PricingLogic: Evaluating LLMs Reasoning on Complex Tourism Pricing Tasks

Anonymous ACL submission

Abstract

We present PricingLogic, the first benchmark that probes whether Large Language Models (LLMs) can reliably automate tourism-booking prices when multiple, overlapping fare rules apply. Travel agencies are eager to offload this error-prone task to AI systems; however, deploying LLMs without verified reliability could result in significant financial losses and erode customer trust. PricingLogic comprises 300 natural-language booking requests derived from 42 real-world pricing policies, spanning two levels of difficulty: (i) basic customer-type pricing and (ii) bundled-tour calculations involving interacting discounts. Evaluations of a line of LLMs reveal a steep performance drop on the harder tier, exposing systematic failures in rule interpretation and arithmetic reasoning. These results highlight that, despite their general capabilities, today's LLMs remain unreliable for revenue-critical applications without further safeguards or domain adaptation. [1]

1 Introduction

Recent advances in Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse domains, such as code generation (Chen et al., 2021, 2022; Hui et al., 2024), mathematical problem-solving (Hendrycks et al., 2021; Ahn et al., 2024) and general-purpose human instruction following (Zhou et al., 2023; Chen et al., 2024; Chiang et al., 2024). However, real-world deployment remains challenging, as practical applications require domain-specific knowledge, navigation of conflicting rules, and high reliability with minimal error tolerance, which are not fully captured by existing benchmarks (Zhou et al., 2024).

In this paper, we focus on a specific yet representative real-world task: automating pricing calculations for tourism bookings, in collaboration with travel agencies interested in using LLM-based systems to process booking requests expressed in *natural language* (Figure 1 left). These requests often involve multiple destinations, varied fare types, and dynamic price policies, making manual processing labor-intensive and error-prone. For LLMs, the task is also non-trivial, as it requires reasoning over complex constraints (Jiang et al., 2023).

To systematically evaluate LLMs on this problem, we introduce PricingLogic, a benchmark specifically designed to evaluate the capabilities of LLMs in handling realistic booking scenarios. We collected 42 real-world pricing policy documents and 300 booking requests. These requests cover two main tasks: basic customer-type pricing and more advanced bundled-tour calculations, presented in increasing levels of difficulty. Notably, in addition to standard prompting approaches, we also investigate code-assisted reasoning, which has been shown to enhance LLM performance on computational and logical tasks (Chen et al., 2022; Gao et al., 2023; Lyu et al., 2023, i.a.). In our approach, LLMs are first prompted to translate pricing policies into executable Python code. For each incoming booking request written in natural language, the model extracts relevant information and converts it into input arguments for the generated code, which then calculates the price. We find that this method significantly improves accuracy; nevertheless, challenges remain for complex booking requests (See Section 4). Our contributions are: (1) The first comprehensive benchmark for evaluating LLMs on real-world tourism pricing; (2) Two tasks with gradient difficulty that test LLMs' ability to handle overlapping discount rules; (3) A code-assisted reasoning approach that improves performance by separating rule interpretation from computation.

2 PricingLogic Construction

In this section we introduce PricingLogic for evaluating LLMs' reasoning abilities in tourism pricing

[1] Code and dataset will be released upon acceptance.

scenarios. We created two tasks with 300 questions distributed across three difficulty levels as shown in Table 1, enabling comprehensive evaluation from basic price lookups to complex calculations.

The difficulty levels test increasing reasoning complexity: Simple questions involve single customer types with basic pricing rules. Medium questions incorporate multiple variables (10+ visitors, mixed demographics, accommodation status) and 2-3 service combinations. Challenging questions present complex scenarios with large groups (25-55 visitors), diverse demographic compositions, region-specific pricing, multiple attractions, and overlapping discount structures.

Category	Count
Data Collection	
Individual attractions	33
Bundled attractions	9
Difficulty Distribution (per task)	
Simple questions	60
Medium questions	50
Challenging questions	40

Table 1: PricingLogic data statistics.

2.1 Collection and Organization of Tourism Products and Discount Policies

We collected PricingLogic through partnerships with travel agencies serving 7 scenic areas with 33 distinct activities. We documented pricing policies for nine customer types (regular visitors, contracted groups, seniors, students, etc.), capturing specific pricing structures, discount thresholds, and special conditions (accommodation benefits, combination incentives). This process revealed the complex conditional rules where prices vary based on customer categories and qualifications. We classified policies by location, activity type, client type, and conditions to generate realistic benchmarking questions.

2.2 Dataset and Task Setups

PricingLogic includes two tasks of increasing complexity, described as follows.

Task 1: Price policies for different customer types. Task 1 evaluates LLMs' ability to compute the total cost of tourism bookings using 33 pricing documents. Bundled packages are excluded from this task. We created 150 scenarios with clearly defined parameters: visitor classification (regular, contract, etc.), demographic thresholds (at least 80% students or at least 70% seniors), group size requirements (10 or more for group rates), and regional pricing variations, etc.

Task 2: Price policies for different customer types with bundled-tour discounts. Task 2 builds upon Task 1 by introducing bundled-tour discounts, which increase the problem's complexity. Multiple feasible pricing options (regular and preferential) can now exist for the same booking requests. This setup mirrors real-world tourism dynamics, where specific combinations of attractions receive preferential rates (lower total price than booking each attraction separately).

3 Methods

We apply two common reasoning approaches to solve the tasks defined in PricingLogic, and provide baseline performance. outlined below are the two methods.

End-to-end prompting (E2E). Our E2E approach processes pricing in a single inference pass. We index and normalize pricing policies, then consolidate relevant documents into a comprehensive context when processing orders. The prompt guides LLMs through two stages: (1) identifying project details, visitor counts, and special conditions, and (2) calculating prices based on applicable policies, including accommodation exemptions and combination requirements.

Code-assisted reasoning (CaR). We propose a two-stage approach for price calculation. First, LLMs generate specialized calculator functions for each tourism product using pricing policies as context, encapsulating conditional logic for customer types and discount rules. Second, our framework processes orders by: (1) identifying relevant items, (2) retrieving corresponding functions, (3) extracting parameter values, and (4) executing these functions to determine the final price.

4 Experiments

4.1 Experimental Setups

Models. We benchmark a line of recent LLMs including both proprietary ones and open-weight ones. Specifically, we have included GPT-4o (OpenAI, 2024), DeepSeek-V3/R1 (DeepSeek-AI et al., 2025b,a), and Qwen2.5-7B/32B/Max (Qwen et al.,

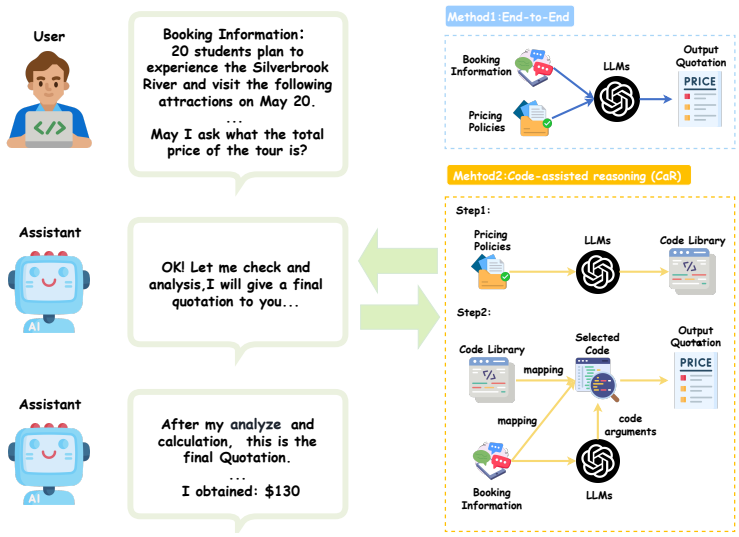

Figure 1: Automatic quotation use case (left) and its two LLM-based realizations (right).

Infer. setting	Model	Question types		
		simple	medium	challenging
E2E	Qwen2.5-7B	63.33	12.00	0.00
	Qwen2.5-32B	86.67	40.00	50.00
	Qwen2.5-Max	**90.00**	54.00	32.50
	DeepSeek-V3	83.33	70.00	40.00
	DeepSeek-R1	78.33	**72.00**	45.00
	GPT-4o	81.67	58.00	**52.50**
CaR	Qwen2.5-7B	66.66$^{3.3\uparrow}$	28.00$^{16.0\uparrow}$	5.00$^{5.0\uparrow}$
	Qwen2.5-32B	93.33$^{6.7\uparrow}$	68.00$^{28.0\uparrow}$	35.00$^{15.0\downarrow}$
	Qwen2.5-Max	92.00$^{2.0\uparrow}$	**78.00**$^{24.0\uparrow}$	55.00$^{22.5\uparrow}$
	DeepSeek-V3	90.00$^{6.7\uparrow}$	70.00$^{0.0\uparrow}$	50.00$^{10.0\uparrow}$
	DeepSeek-R1	93.33$^{15.0\uparrow}$	74.00$^{2.0\uparrow}$	**57.50**$^{12.5\uparrow}$
	GPT-4o	**96.67**$^{15.0\uparrow}$	72.00$^{14.0\uparrow}$	55.00$^{2.5\uparrow}$
CaR-Oracle	Qwen2.5-7B	15.00$^{51.7\downarrow}$	6.00$^{22.0\downarrow}$	0.00$^{5.0\downarrow}$
	Qwen2.5-32B	96.67$^{3.3\uparrow}$	92.00$^{24.0\uparrow}$	30.00$^{5.0\downarrow}$
	Qwen2.5-Max	**100.00**$^{8.0\uparrow}$	82.50$^{4.5\uparrow}$	**55.00**$^{0.0\uparrow}$
	DeepSeek-V3	**100.00**$^{10.0\uparrow}$	85.00$^{15.0\uparrow}$	52.50$^{2.5\uparrow}$
	DeepSeek-R1	**100.00**$^{6.7\uparrow}$	**92.50**$^{18.5\uparrow}$	**55.00**$^{2.5\uparrow}$
	GPT-4o	97.50$^{0.8\uparrow}$	85.00$^{13.0\uparrow}$	50.00$^{0.0\uparrow}$

Table 2: Main Task 1 results. Superscripts denote performance changes between successive evaluation settings (E2E → CaR, CaR → CaR-Oracle).

2025).[2]

Inference settings. As outlined in Section 3, we evaluate LLMs using both E2E and CaR approaches. CaR has two potential failure modes: (1) generating incorrect calculation code and (2) invoking code with incorrect parameters. To isolate error sources, we introduced **CaR-Oracle**, where we manually implemented verified Python code for all pricing policies. In this control condition, LLMs only need to pass correct parameters to human-verified code, enabling precise diagnosis of model limitations by controlling for code quality. We set the temperature to 0.0 across all models for deterministic outputs.

Metrics. We use exact match to compare the model predictions with the correct answer, and report the accuracy.

4.2 Results on Task 1

Table 2 presents the model performance on Task 1. For simple questions, all LLMs, except Qwen2.5-7B, are able to provide correct answers 78% of the time using direct prompting (E2E). However, performance drops as question complexity increases. Upon inspection, we find that models often misidentify customer categories and/or overlook pricing conditions. For challenging questions, all LLMs struggle to answer more than half correctly.

Compared to E2E, the CaR approach improves accuracy in most cases, demonstrating the effectiveness of this two-stage inference framework. The greatest improvement is observed in the Qwen2.5-Max model, suggesting that while it may struggle with end-to-end reasoning, it performs well when tasked with code generation. Nonetheless, even with CaR, no LLM achieves more than 60% accuracy on challenging questions. A notable outlier is Qwen2.5-32B, where CaR underperforms E2E by 15%. Further analysis shows that this model often makes mistakes when converting booking information into correct function calls.

Results from CaR-Oracle confirm that using human-verified code leads to further improvements in accuracy. Strong models like DeepSeek-V3 can fully solve simple tasks with human-written code, achieving a 10% gain compared to using self-

[2]Model versions: `GPT-4o-0129`, `Qwen-Max-1015` and `DeepSeek-V3-0324`.

Infer. setting	Model	Question types		
		simple	medium	challenging
E2E	Qwen2.5-7B	68.33	28.00	0.00
	Qwen2.5-32B	76.67	46.00	27.50
	Qwen2.5-Max	85.00	48.00	22.50
	DeepSeek-V3	83.33	45.00	27.50
	DeepSeek-R1	88.33	40.00	**30.00**
	GPT-4o	**90.00**	**54.00**	27.50
CaR	Qwen2.5-7B	61.67$^{6.7\downarrow}$	22.00$^{6.0\downarrow}$	12.50$^{12.5\uparrow}$
	Qwen2.5-32B	76.67$^{0.0\uparrow}$	42.00$^{4.0\downarrow}$	22.50$^{5.0\downarrow}$
	Qwen2.5-Max	93.33$^{8.3\uparrow}$	**78.00**$^{30.0\uparrow}$	35.00$^{12.5\uparrow}$
	DeepSeek-V3	91.67$^{8.3\uparrow}$	68.00$^{23.0\uparrow}$	30.00$^{2.5\uparrow}$
	DeepSeek-R1	93.33$^{5.0\uparrow}$	70.00$^{30.0\uparrow}$	35.00$^{5.0\uparrow}$
	GPT-4o	**95.00**$^{5.0\uparrow}$	76.00$^{22.0\uparrow}$	**37.50**$^{10.0\uparrow}$

Table 3: Main Task 2 results. Superscripts denote performance changes between E2E → CaR evaluation settings.

generated code. These results indicate that, for simple tasks, LLM-generated code is generally accurate, though it still misses some edge cases. For medium-difficulty tasks, generated code often contains significant issues, but strong LLMs may still map booking requests to correct code arguments. For challenging tasks, even understanding bookings and making correct function calls becomes problematic.

4.3 Results on Task 2

Table 3 presents the model performance on Task 2. With E2E prompting, even the strongest models (GPT-4o) achieve only 27.5% accuracy on challenging questions, demonstrating the difficulty introduced by having to reason about bundled-discount interactions. The CaR approach shows substantial improvements for most models across difficulty levels. Particularly notable are the gains for Qwen2.5-Max, DeepSeek-V3, and DeepSeek-R1 on medium difficulty questions, with improvements of 30%, 23%, and 30% respectively.

The CaR approach's success demonstrates that separating policy interpretation from parameter extraction improves handling of complex pricing logic. Error analysis reveals that models struggle with two specific challenges in Task 2: (1) identifying when bundled discounts should override other customer type pricing, and (2) calculating the optimal combination when multiple valid bundle options exist.

5 Related work

LLMs in real-world scenarios. Recent research has focused on evaluating LLMs in real-world applications. Miserendino et al. (2025) benchmarked LLMs for freelance software engineering, and Huang et al. (2024) assessed their tool utilization in real-world scenarios. Closely related to our work is RuleArena (Zhou et al., 2024) that tests LLMs' rules-following in real-world domains. Unlike RuleArena's linear difficulty scaling (e.g., increasing bag count) and minimal rule conflicts. Our benchmark evaluates LLMs' ability to select optimal pricing among multiple overlapping conditions across diverse demographics, accommodation status, and service combinations, requiring sophisticated comprehension to identify the most favorable option among competing discount rules.

Code-assisted reasoning. Assisting LLMs with code improved their reasoning on computation-intensive tasks, (Lyu et al., 2023), through generating programmatic steps executed by external interpreters. Methods either employ pure code (Chen et al., 2022; Gao et al., 2023), code-language interleaving (Lyu et al., 2023), code with algebraic expressions, (Imani et al., 2023), or code with specialized libraries (Das et al., 2024). While previous work targeted controlled mathematical problems, our approach extends this paradigm to real-world tourism pricing, exceeding textbook problem complexity, through a two-stage pipeline addressing practical constraints such as diverse customer groups and overlapping discount rules.

6 Conclusion

We introduced PricingLogic, a benchmark evaluating LLMs on complex tourism pricing tasks. Our experiments show code-assisted reasoning generally outperforms end-to-end approaches, yet even advanced models struggle with challenging pricing scenarios involving multiple overlapping rules. These findings highlight the gap between theoretical reasoning capabilities and practical deployment needs in revenue-critical applications, emphasizing the importance of rigorous evaluation before implementing AI in financial contexts.

Limitations

We focused only on E2E prompting and CaR methods for evaluating LLMs on pricing tasks. While fine-tuning LLMs specifically for tourism pricing could potentially improve performance, it would require substantial training data, more computational resources, and retraining whenever pricing policies change—making it impractical in dynamic business environments. Our methods offer some flexibility while still providing meaningful performance benchmarks.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Yihan Chen, Benfeng Xu, Quan Wang, Yi Liu, and Zhendong Mao. 2024. Benchmarking large language models on controllable generation under diversified instructions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17808–17816.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot arena: An open platform for evaluating llms by human preference. *Preprint*, arXiv:2403.04132.

Debrup Das, Debopriyo Banerjee, Somak Aditya, and Ashish Kulkarni. 2024. Mathsensei: A tool-augmented large language model for mathematical reasoning. *arXiv preprint arXiv:2402.17231*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025b. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, and 1 others. 2024. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *arXiv preprint arXiv:2401.17167*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2023. Followbench: A multi-level fine-grained constraints following benchmark for large language models. *arXiv preprint arXiv:2310.20410*.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. 2025. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*.

OpenAI. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

Ruiwen Zhou, Wenyue Hua, Liangming Pan, Sitao Cheng, Xiaobao Wu, En Yu, and William Yang Wang. 2024. Rulearena: A benchmark for rule-guided reasoning with llms in real-world scenarios. *CoRR*, abs/2412.08972.

A Prompts

The E2E prompt is shown in Figure 3. The CaR prompt is shown in Figures 4 to 6. The Benchmark information is shown in Figure 2

B annotation process

The PricingLogic dataset was manually collected and annotated by the authors over a four-day period. We first established clear definitions for various customer types and documented their corresponding pricing structures for each tourism attraction. Table 4 illustrates an example of the pricing policy for a specific attraction, showing how prices vary across different customer categories. Table 5 provides detailed definitions of these customer categories, explaining the qualifications and conditions for each pricing tier.

C computing infra

Experiments in this work were conducted on a mixed infrastructure setup, with some models run locally and others accessed via API endpoints. For open-source models, experiments were conducted with different GPU configurations. Qwen2.5-7B was run on a single Nvidia A800 GPU card (80GB), while Qwen2.5-32B required 4 A800 GPUs for inference. The server was equipped with Intel(R) Xeon(R) Platinum 8378A CPU @ 3.00GHz processors. Batch processing was implemented to optimize throughput across all experimental runs. For larger proprietary models (Qwen2.5-Max, DeepSeek-V3, DeepSeek-R1, and GPT-4o), we utilized their respective API endpoints. The API calls were managed through a queuing system to handle rate limits and ensure reliable data collection. All API requests were executed with temperature set to 0.0 to ensure deterministic outputs.

Type	**Price**
Regular retail price	80
Contracted group price	50
Contracted non-group price	50
Non-contracted group price	64
Non-contracted non-group price	72
Senior/Student group price	40
Long-distance and new market price	30
Accommodation package price	50
Travel employee price	50
Free admission with hotel stay	0

Table 4: Example Customer Type Price of one Attraction

Customer Type	**Definition**
Regular retail price	Standard price for individual visitors
Group price	Applies when the number of visitors is ≥ 10 people
Contracted group price	Discounted rate for customers with a signed contract
Contracted non-group price	Price for contracted customers who don't meet group size requirement
Non-contracted group price	Group rate for customers without a contract (requires tour guide certificate)
Non-contracted non-group price	Standard price for customers without a contract
Senior group price	Applies when seniors (55+) constitute $> 70\%$ of the group
Student group price	Applies when students constitute $> 80\%$ of the group
Long-distance market price	Special price for visitors from outside Somerset, Hampshire, and London
Accommodation package price	Preferential prices for contracted groups staying at designated hotels in Clayton Castle
Travel employee price	Special price for travel employees and companions; applies to entire group when led by an employee
Free admission with hotel stay	Visitors at designated Clayton Castle hotels receive free admission to select attractions

Table 5: Customer Type Definitions

Benchmark Information

simple questions:

3 non-contract customers plan to experience the Harrenstadt Bay tour route. What is the total price for the Harrenstadt Bay tour route?

medium questions:

12 tourists (non-contract customers from Essex) are visiting Brighton Cave and St. Elvi Ancient Village. They plan to experience the Brighton Cave entrance ticket and St. Elvi Ancient Village entrance ticket. What is the total price for the Brighton Cave entrance ticket and St. Elvi Ancient Village entrance ticket?

challenging questions:

25 tourists (contract customers, including 12 students and 6 seniors, staying at a designated hotel in Clayton Castle) are visiting Brighton Cave, St. Elvi Ancient Village, and Montfiel Monastery. They plan to experience the Brighton Cave entrance ticket, Brighton Cave boat ride, Brighton Cave magic carpet ascent, St. Elvi Ancient Village entrance ticket, and Montfiel Monastery entrance ticket. What is the total price for the Brighton Cave entrance ticket, Brighton Cave boat ride, Brighton Cave magic carpet ascent, St. Elvi Ancient Village entrance ticket, and Montfiel Monastery entrance ticket?

Figure 2: Benchmark Information.

E2E method prompt

You are a tourism pricing expert. Please complete two tasks based on the following order information and pricing policies: project identification and price calculation.
Order Information:
{order_text}
Pricing Policies:
{all_policy_content}
Please complete the following two tasks:
Task 1: Identify Project Information
Please analyze the order text to identify the tourism projects, number of people, dates, and any special identities or conditions mentioned.
Task 2: Calculate Price
Based on the project information you've identified and the corresponding pricing policies, calculate the total price of the order.
Please follow the rules in the pricing policies, considering information such as the number of people, dates, special identities or conditions in the order to determine the customer type and accurately calculate the price. If the order meets multiple customer conditions, choose the customer type that results in the lowest price.
In your response, first clearly list the project information you've identified (including project name, number of people, etc.), then explain the calculation process in detail.
Finally, be sure to output the total price on the last line of your answer in the following fixed format:
Final Price:XXXX yuan
Please ensure this line stands alone without any other text, where XXXX is the final price number you calculated. This is important for the system to extract the price.

Figure 3: E2E method prompt.

code generation prompt

Please deeply analyze the pricing policy document and generate a price calculator module. Please respond in Chinese.
Task Requirements
1. Create a function named calculate_price
2. The function should calculate the total expense based on the number of people
3. The function should handle various edge cases and special situations
4. If multiple pricing policies apply, choose the one most beneficial to the customer
5. The code should be concise, efficient, and easy to understand
6. Do not add any additional functions or classes
7. Do not import any unnecessary modules
8. Do not add any other content besides the calculation function
Pricing Policy Document
{document_content}
Calculation Function Requirements
Please determine the parameters needed for the function based on the content of the pricing policy document, and implement the complete calculation logic.
The following requirements apply to the function parameters and calculation logic:
1. The entire code's decision logic should be in a complete if-elif-else structure
2. The code should be able to calculate the unit price of each item in the combination policy, as well as the total price of all items in the combination policy
Please generate complete Python code directly, without any additional explanations or modules.

Figure 4: CaR method step1 prompt.

CaR method prompt1

Please analyze the following tourism order text and identify the projects needed by the customer.

Order text:
"{order_text}"

Available project list:
{', '.join(available_items)}

Please only return a JSON array containing the identified project names, for example:
["Project1", "Project2"]

Figure 5: CaR method step2 prompt for booking information analysis.

CaR method prompt2

Please parse the following tourism order text and extract all key information that may affect price calculation.
Order text:
"{order_text}"
Identified projects:
{', '.join(identified_items)}
Corresponding project function code, please carefully analyze the source code of each calculator function, paying special attention to how parameters are actually used within the function.
{calculator_code_prompt}
User question:
"{question if question else 'Calculate the total price of the order'}"
Please first provide a detailed explanation of your understanding of this order, including:
1. The key information you've identified (such as number of people, dates, customer type, etc.)
2. How you determined which parameters to extract based on the price calculator code
3. If you feel the information provided in the order is insufficient, you may refuse to answer and point out where information is lacking
4. Please refer to this supplementary policy document when understanding the order: {policy_content}
Then, please carefully analyze the order text to extract all factors that may affect the price, such as number of people, customer type, special identity, date, etc.
Please carefully understand the price calculation code for specific projects mentioned in the order information, and extract the corresponding information according to its parameter requirements.
Finally, please return the results in strict JSON format, without any comments, and must include the "items" field and all parameters required by the calculator functions.
Your response should include two parts:
1. Your analysis and understanding of the order (textual explanation)
2. Extracted parameters (JSON format)

Figure 6: CaR method step2 prompt for code arguments analysis.