

HOW TO GET YOUR LLM TO GENERATE CHALLENGING PROBLEMS FOR EVALUATION

Anonymous authors

Paper under double-blind review

ABSTRACT

The pace of evolution of Large Language Models (LLMs) necessitates new approaches for rigorous and comprehensive evaluation. Traditional human annotation is increasingly impracticable due to the complexities and costs involved in generating high-quality, challenging problems, particularly for tasks such as long-context reasoning. Moreover, the rapid saturation of existing human-curated benchmarks by LLMs further necessitates the need to develop scalable and automatically renewable evaluation methodologies. In this work, we introduce **CHASE**, a unified framework to synthetically generate challenging problems using LLMs without human involvement. For a given task, our approach builds a hard problem in a bottom-up manner from simpler components. Moreover since we want to generate synthetic data for evaluation, our framework decomposes the generation process into independently verifiable sub-tasks, thereby ensuring a high level of quality and correctness. We implement CHASE to create evaluation benchmarks across three diverse domains: document-based question answering, repository-level code completion, and math reasoning. The performance of state-of-the-art LLMs on these synthetic benchmarks lies in the range of 40-60% accuracy, thereby demonstrating the effectiveness of our framework at generating hard problems. Our experiments further reveal that the Gemini models significantly outperform other LLMs at long-context reasoning, and that the performance of all LLMs drastically drops by as much as 70% when we scale up the context size to 50k tokens.

1 INTRODUCTION

In the past few years, we have witnessed the emergence of powerful Large Language Models (LLMs) (OpenAI Team et al., 2024; Llama Team et al., 2024; Gemini Team et al., 2024) that exhibit remarkable performance over a wide range of tasks. However, the methods and resources for evaluating these models have not kept pace with their rapid evolution and increased capabilities. Contemporary LLMs have saturated many existing reasoning benchmarks (Chen et al., 2021; Cobbe et al., 2021). Developing challenging problems for reasoning tasks, such as math reasoning or code generation, can be both expensive and time-consuming, especially for human annotators with average expertise. Moreover, there are some settings, such as tasks requiring long-context reasoning over hundreds of thousands of tokens, where the generation of high quality data by humans is impracticable. Additionally, we are facing a significant depletion of interesting data that is publicly available (such as SAT exams, Olympiad problems, etc.) for creating benchmarks. Hence, we believe that the conventional approach of developing evaluation benchmarks through human annotation and existing Web content is limited, and it is worthwhile to explore using LLMs for generating evaluation data.

Synthetic data generation has emerged as a powerful paradigm in recent years driven by the widespread availability of cheaper and faster LLMs that can effectively follow instructions. The focus of most prior works, however, has been on improving models by generating synthetic data for pre-training (Gunasekar et al., 2023), instruction fine-tuning (Wang et al., 2023; Xu et al., 2024), alignment (Bai et al., 2022), or task-specific fine-tuning (Yue et al., 2024; Wei et al., 2024). In contrast, using synthetic data for evaluation and benchmarking has been relatively underexplored.

There are considerable advantages in using synthetic data for evaluation: it is comparatively inexpensive, highly scalable, and can be renewed periodically to mitigate contamination concerns. However, there are two main challenges: first, *how can we create **hard** and **realistic** problems?* and second, *how*

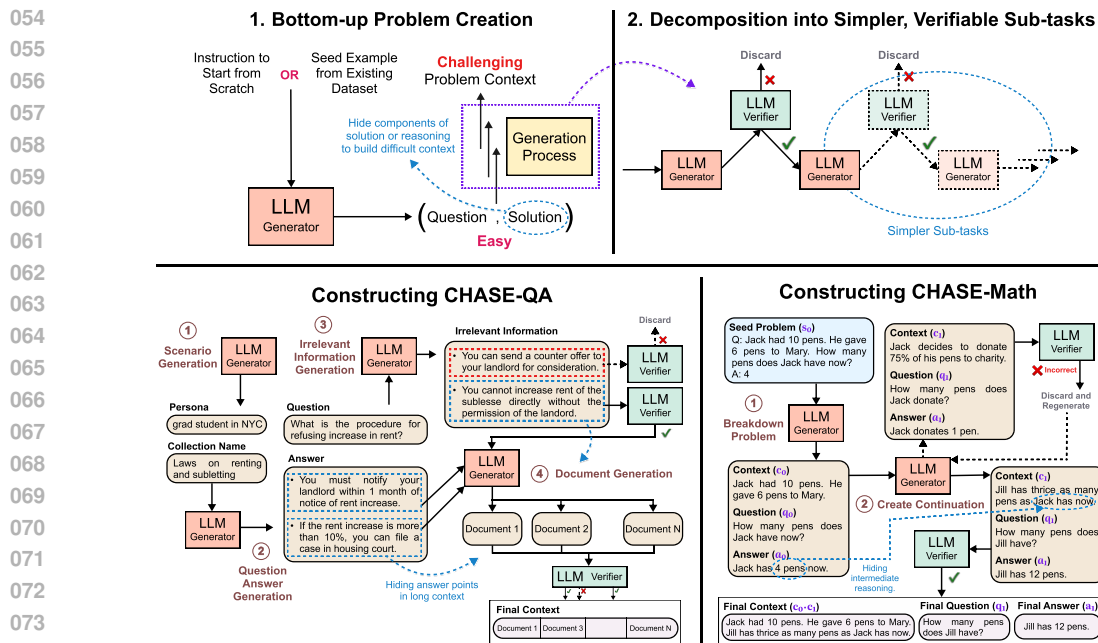


Figure 1: *Top:* Illustrating the high-level ideas behind our proposed CHASE framework. *Bottom left:* Pipeline for creating an example in CHASE-QA. *Bottom right:* Pipeline for creating a math word problem in CHASE-MATH. The pipeline for CHASE-CODE is illustrated in Figure 4 in Appendix.

can we automatically *verify the correctness* of the generated data? Typical synthetic data generation pipelines are either focused on self-improvement or on distillation. In the former, a model generates large-scale synthetic data that provides useful guidance for training. However, a large portion of such model-generated data is incorrect (Wang et al., 2023), which makes it difficult to adapt such pipelines for evaluation. Distillation-based approaches use stronger models to generate complex problems for training weaker models (Xu et al., 2024). However, we would like to design an approach that allows us to craft problems that are *challenging* to solve even for the LLM that generated them.

In this work, we present the **CHASE** framework: **CH**allenging AI with **S**ynthetic **E**valuations. Our methodology is based on two main ideas (see Figure 1) geared towards addressing the above-mentioned challenges. First, we create problems in a *bottom-up* manner where we iteratively hide parts of the solution within the problem’s context. This makes problems challenging because finding the solution will require multiple steps of drawing inferences or reasoning over a longer context. Second, we decompose the generation process into simpler, *individually verifiable sub-tasks*. This facilitates fine-grained verification for correctness at each stage of the generation process.

We implemented our framework to create challenging benchmarks across three diverse domains. (1) **CHASE-QA** is a document-based question answering benchmark. The average length of documents for each problem is 6k tokens. This benchmark simulates the real-world application of answering user queries based on information that is spread across multiple documents, most of which may be irrelevant. (2) **CHASE-CODE** is a repository-level code completion benchmark with an average context length of 17k tokens. It consists of problems from two different domains: *data pre-processing* functions, and *algorithms*. This dataset simulates the real-world application of generating code in existing repositories based on user specifications. (3) **CHASE-MATH** is a grade-school level math word problems benchmark consisting of problems involving arithmetic reasoning. We built this benchmark to show the utility of our framework in building challenging problems for seemingly easy tasks on which models have already saturated existing benchmarks (Cobbe et al., 2021).

Experiments with 15 contemporary LLMs show that the datasets generated using CHASE are challenging for all models. The best performing LLMs only achieve accuracies in the range of $\sim 40 - 60\%$ across the three different domains. We further highlight the utility of CHASE by comparing it with direct prompting-based data generation baselines such as *Evol-Instruct* (Xu et al., 2024), which yield data with significant errors, apart from being relatively easier for models to solve.

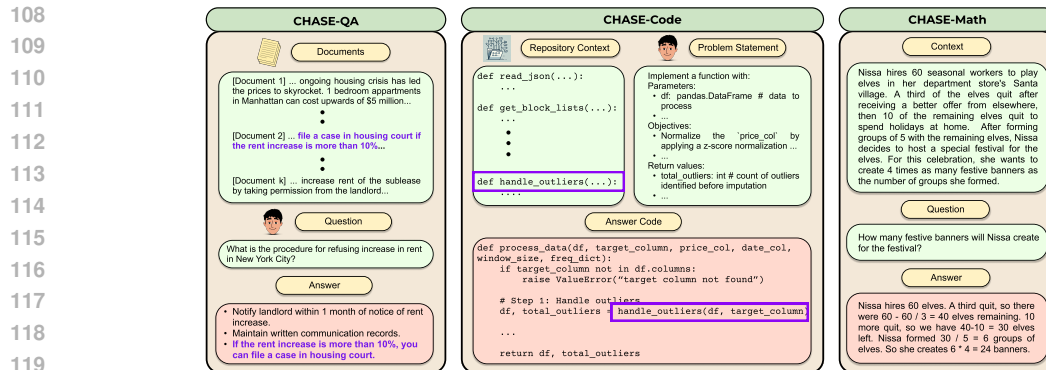


Figure 2: Examples of problems from all three benchmarks created using CHASE.

Our results also reveal large gaps in performance between different LLMs, all of which perform similarly on existing benchmarks like MMLU (Hendrycks et al., 2021) or HumanEval (Chen et al., 2021). Lastly, using our long-context benchmarks, we show that the performance of all LLMs decreases drastically (sometimes by upto 70%) when we increase the context size beyond 50k tokens.

2 RELATED WORK

Synthetic data generation. Prior works have explored generating synthetic data for various stages of an LLM’s development pipeline: pre-training (Gunasekar et al., 2023; Ben Allal et al., 2024), instruction-finetuning (Wang et al., 2023; Xu et al., 2024), alignment (Bai et al., 2022; Lee et al., 2024) and task-specific fine-tuning (Wei et al., 2024; Yu et al., 2024a). The main focus of our work, however, is to generate high-quality challenging problems for evaluation and benchmarking. There is very limited existing literature in this area. Sprague et al. (2024) created a narrative-based question answering (QA) benchmark using a neuro-symbolic pipeline that first samples facts, and then uses an LLM to build a reasoning tree over them, along with generating a narrative. Bohnet et al. (2024) created another synthetic QA benchmark by extracting entities and their reference chains from existing stories and then prompting an LLM to generate questions over them. Gu et al. (2024) create a code understanding benchmark by prompting CodeLlama (Rozière et al., 2024) to generate python functions and their inputs and designing tasks based on predicting either the input or the output for a given function. In contrast to these works, we focus on presenting a general framework which can be used to design scalable pipelines to create challenging benchmarks across multiple domains. Moreover, we focus on realistic tasks such as information-seeking QA and repository-level code generation, both with extremely long contexts that we generate completely from scratch.

Task-specific synthetic data. Recent works have explored generating synthetic datasets for content-grounded QA tasks. Dai et al. (2022) use an LLM to develop information-seeking dialogue datasets based on text extracted from Wikipedia and the Web. Yehudai et al. (2024) prompt an LLM with wikipedia text to generate a question-answer pair. In contrast, we take a more application-oriented view and design a benchmark for document-based information-seeking questions that model realistic situations. Moreover, our pipeline generates the entire context (i.e., the documents) from LLMs, which allows a higher degree of control. There has also been significant interest in generating synthetic data for code. Yu et al. (2024b) employ an LLM-based framework to create examples for tasks based on existing raw code data. Wei et al. (2024) generate code instruction data by prompting an LLM with seed code snippets from existing repositories. Luo et al. (2024) generate complex and diverse code instruction-tuning data by applying the *Evol-Instruct* method (Xu et al., 2024) on existing code benchmarks. In this work, we focus on repository-level code completion, where we generate the repository contexts completely from scratch. Moreover, we also synthetically generate the corresponding test code to evaluate each example. Generating synthetic data to improve math reasoning has recently been a very active area of research. Previous work has explored generating new math problems by prompting LLMs with examples from existing datasets (Liu et al., 2023; Yu et al., 2024a; Lu et al., 2024). 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 to generate

novel and difficult math problems by prompting LLMs with multiple core skills that must be used in the problem. Liu et al. (2024a) employ iterative question composition where they iteratively prompt an LLM with a seed question to generate more complex variations of it. In this work, we design a completely automated pipeline to craft grade-school level math word problems that are challenging to solve even for the LLM that generated them.

3 THE CHASE FRAMEWORK AND BENCHMARKS

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

1. Bottom-up problem creation.

We abandon the forward-thinking approach of first creating a difficult problem and then obtaining the corresponding solution, as followed by most works that create synthetic data for training (Xu et al., 2024; Liu et al., 2024a). If we first synthesize a complex problem and then obtain its corresponding solution from the generating LLM itself, then that problem is inherently solvable by that LLM. However, we wish to craft problems that are challenging even for the model which generates them. Hence, we instead take a different approach where we either generate or start with a simpler problem-solution pair, and then *bottom-up* build a challenging context. We make the problem’s context challenging by systematically *hiding* components of the solution or reasoning such that they need to be either extracted from a long context or inferred based on given information.

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 (i.e., each inference call) 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. We can treat each step as a task by itself and optimize the corresponding inference parameters individually. This also allows us to better manage the complexity and diversity of the generated data depending on our requirements. 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 believe that LLMs can be relied upon for verification because our framework makes each verification task smaller and simpler compared to the main task of generating or solving the problem we are crafting.

We show the effectiveness and ease of adaptation of our framework by implementing it to create challenging problems across three diverse domains:

CHASE-QA is an open-domain document-grounded question answering task consisting of 671 problems. Each example in CHASE-QA consists of a set of documents and a question-answer pair, as illustrated in Figure 2 left. Models need to reason over a long context (more than 6k tokens) because the information relevant for answering the question is spread across multiple documents.

CHASE-CODE is a repository-level code completion benchmark consisting of 220 challenging problems. Given a repository of Python functions, the task is to implement a new function based on a set of objectives provided in natural language, as illustrated in Figure 2 centre. We create data for two broad domains: (1) *data pre-processing* operations such as dataframe manipulation, string processing, file I/O, etc., and (2) *algorithms* such as graph operations, array manipulations, etc.

CHASE-MATH is a challenging benchmark of 150 grade-school level math word problems involving only basic arithmetic operations. An example of the task is provided in Figure 2 right.

4 CONSTRUCTION PIPELINES

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

4.1 CONSTRUCTING CHASE-QA

We generate CHASE-QA completely from scratch without relying on existing contexts or any seed examples from previous datasets. Following the CHASE framework, we create each example in a

216 bottom-up manner by first generating the question-answer pair, and then generating the corresponding
 217 documents. Our pipeline for creating CHASE-QA is illustrated in Figure 1 bottom left. We describe
 218 it in detail below. The exact prompts are provided in Appendix F.1.

219
 220 **Generating diverse scenarios.** We begin by first generating a set of diverse realistic scenarios in
 221 which a *user persona* seeks to find some information from a *collection of documents*. An example
 222 would be a ‘grad student in NYC’ searching the ‘laws on renting and subletting’. We prompt **G** to
 223 generate diverse scenarios in the form of a tuple (*persona*, *collection_name*) by bootstrapping
 224 it with 5 annotated example scenarios, and later prompting it with its own generated scenarios.

225
 226 **Generating question-answer pairs.** Each scenario (*persona*, *collection_name*) defines
 227 the working environment for creating a particular example. We design programmatic prompts with a
 228 given scenario as the variable to prompt **G** to generate a realistic information-seeking question that
 229 the *persona* might want to know about from *collection_name* set of documents. For example,
 230 a ‘grad student’ might pose the question, ‘what is the procedure for refusing increase in rent?’, whose
 231 answer can be found spread across multiple documents about the government’s laws on renting.
 232 Apart from the question, **G** must also generate the corresponding answer. We prompt **G** to generate
 233 questions and answers where the answers are a composition of multiple points or ideas. Additionally,
 234 **G** must generate the outline of the documents (only title and abstract) which will contain the answer.
 235 The idea is that it must separate out the answer points and assign them to these different documents.

236
 237 **Generating irrelevant information.** To make the task more challenging, for each question-answer
 238 pair, we prompt **G** to generate other question-answer pairs where the answer is of a similar type
 239 or domain as the ground-truth answer. An example of a similar question for our running example
 240 with the grad student is ‘what is the process to increase the rent for an apartment I am subletting?’.
 241 The intuition is that the corresponding answers to such similar questions will be of a very similar
 242 flavour to the ground-truth answer, but ultimately *irrelevant* for answering the question. This will
 243 make the generated data challenging since it will confuse the model when all of this similar type of
 244 information is spread across a long context. It is, however, important to verify that none of this newly
 245 generated irrelevant information is actually relevant for the original question (otherwise it will make
 246 our ground-truth answer incomplete). We individually prompt **V** with the original question and each
 247 of the supposed irrelevant information points to check if any part of them is relevant for answering
 248 the question (see Figure 1 bottom left for an example of an irrelevant point discarded by **V** because it
 was relevant for the original question).

249
 250 **Generating documents.** For each example, we have generated a question-answer (QA) pair, along
 251 with some similar but irrelevant QA pairs. For each of these QA pairs, we separately prompt **G**
 252 to generate long documents where the documents must discuss the corresponding answer points
 253 assigned to it, along with many other irrelevant points. All documents discuss distinct information.
 254 Together, all these documents form up the context for that example. We verify two main things to
 255 ensure the correctness of the task: (1) none of the documents should contain any information relevant
 256 for answering the question, apart from the ground-truth answer points, and (2) all of the ground-truth
 257 answer points must be discussed somewhere in the documents. We do this by rigorously prompting
V with individual documents and the ground-truth answer points.

258 4.2 CONSTRUCTING CHASE-CODE

259
 260 We generate CHASE-CODE completely from scratch without relying on existing contexts or any
 261 seed examples from previous datasets. Our pipeline for creating CHASE-CODE is shown in Figure 4
 262 in the Appendix. We describe it in detail below. The exact prompts are provided in Appendix F.2.

263
 264 **Generating Python functions.** We begin by first generating a set of diverse and realistic Python
 265 functions. We prompt **G** to generate Python functions for a particular domain by bootstrapping it
 266 with 3 annotated functions in that domain, and later prompting it with its own generated functions.
 267 These generated functions will act as the *helper* functions in the repository context which may or
 268 may not be called in the answer code function. Given each generated helper function, we prompt
 269 **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 some other pieces of logic. This complex function is our *answer code*. Additionally **G** must elaborate in natural language what objectives the complex function achieves, which makes up 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. Moreover, we need to verify whether the generated 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 the 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 some sample values, and compare its output with its own implementation. We execute the generated test code to check if the corresponding 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.

4.3 CONSTRUCTING CHASE-MATH

We sample math word problems (MWP) from existing datasets as seed examples to build our benchmark. Following the CHASE framework, we bottom-up build a more complex problem by iteratively increasing the reasoning depth of the problem. An illustration of the pipeline used for creating CHASE-MATH can be seen in Figure 1 bottom right. We describe the pipeline in more detail below. The exact prompts are provided in Appendix F.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 original 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 corresponding 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 complex 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 individual 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 has very high performance on the seed dataset. We prompt each \mathbf{V}_k with the generated problem 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 generating again with s_{i-1} as the seed MWP (see Figure 1 bottom right).

5 EXPERIMENTS

5.1 IMPLEMENTATION DETAILS

Generating CHASE-QA. We use GPT-4o (OpenAI Team et al., 2024) as the generator \mathbf{G} , and GPT-4o-mini as the verifier \mathbf{V} . We first sampled 500 different scenarios. For each scenario, we generate 2 different question-answer (QA) pairs. For each of the resulting 1000 unique QA pairs, we obtain *irrelevant* information by generating 4 similar QA pairs. We then generate the corresponding documents containing the ground-truth answer as well as irrelevant 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 \mathbf{G} , and Gemini-1.5-Flash as the verifier \mathbf{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 again use GPT-4o-mini (OpenAI Team et al., 2024) as the generator \mathbf{G} , and an ensemble of Gemini-1.5-Flash and Llama-3.1-70B as the verifier \mathbf{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 8 and 4 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 19 and 28 respectively in Appendix F. For CHASE-MATH, we prompt models with the standard 8-shot chain-of-thought (Wei et al., 2022b). The prompt format is provided in Figure 31 in Appendix F.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

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. DATA and ALGO refer to the *data pre-processing* and *algorithms* sub-domains of CHASE-CODE. Numbers in **bold** indicate best performance on domain while underline indicates best-in-class performance.

MODELS	QA	CODE		MATH
		DATA	ALGO	
Gemini-1.5-Pro	63.2	35.6	40.8	65.4
GPT-4o	55.3	26.8	22.4	59.8
Claude-3.5-Sonnet	36.1	19.6	25.2	64.2
Gemini-1.5-Flash	55.1	25.6	31.6	56.6
GPT-4o-mini	50.2	19.6	18	48.4
Claude-3-Haiku	32.6	18	25.6	44.2
Llama-3.1-70B	41.3	12.4	18.8	53.4
Mistral Large 2	34.1	4.8	5.2	<u>59.6</u>
Qwen2.5-72B	38.3	14.4	8.4	58.4
Command R+	41.7	0	0	43.2
DBRX	15.7	1.2	3.2	21.6
Phi-3.5-MoE	10.6	0.4	1.2	39.4
Mistral Small	35.5	1.2	1.6	50.6
Llama-3.1-8B	<u>25.2</u>	<u>0.8</u>	3.2	32.2
Qwen2.5-7B	22.2	0	<u>4.4</u>	<u>42.8</u>

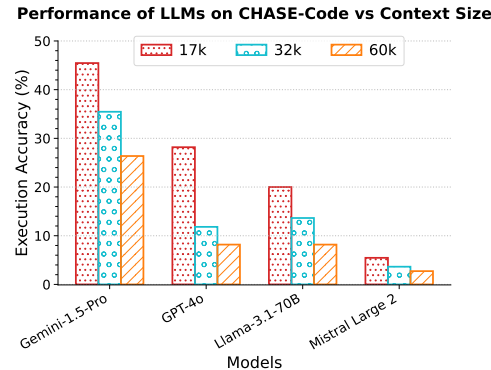
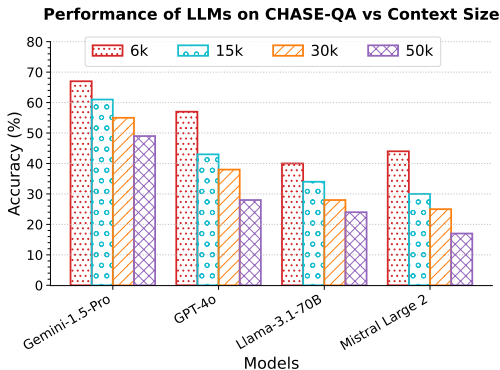


Figure 3: Performance of LLMs decreases uniformly with increasing context sizes for the 100 example subset of CHASE-QA (*top*) and the 55 example subset of CHASE-CODE (*bottom*).

accuracy as the percentage of predictions judged to be correct. The prompt format used for evaluation is provided in Figure 20 in Appendix F.1. 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 evaluated a total of 15 different 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), Llama-3.1 8B and 70B (Llama Team et al., 2024), Mistral Small and Large 2 (Mistral, 2024), Qwen2.5 7B and 72B (Yang et al., 2024a; Team, 2024a), Cohere Command R+ (Cohere, 2024), DBRX-Instruct (Team, 2024b), and Phi-3.5-MoE (Abdin et al., 2024). The details of implementation are provided in Appendix B.

5.2 RESULTS AND DISCUSSION

Performance of models. Table 1 shows the performance of all 15 LLMs on all three benchmarks. For, CHASE-QA, all models, including the generator (and judge) GPT-4o, find the task challenging. The best performing model is Gemini-1.5-Pro which achieves only about 63% accuracy, suggesting massive room for improvement. All the models struggle even more on CHASE-CODE, with the best performing model only achieving 38.2% average accuracy. For CHASE-MATH, we see that even the most powerful LLMs only achieve 65.4% accuracy on a benchmark composed of grade-school level math word problems. Overall, these results clearly indicate the utility of the CHASE framework in crafting challenging problems that even state-of-the-art LLMs struggle to solve. We provide examples of errors made by Gemini-1.5-Pro on all three benchmarks and analyze them in Appendix E.

Table 2: Performance of LLMs on data generated by direct prompting approaches without using CHASE.

MODEL	QA	MATH
Gemini-1.5-Pro	81	85.7
GPT-4o	78	88.9
Claude-3.5-Sonnet	73	82.5

Table 3: Accuracy of LLMs increases marginally on CHASE-MATH when fine-tuned on data generated by Llama-3.1-8B.

MODEL	BASE	FINE-TUNED
Llama-3.1-8B	30	34.7
Mistral-7B	3.3	4.7
Qwen2-7B	12.7	15.3

On all three benchmarks, we see huge variations in performance between the models. Moreover, we observe models different from the generator and verifier (such as Gemini-1.5-Pro) that do better for all three tasks. This stands as a testament to the underlying quality of the generated data and highlights 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) or HumanEval (Chen et al., 2021). We further note some interesting observations. On both the long-context benchmarks, there is a substantial gap between the Gemini models and the other LLMs, clearly exhibiting the strong long-context reasoning capabilities of Gemini. Another interesting observation is that most of the models are stronger on the *algorithms* domain, while GPT-4o is stronger at *data pre-processing*. This demonstrates the utility of our benchmark at identifying such targeted differences in performance which could be very helpful for real tasks. For math reasoning, we have seen even weaker models like Llama-3.1-8B and Phi-3.5 get around 85 – 90% accuracies on the popular GSM8k and SVAMP benchmarks. However, we see a large difference ($\sim 25 - 30\%$) between their performance and that of the state-of-the-art models, which may be attributed in part to some level of contamination of existing benchmarks (Zhang et al., 2024a).

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 in a manner similar to Honovich et al. (2023) and Wang et al. (2023). For math, we adapt the *Evol-Instruct* method (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 100 examples for both tasks. For the math task, we manually examined the generated problems and found that 34 of them had some kind of error such as the problem text being ambiguous or vague or the reasoning and answer being incorrect. Carrying out a detailed manual verification for the QA problems is impracticable, however, we believe it is highly likely that a significant portion of it is incorrect. We evaluated GPT-4o, Gemini-1.5-Pro, and Claude-3.5-Sonnet on these datasets and provide the results in Table 2. For both tasks, we observe that we are unable to generate challenging data by direct prompting.

Impact of context size. We studied the impact of varying the context size for long-context reasoning. For each example in a randomly-sampled 100-example subset of CHASE-QA, we increase the context size by concatenating the documents in that example with irrelevant documents randomly sampled from other examples. For CHASE-CODE, we create a subset of 55 randomly-sampled examples for each of the domains and increase the context size by concatenating irrelevant code functions in the corresponding repository context. Figure 3 plots the performances of 4 LLMs across different context sizes. For both benchmarks, we see a consistent and significant decrease in model performance as we scale up 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.

Human verification of LLM judgements. We measure the correlation of the GPT-4o evaluator’s judgement and 3 human annotators over 100 randomly sampled¹ predictions made by Gemini-1.5-Pro on CHASE-QA. 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 can be interpreted as almost-perfect agreement. Additional details regarding the setup of these experiments can be found in Appendix B.

¹but balanced, according to GPT-4o’s judgement.

486 **Fine-tuning smaller models.** We study whether we can use much smaller models (of around 7B
487 scale) to generate useful fine-tuning data for themselves following CHASE pipelines to perform
488 better on evaluation benchmarks created by stronger models. We generate $\sim 10k$ math problems
489 using Llama-3.1-8B as both the generator and the verifier and fine-tune 3 small models. Table 3
490 shows the accuracies on CHASE-MATH before and after fine-tuning on this dataset for all LLMs.
491 We see marginal performance improvements across all models. These results seem to indicate that
492 the evaluation data generated using significantly stronger models cannot be easily solved by such
493 weak models even when fine-tuned on data generated by themselves using the exact same pipeline.

494 495 6 CONCLUSION

496
497 In this work, we presented CHASE, a framework to synthetically generate challenging problems for
498 evaluation. Our framework offers multiple benefits. First, it is scalable and can efficiently generate
499 hundreds of challenging problems. Second, it is renewable, which means it can be used to keep
500 regenerating new datasets for a given task at periodic intervals thereby mitigating contamination
501 concerns which plague most evaluation benchmarks. Third, it can effectively help evaluate tasks (such
502 as long-context reasoning) which are inherently difficult or impracticable for humans to evaluate.
503 Fourth, it provides a very high level of quality and correctness because of extensive verification.
504 And finally, it can be used to generate *difficult* examples for a given task. We implemented CHASE
505 on three different domains: document-based question answering, repository-level code completion,
506 and math reasoning. With the help of extensive experiments, we showed that CHASE successfully
507 generates examples that are difficult even for the state-of-the-art LLMs to solve. Our results raise
508 several interesting questions to explore in future work: (a) How can we modify this framework to
509 more easily adapt to different tasks? (b) What is the utility of this framework in generating data for
510 training LLMs? (c) How can we verify LLM generations more efficiently? We hope our work will
511 catalyze many such studies on using synthetic data for evaluation.

512 7 LIMITATIONS

513
514 **Size of benchmarks.** The datasets we generated are comparatively smaller in size. Our framework
515 necessitates querying the generator and especially the verifier many times for crafting each example.
516 While this increases the quality and correctness of the data, it significantly increases the cost of
517 generation. Moreover, a large portion of the intermediate generations in our pipeline are discarded
518 because of extensive verification, which significantly reduces the yield. Our focus in this work is to
519 present the CHASE framework and we believe our experiments, albeit on smaller-sized datasets,
520 convincingly show its utility in generating challenging problems for evaluation. Small benchmarks, if
521 generated with a high bar for quality and correctness, can still be very impactful as evidenced by past
522 works like HumanEval (Chen et al., 2021).

523
524 **Instance-level issues.** Some of the examples we generated using CHASE, while being semantically
525 correct, use unnatural or difficult-to-parse language. This is a general trait of text generated from
526 contemporary LLMs, and our framework is unfortunately susceptible to it. Moreover, while we carry
527 out extensive verification of correctness at the instance-level using LLMs, it is still possible that some
528 of the generated examples are incorrect. Verifying long-context reasoning data using non-expert
529 humans is impracticable. To get a sense of the quality, we manually examined 30 random examples
530 each from the generation process of CHASE-QA and CHASE-CODE. We found only 2 errors in
531 CHASE-QA and none in CHASE-CODE. We also examined 100 examples from CHASE-MATH’s
532 pipeline and found 7 that were incorrect. Examples of such errors in generation are discussed in
533 Appendix E. We believe that errors at this scale are acceptable considering the other advantages of
534 the framework and can be factored into the evaluation in future studies.

535
536 **Adaptability.** While we have shown how we implemented CHASE on three different domains, it
537 is not trivial to adapt the framework to other tasks. Although the high level ideas behind CHASE
538 are easy enough to follow, it takes multiple trials and errors to design a working pipeline for any
539 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.

REFERENCES

- 540
541
542 Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen
543 Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko,
544 Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong
545 Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai,
546 Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg,
547 Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J.
548 Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin,
549 Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev
550 Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui
551 Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo,
552 Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes,
553 Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker,
554 Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa,
555 Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael
556 Santacrose, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song,
557 Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan
558 Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping
559 Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali
560 Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong
561 Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and
562 Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024.
563 URL <https://arxiv.org/abs/2404.14219>.
- 564 Vaibhav Adlakha, Shehzaad Dhuliawala, Kaheer Suleman, Harm de Vries, and Siva Reddy. Topi-
565 OCQA: Open-domain conversational question answering with topic switching. *Transactions of the*
566 *Association for Computational Linguistics*, 10:468–483, 2022. doi: 10.1162/tacl.a.00471. URL
<https://aclanthology.org/2022.tacl-1.27>.
- 567 Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong,
568 and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In
569 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting*
570 *of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14388–14411,
571 Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/
572 2024.acl-long.776. URL <https://aclanthology.org/2024.acl-long.776>.
- 573 Anthropic. The claude 3 model family: Opus, sonnet, haiku. [https://www.anthropic.com/
574 news/claude-3-family](https://www.anthropic.com/news/claude-3-family), 2024a.
- 575 Anthropic. Claude 3.5 sonnet. <https://www.anthropic.com/news/claude-3-5-sonnet>,
576 2024b.
- 577 Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones,
578 Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson,
579 Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson,
580 Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile
581 Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado,
582 Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec,
583 Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom
584 Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei,
585 Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness
586 from ai feedback, 2022. URL <https://arxiv.org/abs/2212.08073>.
- 587 Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du,
588 Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilin-
589 gual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and
590 Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Com-
591 putational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August
592 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL
593 <https://aclanthology.org/2024.acl-long.172>.

- 594 Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von
595 Werra. Cosmopedia, 2024. URL [https://huggingface.co/datasets/HuggingFaceTB/
596 cosmopedia](https://huggingface.co/datasets/HuggingFaceTB/cosmopedia).
- 597 Bernd Bohnet, Kevin Swersky, Rosanne Liu, Pranjal Awasthi, Azade Nova, Javier Snaider, Hanie
598 Sedghi, Aaron T Parisi, Michael Collins, Angeliki Lazaridou, Orhan Firat, and Noah Fiedel. Long-
599 span question-answering: Automatic question generation and qa-system ranking via side-by-side
600 evaluation, 2024. URL <https://arxiv.org/abs/2406.00179>.
- 601 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal,
602 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel
603 Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler,
604 Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Ben-
605 jamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and
606 Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell,
607 M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33,
608 pp. 1877–1901. Curran Associates, Inc., 2020. URL [https://proceedings.neurips.cc/
609 paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf).
- 610 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared
611 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,
612 Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,
613 Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,
614 Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios
615 Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino,
616 Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,
617 Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa,
618 Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob
619 McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating
620 large language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>.
- 621 Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li,
622 Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun
623 Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin
624 Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping
625 Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts,
626 Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal
627 of Machine Learning Research*, 25(70):1–53, 2024. URL [http://jmlr.org/papers/v25/
628 23-0870.html](http://jmlr.org/papers/v25/23-0870.html).
- 629 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,
630 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John
631 Schulman. Training verifiers to solve math word problems, 2021. URL [https://arxiv.org/
632 abs/2110.14168](https://arxiv.org/abs/2110.14168).
- 633 Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological
634 measurement*, 20(1):37–46, 1960.
- 635 Cohere. Cohere command r+, August 2024. URL [https://docs.cohere.com/docs/
636 command-r-plus](https://docs.cohere.com/docs/command-r-plus).
- 637 Zhuyun Dai, Arun Tejasvi Chaganty, Vincent Y Zhao, Aida Amini, Qazi Mamunur Rashid,
638 Mike Green, and Kelvin Guu. Dialog inpainting: Turning documents into dialogs. In Ka-
639 malika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato
640 (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of
641 *Proceedings of Machine Learning Research*, pp. 4558–4586. PMLR, 17–23 Jul 2022. URL
642 <https://proceedings.mlr.press/v162/dai22a.html>.
- 643 Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of
644 information-seeking questions and answers anchored in research papers. In Kristina Toutanova,
645 Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cot-
646 terrell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the*

- 648 *North American Chapter of the Association for Computational Linguistics: Human Language*
649 *Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi:
650 10.18653/v1/2021.naacl-main.365. URL [https://aclanthology.org/2021.naacl-main.](https://aclanthology.org/2021.naacl-main.365)
651 365.
- 652
653 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of
654 deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and
655 Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the*
656 *Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and*
657 *Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational
658 Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423>.
- 659 Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner.
660 DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In
661 Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of*
662 *the North American Chapter of the Association for Computational Linguistics: Human Language*
663 *Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June
664 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL <https://aclanthology.org/N19-1246>.
- 665
666 Gemini Team et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of
667 context, 2024. URL <https://arxiv.org/abs/2403.05530>.
- 668
669 Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I.
670 Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL
671 <https://arxiv.org/abs/2401.03065>.
- 672
673 Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth
674 Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital
675 Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai,
676 Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2023. URL [https://arxiv.org/](https://arxiv.org/abs/2306.11644)
677 [abs/2306.11644](https://arxiv.org/abs/2306.11644).
- 678 Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob
679 Steinhardt. Measuring massive multitask language understanding. In *International Conference on*
680 *Learning Representations*, 2021. URL <https://openreview.net/forum?id=d7KBjmI3GmQ>.
- 681
682 Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. Unnatural instructions: Tuning
683 language models with (almost) no human labor. In Anna Rogers, Jordan Boyd-Graber, and
684 Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Com-*
685 *putational Linguistics (Volume 1: Long Papers)*, pp. 14409–14428, Toronto, Canada, July
686 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.806. URL
687 <https://aclanthology.org/2023.acl-long.806>.
- 688 Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekish, Fei Jia, Yang
689 Zhang, and Boris Ginsburg. Ruler: What’s the real context size of your long-context language
690 models?, 2024. URL <https://arxiv.org/abs/2404.06654>.
- 691
692 Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang,
693 and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International*
694 *Conference on Learning Representations*, 2022. URL [https://openreview.net/forum?id=](https://openreview.net/forum?id=nZeVKeeFYf9)
695 [nZeVKeeFYf9](https://openreview.net/forum?id=nZeVKeeFYf9).
- 696 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R
697 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth*
698 *International Conference on Learning Representations*, 2024. URL [https://openreview.net/](https://openreview.net/forum?id=VTF8yNQm66)
699 [forum?id=VTF8yNQm66](https://openreview.net/forum?id=VTF8yNQm66).
- 700
701 Greg Kamradt. Needle in a haystack - pressure testing llms. [https://github.com/gkamradt/](https://github.com/gkamradt/LLMTest_NeedleInAHaystack)
[LLMTest_NeedleInAHaystack](https://github.com/gkamradt/LLMTest_NeedleInAHaystack), 2023.

- 702 Varun Kumar, Ashutosh Choudhary, and Eunah Cho. Data augmentation using pre-trained transformer
703 models. In William M. Campbell, Alex Waibel, Dilek Hakkani-Tur, Timothy J. Hazen, Kevin
704 Kilgour, Eunah Cho, Varun Kumar, and Hadrien Glaude (eds.), *Proceedings of the 2nd Workshop
705 on Life-long Learning for Spoken Language Systems*, pp. 18–26, Suzhou, China, December
706 2020. Association for Computational Linguistics. URL <https://aclanthology.org/2020.lifelongnlp-1.3>.
- 708 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph
709 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model
710 serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems
711 Principles*, pp. 611–626, 2023.
- 712 Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Ren Lu,
713 Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif vs. rlhf: Scaling rein-
714 forcement learning from human feedback with ai feedback. In *Forty-first International Conference
715 on Machine Learning*, 2024.
- 717 Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and
718 Yongbin Li. Evocodebench: An evolving code generation benchmark with domain-specific
719 evaluations. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets
720 and Benchmarks Track*, 2024a. URL <https://openreview.net/forum?id=kvjbfVHpnY>.
- 721 Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context
722 language models understand long contexts? In Lun-Wei Ku, Andre Martins, and Vivek
723 Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computa-
724 tional Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, Bangkok, Thailand, August
725 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.859. URL
726 <https://aclanthology.org/2024.acl-long.859>.
- 727 Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel
728 Ward, and Yi Zhang. Tinygsm: achieving ≥ 80 URL <https://arxiv.org/abs/2312.09241>.
- 730 Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. Augmenting math word problems
731 via iterative question composing, 2024a. URL <https://arxiv.org/abs/2401.09003>.
- 732 Ruibo Liu, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, Daiyi
733 Peng, Diyi Yang, Denny Zhou, and Andrew M. Dai. Best practices and lessons learned on synthetic
734 data, 2024b. URL <https://arxiv.org/abs/2404.07503>.
- 736 Llama Team et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- 738 Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International
739 Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>.
- 742 Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and
743 Hongsheng Li. MathGenie: Generating synthetic data with question back-translation for enhancing
744 mathematical reasoning of LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.),
745 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume
746 1: Long Papers)*, pp. 2732–2747, Bangkok, Thailand, August 2024. Association for Computational
747 Linguistics. URL <https://aclanthology.org/2024.acl-long.151>.
- 748 Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng,
749 Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical
750 reasoning for large language models via reinforced evol-instruct, 2023. URL <https://arxiv.org/abs/2308.09583>.
- 752 Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing
753 Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with
754 evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024. URL
755 <https://openreview.net/forum?id=UnUwSIgK5W>.

- 756 Mistral. Mistral large 2. <https://mistral.ai/news/mistral-large-2407>, 2024.
757
- 758 Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Cudas,
759 Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash
760 Lara, and Ahmed Awadallah. Agentinstruct: Toward generative teaching with agentic flows, 2024.
761 URL <https://arxiv.org/abs/2407.03502>.
- 762 Joao Monteiro, Pierre-Andre Noel, Etienne Marcotte, Sai Rajeswar, Valentina Zantedeschi, David
763 Vazquez, Nicolas Chapados, Christopher Pal, and Perouz Taslakian. Repliq: A question-answering
764 dataset for benchmarking llms on unseen reference content, 2024. URL <https://arxiv.org/abs/2406.11811>.
- 765
766 Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed
767 Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023. URL
768 <https://arxiv.org/abs/2306.02707>.
769
- 770 OpenAI Team et al. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.
771
- 772 Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong
773 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser
774 Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan
775 Leike, and Ryan Lowe. Training language models to follow instructions with human feedback.
776 In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances*
777 *in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates,
778 Inc., 2022. URL [https://proceedings.neurips.cc/paper_files/paper/2022/file/](https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf)
779 [b1efde53be364a73914f58805a001731-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf).
- 780 Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Lixin Su, and Xueqi Cheng. Has-qa: Hierarchical
781 answer spans model for open-domain question answering. *Proceedings of the AAAI Conference on*
782 *Artificial Intelligence*, 33(01):6875–6882, Jul. 2019. doi: 10.1609/aaai.v33i01.33016875. URL
783 <https://ojs.aaai.org/index.php/AAAI/article/view/4664>.
- 784 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor
785 Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward
786 Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner,
787 Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance
788 deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox,
789 and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Cur-
790 ran Associates, Inc., 2019. URL [https://proceedings.neurips.cc/paper/2019/file/](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf)
791 [bdbca288fee7f92f2bfa9f7012727740-Paper.pdf](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf).
- 792 Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple
793 math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek
794 Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou
795 (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association*
796 *for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June
797 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL
798 <https://aclanthology.org/2021.naacl-main.168>.
- 799 Arkil Patel, Siva Reddy, Dzmitry Bahdanau, and Pradeep Dasigi. Evaluating in-context learning of
800 libraries for code generation. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings*
801 *of the 2024 Conference of the North American Chapter of the Association for Computational*
802 *Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2908–2926, Mexico
803 City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.
804 [naacl-long.161](https://aclanthology.org/2024.naacl-long.161). URL <https://aclanthology.org/2024.naacl-long.161>.
- 805 Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language
806 models are unsupervised multitask learners. 2019.
807
- 808 Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions
809 for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings*
of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 2383–2392,

- 810 Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/
811 D16-1264. URL <https://aclanthology.org/D16-1264>.
812
- 813 Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions
814 for squad, 2018. URL <https://arxiv.org/abs/1806.03822>.
815
- 816 Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi
817 Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov,
818 Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre
819 Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas
820 Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL
821 <https://arxiv.org/abs/2308.12950>.
- 822 Timo Schick and Hinrich Schütze. Generating datasets with pretrained language models. In Marie-
823 Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of*
824 *the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6943–6951,
825 Online and Punta Cana, Dominican Republic, November 2021. Association for Computational
826 Linguistics. doi: 10.18653/v1/2021.emnlp-main.555. URL [https://aclanthology.org/](https://aclanthology.org/2021.emnlp-main.555)
827 [2021.emnlp-main.555](https://aclanthology.org/2021.emnlp-main.555).
- 828 Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua
829 Bengio, Sanjeev Arora, and Anirudh Goyal. Ai-assisted generation of difficult math questions,
830 2024. URL <https://arxiv.org/abs/2407.21009>.
831
- 832 Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong,
833 Mor Geva, Jonathan Berant, and Omer Levy. SCROLLS: Standardized Comparison over long
834 language sequences. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of*
835 *the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 12007–12021,
836 Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:
837 10.18653/v1/2022.emnlp-main.823. URL [https://aclanthology.org/2022.emnlp-main.](https://aclanthology.org/2022.emnlp-main.823)
838 [823](https://aclanthology.org/2022.emnlp-main.823).
- 839 Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. ZeroSCROLLS: A zero-
840 shot benchmark for long text understanding. In Houda Bouamor, Juan Pino, and Kalika Bali
841 (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7977–7989,
842 Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.
843 findings-emnlp.536. URL <https://aclanthology.org/2023.findings-emnlp.536>.
844
- 845 Zayne Rea Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. MuSR: Testing the
846 limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on*
847 *Learning Representations*, 2024. URL <https://openreview.net/forum?id=jenyYQzue1>.
- 848 Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. Mathsacle: Scaling instruction
849 tuning for mathematical reasoning, 2024. URL <https://arxiv.org/abs/2403.02884>.
850
- 851 Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL [https://qwenlm.](https://qwenlm.github.io/blog/qwen2.5/)
852 [github.io/blog/qwen2.5/](https://qwenlm.github.io/blog/qwen2.5/).
853
- 854 The Mosaic Research Team. Introducing dbrx: A new state-of-the-art
855 open llm, March 2024b. URL [https://www.databricks.com/blog/](https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm)
856 [introducing-dbrx-new-state-art-open-llm](https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm).
- 857 Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman.
858 Openmathinstruct-1: A 1.8 million math instruction tuning dataset, 2024. URL [https://arxiv.](https://arxiv.org/abs/2402.10176)
859 [org/abs/2402.10176](https://arxiv.org/abs/2402.10176).
860
- 861 Cunxiang Wang, Ruoxi Ning, Boqi Pan, Tonghui Wu, Qipeng Guo, Cheng Deng, Guangsheng Bao,
862 Xiangkun Hu, Zheng Zhang, Qian Wang, and Yue Zhang. Novelqa: Benchmarking question
863 answering on documents exceeding 200k tokens, 2024a. URL [https://arxiv.org/abs/2403.](https://arxiv.org/abs/2403.12766)
[12766](https://arxiv.org/abs/2403.12766).

- 864 Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu,
865 Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. Leave no
866 document behind: Benchmarking long-context llms with extended multi-doc qa, 2024b. URL
867 <https://arxiv.org/abs/2406.17419>.
- 868 Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and
869 Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In
870 Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual
871 Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–
872 13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/
873 2023.acl-long.754. URL <https://aclanthology.org/2023.acl-long.754>.
- 874 Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du,
875 Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International
876 Conference on Learning Representations, 2022a*. URL [https://openreview.net/forum?
877 id=gEZrGCozdqR](https://openreview.net/forum?id=gEZrGCozdqR).
- 878 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V
879 Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language mod-
880 els. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Ad-
881 vances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran As-
882 sociates, Inc., 2022b. URL [https://proceedings.neurips.cc/paper_files/paper/
883 2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf).
- 884 Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering
885 code generation with OSS-instruct. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller,
886 Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the
887 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine
888 Learning Research*, pp. 52632–52657. PMLR, 21–27 Jul 2024. URL [https://proceedings.
889 mlr.press/v235/wei24h.html](https://proceedings.mlr.press/v235/wei24h.html).
- 890 Yeming Wen, Pengcheng Yin, Kensen Shi, Henryk Michalewski, Swarat Chaudhuri, and Alex
891 Polozov. Grounding data science code generation with input-output specifications, 2024. URL
892 <https://arxiv.org/abs/2402.08073>.
- 893 Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi,
894 Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick
895 von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gug-
896 ger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art
897 natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods
898 in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020.
899 Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL
900 <https://aclanthology.org/2020.emnlp-demos.6>.
- 901 Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei
902 Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow
903 complex instructions. In *The Twelfth International Conference on Learning Representations, 2024*.
904 URL <https://openreview.net/forum?id=CfXh93NDgH>.
- 905 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,
906 Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint
907 arXiv:2407.10671*, 2024a.
- 908 Zhicheng Yang, Yinya Huang, Wei Shi, Liang Feng, Linqi Song, Yiwei Wang, Xiaodan Liang, and
909 Jing Tang. Benchmarking llms for optimization modeling and enhancing reasoning via reverse
910 socratic synthesis, 2024b. URL <https://arxiv.org/abs/2407.09887>.
- 911 Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and
912 Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answer-
913 ing. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii (eds.), *Proceedings
914 of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380,
915 Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi:
916 10.18653/v1/D18-1259. URL <https://aclanthology.org/D18-1259>.
- 917

- 918 Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng
919 Kong. ZeroGen: Efficient zero-shot learning via dataset generation. In Yoav Goldberg, Zornitsa
920 Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in
921 Natural Language Processing*, pp. 11653–11669, Abu Dhabi, United Arab Emirates, December
922 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.801. URL
923 <https://aclanthology.org/2022.emnlp-main.801>.
- 924 Asaf Yehudai, Boaz Carmeli, Yosi Mass, Ofir Arviv, Nathaniel Mills, Eyal Shnarch, and Leshem
925 Choshen. Achieving human parity in content-grounded datasets generation. In *The Twelfth
926 International Conference on Learning Representations*, 2024. URL [https://openreview.
927 net/forum?id=RjYKTQ0LOW](https://openreview.net/forum?id=RjYKTQ0LOW).
- 928 Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woomyoung Park. GPT3Mix:
929 Leveraging large-scale language models for text augmentation. In Marie-Francine Moens, Xuanjing
930 Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational
931 Linguistics: EMNLP 2021*, pp. 2225–2239, Punta Cana, Dominican Republic, November 2021.
932 Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.192. URL
933 <https://aclanthology.org/2021.findings-emnlp.192>.
- 934 Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo
935 Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for
936 large language models. In *The Twelfth International Conference on Learning Representations*,
937 2024a. URL <https://openreview.net/forum?id=N8N0hgNDrt>.
- 938 Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and
939 Qiufeng Yin. WaveCoder: Widespread and versatile enhancement for code large language models
940 by instruction tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings
941 of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long
942 Papers)*, pp. 5140–5153, Bangkok, Thailand, August 2024b. Association for Computational
943 Linguistics. URL <https://aclanthology.org/2024.acl-long.280>.
- 944 Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhua Chen.
945 MAMmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth
946 International Conference on Learning Representations*, 2024. URL [https://openreview.net/
947 forum?id=yLClGs770I](https://openreview.net/forum?id=yLClGs770I).
- 948 Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav
949 Raja, Dylan Slack, Qin Lyu, et al. A careful examination of large language model performance on
950 grade school arithmetic. *arXiv preprint arXiv:2405.00332*, 2024a.
- 951 Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han,
952 Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞ bench: Extending long context
953 evaluation beyond 100k tokens, 2024b. URL <https://arxiv.org/abs/2402.13718>.
- 954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

A ROADMAP

The appendix is organized as follows.

- In Section B, we provide the implementation details for our experiments.
- In Section C, we discuss some additional experimental results.
- In Section D, we discuss some additional related work.
- In Section E, we analyze errors made by LLMs while generating and solving CHASE benchmarks.
- In Section F, we provide the exact prompts used in this work.

B IMPLEMENTATION DETAILS

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.² Experiments using Gemini-1.5-Pro and Gemini-1.5-Flash were carried out using the Google AI Studio.³ Experiments with Claude-3.5-Sonnet and Claude-3-Haiku were carried out using Anthropic’s API.⁴ We provide the exact identifier and version for each LLM we experimented with in Table 4.

Table 4: Model identifiers for the 17 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-7B	mistralai/Mistral-7B-Instruct-v0.1
Mistral Small	mistralai/Mistral-Small-Instruct-2409
Mistral Large 2	mistralai/Mistral-Large-Instruct-2407
Qwen2-7B	Qwen/Qwen2-7B-Instruct
Qwen2.5-7B	Qwen/Qwen2.5-7B-Instruct
Qwen2.5-72B	Qwen/Qwen2.5-72B-Instruct
Command R+	CohereForAI/c4ai-command-r-plus-08-2024
DBRX	databricks/dbrx-instruct
Phi-3.5-MoE	microsoft/Phi-3.5-MoE-instruct
GPT-4o-mini	gpt-4o-mini-2024-07-18
GPT-4o	gpt-4o-2024-05-13
Gemini-1.5-Flash	gemini-1.5-flash-001
Gemini-1.5-Pro	gemini-1.5-pro-001
Claude-3-Haiku	claude-3-haiku-20240307
Claude-3.5-Sonnet	claude-3-5-sonnet-20240620

²<https://platform.openai.com>

³<https://aistudio.google.com>

⁴<https://console.anthropic.com>

1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

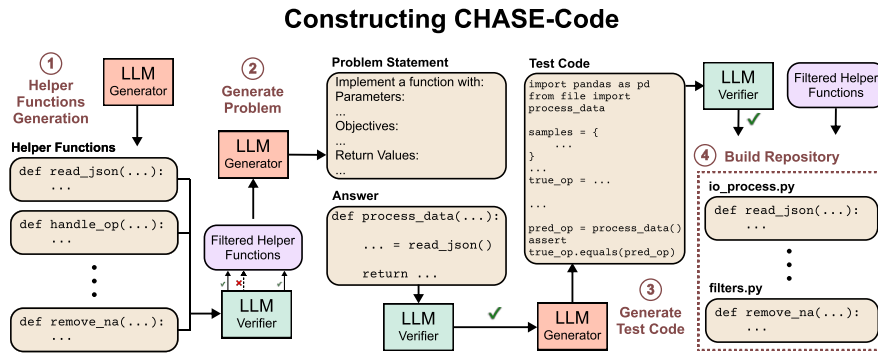


Figure 4: Pipeline for creating an example in CHASE-CODE.

Table 5: 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

Fine-tuning. We use the torchtune library and fine-tune using LoRA (Hu et al., 2022). We did not extensively tune the hyperparameters as that is not the focus of this work. We used the following hyperparameters:

- Learning rate: $5e-4$
- LoRA rank: 16
- LoRA alpha: 32
- Batch size: 2
- Scheduler: Cosine
- Precision: Brain float16 (i.e., bf16)
- Optimizer: AdamW (Loshchilov & Hutter, 2019)

Human verification of LLM judge. 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 20. We paid \$0.5 USD to the workers for every example.

Cost of creation. In Table 5, 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.

Table 6: Comparison of model performances (pass@1) on CHASE-CODE and HumanEval, a widely-used benchmark for code generation.

MODEL	CHASE-CODE	HUMANĒVAL
Gemini-1.5-Pro	38.2	84.1
GPT-4o	24.6	90.2
Claude-3.5-Sonnet	22.4	92.0
Gemini-1.5-Flash	28.6	74.3
GPT-4o-mini	18.8	86.6
Claude-3-Haiku	21.8	75.9
Llama-3.1-70B	15.6	80.5
Mistral Large 2	5.0	92.1
Qwen2.5-72B	11.4	86.6
Command R+	0	70.1
DBRX	2.2	70.1
Phi-3.5-MoE	0.8	70.7
Mistral Small	1.4	73.8
Llama-3.1-8B	2.0	72.6
Qwen2.5-7B	2.2	57.9

C ADDITIONAL RESULTS AND DISCUSSION

C.1 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., 2024b), 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.

CHASE-CODE consists of repository-level code generation problems. HumanEval (Chen et al., 2021) is the most widely-used challenging code generation benchmark. We compare the performances of all models on both datasets in Table 6. We can clearly see that CHASE-CODE is a much more 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 grade-school level math word problems. The most widely-used challenging benchmark for this task is GSM8k (Cobbe et al., 2021), comprising of 1319 examples. We compare the performances of all models on both datasets in Table 7. It is clear that GSM8k has mostly become 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.

Table 7: Comparison of model performances on CHASE-MATH and GSM8k, a widely-used benchmark for grade-school level math word problem solving.

MODEL	CHASE-MATH	GSM8k
Gemini-1.5-Pro	65.4	90.8
GPT-4o	59.8	96.1
Claude-3.5-Sonnet	64.2	96.4
Gemini-1.5-Flash	56.6	86.2
GPT-4o-mini	48.4	94.2
Claude-3-Haiku	44.2	79.2
Llama-3.1-70B	53.4	95.1
Mistral Large 2	59.6	92.7
Qwen2.5-72B	58.4	95.8
Command R+	43.2	70.7
DBRX	21.6	72.7
Phi-3.5-MoE	39.4	88.7
Mistral Small	50.6	87.4
Llama-3.1-8B	32.2	84.5
Qwen2.5-7B	42.8	85.4

C.2 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**, 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 21 and Figure 22 respectively. The results are provided in Table 8.

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).

C.3 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 32. The results for 3 different models are shown in Table 9. While there is a clear increase in performance for all models, the task still remains difficult to solve, in general.

Table 8: 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

Table 9: Effect of prompt (see Figure 32) 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

Examples of errors made by models even with this better prompting technique are provided in Figure 8 and Figure 9.

D ADDITIONAL RELATED WORK

D.1 SYNTHETIC DATA GENERATION

Early works explored the potential of pretrained generative language models such as BERT (Devlin et al., 2019), GPT2 (Radford et al., 2019), and GPT-3 (Brown et al., 2020) to generate datasets for fine-tuning (Schick & Schütze, 2021; Ye et al., 2022) or for data augmentation (Kumar et al., 2020; Yoo et al., 2021). With increasingly powerful LLMs (Ouyang et al., 2022; OpenAI Team et al., 2024) being released in recent years combined with the benefits obtained from instruction fine-tuning (Wei et al., 2022a; Chung et al., 2024), the focus has shifted more towards generating synthetic instructions data. Honovich et al. (2023) and Wang et al. (2023) prompt GPT-3 with seed examples to automatically generate a large set of diverse tasks. Xu et al. (2024) introduced the *Evol-Instruct* pipeline to generate more complex examples from a given seed example. Mukherjee et al. (2023) leverage explanation traces from GPT-4 to create a large synthetic instruction-tuning dataset for

1242 fine-tuning smaller LLMs. Mitra et al. (2024) design an agentic framework that uses raw text and
1243 code data as seeds to generate large scale synthetic data of prompts and responses.

1244 Apart from synthetic instructions datasets, there has also been interest in using synthetic data for
1245 pre-training. Gunasekar et al. (2023) and Ben Allal et al. (2024) generate high-quality textbooks for
1246 pre-training small LLMs to unlock better reasoning performance. Recent works (Bai et al., 2022;
1247 Lee et al., 2024) have also explored using synthetic preference data for aligning language models.
1248 Synthetic data has started to become a major component in the post-training development phase
1249 of contemporary LLMs (Llama Team et al., 2024) to improve their reasoning capabilities. For a
1250 comprehensive discussion of major ideas and issues in synthetic data generation, we refer the reader
1251 to Liu et al. (2024b)’s survey.

1253 D.2 SYNTHETIC DATA FOR CODE GENERATION, AND MATH REASONING

1255 There has been significant recent interest in generating synthetic data for code. Wen et al. (2024)
1256 utilize I/O specification apart from synthetic intents to generate data science code instructions. Patel
1257 et al. (2024) generate code library specifications using GPT-4 to evaluate whether LLMs can learn
1258 new code libraries in-context.

1259 Generating synthetic data to improve math reasoning has also been a very active area of research.
1260 Tang et al. (2024) explored extracting topics and knowledge from math problems to prompt an LLM
1261 to generate new data. Toshniwal et al. (2024) create an instruction-tuning dataset by synthesizing
1262 code-interpreter style solutions for existing math problems using open-source LLMs. Yang et al.
1263 (2024b) create a reverse socratic approach to synthesize math problems from optimization scenarios.
1264 Luo et al. (2023) generate diverse math instructions data using *Evol-Instruct* and then train LLMs
1265 using reinforcement learning. Yue et al. (2024) build an instruction-tuning dataset by using GPT-4 to
1266 generate hybrid CoT and PoT rationales for examples from diverse math datasets. Yu et al. (2024a)
1267 bootstrap problems from existing math datasets by using an LLM to rephrase the question text.

1269 D.3 HUMAN-CURATED EVALUATION BENCHMARKS

1270 **Question Answering.** There have been numerous context-grounded question answering bench-
1271 marks proposed in the past. These include SQuAD (Rajpurkar et al., 2016; 2018), HotpotQA (Yang
1272 et al., 2018), DROP (Dua et al., 2019), HAS-QA, Qasper (Dasigi et al., 2021), (Pang et al., 2019)
1273 TopioCQA (Adlakha et al., 2022), NovelQA Wang et al. (2024a), and RepliQA (Monteiro et al.,
1274 2024). Different from these works, we focus on extremely long contexts, with documents containing
1275 a total of more than 10,000 tokens. Another important aspect of our work is that all the documents
1276 that make up the context for an example are generated by an LLM from scratch, which reduces
1277 contamination risk. Moreover, our questions are designed to simulate everyday queries that users
1278 might ask an LLM-based agent. Correspondingly, the annotated ground-truth answers are at times
1279 very verbose, which make them impossible to objectively evaluate using automated metrics.

1280 **Long Context Reasoning.** Kamradt (2023) introduced the Needle-In-a-Haystack (NIH) task in
1281 which a model is asked to retrieve a fact placed in a long context. However, this only assessed
1282 surface-level long-context retrieval abilities without much reasoning. In the past few months, many
1283 more long-context understanding benchmarks have been created. SCROLLS Shaham et al. (2022)
1284 and ZeroScrolls (Shaham et al., 2023) are long-context benchmarks created by aggregating and
1285 repurposing existing datasets. Compared to the QA datasets in ZeroScrolls which consist of contexts
1286 based on science and literature, CHASE-QA focuses on real-world scenarios where a user is
1287 looking for information across multiple documents. Many benchmarks such as LongBench (Bai
1288 et al., 2024), RULER (Hsieh et al., 2024), L-Eval (An et al., 2024), LooGLE (Li et al., 2024b)
1289 and InfiniteBench (Zhang et al., 2024b) consist of long-context QA tasks. However, the relevant
1290 information for answering the question is comparatively much more centralized in the context,
1291 surrounded by distracting information. In contrast, the relevant information in CHASE-QA is spread
1292 across multiple documents. Moreover, the irrelevant information in the other documents is still very
1293 closely related to the topics of the question, thereby making the task very difficult. Perhaps, the
1294 benchmark most similar to our work is Loong (Wang et al., 2024b). Like CHASE-QA, Loong also
1295 has evidences for answering questions scattered across a long context of documents. Unlike our
benchmark, however, every document in Loong is known to be relevant for answering the question.

1296 In contrast CHASE-QA models the more realistic scenario of searching a bunch of documents that
1297 are on very closely related topics, only some of which contain the answer. Moreover, unlike all these
1298 prior works, we create long-context reasoning benchmarks completely from scratch in which the long
1299 text contexts are also generated using LLMs. Further note that in contrast to these prior works that
1300 use manual annotation, our approach is highly scalable, capable of creating thousands of examples
1301 across diverse domains automatically in a much more cost-efficient manner.

1302 E ERROR ANALYSIS

1303 We provide examples for two types of errors, those made while solving the benchmarks, and those
1304 made while generating the benchmarks.

1305 E.1 ERRORS MADE WHILE SOLVING CHASE BENCHMARKS.

1306 Figure 5 provides an example of an error made by Gemini-1.5-Pro on a problem from CHASE-QA.
1307 The model fails to mention two important points relevant for answering the question, which have
1308 been discussed in the documents. This provides a qualitative example of how even the most powerful
1309 models are unable to properly pay attention to all parts of a long-context and may miss some important
1310 information.

1311 Figure 6 provides an example of an error made by Gemini-1.5-Pro in generating the correct code for
1312 a problem in CHASE-CODE. The model generates most of the code correctly, but for a particular
1313 objective, it gets confused in choosing to call the right helper function from the long-context code
1314 repository. This example qualitatively illustrates that doing well on this task requires not only a good
1315 understanding of the user-specified objectives, but also requires an in-depth understanding of all parts
1316 of the code repository.

1317 Figure 7 provides an example of an error made by Gemini-1.5-Pro in solving a math word problem
1318 from CHASE-MATH. The model executes most of the reasoning steps correctly but fails at the last
1319 one. This example qualitatively shows how even state-of-the-art LLMs start to struggle when we
1320 scale up the reasoning depth of such types of problems.

1321 E.2 ERRORS MADE IN THE GENERATION PROCESS WHEN USING CHASE.

1322 In Figure 10, we show an error made in the generation process of CHASE-QA by GPT-4o. In the
1323 document generation stage, the model generated a document which contained extra information
1324 that was directly relevant for answering the given question but was not included in the ground-truth
1325 answer. This is also a failure case of our verification engine (the one that uses the prompt in Figure
1326 17) which failed to detect the presence of this extra relevant information in the generated document.
1327 We believe such errors can be further reduced by using an ensemble of verifiers to carry out each
1328 verification task.

1329 Figure 11 provides an example of an error made by GPT-4o-mini while generating the CHASE-
1330 MATH benchmark. The model’s generated answer did not correspond to its generated problem. This
1331 is also a failure for both the generator and verifier. Such failures may be reduced by using an ensemble
1332 of stronger LLMs for verification instead.

1333 F PROMPTS

1334 F.1 PROMPTS FOR CHASE-QA

1335 In this section, we outline the exact prompts for all experiments related to CHASE-QA:

- 1336 • Generating scenarios: Figure 12
- 1337 • Generating QA pairs: Figure 13
- 1338 • Generating irrelevant information: Figure 14
- 1339 • Generating documents: Figure 15
- 1340 • Verifying irrelevant information: Figure 16

- 1350 • Verifying absence of relevant information: Figure 17
- 1351 • Verifying presence of answer: Figure 18
- 1352 • Solving the task: Figure 19
- 1353 • LLM-as-a-judge: Figure 20
- 1354

1355 F.2 PROMPTS FOR CHASE-CODE

1357 In this section, we outline the exact prompts for all experiments related to CHASE-CODE:

- 1359 • Generating helper functions: Figure 23
- 1360 • Generating problem statement and answer code: Figure 24
- 1361 • Generating test code: Figure 25
- 1362 • Generating test for verifying if function executes: Figure 26
- 1363 • Verify correctness of problem statement: Figure 27
- 1364 • Solving the task: Figure 28
- 1365

1366 F.3 PROMPTS FOR CHASE-MATH

1368 In this section, we outline the exact prompts for all experiments related to CHASE-MATH:

- 1370 • Breaking down seed MWP: Figure 29
- 1371 • Creating continuation of MWP: Figure 30
- 1372 • Solving MWP: Figure 31
- 1373 • Solving MWP sentence-by-sentence: Figure 32
- 1374

1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403

1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

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.

Irrelevant 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 5: 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.

1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1510
 1511

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(reconstructed_matrix
13      , 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, can_schedule,
17      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), k)
10    extracted_entities = extract_entities(text, entities)
11    can_schedule = can_schedule_meetings(intervals, 0)
12    interesting_cells = find_interesting_cells(reconstructed_matrix
13      , threshold)
14    interesting_cells_with_distances = [(row, col), abs(row) + abs
15      (col)] for row, col in interesting_cells]
16    return reconstructed_matrix, extracted_entities, can_schedule,
17      interesting_cells_with_distances

```

Figure 6: 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.

1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565

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 $12 \text{ hours} / 3 \text{ hours/segment} = 4$ 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 $12 \text{ hours} / 4 \text{ hours/session} = 3$ 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 7: 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.

1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

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 meters/day * 3 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 8: 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.

1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

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. #####120.0

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 $120 + 40 * 3 = 120 + 120 = 240$ crabs altogether to share with the additional friends.**

The answer is 240.

Figure 9: 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.

1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727

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 10: 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.

1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781

Error Analysis: Generating CHASE-QA - 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 11: 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.

1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835

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 12: Prompt for generating diverse scenarios for CHASE-QA.

1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889

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

1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943

CHASE-QA: Generating Irrelevant 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 14: Programmatic prompt for generating irrelevant information question-answer pairs for CHASE-QA.

1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997

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 irrelevant question-answer pairs. You should create documents that discuss topics that are completely different from the following information.

{IRRELEVANT_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 15: Programmatic prompt for generating documents for CHASE-QA.

1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051

CHASE-QA: Verifying Irrelevant Information

System Prompt: You are an expert at verifying data.

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”.

Question: {QUESTION}

Answer:
{IRRELEVANT_ANSWERS}

Give output in the following format:
Relevance: True/False

Figure 16: Programmatic prompt for verifying irrelevance of irrelevant information for CHASE-QA.

2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105

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

2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159

CHASE-QA: Verifying Presence of Answer

System Prompt: You are an expert at verifying data.

You are given a document followed by a question and an answer point. You must check two things:

1. Presence: Is the point mentioned in the document?
2. Relevance: Is the point discussed in a manner such that it can be used to partially answer the question?

Document:
{DOCUMENT}

Question: {QUESTION}

Answer Point:
{ANSWER_POINT}

Give output in the following format:
Presence: True/False
Explanation for Presence:

Relevance: True/False
Explanation for Relevance:

Figure 18: Programmatic prompt for verifying presence of ground-truth answer in the documents for CHASE-QA.

CHASE-QA: Solving the Task

System Prompt: You are an expert at answering questions based on documents.

You are given some documents followed by a question. You need to generate the answer for that question. Provide the answer in bullet points, so that it is easier to objectively evaluate. Answering the question correctly requires information from multiple documents. You must only generate the points necessary for answering the question, without mentioning anything irrelevant to the question. If you find no relevant information in the documents for answering the question, you must only generate “No relevant information found in the documents.” and nothing else.

Documents: {DOCUMENTS}

Question: {QUESTION}

Answer:

Figure 19: Programmatic prompt for solving examples in CHASE-QA.

2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213

CHASE-QA: LLM-as-a-Judge for Calculating Accuracy

System Prompt: You are an expert evaluator.

You are given a question, irrelevant answers, the ground-truth answer, and a prediction. You need to evaluate whether the prediction is correct by matching against the ground truth answer. Do not look for exact phrases or words since the prediction can have points that are a paraphrase of the same information. Based on the question, check for the presence of the same ideas or main points in the prediction as in the ground-truth answer. All the main points in the ground-truth answer must be mentioned in the prediction. The order of points mentioned is irrelevant. It is allowed for the prediction to elaborate or provide more specifics or details over the major points in the ground-truth answer. However, the prediction should not contain additional major points that are contradictory or irrelevant for answering the question. Importantly, the prediction must not discuss any of the points mentioned in the “irrelevant answers”. The first word in your response must be either True or False. If False, explain why you think the prediction is wrong in detail.

Question: {QUESTION}

Irrelevant Answers: {IRRELEVANT_ANSWERS}

Ground-truth Answer: {GROUND_TRUTH_ANSWER}

Prediction: {PREDICTION}

Result:

Figure 20: Programmatic prompt for evaluating accuracy of predictions of models for problems in CHASE-QA.

CHASE-QA: LLM-as-a-Judge for Calculating K-Precision

System Prompt: You are an expert evaluator.

You are given a question, an answer written in points, and some documents. You need to check whether the information in the answer points is discussed in the documents in a manner such that it can be used to at least partially answer the question. You do not need to think about the overall correctness of the answer points, just check whether or not a particular answer point is discussed in the documents. Your goal is to calculate precision, i.e., the percentage (out of 100) of answer points that have been adequately mentioned in the document. The first thing in your response must be “Precision: ” followed by the precision value in decimal form. If precision is less than 100%, explain which answer points are not present in the document.

Documents: {DOCUMENTS}

Question: {QUESTION}

Answer Points: {ANSWER_POINTS}

Result:

Figure 21: Programmatic prompt for evaluating K-Precision of predictions of models for problems in CHASE-QA.

2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267

CHASE-QA: LLM-as-a-Judge for Calculating Recall

System Prompt: You are an expert evaluator.

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.

Question: {QUESTION}

Statement: {STATEMENT}

Reference Points: {REFERENCE_POINTS}

Result:

Figure 22: Programmatic prompt for evaluating recall of predictions of models for problems in CHASE-QA.

2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321

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 23: Prompt for generating helper functions for CHASE-CODE.

2322
 2323
 2324
 2325
 2326
 2327
 2328
 2329
 2330
 2331
 2332
 2333
 2334
 2335
 2336
 2337
 2338
 2339
 2340
 2341
 2342
 2343
 2344
 2345
 2346
 2347
 2348
 2349
 2350
 2351
 2352
 2353
 2354
 2355
 2356
 2357
 2358
 2359
 2360
 2361
 2362
 2363
 2364
 2365
 2366
 2367
 2368
 2369
 2370
 2371
 2372
 2373
 2374
 2375

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

2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429

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

2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483

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

2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537

CHASE-CODE: Verifying Problem Statement

System Prompt: You are an expert programmer.

You are given a codebase with some files and functions in the domain of params[0]. You need to write a single python function to achieve the objectives specified in the problem statement. You may call the functions in the codebase when necessary. Do not give any examples of usage or any explanations.

Codebase:

{RELEVANT_FUNCTIONS}

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

2538
 2539
 2540
 2541
 2542
 2543
 2544
 2545
 2546
 2547
 2548
 2549
 2550
 2551
 2552
 2553
 2554
 2555
 2556
 2557
 2558
 2559
 2560
 2561
 2562
 2563
 2564
 2565
 2566
 2567
 2568
 2569
 2570
 2571
 2572
 2573
 2574
 2575
 2576
 2577
 2578
 2579
 2580
 2581
 2582
 2583
 2584
 2585
 2586
 2587
 2588
 2589
 2590
 2591

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 28: Programmatic prompt for solving examples in CHASE-CODE.

2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645

CHASE-MATH: Breaking-down Seed MWPs

System Prompt: You are an expert mathematician.

You are a research scientist. Your task is to create a hard math word problem to test an advanced math reasoning system. For that, you are given the following problem:

Q: {QUESTION}
A: {ANSWER}

Your job is to first divide up the problem into the “context” and the “question statement”. Isolate the quantity that the problem is inquiring about by looking at the final question statement and the rest of the information provided becomes the context. Also form a brief answer statement by phrasing the answer in a complete sentence. Do not include the answer statement in the context.

Give output in the following format only:
Original context [without question statement]: <>
Question statement: <>
Original answer: <>
Original answer statement: <>

Figure 29: Programmatic prompt for breaking down the seed MWP for CHASE-MATH.

CHASE-MATH: Creating Continuation of MWP

System Prompt: You are an expert mathematician.

You are a research scientist. Your task is to create a hard math word problem to test an advanced math reasoning system. For that, you are given the following problem:

Context: {CONTEXT}
Question statement: {QUESTION_STATEMENT}
Answer: {ANSWER}
Answer statement: {ANSWER_STATEMENT}

You need to further continue the problem over the answer quantity, by introducing a scenario and new question where you need to perform one more operation (such as +,-,/,*, etc.) over this quantity to get the final answer. Crucially, the new context must not mention the original answer - it still has to be inferred based on previous information. Do not make any calculation or inference in the new context. Try to make the new context challenging. Also provide a complete reasoning of how you reached the new answer (never round down or round up decimals).

Give output in the following format only:
New operation over original answer: <>
New context [Do not mention original answer]: <>
New question statement: <>
New answer reasoning: <>
New answer [Number only]: <>

Figure 30: Programmatic prompt for extending the seed MWP for CHASE-MATH.

2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699

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

2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753

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