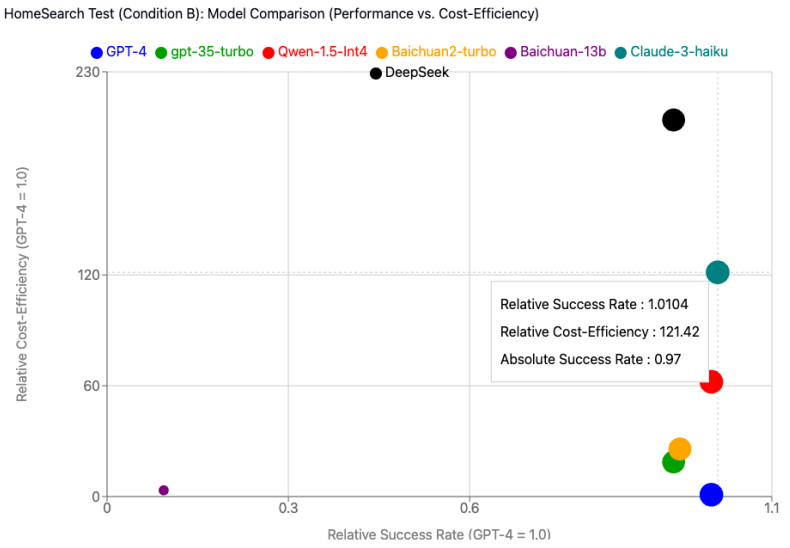

Fig. 17: HomeSearch B

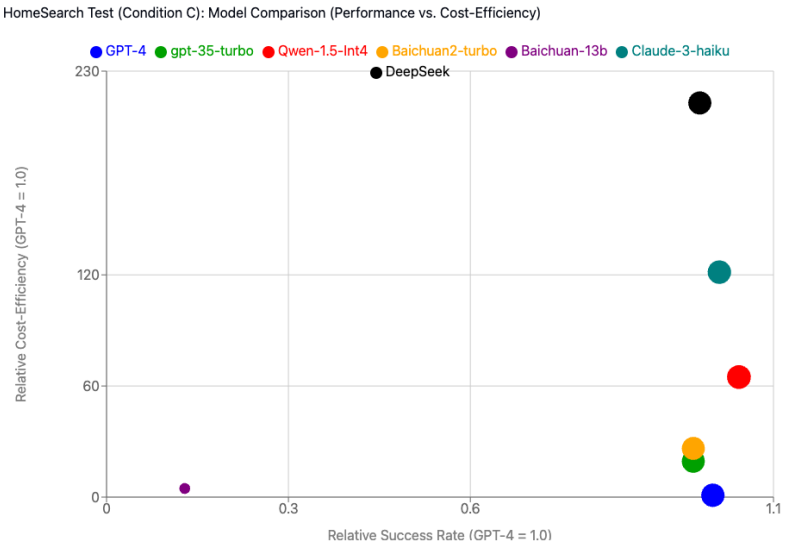

Fig. 18: HomeSearch C

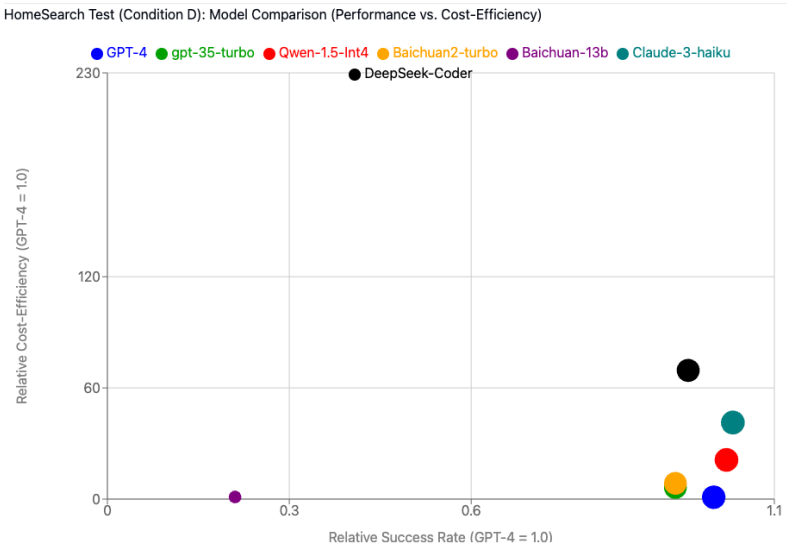

Fig. 19: HomeSearch D

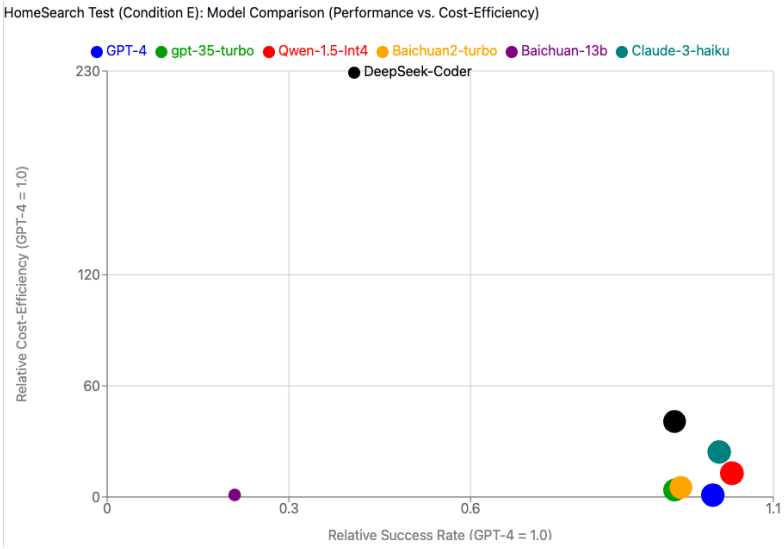

Fig. 20: HomeSearch E

E Model Performance for Booking

Model (Greedy Decoding)	Success Rate	Weak LLM Calls	Strong LLM Calls
GPT-4	0.883	120	0
gpt-3.5-turbo	0.833	120	0
baichuan-13	0.025	120	0
Baichuan2-turbo	0.633	120	0
Qwen-1.5-Int4	0.75	120	0
Claude-3-haiku	0.792	120	0
DeepSeek-Coder	0.175	120	0

Table 17: Booking: Condition A (num=1; vote=1; fallback=no)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
GPT-4	0.900	360	0
gpt-3.5-turbo	0.833	360	0
baichuan-13	0.042	360	0
Baichuan2-turbo	0.692	360	0
Qwen-1.5-Int4	0.742	360	0
Claude-3-haiku	0.767	360	0
DeepSeek	0.167	360	0

Table 18: Booking: Condition B (num=3; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
GPT-4	0.892	600	0
gpt-3.5-turbo	0.858	600	0
baichuan-13	0.058	600	0
Baichuan2-turbo	0.708	600	0
Qwen-1.5-Int4	0.775	600	0
Claude-3-haiku	0.800	600	0
DeepSeek	0.167	600	0

Table 19: Booking: Condition C (num=5; vote=0.5; fallback=weak)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-3.5-turbo	0.925	360	13
baichuan-13	0.792	360	106
Baichuan2-turbo	0.808	360	18
Qwen-1.5-Int4	0.817	360	19
Claude-3-haiku	0.850	360	13
DeepSeek	0.392	360	32
GPT-4 (Greedy Decoding)	0.883	120	0

Table 20: Booking: Condition D (num=3; vote=0.5; fallback=strong)

Model	Success Rate	Weak LLM Calls	Strong LLM Calls
gpt-3.5-turbo	0.908	600	20
baichuan-13	0.850	600	114
Baichuan2-turbo	0.842	600	29
Qwen-1.5-Int4	0.850	600	25
Claude-3-haiku	0.883	600	18
DeepSeek	0.375	600	32
GPT-4 (Greedy Decoding)	0.883	120	0

Table 21: Booking: Condition E (num=5; vote=0.5; fallback=strong)

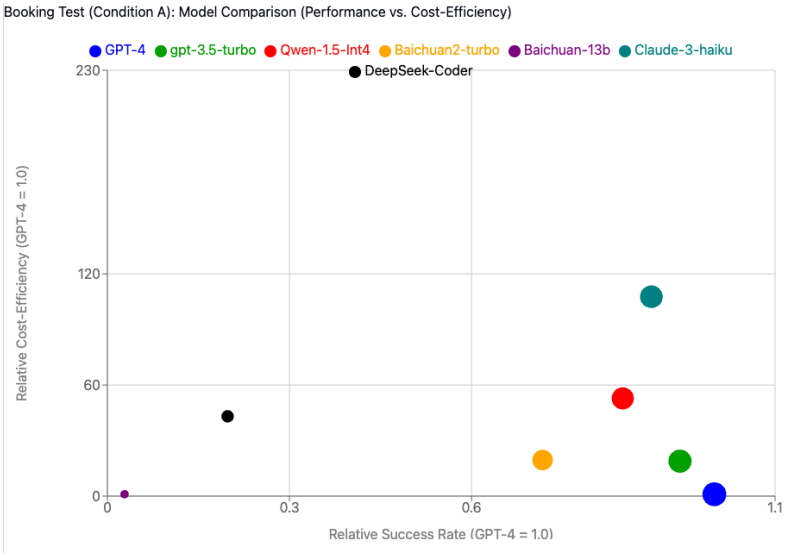

Fig. 21: Booking A

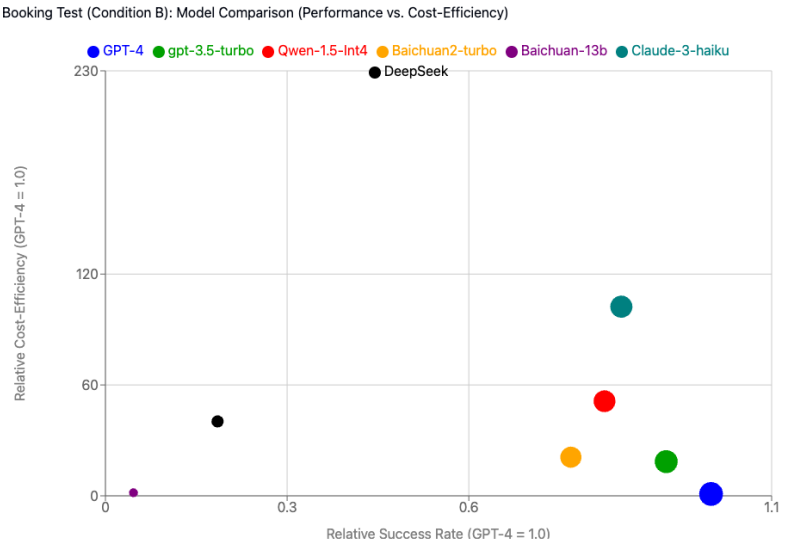

Fig. 22: Booking B

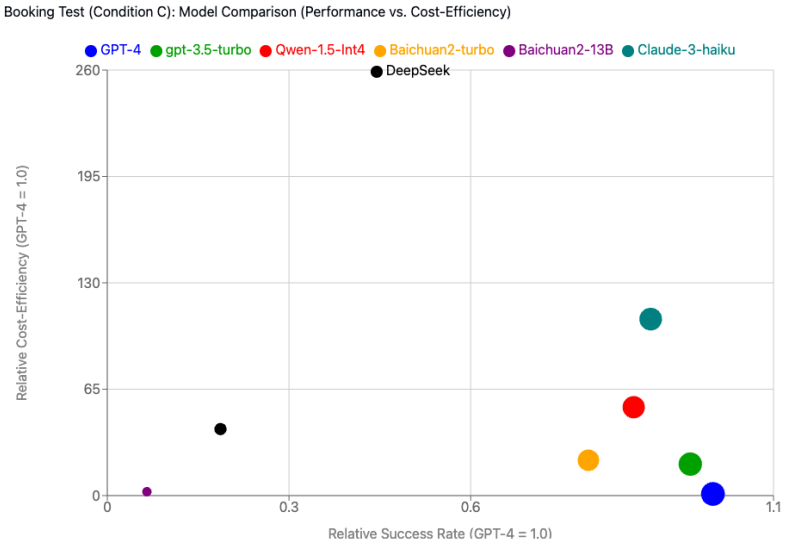

Fig. 23: Booking C

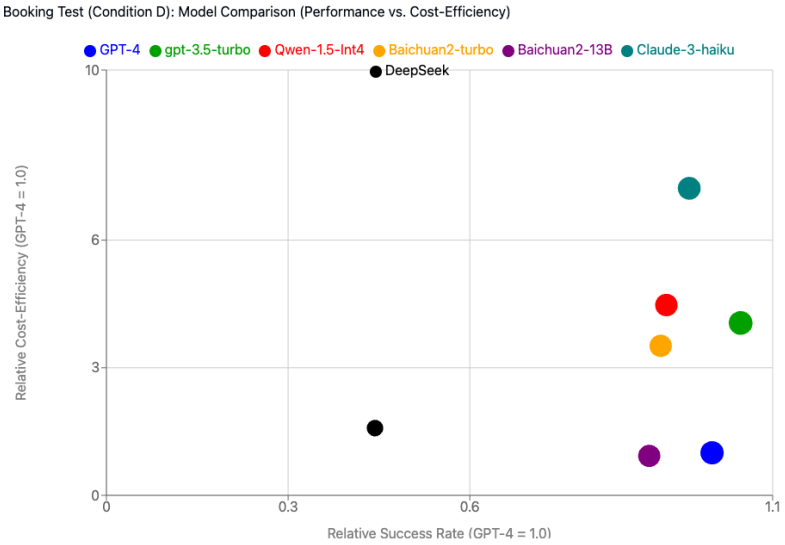

Fig. 24: Booking D. Compared to other cases, this test shows higher costs, but it requires less than 1/3 of the cost of GPT-4.

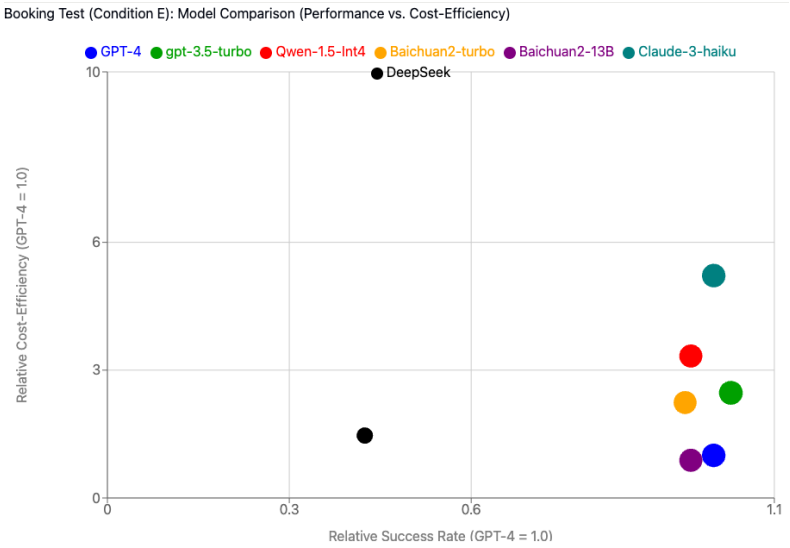

Fig. 25: Booking E