
How to Get Your LLM to Generate Challenging Problems for Evaluation

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 The pace of evolution of Large Language Models (LLMs) necessitates new ap-
2 proaches for rigorous and comprehensive evaluation. Traditional human annotation
3 is increasingly impracticable due to the complexities and costs involved in gener-
4 ating high-quality, challenging problems. In this work, we introduce **CHASE**, a
5 framework to synthetically generate challenging problems using LLMs without
6 human involvement. For a given task, our approach builds a difficult problem in a
7 bottom-up manner from simpler components in a verifiable way. We implement
8 CHASE to create evaluation benchmarks across three diverse domains on which
9 state-of-the-art LLMs demonstrate severe vulnerabilities.

10 1 Introduction

11 We have witnessed the emergence of powerful Large Language Models (LLMs) (OpenAI Team et al.,
12 2024) that exhibit remarkable performance over a wide range of tasks. However, the resources for
13 evaluating these models have not kept pace with their rapid evolution. Contemporary LLMs have
14 saturated many existing reasoning benchmarks (Chen et al., 2021, Cobbe et al., 2021). Developing
15 challenging problems is both expensive and time-consuming, especially for non-expert human
16 annotators. Hence, relying solely on human annotation to build evaluation benchmarks is inherently
17 limited, and LLMs offer a promising alternative for generating high-quality evaluation data.

18 Synthetic data generation has emerged as a powerful paradigm in recent years driven by the wide-
19 spread availability of cheaper and faster LLMs that can effectively follow instructions (Wang et al.,
20 2023, Xu et al., 2024). However, using synthetic data for evaluation has been relatively underex-
21 plored. There are considerable advantages in using synthetic data for evaluation: it is comparatively
22 inexpensive, highly scalable, and can be renewed periodically to mitigate contamination concerns.
23 However, there are two main challenges: first, *how can we create **difficult** and **realistic** problems?*
24 and second, *how can we automatically **verify the correctness** of the generated data?*

25 In this work, we present the **CHASE** framework: **CH**allenging **AI** with **S**ynthetic **E**valuations. Our
26 methodology is based on two straightforward ideas (see Figure 1). First, we create problems in a
27 *bottom-up* manner, iteratively hiding parts of the solution within the problem’s context. Second, we
28 decompose the generation process into simpler, *individually verifiable sub-tasks*. We implemented
29 our framework to create challenging benchmarks across three diverse domains: (1) **CHASE-QA**,
30 a document-based question answering benchmark; (2) **CHASE-CODE**, a repository-level code
31 completion benchmark; (3) **CHASE-MATH**, a benchmark of grade-school level word problems.
32 Experiments with **27** contemporary LLMs show that the datasets generated using CHASE are
33 challenging for all models (accuracies of $\sim 40 - 70\%$). Our results also reveal large gaps in
34 performance between different LLMs that perform similarly on existing benchmarks like MMLU
35 (Hendrycks et al., 2021). Lastly, we show that CHASE is *scalable*, allows a high degree of control
36 over the *difficulty* of generated data, and provides a high level of *correctness*.

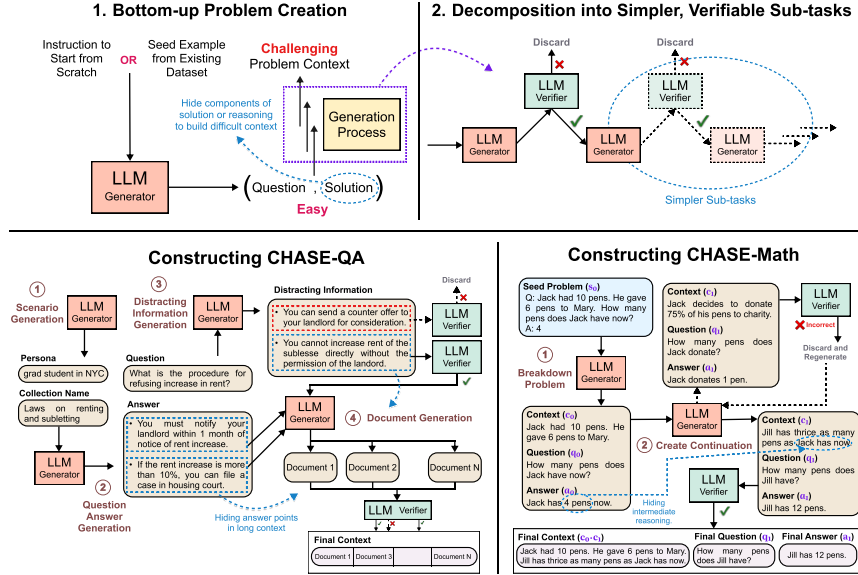


Figure 1: *Top*: Illustrating the high-level ideas behind our proposed CHASE framework. *Bottom left*: Pipeline for creating an example in CHASE-QA. *Bottom right*: Pipeline for creating a math word problem in CHASE-MATH. The pipeline for CHASE-CODE is illustrated in Figure 3 in Appendix.

2 The CHASE Framework and Benchmarks

Our framework for generating synthetic data is based on two simple ideas as illustrated in Figure 1.

1. Bottom-up problem creation. We take a bottom-up approach to synthetic data generation. We either generate or start with a simpler problem-solution pair, and incrementally enrich the problem context in ways that increase reasoning difficulty. Specifically, we obscure, divide, or distribute key elements of the reasoning process or solution, thereby requiring the solver to undertake multiple steps of inferences to arrive at the solution.

2. Decomposition into simpler, verifiable sub-tasks. We design pipelines that break down the generation process into simpler sub-tasks. Each individual LLM in the pipeline performs a simpler, specific function in the generation process. This provides us with multiple benefits. First, it grants us more control over each step of the generation process. Second, and perhaps more importantly, it facilitates fine-grained verification. We deploy LLMs that are not part of the generation process to check the correctness and quality of the generated data at each possible step.

We demonstrate the adaptability of our framework by generating challenging problems in three domains. **CHASE-QA** is a document-grounded QA task with 671 problems (Figure 2 left), requiring reasoning over long contexts ($>6k$ tokens) as relevant information spans multiple documents. **CHASE-CODE** is a repository-level code completion benchmark with 500 problems (Figure 2 centre), where models implement new Python functions based on specified objectives. **CHASE-MATH** comprises 500 grade-school math word problems involving basic arithmetic (Figure 2 right). Below, we briefly describe construction pipelines for CHASE-QA and CHASE-MATH (refer Figure 1). Details for all construction pipelines are provided in Appendix E.

Constructing CHASE-QA. We first generate diverse realistic scenarios where a *user persona* seeks information from a *collection of documents*. For example, a ‘grad student in NYC’ searching ‘laws on renting and subletting’. We prompt the generator G to produce scenarios as tuples $(\text{persona}, \text{collection_name})$, bootstrapping with 5 annotated examples and then iteratively expanding with its own outputs. Given a scenario, we then prompt G to create a realistic information-seeking question and its answer. For instance, a ‘grad student’ might ask, ‘what is the procedure for refusing increase in rent?’, with the answer spread across multiple government documents. G must produce answers composed of multiple points, along with an outline of documents (title and abstract) where each point is assigned. To increase difficulty, for each QA pair, G also generates

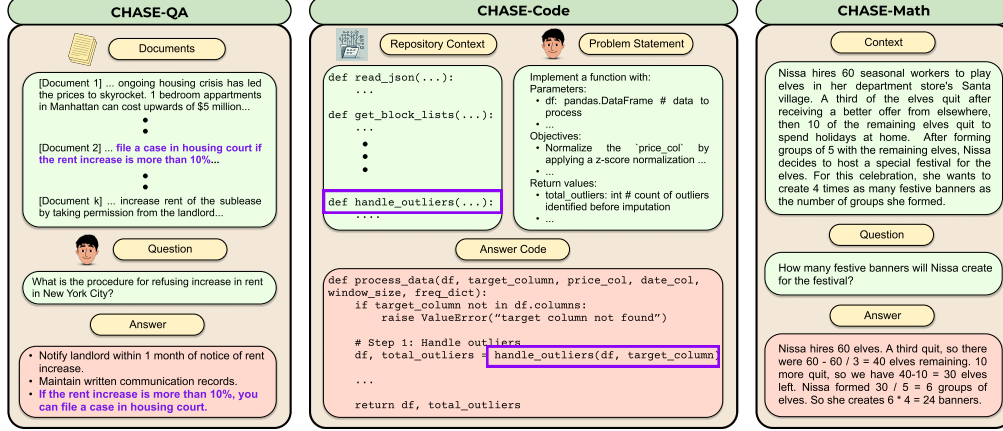


Figure 2: Examples of problems from all three benchmarks created using CHASE.

67 *distracting* QA pairs with answers of a similar type but irrelevant to the original question (e.g., ‘how
68 do I increase the rent for an apartment I am subletting?’). To ensure correctness, we prompt the
69 verifier **V** to check that none of the distractors are actually relevant (Figure 1 bottom left). Finally,
70 for each QA and distracting QA pair, **G** generates long documents that include the assigned answer
71 points plus irrelevant details. These documents collectively form the context. We verify correctness
72 with **V** to ensure: (1) no extra relevant information appears beyond ground-truth answers, and (2) all
73 answer points are fully covered.

74 **Constructing CHASE-MATH.** We start by sampling seed math word problems (MWP) from
75 existing datasets. A seed MWP $s = (p, a)$ has problem p and answer a . We prompt generator **G** to
76 split p into context c (given information) and question q (unknown to find). Given the initial seed
77 MWP, $s_0 = (p_0, a_0)$, **G** produces $s_1 = (p_1, a_1)$ where the new context c_1 implicitly assumes a_0 as
78 known, and generates a new question q_1 whose answer a_1 is obtained by an arithmetic operation over
79 a_0 . Merging s_0 with s_1 yields a higher-depth MWP $s = (p, a)$ with context $c = c_0 \cdot c_1$, question
80 $q = q_1$, and answer $a = a_1$ (Figure 1, bottom right). We repeat this process: each s_i becomes the
81 seed for s_{i+1} . After j successful iterations (depth $j+1$), the final problem has $c = c_0 \cdot c_1 \cdots c_j$,
82 question q_j , and answer $a = a_j$. Because each s_i is low-depth and comparable to seed data, we verify
83 correctness using a non-identical ensemble of verifiers $\{V_1, \dots, V_n\}$ that perform well on the seed
84 set. Each V_k is prompted with (c_i, q_i) ; if any prediction differs from a_i , we discard s_i and restart
85 from s_{i-1} .

86 3 Experiments

87 **Implementation Details.** The exact details associated with generating each benchmark are provided
88 in Appendix F. The prompts are provided in Appendix J. For evaluation of CHASE-CODE, we
89 measure the pass@1 execution accuracy. For CHASE-MATH, we measure the exact match accuracy
90 against the ground-truth numerical answer. Since, the ground-truth answers for CHASE-QA are
91 verbose text, we deploy an LLM-as-a-judge to automatically assess the correctness of predictions.
92 We use GPT-4o as the judge and measure the accuracy as the percentage of predictions judged to
93 be correct. We evaluated a total of 27 different LLMs, including reasoning models. Other details
94 of implementation, such as details of LLMs and human evaluation of LLM judge are provided in
95 Appendix F.

96 **Results and Discussion.** The performance of all LLMs on CHASE benchmarks can be seen in
97 Table 1. Even the state-of-the-art reasoning models achieve performance only in the range of 50-70%,
98 which attests to the *difficulty* of the benchmark. To verify *correctness*, we manually reviewed 200
99 random examples from each of the 3 benchmarks. We detected errors in 7%, 5%, and 8% of the
100 examples in CHASE-QA, CHASE-CODE, and CHASE-MATH respectively. These error rates are
101 similar to those observed for other human and synthetic benchmarks (Chong et al., 2022, Li et al.,

Table 1: The performance of various LLMs on all 3 domains of the CHASE benchmark. We measure the accuracy of the predictions for CHASE-QA and CHASE-MATH, and pass@1 for CHASE-CODE. Numbers in **bold** indicate best performance on domain while underline indicates best-in-class performance. Full model names for the identifiers used here are provided in Table 2.

(a) General LLMs.				(b) Reasoning Models.			
MODELS	QA	CODE	MATH	MODELS	QA	CODE	MATH
Gemini-1.5-Pro	63.2	38.2	65.4	o4-mini	66.5	38.0	69.6
GPT-4o	55.3	24.6	59.8	Gemini-2.5-FP	64.9	40.8	56.8
Claude-3.5-Sonnet	36.1	22.4	64.2	DeepSeek-R1	45.2	36.6	55.0
DeepSeek-V3	33.8	36.6	58.6	R1-Llama-70B	50.7	30.4	53.6
Gemini-1.5-Flash	<u>55.1</u>	<u>28.6</u>	56.6	R1-Qwen-32B	50.1	32.6	<u>59.4</u>
GPT-4o-mini	50.2	18.8	48.4	OThinker-32B	49.2	<u>33.8</u>	53.4
Claude-3-Haiku	32.6	21.8	44.2	s1-32B	48.1	6.2	45.2
Llama-3.1-70B	41.3	15.6	53.4	LIMO-32B	<u>53.6</u>	8.4	46.0
Qwen2.5-32B	43.9	25	<u>61.2</u>	R1-Llama-8B	<u>34.3</u>	<u>5.2</u>	26.4
Llama-3.1-8B	<u>25.2</u>	2.0	32.2	R1-Qwen-7B	18.5	1.4	<u>35.2</u>
Qwen2.5-7B	22.2	<u>2.2</u>	<u>42.8</u>	R1-Qwen-1.5B	7.2	0.0	17.6

2025). Finally, we observed that all 3 CHASE benchmarks exhibit high *diversity*, with a detailed qualitative and quantitative discussion provided in Appendix G.5.

We note several interesting insights. We see that the Gemini models perform significantly better than other models, especially on long-context reasoning tasks. Surprisingly, we also observe that DeepSeek-R1, which is specialized for reasoning performs similar to its non-reasoning base model DeepSeek-V3 on CHASE-CODE and CHASE-MATH. Another interesting result is that while many models have saturated reasoning benchmarks like AIME-2024 (MAA, 2024) and GSM8k (as seen in Table 10), they still struggle on our simple math word problems dataset. Indeed, on all CHASE benchmarks, we see huge variations in performance between the models. These results highlight our framework’s potential for differentiating between state-of-the-art LLMs that all perform similarly on standard benchmarks like MMLU (Hendrycks et al., 2021), GSM8k (Cobbe et al., 2021), or HumanEval (Chen et al., 2021). We provide examples of errors made by Gemini-1.5-Pro and o4-mini on the CHASE benchmarks and analyze them in Appendix I.

Additional analyses. We compare CHASE with a direct-generation baseline in Appendix G.2. We see that the baseline yields high error rates and fails to challenge strong LLMs (Table 5). We also observed that CHASE scales nearly linearly with budget (Appendix G.3; Figure 4) and enables a tunable cost-size-accuracy trade-off via rejection sampling. Lastly, we show that we can programmatically control the difficulty (Appendix G.4; Figure 5) of the generated problems.

4 Conclusion

In this work, we presented CHASE, a framework for generating challenging evaluation problems. It (i) efficiently generates hundreds of examples; (ii) can be periodically regenerated to mitigate contamination; (iii) enables evaluation of hard-to-judge skills (e.g., long-context reasoning); and (iv) ensures correctness via extensive verification. We instantiate CHASE in document-based QA, repository-level code completion, and math reasoning, and show that even state-of-the-art LLMs struggle on these tasks. Limitations and clarifications are provided in Appendices B and C.

References

- Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacrose, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL <https://arxiv.org/abs/2404.14219>.
- Vaibhav Adlakha, Shehzaad Dhuliawala, Kaheer Suleman, Harm de Vries, and Siva Reddy. Topi-OCQA: Open-domain conversational question answering with topic switching. *Transactions of the Association for Computational Linguistics*, 10:468–483, 2022. doi: 10.1162/tacl_a_00471. URL <https://aclanthology.org/2022.tacl-1.27>.
- Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitt, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussaleem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report, 2023. URL <https://arxiv.org/abs/2305.10403>.
- Anthropic. The claude 3 model family: Opus, sonnet, haiku. <https://www.anthropic.com/news/claude-3-family>, 2024a.
- Anthropic. Claude 3.5 sonnet. <https://www.anthropic.com/news/claude-3-5-sonnet>, 2024b.
- Bernd Bohnet, Kevin Swersky, Rosanne Liu, Pranjal Awasthi, Azade Nova, Javier Snider, Hanie Sedghi, Aaron T Parisi, Michael Collins, Angeliki Lazaridou, Orhan Firat, and Noah Fiedel. Long-span question-answering: Automatic question generation and qa-system ranking via side-by-side evaluation, 2024. URL <https://arxiv.org/abs/2406.00179>.

183 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared
184 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,
185 Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,
186 Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,
187 Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios
188 Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino,
189 Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,
190 Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa,
191 Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob
192 McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating
193 large language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>.

194 Yulong Chen, Yang Liu, Jianhao Yan, Xuefeng Bai, Ming Zhong, Yinghao Yang, Ziyi Yang, Chen-
195 guang Zhu, and Yue Zhang. See what LLMs cannot answer: A self-challenge framework
196 for uncovering LLM weaknesses. In *First Conference on Language Modeling*, 2024. URL
197 <https://openreview.net/forum?id=18iNTRPx8c>.

198 Derek Chong, Jenny Hong, and Christopher Manning. Detecting label errors by using pre-trained
199 language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the*
200 *2022 Conference on Empirical Methods in Natural Language Processing*, pages 9074–9091, Abu
201 Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:
202 10.18653/v1/2022.emnlp-main.618. URL [https://aclanthology.org/2022.emnlp-main.](https://aclanthology.org/2022.emnlp-main.618/)
203 618/.

204 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,
205 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John
206 Schulman. Training verifiers to solve math word problems, 2021. URL [https://arxiv.org/](https://arxiv.org/abs/2110.14168)
207 [abs/2110.14168](https://arxiv.org/abs/2110.14168).

208 Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological*
209 *measurement*, 20(1):37–46, 1960.

210 Cohere. Cohere command r+, August 2024. URL [https://docs.cohere.com/docs/](https://docs.cohere.com/docs/command-r-plus)
211 [command-r-plus](https://docs.cohere.com/docs/command-r-plus).

212 Zhuyun Dai, Arun Tejasvi Chaganty, Vincent Y Zhao, Aida Amini, Qazi Mamunur Rashid, Mike
213 Green, and Kelvin Guu. Dialog inpainting: Turning documents into dialogs. In Kamalika
214 Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, edi-
215 tors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of
216 *Proceedings of Machine Learning Research*, pages 4558–4586. PMLR, 17–23 Jul 2022. URL
217 <https://proceedings.mlr.press/v162/dai22a.html>.

218 Google DeepMind. Gemini 2.5 flash preview model card, 2025. URL [https://storage.](https://storage.googleapis.com/model-cards/documents/gemini-2.5-flash-preview.pdf)
219 [googleapis.com/model-cards/documents/gemini-2.5-flash-preview.pdf](https://storage.googleapis.com/model-cards/documents/gemini-2.5-flash-preview.pdf).

220 DeepSeek-AI et al. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement
221 learning, 2025a. URL <https://arxiv.org/abs/2501.12948>.

222 DeepSeek-AI et al. DeepSeek-V3 Technical Report, 2025b. URL [https://arxiv.org/abs/2412.](https://arxiv.org/abs/2412.19437)
223 19437.

224 Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. Cyclomatic
225 complexity. *IEEE software*, 33(6):27–29, 2016.

226 Gemini Team et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of
227 context, 2024. URL <https://arxiv.org/abs/2403.05530>.

228 Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I.
229 Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL
230 <https://arxiv.org/abs/2401.03065>.

231 T Hariprasad, G Vidhyagaran, K Seenu, and Chandrasegar Thirumalai. Software complexity analysis
232 using halstead metrics. In *2017 International Conference on Trends in Electronics and Informatics*
233 *(ICEI)*, pages 1109–1113. IEEE, 2017.

- 234 Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob
235 Steinhardt. Measuring massive multitask language understanding. In *International Conference on*
236 *Learning Representations*, 2021. URL <https://openreview.net/forum?id=d7KBjmI3GmQ>.
- 237 Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. Unnatural instructions: Tuning
238 language models with (almost) no human labor. In Anna Rogers, Jordan Boyd-Graber, and
239 Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Com-*
240 *putational Linguistics (Volume 1: Long Papers)*, pages 14409–14428, Toronto, Canada, July
241 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.806. URL
242 <https://aclanthology.org/2023.acl-long.806>.
- 243 Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando
244 Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free
245 evaluation of large language models for code. In *The Thirteenth International Conference on*
246 *Learning Representations*, 2025. URL <https://openreview.net/forum?id=chfJJYC3iL>.
- 247 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R
248 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth*
249 *International Conference on Learning Representations*, 2024. URL [https://openreview.net/](https://openreview.net/forum?id=VTF8yNQM66)
250 [forum?id=VTF8yNQM66](https://openreview.net/forum?id=VTF8yNQM66).
- 251 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph
252 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model
253 serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems*
254 *Principles*, pages 611–626, 2023.
- 255 Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and
256 Yongbin Li. Evocodebench: An evolving code generation benchmark with domain-specific
257 evaluations. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets*
258 *and Benchmarks Track*, 2024a. URL <https://openreview.net/forum?id=kvjbfVHpnY>.
- 259 Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context
260 language models understand long contexts? In Lun-Wei Ku, Andre Martins, and Vivek
261 Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computa-*
262 *tional Linguistics (Volume 1: Long Papers)*, pages 16304–16333, Bangkok, Thailand, August
263 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.859. URL
264 <https://aclanthology.org/2024.acl-long.859>.
- 265 Xiang Lisa Li, Farzaan Kaiyom, Evan Zheran Liu, Yifan Mai, Percy Liang, and Tatsunori Hashimoto.
266 Autobench: Towards declarative benchmark construction. In *The Thirteenth International*
267 *Conference on Learning Representations*, 2025. URL [https://openreview.net/forum?id=](https://openreview.net/forum?id=ynt4crbbXh)
268 [ymt4crbbXh](https://openreview.net/forum?id=ynt4crbbXh).
- 269 Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel
270 Ward, and Yi Zhang. Tinygsm: achieving >80URL <https://arxiv.org/abs/2312.09241>.
- 271 Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. Augmenting math word problems
272 via iterative question composing, 2024. URL <https://arxiv.org/abs/2401.09003>.
- 273 Llama Team et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- 274 Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and
275 Hongsheng Li. MathGenie: Generating synthetic data with question back-translation for enhancing
276 mathematical reasoning of LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, edi-
277 tors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*
278 *(Volume 1: Long Papers)*, pages 2732–2747, Bangkok, Thailand, August 2024. Association for
279 Computational Linguistics. URL <https://aclanthology.org/2024.acl-long.151>.
- 280 MAA. American invitational mathematics examination - AIME. In *American Invitational Mathemat-*
281 *ics Examination - AIME 2024*, February 2024. URL [https://maa.org/math-competitions/](https://maa.org/math-competitions/american-invitational-mathematics-examination-aime)
282 [american-invitational-mathematics-examination-aime](https://maa.org/math-competitions/american-invitational-mathematics-examination-aime).

283 Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing
284 English math word problem solvers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel
285 Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational*
286 *Linguistics*, pages 975–984, Online, July 2020. Association for Computational Linguistics. doi:
287 10.18653/v1/2020.acl-main.92. URL <https://aclanthology.org/2020.acl-main.92/>.

288 Mistral. Mistral large 2. <https://mistral.ai/news/mistral-large-2407>, 2024.

289 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke
290 Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time
291 scaling, 2025. URL <https://arxiv.org/abs/2501.19393>.

292 Marianna Nezhurina, Lucia Cipolina-Kun, Mehdi Cherti, and Jenia Jitsev. Alice in wonderland:
293 Simple tasks showing complete reasoning breakdown in state-of-the-art large language models,
294 2025. URL <https://arxiv.org/abs/2406.02061>.

295 OpenAI. Introducing openai o3 and o4-mini, 2025. URL <https://openai.com/index/introducing-o3-and-o4-mini/>.

297 OpenAI Team et al. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.

298 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor
299 Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward
300 Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner,
301 Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance
302 deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox,
303 and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Cur-
304 ran Associates, Inc., 2019. URL [https://proceedings.neurips.cc/paper/2019/file/](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf)
305 [bdbca288fee7f92f2bfa9f7012727740-Paper.pdf](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf).

306 Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple
307 math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek
308 Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou,
309 editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association*
310 *for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online, June
311 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL
312 <https://aclanthology.org/2021.naacl-main.168>.

313 Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi
314 Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov,
315 Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre
316 Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas
317 Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL
318 <https://arxiv.org/abs/2308.12950>.

319 Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua
320 Bengio, Sanjeev Arora, and Anirudh Goyal. Ai-assisted generation of difficult math questions,
321 2024. URL <https://arxiv.org/abs/2407.21009>.

322 Zayne Rea Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. MuSR: Testing the
323 limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference*
324 *on Learning Representations*, 2024. URL <https://openreview.net/forum?id=jenyYQzue1>.

325 OpenThoughts Team. Open Thoughts. <https://open-thoughts.ai>, February 2025.

326 Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL [https://qwenlm.](https://qwenlm.github.io/blog/qwen2.5/)
327 [github.io/blog/qwen2.5/](https://qwenlm.github.io/blog/qwen2.5/).

328 The Mosaic Research Team. Introducing dbrx: A new state-of-the-art open llm, March 2024b. URL
329 <https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm>.

- 330 Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. LLMs cannot find
 331 reasoning errors, but can correct them given the error location. In Lun-Wei Ku, Andre Martins,
 332 and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL*
 333 *2024*, pages 13894–13908, Bangkok, Thailand, August 2024. Association for Computational
 334 Linguistics. doi: 10.18653/v1/2024.findings-acl.826. URL [https://aclanthology.org/2024.](https://aclanthology.org/2024.findings-acl.826/)
 335 [findings-acl.826/](https://aclanthology.org/2024.findings-acl.826/).
- 336 Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang
 337 Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. Leave
 338 no document behind: Benchmarking long-context llms with extended multi-doc qa, 2024. URL
 339 <https://arxiv.org/abs/2406.17419>.
- 340 Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and
 341 Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions.
 342 In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st*
 343 *Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages
 344 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi:
 345 10.18653/v1/2023.acl-long.754. URL <https://aclanthology.org/2023.acl-long.754>.
- 346 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V
 347 Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models.
 348 In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in*
 349 *Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates,
 350 Inc., 2022. URL [https://proceedings.neurips.cc/paper_files/paper/2022/file/](https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf)
 351 [9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf).
- 352 Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering
 353 code generation with OSS-instruct. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian
 354 Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st*
 355 *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning*
 356 *Research*, pages 52632–52657. PMLR, 21–27 Jul 2024. URL [https://proceedings.mlr.](https://proceedings.mlr.press/v235/wei24h.html)
 357 [press/v235/wei24h.html](https://proceedings.mlr.press/v235/wei24h.html).
- 358 Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony
 359 Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer,
 360 Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain
 361 Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-
 362 art natural language processing. In *Proceedings of the 2020 Conference on Empirical Meth-*
 363 *ods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October
 364 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL
 365 <https://aclanthology.org/2020.emnlp-demos.6>.
- 366 Roy Xie, Chengxuan Huang, Junlin Wang, and Bhuwan Dhingra. Adversarial math word prob-
 367 lem generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings*
 368 *of the Association for Computational Linguistics: EMNLP 2024*, pages 5075–5093, Miami,
 369 Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/
 370 [2024.findings-emnlp.292](https://aclanthology.org/2024.findings-emnlp.292/). URL <https://aclanthology.org/2024.findings-emnlp.292/>.
- 371 Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei
 372 Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow
 373 complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024.
 374 URL <https://openreview.net/forum?id=CfXh93NDgH>.
- 375 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,
 376 Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint*
 377 *arXiv:2407.10671*, 2024.
- 378 Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for
 379 reasoning, 2025. URL <https://arxiv.org/abs/2502.03387>.
- 380 Asaf Yehudai, Boaz Carmeli, Yosi Mass, Ofir Arviv, Nathaniel Mills, Eyal Shnarch, and Leshem
 381 Choshen. Achieving human parity in content-grounded datasets generation. In *The Twelfth*

- 382 *International Conference on Learning Representations*, 2024. URL [https://openreview.net/](https://openreview.net/forum?id=RjYKTQOLOW)
383 [forum?id=RjYKTQOLOW](https://openreview.net/forum?id=RjYKTQOLOW).
- 384 Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo
385 Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for
386 large language models. In *The Twelfth International Conference on Learning Representations*,
387 2024a. URL <https://openreview.net/forum?id=N8NOhgNDRt>.
- 388 Xiaodong Yu, Hao Cheng, Xiaodong Liu, Dan Roth, and Jianfeng Gao. ReEval: Automatic hal-
389 lucination evaluation for retrieval-augmented large language models via transferable adversarial
390 attacks. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association*
391 *for Computational Linguistics: NAACL 2024*, pages 1333–1351, Mexico City, Mexico, June
392 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.85. URL
393 <https://aclanthology.org/2024.findings-naacl.85/>.
- 394 Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and
395 Qiufeng Yin. WaveCoder: Widespread and versatile enhancement for code large language models
396 by instruction tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings*
397 *of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*
398 *Papers)*, pages 5140–5153, Bangkok, Thailand, August 2024c. Association for Computational
399 Linguistics. URL <https://aclanthology.org/2024.acl-long.280>.
- 400 Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. Gsm-infinite: How do
401 your llms behave over infinitely increasing context length and reasoning complexity?, 2025. URL
402 <https://arxiv.org/abs/2502.05252>.

A Roadmap

The appendix is organized as follows.

- In Section B, we discuss some concrete limitations of our work.
- In Section C, we clarify some common concerns regarding our work.
- In Section D, we discuss the broader impacts of our work.
- In Section E, we discuss the construction pipelines in detail.
- In Section F, we provide the implementation details for our experiments.
- In Section G, we discuss some additional experimental results.
- In Section H, we discuss some related works.
- In Section I, we analyze errors made by LLMs while generating and solving CHASE benchmarks.
- In Section J, we provide the exact prompts used in this work.

B Limitations

Size of benchmarks. The datasets we generated are comparatively smaller in size. Our framework necessitates querying the generator and especially the verifier many times for crafting each example. While this increases the quality and correctness of the data, it significantly increases the cost of generation. Moreover, a large portion of the intermediate generations in our pipeline are discarded because of extensive verification, which significantly reduces the yield. Our focus in this work is to present the CHASE framework and we believe our experiments, albeit on smaller-sized datasets, convincingly show its utility in generating challenging problems for evaluation. Small benchmarks, if generated with a high bar for quality and correctness, can still be very impactful as evidenced by past works like HumanEval (Chen et al., 2021).

Presence of Errors. Some of the examples we generated using CHASE, while being semantically correct, use unnatural or difficult-to-parse language. This is a general trait of text generated from contemporary LLMs, and our framework is unfortunately susceptible to it. Moreover, as noted in Section 3, a small percentage of correctness errors may exist in the CHASE-generated benchmarks. Examples of such errors in generation are discussed in Appendix I. We believe that errors at this scale are acceptable considering the other advantages of the framework and can be factored into the evaluation in future studies.

Adaptability. While we have shown how we implemented CHASE on three different domains, it is not trivial to adapt the framework to other tasks. Although the high level ideas behind CHASE are easy enough to follow, it takes multiple trials and errors to design a working pipeline for any given task. However, we are optimistic that advances in LLMs’ abilities to more precisely follow instructions will make such pipelines easier to construct in the future.

C Clarifications

Does LLM-verification create a circular dependency, where LLMs validate the work of other LLMs, potentially propagating the same limitations?

One of the key features of CHASE is that the generation and verification tasks are very different. We use LLMs to only verify simple sub-steps in a multi-stage pipeline. Consider, for e.g., the task of verifying whether a particular document contains a specific point of information for CHASE-QA. This verification task is much more simple and straightforward than the generation tasks of generating the questions, answers, or documents. We also carried out human verification of these individual verification sub-tasks as discussed in Appendix G.8.

How does CHASE differ from works that artificially increase benchmark difficulty using simple perturbations?

448 Unlike benchmarks that rely on simple perturbations to expose model weaknesses (Nezhurina et al.,
449 2025, Xie et al., 2024, Zhou et al., 2025), CHASE aims to evaluate LLMs in realistic scenarios
450 that require a higher level of reasoning. Our goal is not to construct adversarial edge cases, but
451 to develop scalable evaluation pipelines for tasks that naturally arise in practical settings, such as
452 information-seeking QA and repository-level code synthesis. Even in math reasoning, CHASE
453 presents rich, scenario-based problems written in ambiguous, naturalistic language, reflecting the
454 challenges of real-world comprehension. While some of our findings (e.g., LLMs still struggle at
455 higher-depth math word problems) may echo those of perturbation-based studies, our methodology is
456 fundamentally different in motivation, construction, and scope.

457 *What is the value of CHASE-Code when we can automatically create benchmarks such as SWE-Bench*
458 *(Jimenez et al., 2024) using real-world GitHub repositories.*

459 CHASE-CODE is not intended to replace or compete with benchmarks like SWE-Bench, but rather
460 to complement them. SWE-Bench targets a fundamentally different task: code editing and bug
461 fixing, where models are expected to generate patches for existing issues in real repositories. In
462 contrast, CHASE-CODE focuses on generating new functionality based on its precise description.
463 Such problems may be present in SWE-Bench, but they are a clear minority. Moreover, a key
464 advantage of CHASE-CODE is its controllability. Users can specify parameters such as the target
465 domain (e.g., algorithms, data pre-processing), function difficulty, or repository length. This level
466 of customization is not feasible with current real-world benchmarks like SWE-Bench, which rely
467 on naturally occurring issues in a small number of curated repositories. Finally, benchmarks like
468 SWE-Bench are bottlenecked by the availability of high-quality repositories with extensive testing
469 infrastructure. In contrast, CHASE-CODE’s pipeline enables automated test generation, thereby
470 improving its scalability.

471 **D Broader Impact Statement**

472 Our work presents a novel framework that leverages Large Language Models (LLMs) to synthetically
473 generate challenging evaluation benchmarks. As LLMs continue to advance, rigorously assessing
474 their capabilities only based on human-annotation has become increasingly difficult. Our approach has
475 the potential to drive significant progress in the field of evaluation, addressing one of the most critical
476 challenges in machine learning by providing a scalable, transparent, and cost-effective mechanism
477 for benchmarking new models over time.

478 From an ethical standpoint, our framework helps mitigate the risks of over-reliance on static bench-
479 marks, which can become obsolete or compromised through test data contamination. By enabling the
480 periodic renewal of evaluation datasets, our method promotes a more robust and adaptive assessment
481 of LLMs. However, since synthetic benchmarks are themselves generated by LLMs, they may
482 encode biases, fail to capture certain real-world complexities, or reinforce existing limitations in
483 model-generated data. Ensuring that these benchmarks remain diverse, representative, and aligned
484 with human values is an essential direction for future research.

485 On a broader societal level, improved evaluation methods contribute to the responsible development
486 of AI systems by enabling better monitoring of model performance, safety, and fairness. However,
487 the automation of evaluation also raises questions about the role of human oversight and the potential
488 risks of relying on AI-generated assessments. Future work should seek to investigate such questions
489 towards ensuring rigorous, unbiased, and socially beneficial evaluations of AI systems.

490 **E Construction Pipelines**

491 In this section, we discuss in detail our implementation of the CHASE framework for all three
492 domains. Our pipelines use two different LLMs: the generator **G** and verifier **V**.

493 **E.1 Constructing CHASE-QA**

494 We generate CHASE-QA completely from scratch without relying on existing contexts or any seed
495 examples from previous datasets. Following the CHASE framework, we create each example in a
496 bottom-up manner by first generating the question-answer pair, and then generating the corresponding

documents. Our pipeline for creating CHASE-QA is illustrated in Figure 1 bottom left. We describe it in detail below. The prompts are provided in Appendix J.1.

Generating diverse scenarios. We generate a set of diverse realistic scenarios in which a *user persona* seeks to find some information from a *collection of documents*. For example, a ‘grad student in NYC’ searching the ‘laws on renting and subletting’. We prompt **G** to generate scenarios in the form of a tuple (*persona*, *collection_name*) by bootstrapping it with 5 annotated scenarios, and later prompting it with its own generated scenarios.

Generating question-answer (QA) pairs. We design programmatic prompts with a given scenario as the variable to prompt **G** to generate a realistic information-seeking question that the *persona* might want to know about from *collection_name* set of documents. For example, a ‘grad student’ might pose the question, ‘what is the procedure for refusing increase in rent?’, whose answer can be found spread across multiple documents on the government’s laws on renting. Additionally, **G** must generate the corresponding answer. We prompt **G** to generate questions and answers where the answers are a composition of multiple points or ideas. Lastly, **G** must generate the outline of the documents (only title and abstract) which will contain the answer. The idea is that it must separate out the answer points and assign them to these different documents.

Generating distracting information. To make the task more challenging, for each QA pair, we prompt **G** to generate *distracting* QA pairs where the answer is of a similar type as the ground-truth answer. An example of a distracting question for our running example with the grad student is, ‘how do I increase the rent for an apartment I am subletting?’. The corresponding answers to such questions will be of a similar flavour to the ground-truth answer, but ultimately *distracting* for answering the question. This will make the generated data challenging since it will confuse the model when all this similar type of information is spread across multiple documents. It is, however, important to verify that none of this generated distracting information is actually relevant for the question (otherwise it will make our ground-truth answer incomplete). We individually prompt **V** with the original question and each of the supposed distracting information points to check if any part of them is relevant for answering the question (see Figure 1 bottom left for an example of a distracting point discarded by **V** because it was relevant for the original question).

Generating documents. For each example, we have generated a QA pair, along with some similar but distracting QA pairs. For each of these QA pairs, we separately prompt **G** to generate long documents where the documents must discuss the corresponding answer points assigned to it, along with many other irrelevant points. Together, all these documents form up the context for that example. We verify two things to ensure the correctness of the task: (1) none of the documents should contain any extra information related to the question, apart from the ground-truth answer points, and (2) all of the ground-truth answer points must be discussed in the documents. We do this by rigorously prompting **V** with individual documents and ground-truth answer points.

E.2 Constructing CHASE-MATH

We sample math word problems (MWP) from existing datasets as seed examples to build our benchmark. Following CHASE, we bottom-up build a complex problem by iteratively increasing the reasoning depth of the problem. Our pipeline used for creating CHASE-MATH can be seen in Figure 1 bottom right. We describe it in more detail below. The prompts are provided in Appendix J.3.

Breaking down seed MWP. A seed MWP s is characterised by the tuple $s = (p, a)$ where p is the problem, and a is the answer. We prompt **G** to break down p into two parts: the context c , which provides all the information, and the question q , which asks about some unknown quantity.

Create continuation of MWP. We prompt **G** with an initial seed MWP $s_0 = (p_0, a_0)$ to build a new problem which is a continuation of the previous problem. More precisely, **G** should output a new problem $s_1 = (p_1, a_1)$, where the context of p_1 , i.e., c_1 assumes a_0 as given information (without explicitly stating it). For example, in Figure 1 bottom right, the model assumes *Jack has 4 pens* as given information, and creates a new continuation context, *Jill has thrice as many pens as Jack has now*. The model also generates a new question q_1 , *how many pens does Jill have?* whose answer $a_1 = 12$ is obtained by performing an arithmetic operation (here, *multiplication by 3*) over $a_0 = 4$.

Combining seed MWP with its continuation. By combining the seed problem with its continuation, we get a new MWP $s = (p, a)$ with a higher reasoning depth, where the context c of the combined

problem p is a concatenation of the contexts of the seed problem and the continuation $c = c_0 \cdot c_1$. The question for the combined problem will be the one generated by the model, i.e., q_1 , and the answer $a = a_1$. We refer to Figure 1 bottom right for illustration.

Iteratively increase reasoning depth. We increase the reasoning depth of a given seed MWP by creating new continuations in an iterative manner. Each new continuation s_i formed after the i^{th} iteration becomes the seed problem for the $(i + 1)^{\text{th}}$ iteration. The final problem after j successful iterations, i.e., with a reasoning depth of $j + 1$, is given by context $c = c_0 \cdot c_1 \dots c_j$, question q_j , and answer $a = a_j$.

Since each new problem created by **G** has a low reasoning depth of the same difficulty as the problems in the seed datasets, we verify their correctness using a non-identical ensemble of verifier models $\{\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_n\}$, each of which performs well on the seed dataset. We prompt each \mathbf{V}_k with the generated context c_i and question q_i and check whether the prediction is the same as the generated answer a_i . If this fails for any verifier, we discard s_i and begin again with s_{i-1} as the seed MWP (see Figure 1 bottom right).

E.3 Constructing CHASE-CODE

We generate CHASE-CODE completely from scratch without relying on existing contexts or any seed examples from previous datasets. Our pipeline for creating CHASE-CODE is shown in Figure 3 in the Appendix. We describe it in detail below. The prompts are provided in Appendix J.2.

Generating Python functions. We begin by first generating a set of diverse and realistic Python functions. We prompt **G** to generate Python functions for a particular domain by bootstrapping it with 3 annotated functions in that domain, and later prompting it with its own generated functions. These generated functions will act as the *helper* functions in the repository context which may or may not be called in the answer code function. Given each generated helper function, we prompt **V** to generate a Python code which initializes sample inputs for the function and then calls it using them. We then execute this code to verify whether the generated helper function executes correctly.

Generating problem statement and answer code. To create a single example, we randomly sample n of the previously generated helper functions, and prompt **G** to create a complex function that calls at least k of these provided helper functions (hereafter called *relevant* helper functions) apart from implementing additional logic. This complex function is our *answer code*. Additionally **G** must elaborate in natural language what objectives the complex function achieves, which forms our *problem statement*. Similar to the case of helper functions, we prompt **V** to generate test code to check if the generated answer code executes correctly. To verify whether the problem statement sufficiently specifies the answer code, we prompt **V** with the problem statement and corresponding *relevant* helper functions and check whether the output is semantically equivalent to the answer code (using the test code obtained in the next step).

Generating test code. To enable automatic execution-based testing, we prompt **G** with the generated answer function to implement a test code for it in Python. The test code must independently implement the logic of the answer code without access to the corresponding helper functions. It must then initialize the parameters of the answer function with sample values, and compare the output with its own implementation. We execute the generated test code to check if the answer code passes. We discard all examples for which (1) the test code does not execute properly, or (2) the test code executes but the answer code fails the test.

Building code repository. For each example, we build a unique repository of Python files. The repository consists of the *relevant* helper functions spread across different files, along with m randomly sampled irrelevant Python functions from our previously generated set. The core difficulty of this task arises from understanding the entire long context of code functions, and identifying which ones are relevant for the provided problem statement.

F Implementation Details

Generating CHASE-QA. We use GPT-4o (OpenAI Team et al., 2024) as the generator **G**, and GPT-4o-mini as the verifier **V**. We first sampled 500 unique scenarios. For each scenario, we generate 2 QA pairs. For each of the resulting 1000 unique QA pairs, we obtain *distracting* information

by generating 4 similar QA pairs. We then generate the corresponding documents containing the ground-truth answer as well as distracting information for each of the 1000 examples. To increase the complexity of the resulting benchmark, we carry out a form of rejection sampling. We evaluate GPT-4o-mini twice on the task, and randomly discard half of the problems on which it was correct both times. This yielded the final benchmark of 671 examples.

Generating CHASE-CODE. We use GPT-4o-mini (OpenAI Team et al., 2024) as the generator G , and Gemini-1.5-Flash as the verifier V . We made this choice because generating even a small amount of challenging code problems required a large number of iterations, since a lot of the model-generated code at various stages would fail to execute or be semantically incorrect. For each domain, we first sampled 500 different helper functions that execute without errors. Then we prompt the model with $n = 10$ random helper functions to generate a problem statement and corresponding answer code that calls at least $k = 4$ helper functions. We do this to create 1000 different examples for each domain. Next, we generate up to 10 test codes for each example and keep only those examples for which a generated test code successfully passed for the corresponding answer code. We also carry out the verification of correctness of problem statement as describe before. This way, we end up with 290 examples for the *algorithms* domain and 300 examples for the *data pre-processing* domain. We again use GPT-4o-mini for rejection sampling and randomly discard around half of the problems on which it was correct. This way, we end up with a total of 500 examples in the benchmark, with 250 examples for each domain. For each example, we randomly sample $m = 100$ *irrelevant* helper functions and distribute them into 10 Python files to constitute the repository context.

Generating CHASE-MATH. We use GPT-4o-mini (OpenAI Team et al., 2024) as the generator G , and an ensemble of Gemini-1.5-Flash and Llama-3.1-70B as the verifier V . In practice, we observed that many of the model generated problems would fail at various stages of verification, so it is faster and cheaper to query the smaller models. We start with 2.3k seed problems taken from the test sets of GSM8k (Cobbe et al., 2021) and SVAMP (Patel et al., 2021). We set the maximum and minimum reasoning depth at 5 and 2 respectively. For each problem, we iterate 15 times to generate a problem continuation. Note that many of these iterations fail to produce a correct continuation of the problem, in which case we discard that generation and retry from that point in the subsequent iteration. We carry out this process 3 times. In this manner, we generated around 1500 problems. We then carry out rejection sampling and roughly discarded 75% of the problems that GPT-4o-mini could solve. In the end, we end up with a total of 500 challenging MWPs.

Task parameters. For CHASE-QA and CHASE-CODE, we prompt models with the instruction for the task, along with the corresponding long-context and question. The prompt formats are provided in Figure 24 and 33 respectively in Appendix J. For CHASE-MATH, we prompt models with 8-shot chain-of-thought (Wei et al., 2022) as shown in Figure 36 in Appendix J.3. We decode for a maximum of 1024 tokens with a temperature of 0.5.

Evaluation. The ground-truth answers for CHASE-QA are verbose text, organized in bullet points. While this simulates real-world complexity, it also makes evaluation difficult. Since it is intractable to employ expert humans for evaluation, we deploy an LLM-as-a-judge to automatically assess the correctness of predictions. A prediction is considered to be correct if and only if it is (1) *complete*, i.e., it includes all the points mentioned in the ground-truth answer, and (2) *relevant*, i.e., it provides information only pertaining to the current question. We use GPT-4o as the judge and measure the accuracy as the percentage of predictions judged to be correct. The prompt format used for evaluation is provided in Figure 25 in Appendix J.1. To assess the validity of the GPT-4o judge, we carry out human evaluation as discussed below which shows an almost-perfect agreement between the judge and humans. For CHASE-CODE, we measure the pass@1 execution accuracy, i.e., whether the model generated code correctly passes when we execute the corresponding test code in the first attempt. For CHASE-MATH, we measure the exact match accuracy against the ground-truth numerical answer.

Models. We experimented with 27 different LLMs. These include *general LLMs*: Gemini-1.5-Pro and Flash (Gemini Team et al., 2024), GPT-4o and GPT-4o-mini (OpenAI Team et al., 2024), Claude-3.5-Sonnet Anthropic (2024b), Claude-3-Haiku (Anthropic, 2024a), DeepSeek-V3 (DeepSeek-AI et al., 2025b), Llama-3.1 8B and 70B (Llama Team et al., 2024), Mistral Small and Large 2 (Mistral, 2024), Qwen2.5 7B and 32B (Team, 2024a, Yang et al., 2024), Cohere Command R+ (Cohere, 2024), DBRX-Instruct (Team, 2024b), and Phi-3.5-MoE (Abdin et al., 2024); and *reasoning LLMs*: o4-mini (OpenAI, 2025), Gemini-2.5-Flash-Preview (DeepMind, 2025), DeepSeek-R1 (DeepSeek-AI et al.,

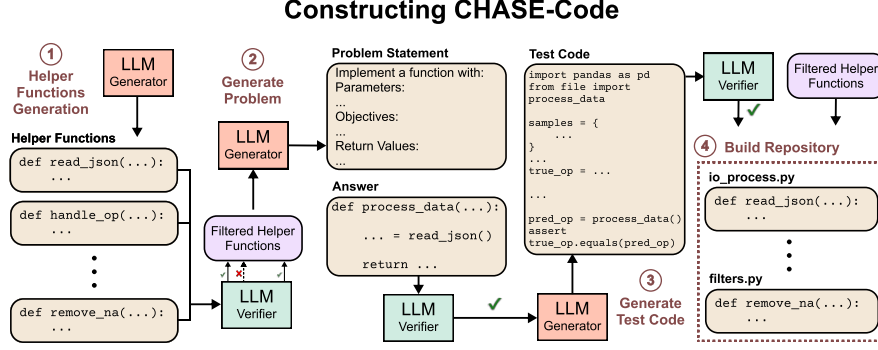


Figure 3: Pipeline for creating an example in CHASE-CODE.

2025a), R1-Distill-Llama-70B, R1-Distill-Qwen-32B, R1-Distill-Llama-8B, R1-Distill-Qwen-7B, R1-Distill-Qwen-1.5B, OpenThinker2-32B (Team, 2025), s1.1-32B (Muennighoff et al., 2025), and LIMO-32 (Ye et al., 2025).

Our code is implemented in PyTorch (Paszke et al., 2019) and makes use of the HuggingFace Transformers library (Wolf et al., 2020) and the vLLM library (Kwon et al., 2023) for running efficient inference locally on LLMs. All experiments with open models were done on our cluster with 8 NVIDIA A6000 GPUs with 48 GB memory. Experiments using GPT-4o and GPT-4o-mini were carried out using the OpenAI API.¹ Experiments using Gemini-1.5-Pro and Gemini-1.5-Flash were carried out using the Google AI Studio.² Experiments with Claude-3.5-Sonnet and Claude-3-Haiku were carried out using Anthropic’s API.³ Experiments with DeepSeek-V3 and DeepSeek-R1 were carried out using Together API.⁴ We provide the exact identifier and version for each LLM we experimented with in Table 2.

Human verification of LLM judgements. For CHASE-QA, we measure the correlation of the GPT-4o evaluator’s judgement and 3 human annotators. We carry out human verification on Amazon Mechanical Turk. We first randomly sampled 10 of the predictions made by Gemini-1.5-Pro on CHASE-QA and manually evaluated them. We then publish them as a batch of 10 Human Intelligence Tasks (HITS) to serve as a qualification task to identify workers who will do the task properly. Note that each model prediction that needs to be judged is a HIT. Once we identified 3 workers that did perfectly on our qualification task, we published a batch of 100 randomly sampled predictions accessible only to those workers. Note that we sampled a balanced set based on the LLM judge’s evaluation: 50 that were marked by GPT-4o as correct and 50 that were marked as incorrect. The instructions provided to the workers and the setup of the task is kept exactly the same as the one provided to the LLM judge as shown by the prompt in Figure 25. We paid \$0.5 USD to the workers for every example. The accuracy of GPT-4o’s judgement as measured against the majority vote of the annotators was 91%. Moreover, Cohen’s kappa (Cohen, 1960) between the majority vote of the annotators and the LLM judge came out to be 0.82, which indicates almost-perfect agreement.

Cost of creation. In Table 3, we report the estimated cost of creating the three benchmarks, both in terms of inference time and API expenses. Note that the inference time assumes sequential execution of each part of the pipeline with only one process running at a time. Hence, the generation can be made considerably faster with increased parallelism. This table does not include the cost of other experiments in the paper nor does it include the cost of background experiments that went into designing the pipelines. We estimate the total of these costs to be over \$1000 USD.

¹<https://platform.openai.com>

²<https://aistudio.google.com>

³<https://console.anthropic.com>

⁴<https://api.together.xyz/>

Table 2: Model identifiers for the 27 models we studied in our work. Models that are openly available are provided with links to their corresponding pages on Huggingface Hub.

MODEL	EXACT IDENTIFIER
Llama-3.1-8B	meta-llama/Llama-3.1-8B-Instruct
Llama-3.1-70B	meta-llama/Llama-3.1-70B-Instruct
Mistral Small	mistralai/Mistral-Small-Instruct-2409
Mistral Large 2	mistralai/Mistral-Large-Instruct-2407
Qwen2.5-7B	Qwen/Qwen2.5-7B-Instruct
Qwen2.5-32B	Qwen/Qwen2.5-32B-Instruct
Command R+	CohereForAI/c4ai-command-r-plus-08-2024
DBRX	databricks/dbrx-instruct
Phi-3.5-MoE	microsoft/Phi-3.5-MoE-instruct
DeepSeek-R1	deepseek-ai/DeepSeek-R1
DeepSeek-V3	deepseek-ai/DeepSeek-V3
R1-Llama-70B	deepseek-ai/DeepSeek-R1-Distill-Llama-70B
R1-Qwen-32B	deepseek-ai/DeepSeek-R1-Distill-Qwen-32B
R1-Llama-8B	deepseek-ai/DeepSeek-R1-Distill-Llama-8B
R1-Qwen-7B	deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
R1-Qwen-1.5B	deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
OTinker-32B	open-thoughts/OpenThinker2-32B
s1-32B	simplescaling/s1.1-32B
LIMO-32B	GAIR/LIMO
GPT-4o-mini	gpt-4o-mini-2024-07-18
GPT-4o	gpt-4o-2024-05-13
o4-mini	o4-mini-2025-04-16
Gemini-1.5-Flash	gemini-1.5-flash-001
Gemini-1.5-Pro	gemini-1.5-pro-001
Gemini-2.5-FP	gemini-2.5-flash-preview-04-17
Claude-3-Haiku	claude-3-haiku-20240307
Claude-3.5-Sonnet	claude-3-5-sonnet-20240620

G Additional Results and Discussion

G.1 Results for Other Models

In Table 4, we provide results for some additional LLMs that we experimented with.

G.2 Direct Generation Baseline

We experimented with directly prompting models to generate challenging data for the QA and math tasks, without using the CHASE framework. For QA, we prompt GPT-4o with unique examples from CHASE-QA as the seed task and instruct it to generate new examples (Honovich et al., 2023, Wang et al., 2023). For math, we adapt *Evol-Instruct* (Xu et al., 2024) to generate more complex problems given seed examples from GSM8k. We carry out the same proportion of rejection sampling as we did for CHASE-QA and CHASE-MATH for fair comparison. We generated a total of 200

Table 3: Estimated cost of creating the benchmarks in terms of inference time and money.

BENCHMARK	INFERENCE TIME (HOURS)	COST (USD)
CHASE-QA	40	100
CHASE-CODE	55	150
CHASE-MATH	200	40

Table 4: The performance of various LLMs on all 3 domains of the CHASE benchmark. We measure the accuracy of the predictions for CHASE-QA and CHASE-MATH, and pass@1 for CHASE-CODE. DATA and ALGO refer to the *data pre-processing* and *algorithms* sub-domains of CHASE-CODE.

MODELS	QA	CODE	MATH
Command R+	41.7	0	43.2
DBRX	15.7	2.2	21.6
Mistral Large 2	34.1	5.0	59.6
Phi-3.5-MoE	10.6	0.8	39.4
Mistral Small	35.5	1.4	50.6

examples for both tasks. We evaluated o4-mini, GPT-4o, Gemini-1.5-Pro, and Claude-3.5-Sonnet on these datasets and provide the results in Table 5. For both tasks, based on manual verification of examples, we observe that the error % is very high. Moreover, the generated data is not challenging for the LLMs we evaluated, all of which perform similarly.

G.3 Scalability of CHASE

We evaluate the scalability of CHASE by running the construction pipelines under varying cost budgets (in \$USD) and measuring the amount of data generated. As shown in Figure 4, for CHASE-CODE, the size of the generated dataset scales approximately linearly with the cost budget. We observed similar scaling behavior for CHASE-QA and CHASE-MATH. Further, we measure the average accuracy of a target model (GPT-4o) after removing different fractions of the data via rejection sampling using GPT-4o-mini. Our results show a controllable trade-off between cost, dataset size, and downstream accuracy, enabling flexible tuning based on evaluation requirements.

G.4 Controlling the Difficulty of CHASE

A key strength of our approach is the ability to finely control various parameters of the data generation process, particularly to produce *challenging* evaluation benchmarks. Figure 5 illustrates this for CHASE-QA and CHASE-MATH, showing that increasing the number of distracting documents (while keeping the total context size constant) and the required depth of reasoning respectively leads to a measurable drop in model performance, thereby demonstrating effective control over task difficulty. In Appendix G.9, we further show the impact of context size on task difficulty.

G.5 Diversity of Problems in CHASE

We assess the diversity of the CHASE benchmarks across all three domains. Each benchmark is constructed in a way that causes variation in problem structure, content, and reasoning skills required. We support this claim both qualitatively—by highlighting diversity-oriented design choices—and quantitatively, by computing the lexicon usage diversity (LD) metric (Miao et al., 2020), which measures lexical diversity across a dataset.

Table 5: Performance of LLMs (Accuracy %) on data generated by direct prompting approaches without using CHASE. All models achieve similarly high accuracies. The last line reports the manual verification errors (%) in each dataset.

MODEL	QA	MATH
o4-mini	80.5	75.5
Gemini-1.5-Pro	82.0	74.0
GPT-4o	77.0	72.0
Claude-3.5-Sonnet	78.5	74.5
Error	16.5	23.0

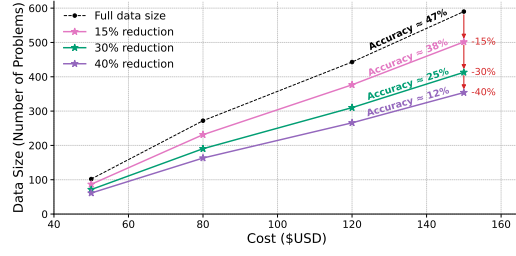


Figure 4: Scaling the size of generated CHASE-CODE data as a function of cost. We also compute the average accuracies (for GPT-4o) when the size is reduced using rejection sampling.

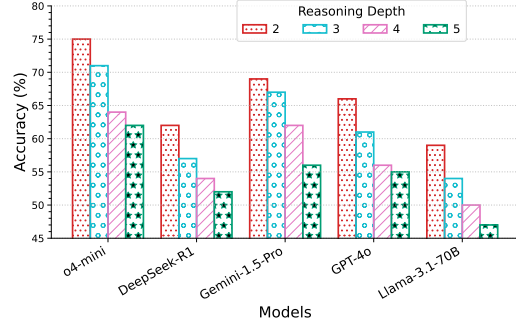
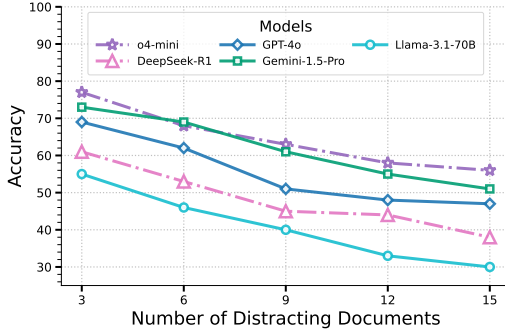


Figure 5: Controlling the difficulty of CHASE-QA by increasing the number of distracting documents (*left*) and CHASE-MATH by increasing the depth of reasoning (*right*). We hold the size of the data for each parameter setting constant at 100 examples.

724 Given a set of n problems $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$, the *lexicon usage diversity* (LD) for a problem p_i is
725 defined as:

$$LD_i = 1 - \max_{j \neq i} \left(\frac{\text{BLEU}(p_i, p_j) + \text{BLEU}(p_j, p_i)}{2} \right)$$

726 where $\text{BLEU}(p_i, p_j)$ is the BLEU score (using up to 4-grams) of p_i with respect to p_j , with smoothing
727 applied for short sequences.

728 The dataset-level LD is then computed as the mean across all individual scores:

$$LD(\mathcal{P}) = \frac{1}{n} \sum_{i=1}^n LD_i$$

729 Higher LD values indicate greater lexical diversity among the problem statements in the dataset. For
730 reference, the LD value of the ASDiv-A dataset (Miao et al., 2020), which was considered to be very
731 diverse, is 0.5.

732 **Diversity of CHASE-MATH.** The problems in CHASE-MATH are built using the examples of
733 GSM8k (Cobbe et al., 2021) as the starting point. Since GSM8k is considered to be a high-quality
734 and diverse dataset, this ensures a high level of diversity in CHASE-MATH by design. The LD values
735 for GSM8k and CHASE-MATH are 0.89 and 0.88 respectively, which quantitatively illustrates that
736 our synthetic benchmark is as diverse as the human-curated GSM8k.

737 **Diversity of CHASE-QA.** Each question in CHASE-QA is constructed using a scenario that
738 involves a particular user persona (e.g., lawyer, doctor, etc.) searching an environment (e.g., laws
739 on renting, etc.). We used 300 unique scenarios to craft the 671 questions in CHASE-QA, which
740 encourages diversity by design. Furthermore, the LD value of the questions in CHASE-QA is 0.75.

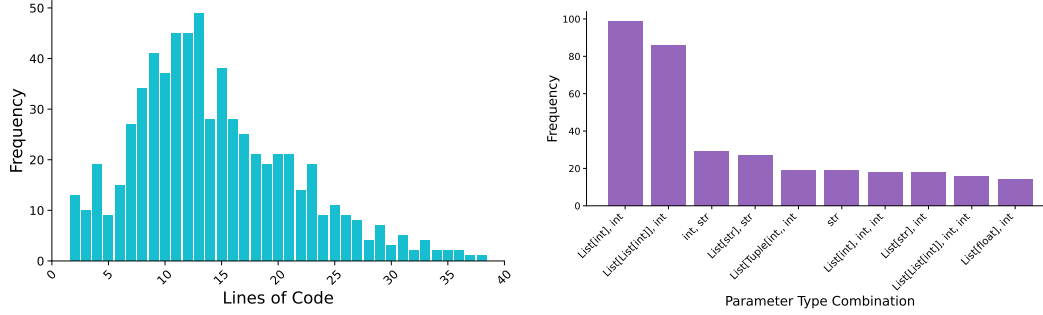


Figure 6: Analysing the diversity of generated helper functions using frequency distributions over number of lines of code (*left*), and the 10 most frequent unique parameter-type combinations (*right*).

Table 6: Dataset statistics of CHASE-QA and intrinsic complexity metrics to measure the difficulty of examples.

CRITERIA	
Number of examples	671
Average context size	6000
Average number of answer points	3.61
Average number of relevant documents	3.32
Average number of distracting documents	7.72

Diversity of CHASE-CODE. We generate problems for two diverse domains: *algorithms* and *data pre-processing*. Each problem in CHASE-CODE is created using a distinct set of randomly sampled helper functions, which encourages diversity by construction. The diversity of the generated helper functions is quite high as evidenced by the frequency distributions over number of lines of code and unique parameter-type combinations shown in Figure 6. Lastly, the LD value of problem statements in the *algorithms* subset of CHASE-CODE is 0.85 and that of the *data pre-processing* subset is 0.83.

G.6 Intrinsic Complexities of CHASE Benchmarks

For all three CHASE benchmarks, we evaluate the intrinsic complexity using various metrics. For CHASE-QA, we measure the average number of answer points per question along with the average number of relevant and distracting documents. For CHASE-CODE we use AST tree-based metrics such as Cyclomatic Complexity (Ebert et al., 2016) and Halstead Difficulty (Hariprasad et al., 2017) of the answer code combined with the helper functions. For CHASE-MATH, we measure the average reasoning depth of the problems. The statistics and complexity values for CHASE-QA, CHASE-CODE, and CHASE-MATH are provided in Tables 6, 7, 8 respectively.

G.7 Comparison of Model Performances On Similar Datasets

CHASE-QA consists of long-context realistic-situation-based information-seeking QA problems. The most similar benchmarks are Loong (Wang et al., 2024), which consists of long-context QA problems requiring reasoning over documents (more than 100k tokens long) from domains such as academic papers and financial reports, and LooGLE (Li et al., 2024b), which consists of long-dependency QA problems over wikipedia and movie scripts (around 32k tokens context). The best performing models on these datasets achieve scores of around 53% and 54% respectively. The best performing model on CHASE-QA achieves a score of around 63%, which reduces to around 55% when we scale the context size to comparable levels of 30k tokens (as seen in Figure 7 *right*).

CHASE-CODE consists of repository-level code generation problems. HumanEval (Chen et al., 2021) and LiveCodeBench (Jain et al., 2025) are some of the most widely-used challenging code

Table 7: Dataset statistics of CHASE-CODE and intrinsic complexity metrics to measure the difficulty of examples.

CRITERIA	ALGO	DP
Number of examples	250	250
Average context size	17000	17000
Average number of statements	22.04	11.37
Average Cyclomatic Complexity	15.40	8.23
Average Halstead Difficulty	19.59	12.24

Table 8: Dataset statistics of CHASE-MATH and intrinsic complexity metrics to measure the difficulty of examples.

CRITERIA	
Number of examples	500
Average reasoning depth	3.92
Average number of words in question	278.56

generation benchmarks. We compare the performances of both, general and reasoning LLMs on these datasets in Table 9 and Table 10 respectively. We can clearly see that CHASE-CODE is also a challenging benchmark. Recently, some repository-level code benchmarks have also been proposed. SWE-Bench (Jimenez et al., 2024) is a benchmark of around 2300 software engineering problems compiled from GitHub issues in popular repositories. EvoCodeBench (Li et al., 2024a) consists of 275 repository-level code generation problems based on popular GitHub repositories. The best performing models on these benchmarks achieve around 42% and 20% scores respectively.

CHASE-MATH consists of math reasoning problems. The most widely-used challenging benchmarks for this task are GSM8k (Cobbe et al., 2021), and AIME-2024 (MAA, 2024). We compare the performances of both, general and reasoning LLMs on these datasets in Table 9 and Table 10 respectively. It is clear that GSM8k and AIME-2024 are becoming saturated, with many state-of-the-art models achieving more than 90% accuracies. In comparison, CHASE-MATH is still very difficult for all models to solve. Moreover, the differences in performance between different models is much larger, which enables more confident comparison.

G.8 Human verification of LLM verification

We carry out human verification of each LLM verification stage. For each stage, we manually look at 30 random instances and report the errors.

For CHASE-QA, there are two main verification steps: (1) verifying whether the information in distracting answer points is not relevant for answering the question, and (2) verifying whether the generated documents contain only the same information as the ground truth answer (not missing anything, nor adding anything). We did not find any errors in the former, while we found 2 errors in the latter.

For CHASE-CODE, the verification steps are: (1) verifying the execution of helper functions, (2) verifying the execution of the answer code, and (3) verifying whether the problem statement sufficiently specifies the answer code. We found only one error in the last step.

For CHASE-MATH, the verification involves verifying a single problem continuation. We found two errors in this step.

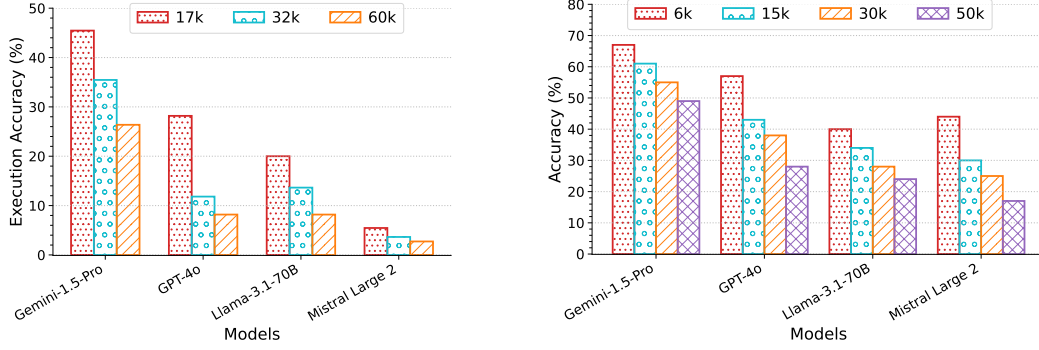


Figure 7: Performance of LLMs decreases with increasing context sizes for 110-example subset of CHASE-CODE (*left*), and 100-example subset of CHASE-QA (*right*).

Table 9: Comparison of performances of general LLMs on CHASE benchmarks with other human-curated benchmarks in the domain. We compare CHASE-MATH and GSM8k, a widely-used benchmark for grade-school level math word problem solving. We also compare CHASE-CODE and HumanEval, a widely-used benchmark for code generation (pass@1 for both).

MODEL	CHASE-MATH	GSM8K	CHASE-CODE	HUMANEVAL
Gemini-1.5-Pro	65.4	90.8	38.2	84.1
GPT-4o	59.8	96.1	24.6	90.2
Claude-3.5-Sonnet	64.2	96.4	22.4	92.0
DeepSeek-V3	58.6	89.3	36.6	65.2
Gemini-1.5-Flash	56.6	86.2	28.6	74.3
GPT-4o-mini	48.4	94.2	18.8	86.6
Claude-3-Haiku	44.2	79.2	21.8	75.9
Llama-3.1-70B	53.4	95.1	15.6	80.5
Mistral Large 2	59.6	92.7	5.0	92.1
Qwen2.5-32B	61.2	92.9	25.0	86.6
Command R+	43.2	70.7	0.0	70.1
DBRX	21.6	72.7	2.2	72.1
Phi-3.5-MoE	39.4	88.7	0.8	70.7
Mistral Small	50.6	87.4	1.4	73.8
Llama-3.1-8B	32.2	84.5	2.0	72.6
Qwen2.5-7B	42.8	85.4	2.2	57.9

793 G.9 Effect of Context Size

794 We studied the impact of varying the context size for long-context reasoning. For each example
795 in a randomly-sampled 100-example subset of CHASE-QA, we increase the context size by con-
796 catenating the documents in that example with distracting documents randomly sampled from other
797 examples. For CHASE-CODE, we create a subset of 55 randomly-sampled examples for each of the
798 domains and increase the context size by concatenating irrelevant code functions in the corresponding
799 repository context. Figure 7 plots the performances of 4 LLMs across different context sizes. For
800 both benchmarks, we see a consistent and significant decrease in model performance as we scale up

Table 10: Comparison of performances of reasoning LLMs on CHASE benchmarks with other human-curated benchmarks in the domain. We compare CHASE-MATH and AIME-24 (MAA, 2024), a widely-used benchmark for olympiad-level math reasoning. We also compare CHASE-CODE and LiveCodeBench (LCB) (Jain et al., 2025), a widely-used benchmark for code reasoning.

MODEL	CHASE-MATH	AIME-24	CHASE-CODE	LCB
o4-mini	69.6	93.4	38	72.2
Gemini-2.5-FP	56.8	88.0	40.8	63.5
DeepSeek-R1	55.0	79.8	36.7	65.9
R1-Llama-70B	53.6	70.0	30.4	57.5
R1-Qwen-32B	59.4	72.6	32.6	57.2
R1-Llama-8B	26.4	50.4	5.2	39.6
R1-Qwen-7B	35.2	55.5	1.4	37.6
R1-Qwen-1.5B	17.6	28.9	0.0	16.9

Table 11: Measuring performance of all models on CHASE-QA with alternative soft metrics, K-Precision and Recall.

MODEL	ACCURACY	K-PRECISION	RECALL
Gemini-1.5-Pro	63.2	85.1	68.6
GPT-4o	55.3	86.7	58.3
Claude-3.5-Sonnet	36.1	77.6	49.0
Gemini-1.5-Flash	<u>55.1</u>	<u>82.3</u>	<u>61.7</u>
GPT-4o-mini	50.2	74.1	50.7
Claude-3-Haiku	32.6	70.9	40.9
Llama-3.1-70B	41.3	76.3	46.1
Mistral Large 2	34.1	72.4	42.9
Qwen2.5-72B	38.3	78.2	47.9
Command R+	41.7	71.7	47.4
DBRX	15.7	53.2	35.0
Phi-3.5-MoE	10.6	45.0	25.6
Mistral Small	35.5	77.2	41.1
Llama-3.1-8B	<u>25.2</u>	<u>61.3</u>	<u>32.0</u>
Qwen2.5-7B	22.2	56.9	30.3

the context size. Hence, even though most modern LLMs have large context sizes (upwards of 128k), they still struggle to reason even at the scale of 30-40k tokens.

G.10 Alternative Metrics of Evaluation for CHASE-QA

The metric of accuracy for CHASE-QA punishes models for not being concise and generating too many answer points that are not a part of the ground-truth answer. In this section, we present our experimental results with other softer evaluation metrics. We adapt two metrics that have been used by previous works for open-domain question answering (Adlakha et al., 2022): (1) **K-Precision**,

Table 12: Effect of prompt (see Figure 37) that explicitly instructs the model to solve CHASE-MATH problems by processing one sentence at a time.

MODEL	8-SHOT CoT (DEFAULT)	8-SHOT SENTENCE-BY-SENTENCE
Gemini-1.5-Pro	65.4	69.2
GPT-4o	59.8	61.4
Llama-3.1-70B	53.4	56.8

which for a particular example, evaluates whether all of the answer points in the model’s prediction are discussed in the documents, and (2) **Recall**, which evaluates whether all the ground truth answer points are a part of the model’s prediction. K-Precision is used to measure the *faithfulness* of the model’s prediction to the provided documents. Recall is used to measure the *correctness* of the model’s prediction compared to the ground-truth. We define both the metrics as binary per example. Similar to how we calculated accuracy, we use GPT-4o as a judge with the prompts provided in Figure 26 and Figure 27 respectively. The results are provided in Table 11.

Note that the errors in CHASE-QA pertain to the cases where the ground-truth answer may not completely encompass all the relevant information about the question that is mentioned in the documents. We believe that comparisons of models on the basis of recall is relatively less affected by the presence of such errors. This is because if a model has comparatively lesser recall, that means that it generated more responses where it did not include the ground-truth information (irrespective of whether it generated any extra relevant information for the question that is not in the ground truth).

G.11 Effect of Prompt for Solving CHASE-MATH

Considering the fact that CHASE-MATH is built by increasingly concatenating problems, we experiment with solving it using a different prompt format that explicitly instructs the model to process one sentence at a time, from the first to the last until it arrives at the final answer. We also illustrate this methodology in the prompt using 8 problems different from the original chain-of-thought prompt examples. Each of these new problems have a much higher reasoning depth. The prompt is provided in Figure 37. The results for 3 different models are shown in Table 12. While there is a clear increase in performance for all models, the task still remains difficult to solve, in general. Examples of errors made by models even with this better prompting technique are provided in Figure 13 and Figure 14.

H Related Work

Synthetic data for evaluation. Multiple prior works have used synthetic data in some form for evaluation. Sprague et al. (2024) generate a narrative QA dataset by sampling facts and prompting an LLM to write narratives. Bohnet et al. (2024) extract entities and reference chains from existing stories to generate questions. Gu et al. (2024) prompt CodeLlama (Rozière et al., 2024) to generate Python functions and inputs for code understanding. Chen et al. (2024) propose a self-challenge framework that generates new problems based on model errors. Yu et al. (2024b) use prompt-chaining to perturb existing evidences for generating new test cases to evaluate hallucination. Tyen et al. (2024) create the BIG-Bench Mistake dataset using CoT traces from PaLM 2 (Anil et al., 2023). Most of these works rely on seed data (Bohnet et al., 2024, Chen et al., 2024, Tyen et al., 2024, Yu et al., 2024b), human involvement (Chen et al., 2024, Tyen et al., 2024), or focus on niche sub-domains (Gu et al., 2024, Sprague et al., 2024). In contrast, we propose general principles for scalable, fully automatic benchmark generation across diverse tasks, such as information-seeking QA and repository-level code generation, without always relying on pre-existing data. Similar to our work, AutoBench (Li et al., 2025) automates benchmark creation across domains, but its difficulty stems from knowledge/tool-use gaps, whereas ours emphasizes complex reasoning. Our approach also leverages the idea of verifying *simpler sub-steps* during generation to ensure data correctness.

Task-specific synthetic data. Recent works have explored generating synthetic datasets for content-grounded QA tasks. Dai et al. (2022) and Yehudai et al. (2024) use LLMs to generate questions based

on Wikipedia and Web text. In contrast, we design a benchmark for document-based information-seeking questions that model realistic situations. Moreover, we generate the entire documents using LLMs, allowing a higher degree of control. In the code domain, Yu et al. (2024c) and Wei et al. (2024) leverage existing code data to generate examples for instruction tuning. We instead focus on repository-level code completion, generating both the repositories and their test suites from scratch. Synthetic math data generation has also been explored by previous works (Liu et al., 2023, Lu et al., 2024, Yu et al., 2024a) that generate new problems by directly prompting with existing examples. Similar to our work, some prior works have focused on creating challenging math problems. Shah et al. (2024) employ a human-in-the-loop approach and prompt LLMs to design problems requiring a given set of core skills. Liu et al. (2024) iteratively prompt an LLM with a seed question to produce increasingly complex variants. In this work, we design a fully automated pipeline to craft grade-school level math problems that are challenging to solve even for the LLM that generated them.

I Error Analysis

We provide examples for two types of errors, those made while solving the benchmarks, and those made while generating the benchmarks.

I.1 Errors made while solving CHASE benchmarks.

Figure 8 provides an example of an error made by Gemini-1.5-Pro on a problem from CHASE-QA. The model fails to mention two important points relevant for answering the question, which have been discussed in the documents. Figure 9 provides an example of an error made by o4-mini on a problem from CHASE-QA. The model falls into the trap of thinking that the distracting information is relevant for answering the question. These instances provide qualitative examples of how even the most powerful models are unable to properly pay attention to all parts of a long-context and may miss some important information or fail to make the appropriate connections.

Figure 10 provides an example of an error made by Gemini-1.5-Pro in generating the correct code for a problem in CHASE-CODE. The model generates most of the code correctly, but for a particular objective, it gets confused in choosing to call the right helper function from the long-context code repository. This example qualitatively illustrates that doing well on this task requires not only a good understanding of the user-specified objectives, but also requires an in-depth understanding of all parts of the code repository.

Figure 11 provides an example of an error made by Gemini-1.5-Pro in solving a math word problem from CHASE-MATH. The model executes most of the reasoning steps correctly but fails at the last one. Figure 12 provides an example of an error made by o4-mini in solving a math word problem from CHASE-MATH. The model misunderstood a crucial aspect of the problem and made a mistake. Such errors show that while these models are very capable of solving even advanced olympiad-level math problems, they can still sometimes fail at language-heavy math reasoning.

I.2 Errors made in the generation process when using CHASE.

In Figure 15, we show an error made in the generation process of CHASE-QA by GPT-4o. In the document generation stage, the model generated a document which contained extra information that was directly relevant for answering the given question but was not included in the ground-truth answer. This is also a failure case of our verification engine (the one that uses the prompt in Figure 22) which failed to detect the presence of this extra relevant information in the generated document. We believe such errors can be further reduced by using an ensemble of verifiers to carry out each verification task.

Figure 16 provides an example of an error made by GPT-4o-mini while generating the CHASE-MATH benchmark. The model’s generated answer did not correspond to its generated problem. This is also a failure for both the generator and verifier. Such failures may be reduced by using an ensemble of stronger LLMs for verification instead.

897 **J Prompts**

898 **J.1 Prompts for CHASE-QA**

899 In this section, we outline the exact prompts for all experiments related to CHASE-QA:

- 900 • Generating scenarios: Figure 17
- 901 • Generating QA pairs: Figure 18
- 902 • Generating distracting information: Figure 19
- 903 • Generating documents: Figure 20
- 904 • Verifying distracting information: Figure 21
- 905 • Verifying absence of relevant information: Figure 22
- 906 • Verifying presence of answer: Figure 23
- 907 • Solving the task: Figure 24
- 908 • LLM-as-a-judge: Figure 25

909 **J.2 Prompts for CHASE-CODE**

910 In this section, we outline the exact prompts for all experiments related to CHASE-CODE:

- 911 • Generating helper functions: Figure 28
- 912 • Generating problem statement and answer code: Figure 29
- 913 • Generating test code: Figure 30
- 914 • Generating test for verifying if function executes: Figure 31
- 915 • Verify correctness of problem statement: Figure 32
- 916 • Solving the task: Figure 33

917 **J.3 Prompts for CHASE-MATH**

918 In this section, we outline the exact prompts for all experiments related to CHASE-MATH:

- 919 • Breaking down seed MWPs: Figure 34
- 920 • Creating continuation of MWP: Figure 35
- 921 • Solving MWP: Figure 36
- 922 • Solving MWP sentence-by-sentence: Figure 37

Error Analysis: Solving CHASE-QA - Gemini-1.5-Pro

Question: What are the regulations and financial support options available for starting an artisanal cheese-making business in Vermont?

Ground-truth Answer:

- Obtain a Dairy Processing License from the Vermont Agency of Agriculture, Food & Markets.
- Adhere to the Vermont Department of Environmental Conservation's (VT DEC) regulations regarding wastewater management.
- Apply for the Vermont Economic Development Authority (VEDA) Entrepreneurial Loan Program for startup financing.
- Comply with the Vermont Department of Taxes' requirements for sales and use tax on dairy products.
- Secure a Vermont Working Lands Enterprise Initiative Grant for equipment and facility improvement.

distracting Information:

- Obtain a Microbrewery License from the New York State Liquor Authority (NYSLA).
- Adhere to the New York State Department of Environmental Conservation (NYSDEC) regulations regarding waste disposal.

[Redacted]

Prediction:

- Comply with the Vermont Department of Taxes' requirements for sales and use tax on dairy products, including registering for a sales tax permit, tracking sales, and remitting collected tax. (Document 10)
- Adhere to the Vermont Department of Environmental Conservation's (VT DEC) regulations regarding wastewater management, ensuring waste byproducts are treated and disposed of properly to prevent water contamination. (Document 13)
- Obtain a Dairy Processing License from the Vermont Agency of Agriculture, Food & Markets. (Document 12)

LLM Evaluation:

Result: False. The prediction is missing two main points from the ground-truth answer:

1. Applying for the Vermont Economic Development Authority (VEDA) Entrepreneurial Loan Program for startup financing.
2. Securing a Vermont Working Lands Enterprise Initiative Grant for equipment and facility improvement.

These are crucial financial support options that are not mentioned in the prediction.

Figure 8: Example of error made by Gemini-1.5-Pro on CHASE-QA. The GPT-4o judge correctly evaluates the prediction to be wrong because it fails to mention some major points from the ground-truth answer which were discussed in the documents.

Error Analysis: Solving CHASE-QA - o4-mini

Question: Which elements are essential when evaluating the effectiveness and satisfaction of remote work policies for employees in the healthcare industry?

Ground-truth Answer:

- Availability and quality of remote work tools and technology.
 - Training and support provided for remote work adaptation.
 - Communication and collaboration effectiveness within remote teams
-

Distracting Information:

- Availability and diversity of training formats (e.g., online courses, workshops, mentoring).
 - Alignment of professional development opportunities with career advancement goals.
 - Impact of professional development on teaching methodologies and student performance.
 - Frequency and quality of feedback from employees regarding the professional development programs.
-

Prediction:

- Comprehensive training and support programs (e.g., time management, conducting uninterrupted patient consultations, digital-health outreach).
 - High-quality remote-work technology and tools (e.g., high-speed internet, secure communication platforms, specialized telehealth software).
 - Effective communication and collaboration platforms and practices (e.g., real-time case discussions, coordinated care planning, digital teamwork).
 - Evaluate professional development programs using online courses, in-person workshops, and mentoring sessions.
-

LLM Evaluation:

Result: False. While the prediction correctly includes comprehensive training and support for remote work adaptation, high-quality remote work tools and technology, and effective communication and collaboration within remote teams—matching the main points in the ground-truth answer—it also mentions the distracting answer point of evaluating professional development programs.

Figure 9: Example of error made by o4-mini on CHASE-QA. The GPT-4o judge correctly evaluates the prediction to be wrong because it mentions the distracting point which was discussed in the documents.

Error Analysis: Solving CHASE-CODE (Algorithms) - Gemini-1.5-Pro

Objectives:

[Redacted]

- For each coordinate from the previous sub-goal, calculate the Manhattan distance from the origin (0,0) for the interesting cells found, returning a list of tuples.

Ground-truth Answer Code:

```
1 import numpy as np
2 from matrix_decomposition import svd_reconstruction
3 from entity_extractor import extract_entities
4 from meeting_scheduling import can_schedule_meetings
5 from grid_analyzer import high_value_coordinates
6
7 def complex_analysis(matrix, k, text, entities, intervals,
8   threshold):
9     reconstructed_matrix = svd_reconstruction(matrix, k)
10    extracted_entities = extract_entities(text, entities)
11    can_schedule = can_schedule_meetings(intervals, 0)
12    interesting_cells = high_value_coordinates(
13        reconstructed_matrix, threshold)
14    interesting_cells_with_distances = [((i, j), abs(i) + abs(j))
15        for ((i, j), _) in interesting_cells]
16    return reconstructed_matrix, extracted_entities,
17        can_schedule, interesting_cells_with_distances
```

Predicted Code:

```
1 import numpy as np
2 from matrix_decomposition import svd_reconstruction
3 from entity_extractor import extract_entities
4 from meeting_scheduling import can_schedule_meetings
5 from matrix_analyzer import find_interesting_cells
6
7 def complex_analysis(matrix, k, text, entities, intervals,
8   threshold):
9     reconstructed_matrix = svd_reconstruction(np.array(matrix),
10        k)
11     extracted_entities = extract_entities(text, entities)
12     can_schedule = can_schedule_meetings(intervals, 0)
13     interesting_cells = find_interesting_cells(
14         reconstructed_matrix, threshold)
15     interesting_cells_with_distances = [((row, col), abs(row) +
16         abs(col)) for row, col in interesting_cells]
17     return reconstructed_matrix, extracted_entities,
18         can_schedule, interesting_cells_with_distances
```

Figure 10: Example of error made by Gemini-1.5-Pro on an example from the *algorithms* split of CHASE-CODE. The prediction incorrectly calls the `find_interesting_cells` helper function (see line number 11) which does not calculate manhattan distances as requested by the objective.

Error Analysis: Solving CHASE-MATH - 8-shot chain-of-thought - Gemini-1.5-Pro

Question:

Harry slept 9 hours last night. His friend James slept only $\frac{2}{3}$ of what Harry slept. After their sleep, Harry decided to wake up early and went for a jog. He invited James to join him, and they jogged for 4 times the additional hours Harry slept compared to James. They both felt energized and decided to go for breakfast, where they calculated the total hours they spent jogging. After their breakfast, Harry and James sat down to review their fitness goals. They discussed their jogging session and decided it would be beneficial to repeat such sessions. However, they wanted to gauge how many smaller jogging sessions equal the time they previously spent jogging. James suggested that they break down their jogging time into segments of 3 hours to see how many complete segments they can fit within their total jogging time. After evaluating their jogging goals, Harry and James decided to increase the duration of their future jogging sessions. They considered adding an additional hour to each segment to further enhance their fitness regimen, while keeping the same total jog time of all sessions combined. After successfully completing their new jogging regimen for a month, Harry and James noticed that their stamina had increased significantly. They decided to celebrate this achievement by doubling their number of sessions for the future. They planned to increase from their current number of complete sessions to a total that is double their current sessions. They are excited to see how their increased devotion to jogging will impact their overall fitness. After a successful month of their increased jogging regimen, Harry and James wanted to keep track of their progress. They decided to add rest sessions into their training schedule to avoid burnout. For every $\frac{1}{3}$ of the number of jogging sessions in their regimen, they would incorporate 1 rest session. If they have done 4 rest sessions so far, how many total jogging sessions have they finished?

Ground-truth Answer:

James slept 9 hours * $\frac{2}{3}$ = 6 hours. So, Harry slept 9 - 6 = 3 hours more than James. They jogged for 4 times that amount, so 12 hours together. We then divide 12 by 3 to get 4 complete sessions. Now divide the 12 hours by the new session duration (4 hours) which yields 3 sessions. Doubling that quantity results 6. They add rest sessions for every $\frac{1}{3}$ * 6 = 2 jogging sessions. So for 4 rest sessions, they have 4 * 2 = 8 jogging sessions. ##### 8.0

Prediction:

Let's break down this problem step-by-step:

1. ****James's sleep:**** James slept $\frac{2}{3}$ of Harry's 9 hours, which is $(\frac{2}{3}) * 9 = 6$ hours.
 2. ****Additional hours Harry slept:**** Harry slept 9 hours - 6 hours = 3 more hours than James.
 3. ****Jogging time:**** They jogged for 4 times the additional hours Harry slept, meaning they jogged for $4 * 3 = 12$ hours.
 4. ****Segments of jogging:**** They want to break down their jogging time into 3-hour segments, so they have 12 hours / 3 hours/segment = 4 segments.
 5. ****Increased segment duration:**** They increase each segment by 1 hour, making each segment 3 hours + 1 hour = 4 hours long.
 6. ****Total sessions:**** Since they keep the same total jogging time, they still have 12 hours of jogging. With the new segment length, they have 12 hours / 4 hours/session = 3 sessions.
 7. ****Doubled sessions:**** They double their number of sessions from 3 to $3 * 2 = 6$ sessions.
 8. ****Rest sessions and total jog sessions:**** They incorporate 1 rest session for every $\frac{1}{3}$ of their jogging sessions. Since they've done 4 rest sessions, they have completed 4 rest sessions * 3 jog sessions/rest session = 12 jogging sessions.
- The answer is 12.

Figure 11: Example of error made in the last step by Gemini-1.5-Pro on an example from CHASE-MATH when prompted with the default 8-shot chain-of-thought.

Error Analysis: Solving CHASE-MATH - 8-shot chain-of-thought - o4-mini

Question:

Marcus ordered 5 croissants at \$3.00 apiece, 4 cinnamon rolls at \$2.50 each, 3 mini quiches for \$4.00 apiece and 13 blueberry muffins that were \$1.00 apiece. At checkout, Marcus shows his loyalty card that gives him 10% off of his purchase. Marcus then sees that the bakery offers a special deal: If a customer spends over \$30 after discounts, they can add an extra item worth \$10 for an additional \$5. Marcus considers whether to purchase this extra item based on his current total; however, he instead decides to spend 30% of his potential total bill at the bakery on a different treat and save the rest of the money. After determining how much he has left, Marcus decides to invest the remaining amount into a savings account that offers a fixed interest rate of 5% annually. He wonders how much interest he would earn if he left the remaining amount in the savings account for one year. How much interest will Marcus earn after one year if he invests his remaining amount at a 5% interest rate?

Ground-truth Answer:

5 croissants cost \$3.00 each so they cost $5 \times 3 = \$15.00$ 4 cinnamon rolls cost \$2.50 each so they cost $4 \times 2.5 = \$10.00$ 3 mini quiches cost \$4.00 each so they cost $3 \times 4 = \$12.00$ 13 blueberry muffins cost \$1.00 each so the cost $13 \times 1 = \$13.00$ All total, his pastries costs $15 + 10 + 12 + 13 = \$50.00$ His has a 10% coupon for his \$50.00 purchase so that saves him $.10 \times 50 = \$5.00$ His total bill is \$50 minus his discount of \$5.00 so $50 - 5 = \$45.00$ To determine how much Marcus will have remaining after spending 30% of his total bill, we first calculate 30% of his total bill of \$45. That is calculated as $0.30 \times 45 = \$13.50$. Then, we subtract this amount from his original total bill: $45 - 13.50 = \$31.50$. This gives us the remaining amount Marcus has after his additional expenditure. To find out how much interest Marcus will earn, we apply the formula for simple interest, which is $I = P \times r \times t$, where I is the interest, P is the principal amount (the remaining amount), r is the rate (in decimal), and t is the time in years. Here, P is \$31.50, r is 0.05, and t is 1. Therefore, the interest earned after one year would be $31.50 \times 0.05 \times 1 = \1.575 . ##### 1.575

Prediction:

Lets work through step by step.

1. Compute the original subtotal: 5 croissants at \$3.00 each: $5 \times 3.00 = \$15.00$ 4 cinnamon rolls at \$2.50 each: $4 \times 2.50 = \$10.00$ 3 mini quiches at \$4.00 each: $3 \times 4.00 = \$12.00$ 13 blueberry muffins at \$1.00 each: $13 \times 1.00 = \$13.00$ Subtotal = $15.00 + 10.00 + 12.00 + 13.00 = \50.00
2. Apply the 10% loyalty card discount: Discount = $0.10 \times 50.00 = \$5.00$ Total after discount = $50.00 - 5.00 = \$45.00$
3. Hes eligible for the spend over \$30 special. If hed taken it, he would pay an extra \$5 for a \$10 item, making a potential total of $45.00 + 5.00 = \$50.00$
4. Instead of actually buying that extra item, he would be spending \$50, but chooses to spend 30% of that (\$50) on another treat and save the rest. Amount actually spent on the treat: $0.30 \times 50.00 = \$15.00$ Amount saved (and available to invest): $0.70 \times 50.00 = \$35.00$
5. He invests \$35.00 at 5% for one year. The interest earned is Interest = $0.05 \times 35.00 = \$1.75$ Therefore, the one year interest is 1.75.

Figure 12: Example of error made by o4-mini on an example from CHASE-MATH when prompted with the default 8-shot chain-of-thought.

Error Analysis: Solving CHASE-MATH - 8-shot sentence-by-sentence - Gemini-1.5-Pro

Question:

James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. After a month of consistent training, James feels he needs to increase his speed. He decides to take a break from sprinting for one week to focus on recovery. During this recovery week, he runs a shorter distance of 30 meters for only 2 sprints, 3 times that week instead. After his recovery week, James feels rejuvenated and ready to resume his training. He decides to continue running but at double the distance he covered during his recovery week for each day the next week, aiming to improve his overall performance. He runs this increased distance for 7 days straight, maintaining consistency every day. After running consistently throughout the week, James notices a small injury starting to develop in his leg. Concerned about this, he decides to cut back on his total running distance for the following week. To ensure he allows for proper recovery and does not worsen the injury, he plans to reduce his total running distance from the prior week by 420 meters. How much distance will James run in the following week after the reduction?

Ground-truth Answer:

He sprints $3 * 3 = 9$ times So he runs $9 * 60 = 540$ meters In the recovery week, James runs 2 sprints each day for 3 days, covering 30 meters each sprint. Therefore, the total distance covered during the recovery week is calculated as follows: he runs 2 sprints/day * 3 days = 6 sprints. Then, multiplying by the distance per sprint gives 6 sprints * 30 meters/sprint = 180 meters. To find out how much less he runs compared to his regular week, we subtract the total distance he runs that week from the distance he typically runs in a week, which is $540 - 180 = 360$ meters less. James runs a distance that is double what he covered in the recovery week, which is 180 meters. Therefore, if he runs this doubled distance for 7 days, we first calculate the distance he covers in one day, which is $180 \text{ meters} * 2 = 360$ meters. Over the course of 7 days, the total distance will be $360 \text{ meters/day} * 7 \text{ days}$. To find the distance James will run in the following week after reducing his initial total distance by 420 meters, I will need to subtract 420 from the total distance he ran in the previous week, which was 2520.0 meters. Hence, the calculation will be: $2520.0 - 420 = 2100.0$ meters. #####2100.0

Prediction:

Sentence 1: James runs 3 sprints * 60 meters/sprint = 180 meters per day.

Sentence 2: He runs 180 meters/day * 3 days/week = 540 meters per week.

Sentence 3: Nothing to solve.

Sentence 4: During his recovery week, he runs 30 meters/sprint * 2 sprints = 60 meters per day.

Sentence 5: He runs this shorter distance 3 times a week, so $60 \text{ meters/day} * 3 \text{ days/week} = 180$ meters.

Sentence 6: Nothing to solve.

Sentence 7: He doubles his recovery week distance, running $60 \text{ meters} * 2 = 120$ meters per day.

Sentence 8: He runs this distance for 7 days straight, totaling $120 \text{ meters/day} * 7 \text{ days} = 840$ meters.

Sentence 9: Nothing to solve.

Sentence 10: He reduces his total running distance by 420 meters, meaning he'll run $840 \text{ meters} - 420 \text{ meters} = 420$ meters.

The answer is 420.

Figure 13: Example of error made by Gemini-1.5-Pro on an example from CHASE-MATH when prompted with 8-shot sentence-by-sentence solving examples.

Error Analysis: Solving CHASE-MATH - 8-shot sentence-by-sentence - Gemini-1.5-Pro

Question:

Rani has ten more crabs than Monic, who has 4 fewer crabs than Bo. If Bo has 40 crabs, calculate the total number of crabs the three have together. After counting their crabs, Rani, Monic, and Bo decide to share their crabs equally among themselves for a crab feast. However, before they can do that, Rani finds out that they must provide 2 additional crabs to a local aquarium as a donation. After their crab feast, Rani, Monic, and Bo decide to host another gathering for their friends. They plan to invite 5 additional friends to enjoy crabs. Each person, including Rani, Monic, Bo, and their friends, should receive an equal amount of crabs for this new gathering. They go and buy new crabs which are in total 3 times the number of crabs each person ate after the donation. How many crabs did they have altogether to share with the additional friends in the end?

Ground-truth Answer:

If Bo has 40 crabs, then Monic, who has 4 fewer crabs than Bo, has $40 - 4 = 36$ crabs. Bo and Monic have a total of $36 + 40 = 76$ crabs. Since Monic has 36 crabs, Rani's number of crabs is $36 + 10 = 46$. Together, the three have $76 + 46 = 122$ crabs. First, calculate how many crabs remain after the donation: $122 - 2 = 120$. Now, to find out how many crabs each person will receive, divide the remaining number of crabs (120) by 3, since there are three individuals: $120 / 3 = 40$. Since each of them originally ate 40 crabs, if we multiply that number by 3, we calculate the total number of crabs available in the end. $120 * 3 = 360$.

Prediction:

Sentence 1: Monic has $40 - 4 = 36$ crabs.

Sentence 2: Rani has $36 + 10 = 46$ crabs.

Sentence 3: Bo has 40 crabs.

Sentence 4: Together they have $46 + 36 + 40 = 122$ crabs.

Sentence 5: They donate 2 crabs, leaving them with $122 - 2 = 120$ crabs.

Sentence 6: They share the 120 crabs equally, so each person gets $120 / 3 = 40$ crabs.

Sentence 7: They invite 5 more friends, making a total of $3 + 5 = 8$ people.

Sentence 8: They buy new crabs totaling 3 times the number each person ate after the donation, which is $3 * 40 = 120$ crabs.

Sentence 9: Including the 120 crabs they bought and the remaining crabs they shared after donating, they had $120 + 120 * 3 = 120 + 360 = 480$ crabs altogether to share with the additional friends.

The answer is 480.

Figure 14: Example of error made by Gemini-1.5-Pro on an example from CHASE-MATH when prompted with 8-shot sentence-by-sentence solving examples. The model fails to understand that the 120 crabs they had earlier have already been eaten and that only the new 120 crabs bought in the end are available to share with the new friends.

Error Analysis: Generating CHASE-QA - GPT-4o

Documents:

[Redacted]

Document 2: Title: Semiconductor Advancements and Emerging Trends Text: The semiconductor industry has witnessed numerous advancements over the past few years, driven by the rapid pace of technological innovation. One of the most promising materials being explored for next-generation processors is Gallium Nitride (GaN). GaN is being investigated for its potential to significantly improve the speed and efficiency of processors. Its unique properties, such as high electron mobility and wide bandgap, enable faster electron transmission and greater thermal conductivity. This makes GaN an ideal candidate for applications requiring high power and high-frequency operation.

Developments in artificial intelligence algorithms for natural language processing continue to push the boundaries of what machines can understand and generate in human language. Meanwhile, the impact of 5G technology on mobile communications and IoT devices opens up new possibilities for high-speed, low-latency connectivity.

Additionally, **Graphene-based semiconductors hold the potential to revolutionize processor technology**. Graphene, a single layer of carbon atoms arranged in a hexagonal lattice, exhibits exceptional electrical properties. Its high electrical conductivity and mechanical strength make it an attractive material for creating faster and more efficient processors. Researchers are working on overcoming the challenges associated with mass production and integration of graphene into existing manufacturing processes.

[Redacted]

Question: What new semiconductor materials are being explored for next-generation processors, and which startups have recently secured funding to advance these technologies?

Ground-truth Answer:

- Gallium Nitride (GaN) is being investigated for its potential to improve the speed and efficiency of processors.
- Anokiwave, a pioneering startup in GaN-based semiconductors, has recently raised \$45 million in Series C funding.
- Paragraf, a startup developing electronic devices for the semiconductor industry, secured \$30 million in a recent funding round.
- The development of molybdenum disulfide (MoS2) as a semiconductor material could lead to significant advancements in processor miniaturization.

Figure 15: Example of an error made in the generation process of CHASE-QA. The model generated a document which contained additional relevant information about the question (the point about graphene-based semiconductors highlighted in bold in the text) which is not a part of the ground-truth answer.

Error Analysis: Generating CHASE-MATH - GPT-4o

Question: Rani has ten more crabs than Monic, who has 4 fewer crabs than Bo. If Bo has 40 crabs, calculate the total number of crabs the three have together. After counting their crabs, Rani, Monic, and Bo decide to share their crabs equally among themselves for a crab feast. However, before they can do that, Rani finds out that they must provide 2 additional crabs to a local aquarium as a donation. After their crab feast, Rani, Monic, and Bo decide to host another gathering for their friends. They plan to invite 5 additional friends to enjoy the remaining crabs. Each person, including Rani, Monic, Bo, and their friends, should receive an equal amount of crabs for this new gathering. They want to figure out how many crabs they have available for their friends while ensuring that each person gets an equal share. If they multiply the number of crabs each had after the donation by 3, how many crabs did they have altogether to share with the additional friends?

Ground-truth Answer:

If Bo has 40 crabs, then Monic, who has 4 fewer crabs than Bo, has $40 - 4 = 36$ crabs. Bo and Monic have a total of $36 + 40 = 76$ crabs. Since Monic has 36 crabs, Rani's number of crabs is $36 + 10 = 46$. Together, the three have $76 + 46 = 122$ crabs. First, calculate how many crabs remain after the donation: $122 - 2 = 120$. Now, to find out how many crabs each person will receive, divide the remaining number of crabs (120) **by 3, since there are three individuals**: $120 \div 3 = 40$. Since each of them originally received 40 crabs, if we multiply that number by 3 individuals (Rani, Monic, and Bo), we calculate the total number of crabs available for sharing. ####120.0

Figure 16: Example of an error made in the generation process of CHASE-MATH. The reasoning in the answer wrongly considers the number of people for which to divide crabs to be 3 while the context has added new information that there are 5 additional friends to consider.

CHASE-QA: Generating Scenarios

System Prompt: You are an expert generator of data.

You are a research scientist. You want to make data to test an advanced question answering system.

Give me 5 examples of real-life scenarios where a `USER_PERSONA` may seek information in a `COLLECTION_OF_DOCS`. Do not consider educational or historical scenarios.

Some examples are:

`USER_PERSONA`: College student
`COLLECTION_OF_DOCS`: Intranet on the university website

`USER_PERSONA`: Intern doctor at a hospital
`COLLECTION_OF_DOCS`: Encyclopedia of diseases

`USER_PERSONA`: Immigrant in NYC
`COLLECTION_OF_DOCS`: Laws on renting and subletting

`USER_PERSONA`: HR manager at a top law firm
`COLLECTION_OF_DOCS`: Court and newspaper records

`USER_PERSONA`: Scientist at an NGO
`COLLECTION_OF_DOCS`: Government website for Income Tax

Answer in the following format:

`USER_PERSONA`:

`COLLECTION_OF_DOCS`:

Figure 17: Prompt for generating diverse scenarios for CHASE-QA.

CHASE-QA: Generating QA Pairs

System Prompt: You are an expert generator of data. Do not use ** to start lines or denote points.

You are a research scientist. You want to make data to test an advanced question answering system.

Give me an example question and corresponding answer that a {USER_PERSONA} may ask that compulsorily requires searching a {COLLECTION_OF_DOCS}. Make questions that cannot be answered directly with general knowledge but necessarily require some uncommon information that is present in some documents. The answer must be very specific and written in bullet points, so that it is easier to objectively evaluate. Depending on the question, the answer can have anything between 3-6 bullet points without any sub-points.

The answer to the question you create must be scattered across different documents (at least 3). Assign each point of the answer to a specific document in which that point will be discussed. You may assign multiple points to the same document, but each point must only be assigned to a single document. You must state the title and answer points assigned for each of the documents.

Answer in the following format:

Question: <Question>
Answer: <Answer>

Document 1 Title: <Title>
Document 1 Answer points assigned: <Points>

Document 2 Title: <Title>
Document 2 Answer points assigned: <Points>

and so on...

Figure 18: Programmatic prompt for generating question-answer pairs for CHASE-QA.

CHASE-QA: Generating distracting Information QA Pairs

System Prompt: You are an expert generator of data. Do not use ** to start lines or denote points.

You are a research scientist. You want to make hard data to test an advanced question answering system. You are given a question that a {USER_PERSONA} might want answered, along with the corresponding answer, and information of documents from {COLLECTION_OF_DOCS} that are important for answering that question.

Original Question: {QUESTION}

Original Answer:
{ANSWER}

Original Documents Information:
{DOCS_INFORMATION}

You must generate an adversarial question, adversarial answer, and corresponding adversarial documents that ask for something different but on similar topics or type so that it is difficult to answer the original question. Examples of how adversarial questions should look like are provided below:

Original Question: What are the best activities to do in Montreal, Canada during the winter season?

Adversarial Question: What activities should I look at when visiting Tokyo during the summer?

[Redacted]

Also provide an answer to the adversarial question, which is similar in style to the original answer, but differs significantly in information or specifics. The answer points for the adversarial question should be written in context of that adversarial question, so that they cannot be confused with the original question. Note that none of the points appearing in the original answer should be present in the answer to the adversarial question.

The answer to the adversarial question you craft must be scattered across different documents (at least 3) separate from the original answer documents. Assign each point of the adversarial answer to a specific document in which that point will be discussed. You may assign multiple points to the same adversarial document, but each point must only be assigned to a single adversarial document. You must state the title and adversarial answer points assigned for each of the adversarial documents. These adversarial documents should not have any overlapping information with the original answer documents.

Answer in the following format:

[Redacted]

Figure 19: Programmatic prompt for generating distracting information question-answer pairs for CHASE-QA.

CHASE-QA: Generating Documents

System Prompt: You are an expert data generator. Following the instruction, you must generate long and correct documents.

You need to generate the documents for an example of a retrieval based Question Answering Task.

The task consists of n documents provided in English text that consist of information about different topics and a question. To answer the question correctly compulsorily requires using some of the information in some subset of the documents provided.

Given below is a situation faced by {USER_PERSONA} when searching {COLLECTION_OF_DOCS}. The question-answer pair is:

Question: {QUESTION}

Answer: {ANSWER}

Given below are the assigned answer points for each document.

{DOCS_INFORMATION}

Your job is to create long documents according to this information. For each document, first create 10-12 unique other points that are in no way related to the topic of the question and answer (different points for each document). These points should discuss very different things about a similar but different topic. Then use these points along with the assigned answer points to create a long document (at least 700 words long). The assigned answer points must be discussed taking into account the question. You must only discuss about these points and nothing else. Change the order of the points so that the answer points are embedded inside the document. Assign an appropriate title to the document. Do not summarize or conclude the document in the end.

Additionally, ensure that the documents you create do not have any information related to the following distracting question-answer pairs. You should create documents that discuss topics that are completely different from the following information.

{DISTRACTING_QUESTIONS_ANSWERS}

Give output in the following format:

Document 1:

Title: <Title>

Question: {QUESTION}

Answer points assigned [Only these points must be covered with respect to the question]:

<Points>

Other unrelated points created: <Points>

Text:

<Document Text>

[Redacted]

and so on...

Figure 20: Programmatic prompt for generating documents for CHASE-QA.

CHASE-QA: Verifying Distracting Information
<p>System Prompt: You are an expert at verifying data.</p> <hr/>
<p>You are given a question and an answer. You must check whether the answer is even partially relevant for answering the question. If the answer is not relevant at all, output "False" to "Relevance". Otherwise, if and only if the answer discusses information that is at least partially necessary to answer the question, output "True".</p>
<p>Question: {QUESTION}</p>
<p>Answer: {DISTRACTING_ANSWERS}</p>
<p>Give output in the following format: Relevance: True/False</p>

Figure 21: Programmatic prompt for verifying irrelevance of distracting information for CHASE-QA.

CHASE-QA: Verifying Absence of Relevant Information

System Prompt: You are an expert at verifying data.

You are given a document followed by a question and some answer points. You must check whether there are any additional major points in the document that provide relevant information for answering the question that are currently missing from the answer. Follow these instructions:

1. Do not look for exact phrases or explicit mentions since the answer can have points that are a paraphrase of the same broad information.
2. It is ok if the document provides more specifics or details about the points already in the answer or if it discusses them in more depth by introducing related information so you can ignore that.
3. Check if the document introduces a new “major” idea or point that is crucial for answering the question and is not at all mentioned in the answer and is not an extension of the existing points in the answer.
4. Your job is not to check if the question can be sufficiently answered. You should ignore if the document or answer points are missing any points that are needed in the answer to the question.

If the document is not introducing major new points pertaining to the answer, output “False” to “Presence of Extra Points” without giving any explanation. Otherwise, if and only if the document discusses major additional points that are necessary to answer the question, output “True” and mention only the extra major points discussed.

Document:
{Document}

Question: {QUESTION}

Answer Points:
{ANSWER}

Give output in the following format:
Presence of Extra Points: True/False
Extra Points Mentioned (if any):

Figure 22: Programmatic prompt for verifying absence of relevant information in the documents for CHASE-QA.

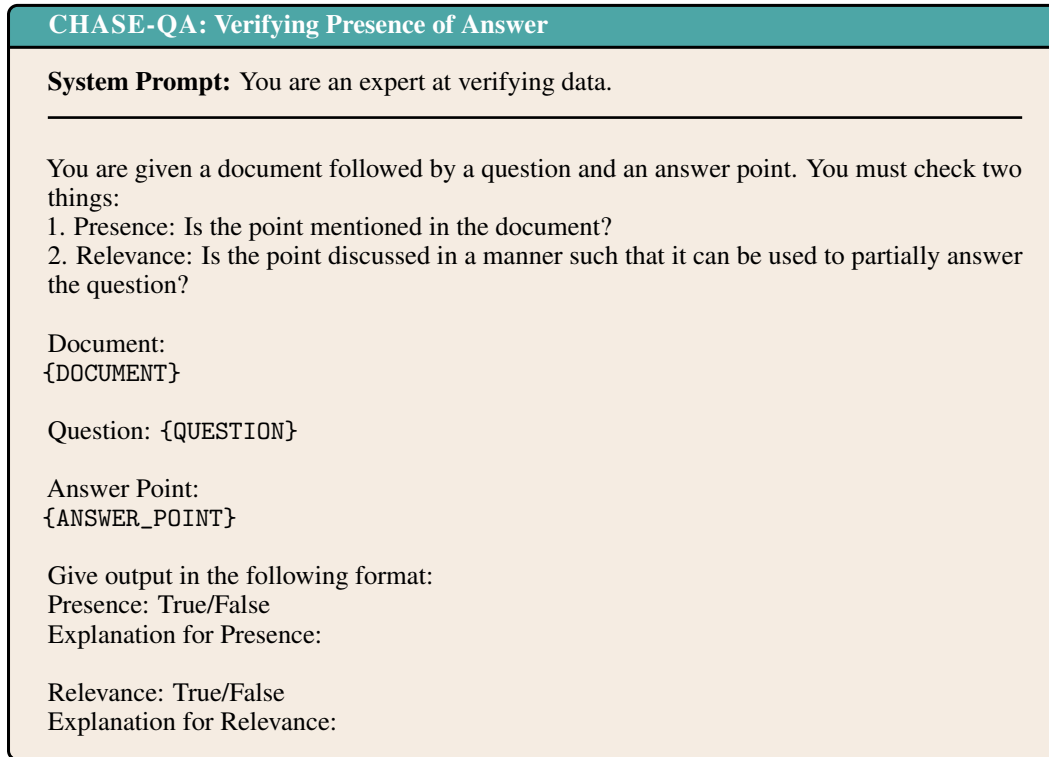


Figure 23: Programmatic prompt for verifying presence of ground-truth answer in the documents for CHASE-QA.

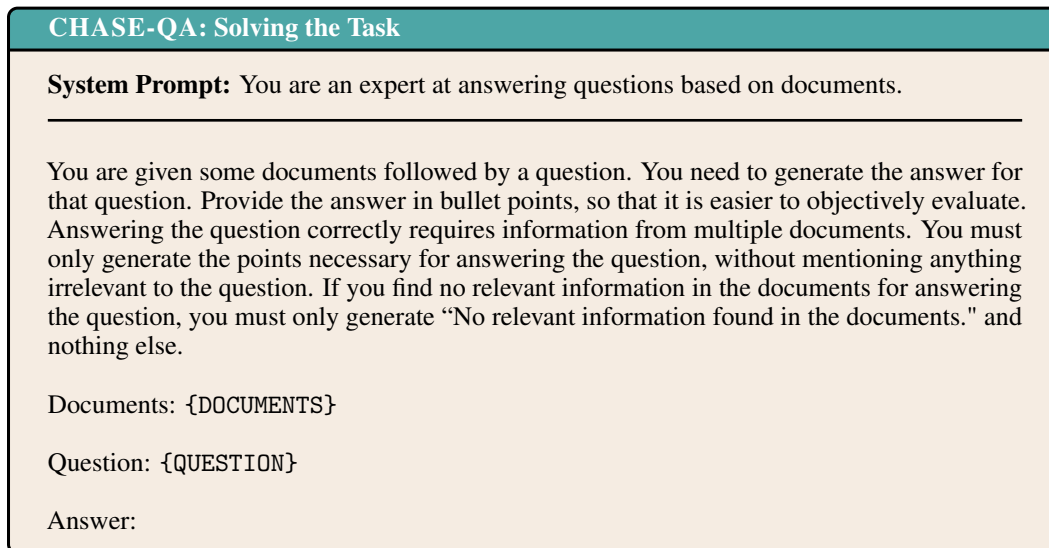


Figure 24: Programmatic prompt for solving examples in CHASE-QA.

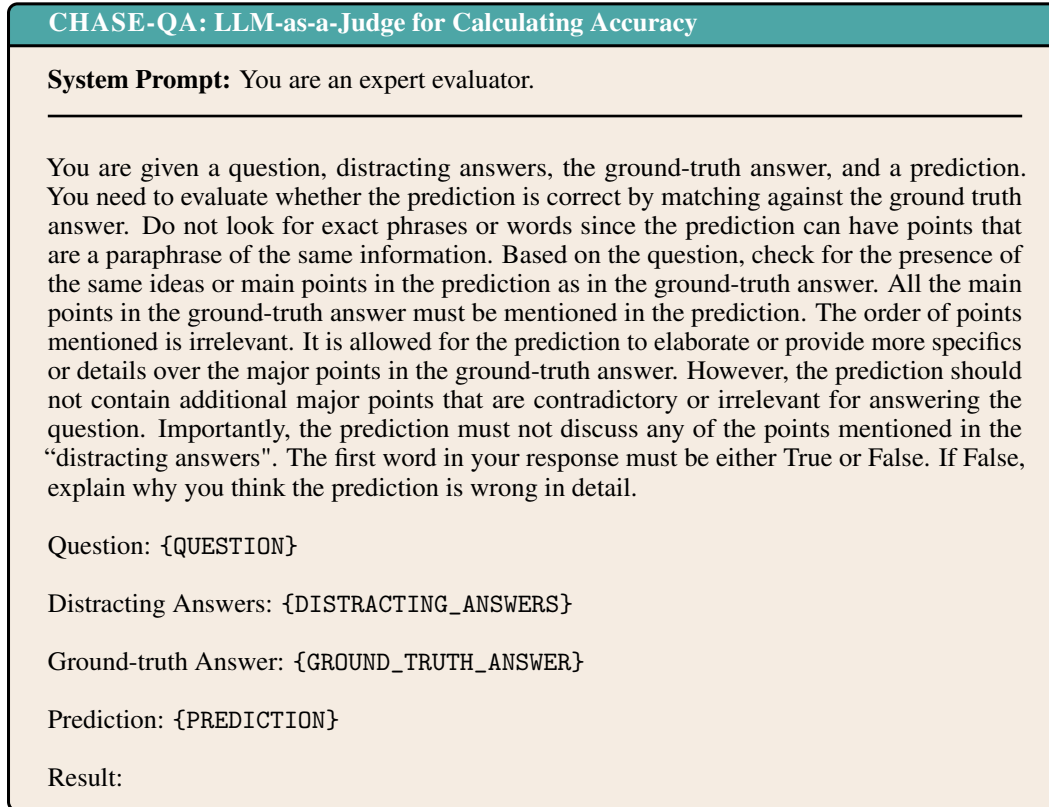


Figure 25: Programmatic prompt for evaluating accuracy of predictions of models for problems in CHASE-QA.

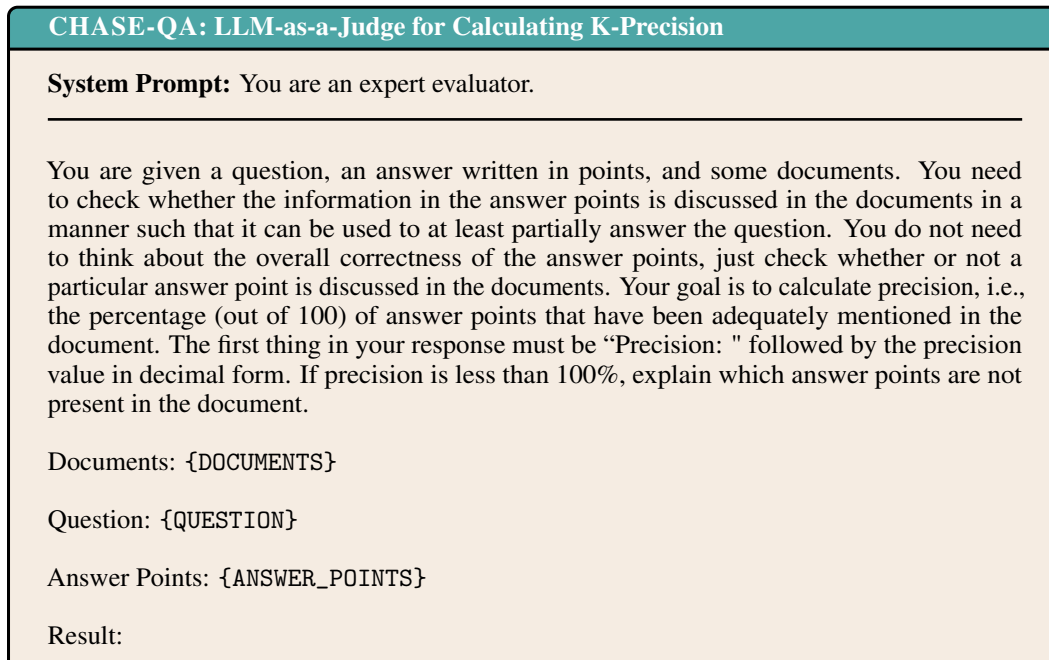


Figure 26: Programmatic prompt for evaluating K-Precision of predictions of models for problems in CHASE-QA.

CHASE-QA: LLM-as-a-Judge for Calculating Recall
<p>System Prompt: You are an expert evaluator.</p> <hr/>
<p>You are given a question, a statement, and some reference points. You need to check whether the information in the statement is discussed in the reference points in a manner such that it can be used to at least partially answer the question. It is okay if the reference points contain a lot more information, your goal is to only check whether the statement is included in the reference points. The first word in your response must be either True or False. If False, explain why in detail.</p>
<p>Question: {QUESTION}</p>
<p>Statement: {STATEMENT}</p>
<p>Reference Points: {REFERENCE_POINTS}</p>
<p>Result:</p>

Figure 27: Programmatic prompt for evaluating recall of predictions of models for problems in CHASE-QA.

CHASE-CODE: Generating Helper Functions

System Prompt: You are an expert generator of code data.

You are a research scientist. You want to make data to test an advanced code generation system. You are given a domain. Assume that there is a large python code base 'C' with at least 10 python files on that domain.

Domain: {DOMAIN}

You need to create 5 functions in this codebase for achieving various objectives. First define the parameters that will be input to the function. Then define the objective of the function. The objective must consist of 3-4 sub-goals, each of which must involve complex logic that make it very difficult to implement the function. However, each sub-goal must be well-specified such that there is only one way to implement the sub-goal. Then based on the objective, you need to create a single function (do not create other functions inside this).

Some examples are:

Parameters:

- data: pandas.DataFrame
- k: int

Objectives:

- In the dataframe "data", find the "frequency" of occurrence of rows that have at least one string field with the number of letters divisible by "k".

[redacted]

Function "filter_k_frequency" in file "string_filters.py":

```
1 import pandas as pd
2
3 def filter_k_frequency(data, k):
4     [redacted]
5     return frequency, filtered_df
```

Now you need to create 5 unique, diverse, and complex functions. Answer in the following format:

Function <Number>:

Parameters:

- <para_name>: <data_type>

...

Objectives:

- <sub_goal>

...

Function "function_name" in file "file_name.py":

<import statements>

<function definition>

Figure 28: Prompt for generating helper functions for CHASE-CODE.

CHASE-CODE: Generating Problem Statement and Answer Code

System Prompt: You are an expert generator of code data.

You are a research scientist. You want to make data to test an advanced code generation system.

Below, you are given 10 functions from a codebase "C" in the domain of {DOMAIN}.

Parameters:

- data: pandas.DataFrame

- k: int

Objectives:

- In the dataframe "data", find the "frequency" of occurrence of rows that have at least one string field with the number of letters divisible by "k".

[redacted]

Function "filter_k_frequency" in file "string_filters.py":

```
1 import pandas as pd
2
3 def filter_k_frequency(data, k):
4     [redacted]
5     return frequency, filtered_df
```

[redacted]

You need to create a complex function that calls at least 4 (but not more than 6) of these functions to achieve various objectives. Apart from just calling these functions, it should also implement some other pieces of complex logic. You first need to define the parameters that will be input to the function. Then you need to define the objective of the function. Follow these instructions for creating the objective:

1. The objective must consist of 6-8 sub-goals. Each sub-goal must be detailed and well-specified such that there is only one way to implement the sub-goal.
2. VERY IMPORTANT: The objective must not explicitly specify which functions should be called.
3. Always give names for variables you are talking about in the objective.
4. You must explicitly mention what parameters are to be used for a specific sub-goal by the name of the parameter.
5. Whenever a variable is obtained that must be returned by the function, you must explicitly state that in the sub-goal.
6. At least 2 of the sub-goals must involve some complex logic, apart from just calling helper functions that make it very difficult to implement the function.

Once you write down the objective, you need to create the function that achieves this objective. Import the required functions from the codebase "C" and use them in your function.

You must give output in the following format:

[redacted]

Figure 29: Programmatic prompt for generating problem statement and answer code for CHASE-CODE.

CHASE-CODE: Generating Test Code

System Prompt: You are an expert tester of code systems.

You are given a function. You need to define an input-output test case for that function to exhaustively test all scenarios.

{ANSWER_FUNCTION}

Follow these instructions:

1. You must output only a single long python code.
2. First initialize the input parameters for the function in python code. If the function reads data from files, you should create and store the necessary files with sample data in the corresponding filepath in the python code. Call the function and assign the return values to variables named as return_<variable_name>.
3. Then write new code to implement the exact logic of the function. This way, you need to simulate step-by-step how the values of the input parameters will be used to obtain the final return values. Call these values as correct_<variable_name>.
4. Finally, and most importantly use assert statements to compulsorily check if the returned outputs of the function (return_<variable_name> variables) match with the ones you computed yourself (correct_<variable_name> variables).

Give output in the following format:

```
1 # Import statements if required
2 import <>
3 ...
4
5 # Import function from file
6 from <filename> import <function_name>
7
8 # Initialize input parameters
9 <param1> = <value1>
10 ...
11
12 # Call function with input parameters
13 return_<variable1>, return_<variable2>, ... = $<function_name>(<
    param1>, <param2>, ...)
14
15 # Step-by-step run-through of function to obtain intermediate
    outputs:
16
17 # Step 1
18 # Explanation: <>
19 <Code for step-1>
20
21 [redacted]
22
23 # Final Expected Output:
24 correct_<variable1> = <value1>
25 ...
26
27 # Assert statements (compulsory) to check if the function
    returns the correct values:
28 assert return_<variable1> == correct_<variable1>
29 ...
```

Figure 30: Programmatic prompt for generating the test code for CHASE-CODE.

CHASE-CODE: Verifying if Function Executes

System Prompt: You are an expert tester of code systems.

You are given a function in a file. You need to check whether the function correctly executes.

{FUNCTION}

Follow these instructions:

1. You must output only a single long python code.
2. First initialize the input parameters for the function in python code. If the function reads data from files, you should create and store the necessary files with sample data in the corresponding filepath in the python code.
3. Finally, call the function with the input parameters.

Give output in the following format:

```
1 # Import statements if required
2 import <>
3 ...
4
5 # Import function from file
6 from <filename> import <function_name>
7
8 # Initialize input parameters
9 <param1> = <value1>
10 ...
11
12 # Call function with input parameters
13 return_<variable1>, return_<variable2>, ... = <function_name>(<
    param1>, <param2>, ...)
```

Figure 31: Programmatic prompt for generating the test code for verifying if a function executes correctly for CHASE-CODE.

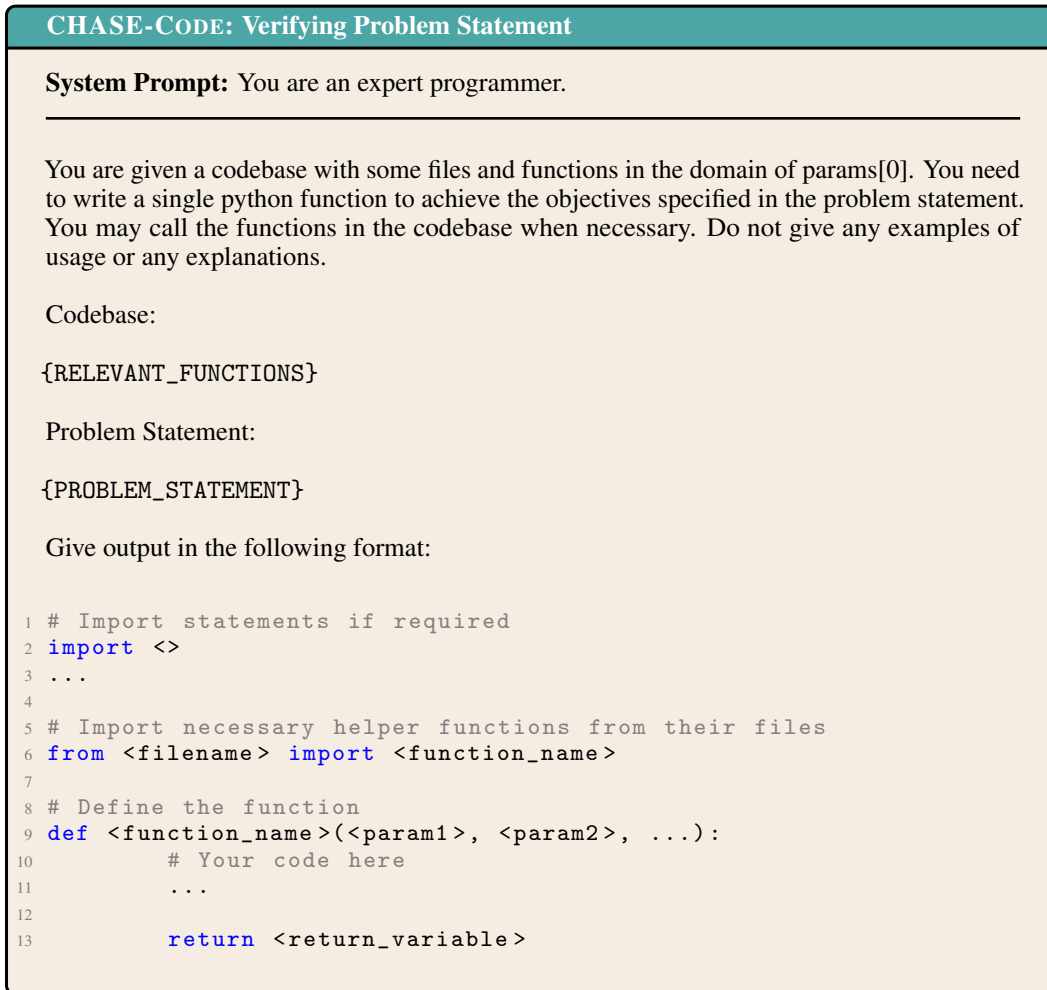


Figure 32: Programmatic prompt for verifying if the problem statement sufficiently specifies the answer code for CHASE-CODE.

CHASE-CODE: Solving the Task

System Prompt: You are an expert programmer. You must output only python code.

You are given a codebase. You need to write a single python function to achieve the objectives specified in the problem statement. In your function, you should call some of the functions in the codebase to achieve specific objectives. Do not give any examples of usage or any explanations.

Codebase:

{CODEBASE}

Problem Statement:

{PROBLEM_STATEMENT}

Give output in the following format:

```
1 # Import statements if required
2 import <>
3 ...
4
5 # Import necessary helper functions from their files
6 from <filename> import <function_name>
7
8 # Define the function
9 def <function_name>(<param1>, <param2>, ...):
10     # Your code here
11     ...
12
13     return <return_variable>
```

Figure 33: Programmatic prompt for solving examples in CHASE-CODE.

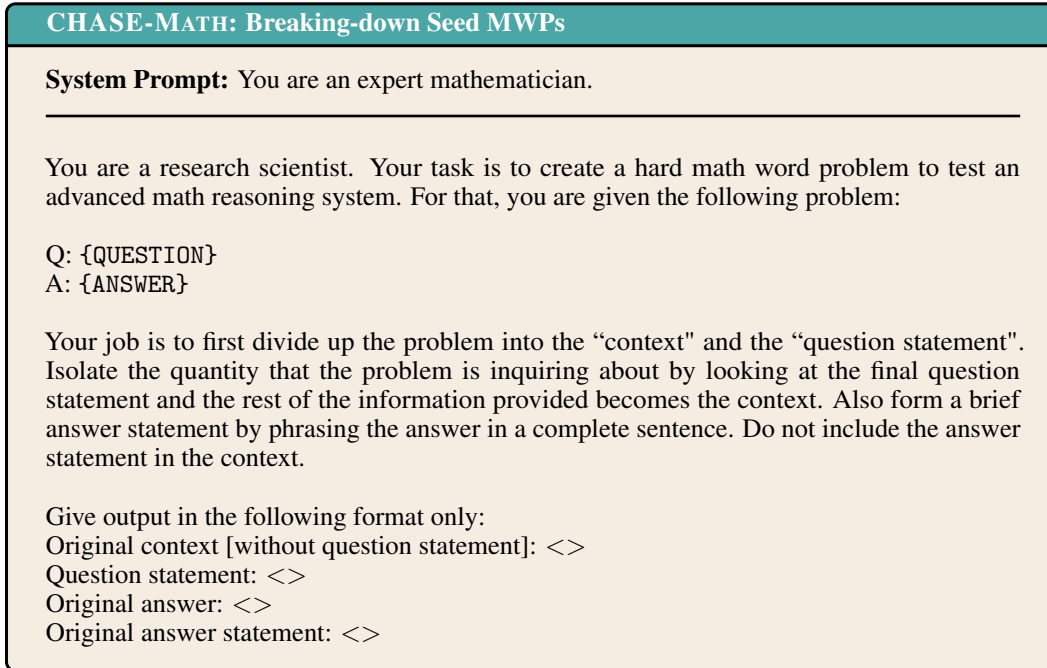


Figure 34: Programmatic prompt for breaking down the seed MWP for CHASE-MATH.

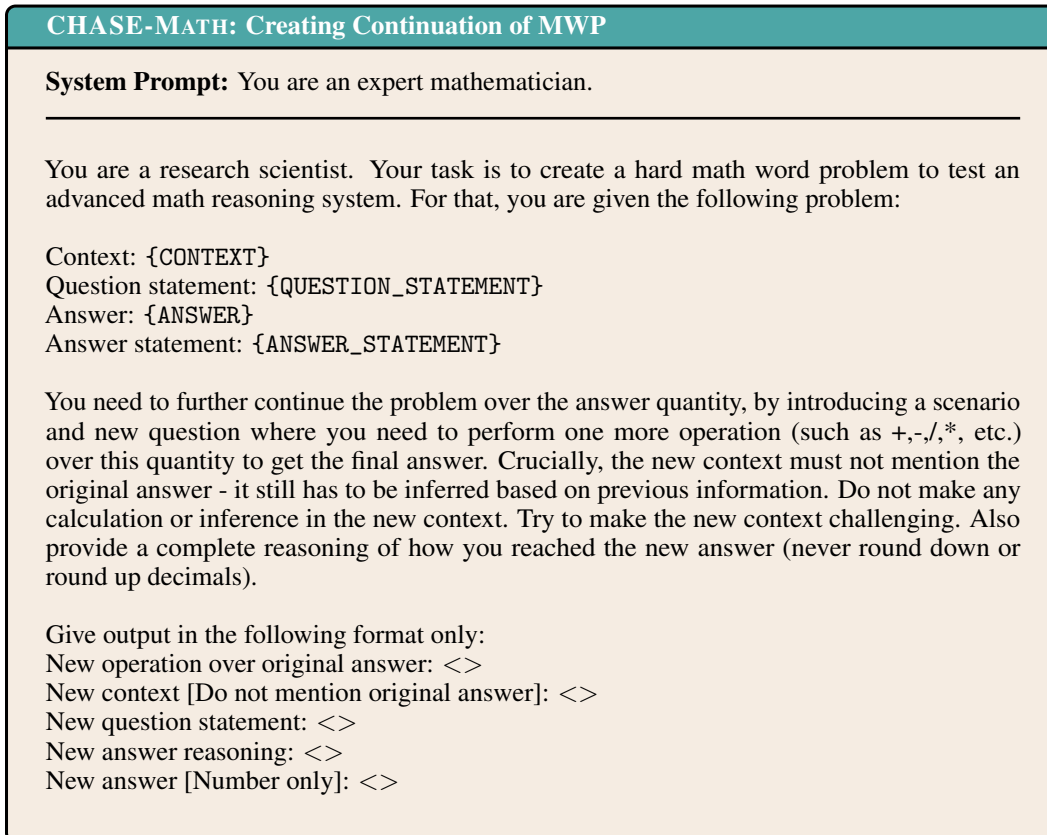


Figure 35: Programmatic prompt for extending the seed MWP for CHASE-MATH.

CHASE-MATH: Solving MWP - 8-shot chain-of-thought

System Prompt: You are an expert mathematician. Your final statement must be of the form 'The answer is <answer>'.

Solve the final math word problem given below by thinking step-by-step. You should always work with exact numbers - never round down or round up decimals based on context. Give the final answer in the end by saying "The answer is <number>".

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been $21 - 15 = 6$. The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

A: There are originally 3 cars. 2 more cars arrive. $3 + 2 = 5$. The answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

A: Originally, Leah had 32 chocolates. Her sister had 42. So in total they had $32 + 42 = 74$. After eating 35, they had $74 - 35 = 39$. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

A: Jason started with 20 lollipops. Then he had 12 after giving some to Denny. So he gave Denny $20 - 12 = 8$. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

A: Shawn started with 5 toys. If he got 2 toys each from his mom and dad, then that is 4 more toys. $5 + 4 = 9$. The answer is 9.

Q: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?

A: There were originally 9 computers. For each of 4 days, 5 more computers were added. So $5 * 4 = 20$ computers were added. $9 + 20$ is 29. The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

A: Michael started with 58 golf balls. After losing 23 on tuesday, he had $58 - 23 = 35$. After losing 2 more, he had $35 - 2 = 33$ golf balls. The answer is 33.

Q: Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?

A: Olivia had 23 dollars. 5 bagels for 3 dollars each will be $5 * 3 = 15$ dollars. So she has $23 - 15$ dollars left. $23 - 15$ is 8. The answer is 8.

Q: {QUESTION}

A:

Figure 36: Programmatic prompt for solving an example in CHASE-MATH using chain-of-thought.

CHASE-MATH: Solving MWP - 8-shot sentence-by-sentence

System Prompt: You are an expert mathematician. Your final statement must be of the form 'The answer is <answer>'.

You need to solve the given math word problem. You should break down the problem sentence by sentence, and solve each sentence, one at a time, from start to finish until you get the final answer. You should always work with exact numbers - never round down or round up decimals based on context. Give the final answer in the end by saying "The answer is <number>".

Given below are illustrations of solving sentence-by-sentence:

Q: In a store, an Uno Giant Family Card costs \$12. When Ivan bought ten pieces, he was given a discount of \$2 for each. The store has a 8% sales tax added to all purchases. Ivan decides to save 25% of this expenditure for a future vacation. After saving, Ivan instead decides to split this amount between two of his friends who are always helping him out. One of the friends decided to split their received amount equally among the five children in Ivan's neighborhood who helped him move the previous day. How much will each child receive?

A: Sentence 1: Uno card costs \$12.

Sentence 2: Ivan bought 10 cards and there was a discount of \$2 each. So, $10 * \$12 = \120 total cost of cards and, $10 * \$2 = \20 discount.

Sentence 3: Sales tax is 8% of $(\$120 - \$20 = \$100)$. So, 8% of \$100 = \$8.

Sentence 4: Ivan saves 25% of $(\$100 + \$8) = \$108$. So, 25% of \$108 = \$27.

Sentence 5: Ivan splits \$27 between 2 friends. So, $\$27 / 2 = \13.50 each.

Sentence 6: One friend splits \$13.50 among 5 children. So, $\$13.50 / 5 = \2.70 each.

The answer is 2.70.

[redacted]

Q: Carly is trying to get in shape to try out for the soccer team. She starts by running 2 miles a week. The second week, she runs twice as long plus 3 extra miles per week. The third week she runs $9/7$ as much as she ran the second week. The week after that, she sprains her ankle and has to reduce her running time by 5 miles this week compared to the previous week. After a few weeks of recovering from her ankle injury, Carly starts to feel better and decides to gradually increase her running time. She starts with a shorter routine that is one quarter of the amount she ran the week she was injured. After a week of intense training, Carly decides to boost her speed and endurance by increasing her weekly running routine to 2.5 times longer. Carly is planning increase the weekly routine by 5 times now. How much is Carly planning to run every week?

A: Sentence 1: Nothing to solve.

Sentence 2: Carly runs 2 miles a week.

Sentence 3: Carly runs $2 * 2 + 3 = 7$ miles in the second week.

Sentence 4: Carly runs $9/7 * 7 = 9$ miles in the third week.

Sentence 5: Carly reduces her running time by 5 miles this week. So, $9 - 5 = 4$ miles.

Sentence 6: Nothing to solve.

Sentence 7: Carly starts with $1/4$ of 4 miles = 1 mile.

Sentence 8: Carly increases her running routine to 2.5 times longer. So, $1 * 2.5 = 2.5$ miles.

Sentence 9: Carly wants to make her long run 5 times as long as her new routine. So, $2.5 * 5 = 12.5$ miles.

The answer is 12.5.

Q: {QUESTION}

A:

Figure 37: Programmatic prompt for solving an example in CHASE-MATH sentence-by-sentence.