

How to Get Your LLM to Generate Challenging Problems for Evaluation

Arkil Patel[†]

Siva Reddy[‡]

Dzmitry Bahdanau[§]

[†]Mila and McGill University

[‡]ServiceNow Research

[§]Canada CIFAR AI Chair

{arkil.patel, siva.reddy, bahdanau}@mila.quebec

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. 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, 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: (1) document-based question answering, (2) repository-level code completion, and (3) 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 challenging problems. We publicly release our benchmarks and code.¹

1 Introduction

In the past few years, we have witnessed the emergence of powerful Large Language Models (LLMs) ([Gemini Team et al., 2024](#), [Llama Team et al., 2024](#), [OpenAI Team et al., 2024](#)) that exhibit remarkable performance over a wide range of tasks. However, the 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 such reasoning tasks can be both expensive and time-consuming, especially for non-expert human annotators. Moreover, there are some settings, such as tasks requiring long-context reasoning over 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 wide-spread 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 some form of training ([Gunasekar et al., 2023](#), [Wang et al., 2023](#), [Xu et al., 2024](#)). In contrast, using synthetic data for evaluation has been relatively underexplored.

¹Data available on [huggingface](#), Code available at: <https://github.com/McGill-NLP/CHASE>

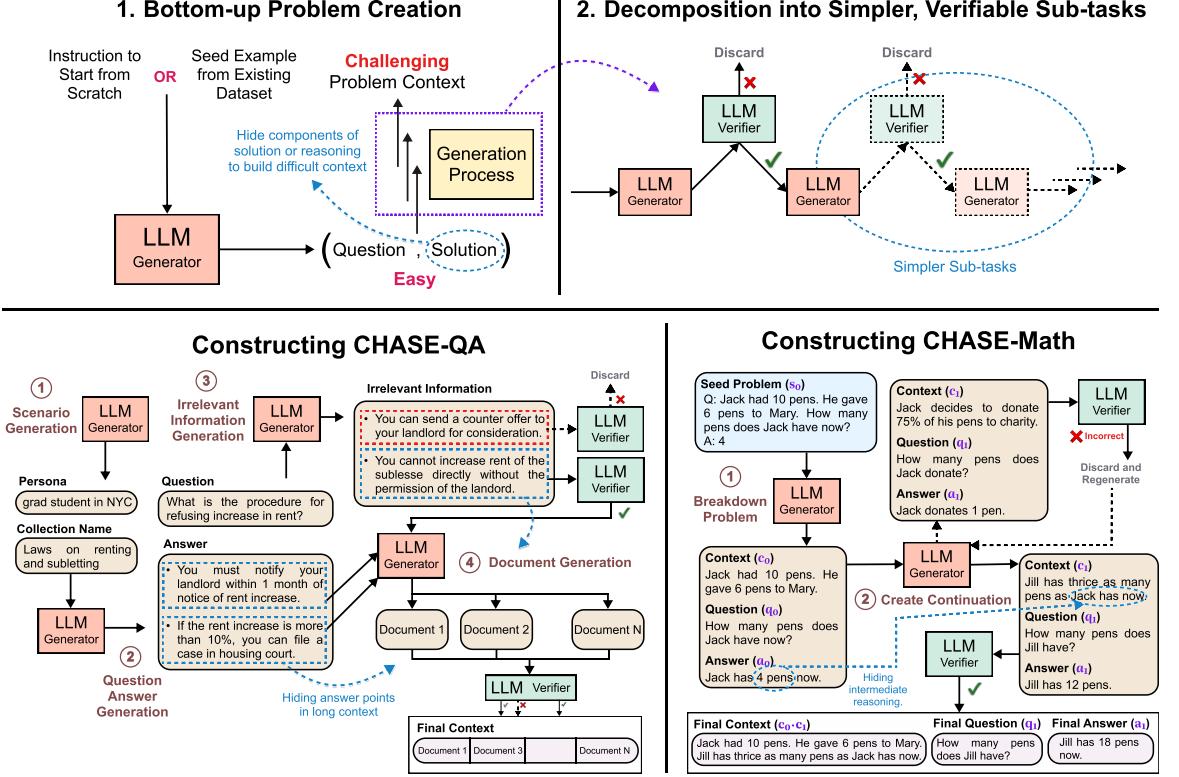


Figure 1: *Top*: Illustrating the high-level ideas behind our proposed CHASE framework. *Bottom left*: Pipeline for creating an example in CHASE-QA. *Bottom right*: Pipeline for creating a math word problem in CHASE-MATH. The pipeline for CHASE-CODE is illustrated in Figure 4 in the Appendix.

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 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 long-context document-based question answering benchmark. 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. It consists of problems from two different domains: *data pre-processing* functions, and *algorithms*.

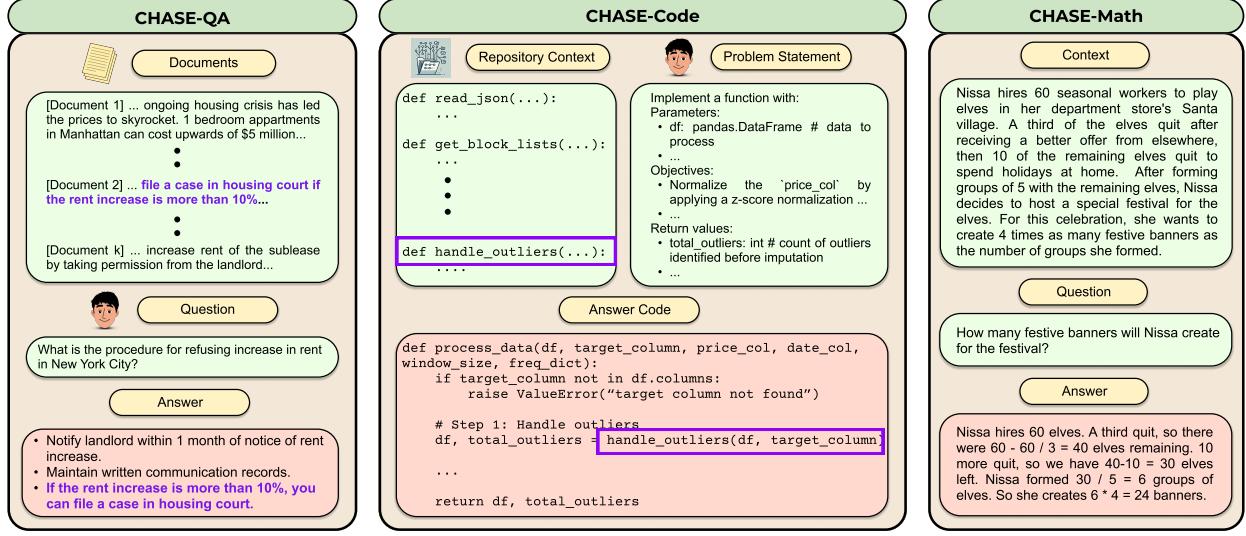


Figure 2: Examples of problems from all three benchmarks created using CHASE.

This dataset attempts to simulate 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 other 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. 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 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 (Ben Allal et al., 2024, Gunasekar et al., 2023), 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 generate a narrative. Bohnet et al. (2024) created a 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 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 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 design a benchmark for document-based information-seeking questions that model realistic situations. Moreover, our pipeline generates entire the documents using LLMs, allowing 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. 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](#), [Lu et al., 2024](#), [Yu et al., 2024a](#)). Similar to our work, some prior works have focused on creating challenging math problems. [Ye et al. \(2024\)](#) and [Zhou et al. \(2025\)](#) use computational graphs to craft problems of high complexity. While the problem contexts in these benchmarks state the entity-value relationships directly, we focus on generating more traditional word problems that require reasoning about the complexities and ambiguities arising from unconstrained use of natural language. [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 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 ([Liu et al., 2024a](#), [Xu et al., 2024](#)). 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 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 a 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 relevant information is spread across multiple documents. **CHASE-CODE** is a repository-level code completion benchmark consisting of 500 problems. Given a repository of Python functions, the task is to implement a new function based on a set of objectives, as shown in Figure 2 centre. We create data for two domains: (1) *data pre-processing* operations such as dataframe manipulation, string processing, etc., and (2) *algorithms* such as graph operations, array manipulations, etc. **CHASE-MATH** consists of 500 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. Our pipelines use two different LLMs: the generator **G** and 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 bottom-up manner by first generating the question-answer pair, and then generating the corresponding documents. Our pipeline for creating CHASE-QA is illustrated in Figure 1 bottom left. We describe it in detail below. The prompts are provided in Appendix F.1.

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

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

Generating irrelevant information. To make the task more challenging, for each QA pair, we prompt **G** to generate other QA pairs where the answer is of a similar type as the ground-truth answer. An example of a similar question for our running example with the grad student is, ‘how do I increase the rent for an apartment I am subletting?’. The intuition is that the corresponding answers to such similar questions will be of a similar flavour to the ground-truth answer, but ultimately *irrelevant* for answering the question. This will make the generated data challenging since it will confuse the model when all of this similar type of information is spread across a long context. It is, however, important to verify that none of this generated irrelevant information is actually relevant for the question (otherwise it will make our ground-truth answer incomplete). We individually prompt **V** with the original question and each of the supposed irrelevant

information points to check if any part of them is relevant for answering 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).

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

4.2 Constructing CHASE-CODE

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

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

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

Generating test code. To enable automatic execution-based testing, we prompt G with the generated answer function to implement a test code for it in Python. The test code must independently implement the logic of the answer code. It must then initialize the parameters of the answer function with sample values, and compare the output with its own implementation. We execute the generated test code to check if the answer code passes. We discard all examples for which (1) the test code does not execute properly, or (2) the test code executes but the answer code fails the test.

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

4.3 Constructing CHASE-MATH

We sample math word problems (MWP) from existing datasets as seed examples to build our benchmark. Following CHASE, we bottom-up build a complex problem by iteratively increasing the reasoning depth of the problem. Our pipeline used for creating CHASE-MATH can be seen in Figure 1 bottom right. We describe it in more detail below. The prompts are provided in Appendix 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 \mathbf{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 \mathbf{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, \mathbf{G} should output a new problem $s_1 = (p_1, a_1)$, where the context of p_1 , i.e., c_1 assumes a_0 as given information (without explicitly stating it). For example, in Figure 1 bottom right, the model assumes *Jack has 4 pens* as given information, and creates a new continuation context, *Jill has thrice as many pens as Jack has now*. The model also generates a new question q_1 , *how many pens does Jill have?* whose answer $a_1 = 12$ is obtained by performing an arithmetic operation (here, *multiplication by 3*) over $a_0 = 4$.

Combining seed MWP with its continuation. By combining the seed problem with its continuation, we get a new MWP $s = (p, a)$ with a higher reasoning depth, where the context c of the combined problem p is a concatenation of the contexts of the seed problem and the continuation $c = c_0 \cdot c_1$. The question for the combined problem will be the one generated by the model, i.e., q_1 , and the answer $a = a_1$. We refer to Figure 1 bottom right for illustration.

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

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

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 unique scenarios. For each scenario, we generate 2 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 described before. This way, we end up with 290 examples for the *algorithms* domain and 300 examples for the *data pre-processing* domain. We again use GPT-4o-mini for rejection sampling and randomly discard around half of the problems on which it was correct. This way, we end up with a total of 500 examples in the benchmark, with 250 examples for each domain. For each example, we randomly sample $m = 100$ *irrelevant* helper functions and distribute them into 10 Python files to constitute the repository context.

Generating CHASE-MATH. We use GPT-4o-mini (OpenAI Team et al., 2024) as the generator \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 2 respectively. For each problem, we iterate 15 times to generate a problem continuation. Note that many of these iterations fail to produce a correct continuation of the problem, in which case we discard that generation and retry from that point in the subsequent iteration. We carry out this process 3 times. In this manner, we generated around 1500 problems. We then carry out rejection sampling and roughly discarded 75% of the problems that GPT-4o-mini could solve. In the end, we end up with a total of 500 challenging MWPs.

Task parameters. For CHASE-QA and CHASE-CODE, we prompt models with the instruction for the task, along with the corresponding long-context and question. The prompt formats are provided in Figure 19 and 28 respectively in Appendix F. For CHASE-MATH, we prompt models with 8-shot chain-of-thought (Wei et al., 2022b) as shown 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 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 (Team, 2024a, Yang et al., 2024a), Cohere Command R+ (Cohere, 2024), DBRX-Instruct (Team, 2024b), and Phi-3.5-MoE (Abdin et al., 2024). Implementation details 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

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	<u>35.6</u>	40.8	<u>65.4</u>
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	<u>25.6</u>	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>

performing model is Gemini-1.5-Pro which achieves only about 63% accuracy, suggesting massive room for improvement. 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.

On all benchmarks, we see huge variations in performance between the models. These results highlight our framework’s potential for differentiating between state-of-the-art LLMs that all perform similarly on standard benchmarks like MMLU (Hendrycks et al., 2021) or HumanEval (Chen et al., 2021). We further note some interesting observations. On both long-context benchmarks, there is a substantial gap between the Gemini models and other LLMs, clearly exhibiting the strong long-context reasoning capabilities of Gemini. Another interesting observation is that most LLMs 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 weak LLMs 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

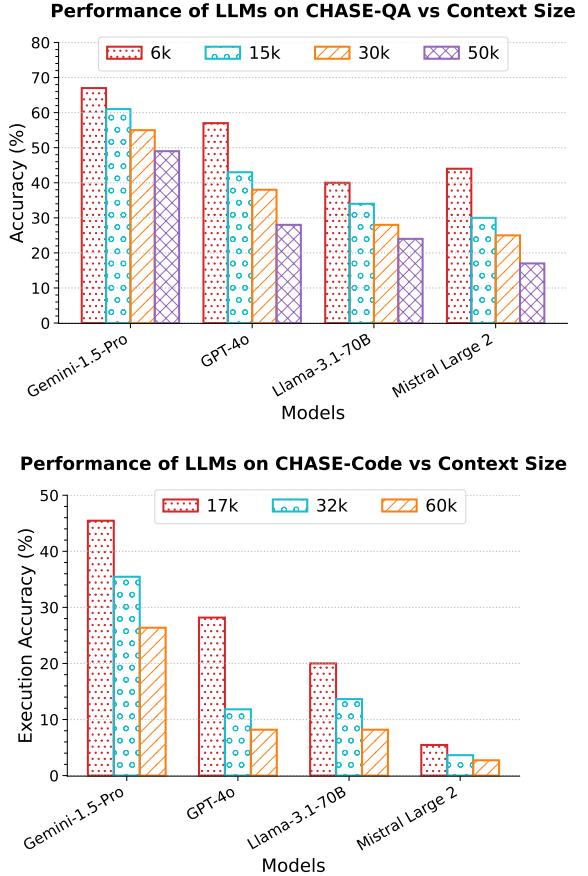


Figure 3: Performance of LLMs decreases uniformly with increasing context sizes for the 100 example subset of CHASE-QA (top) and the 55 example subset of CHASE-Code (bottom).

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

that of the state-of-the-art models, which may indicate 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 baselines.

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 sampled 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 indicates almost-perfect agreement. Additional details regarding the setup of these experiments can be found in Appendix B.

Fine-tuning smaller models. We study whether we can use smaller models (around 7B scale) to generate useful fine-tuning data for themselves following CHASE pipelines to perform better on evaluation benchmarks created by stronger models. We generate ~ 10 k math problems using Llama-3.1-8B as both the generator and the verifier and fine-tune 3 small models. Table 3 shows the accuracies on CHASE-MATH before and after fine-tuning. We see marginal performance improvements across all models. These results

indicate that the evaluation data generated using significantly stronger models cannot be easily solved by such weak models even when fine-tuned on data generated by themselves using the exact same pipeline.

6 Conclusion

In this work, we presented CHASE, a framework to synthetically generate challenging problems for evaluation. Our framework offers multiple benefits. First, it is scalable and can efficiently generate hundreds of examples. Second, it is renewable and can be used to regenerate fresh datasets for a given task at periodic intervals, thereby mitigating contamination concerns. Third, it can effectively help evaluate tasks (such as long-context reasoning) which are inherently difficult for humans to evaluate. Fourth, it provides a high level of quality (compared to existing synthetic data generation approaches) because of extensive verification. And finally, it can be used to generate *difficult* examples for a given task. We implemented CHASE on three different domains: document-based question answering, repository-level code completion, and math reasoning. Empirically, we showed that CHASE successfully generates examples that are difficult even for the state-of-the-art LLMs to solve. We believe the CHASE benchmarks also help advance the evaluation paradigms in their respective domains significantly. CHASE-QA provides the first QA benchmark that reasons over long documents of publicly unavailable text, where only some of the documents contain relevant information. Unlike contemporary repo-level benchmarks, CHASE-CODE is not bottle-necked by the availability of high-quality repositories and existing tests, and allows targeted evaluation of code generation abilities. CHASE-MATH raises the bar for reasoning about simple math operations in natural language word problems. Our results raise several interesting questions to explore in future work: (a) How can we modify this framework to easily adapt to different tasks? (b) How can we verify LLM generations more effectively? We hope our work will catalyze many such studies on using synthetic data for evaluation.

7 Limitations

Quality and Correctness. Since we do not manually verify every example in the benchmarks and only rely on LLM-based *soft* verification, it is likely that for some proportion of examples, the question is objectively ambiguous or the ground-truth annotation is incorrect. We briefly discuss manual verification in Appendix C.1. Note that this work is a preliminary exploration into using synthetic data for evaluation. We believe that errors at a small scale are acceptable considering the other advantages of the framework. We look forward to future studies developing more effective automatic verification strategies. Further note that some of the examples we generated using CHASE, while being semantically correct, use unnatural or difficult-to-parse language which may lead to ambiguity. This is a general trait of text generated from LLMs, and our framework is unfortunately susceptible to it.

Size of benchmarks. The datasets we release are comparatively smaller in size. Our framework necessitates querying the generator and especially the verifier many times for crafting each example. While this increases the quality and correctness of the data, it significantly increases the cost of generation. Moreover, a large portion of the intermediate generations in our pipeline are discarded because of extensive verification, which significantly reduces the yield. Further note that we wanted to keep our benchmarks accessible. It would have been prohibitively expensive to run experiments on long-context benchmarks with a large number of examples. Our focus in this work is to present the CHASE framework and we believe our experiments, albeit on smaller-sized datasets, convincingly show its utility in generating challenging problems for evaluation.

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

Acknowledgments

Arkil was partly supported by the Canada Graduate Scholarship – Master’s (CGS-M) funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). We thank our colleagues at Mila and McGill University for helpful discussions and for providing valuable feedback.

References

- Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragozo, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyra Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL <https://arxiv.org/abs/2404.14219>. (Cited on page 8)
- Vaibhav Adlakha, Shehzaad Dhuliawala, Kaheer Suleman, Harm de Vries, and Siva Reddy. TopiOCQA: Open-domain conversational question answering with topic switching. *Transactions of the Association for Computational Linguistics*, 10:468–483, 2022. doi: 10.1162/tacl_a_00471. URL <https://aclanthology.org/2022.tacl-1.27>. (Cited on pages 25 and 28)
- Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14388–14411, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.776. URL <https://aclanthology.org/2024.acl-long.776>. (Cited on page 28)
- Anthropic. The claude 3 model family: Opus, sonnet, haiku. <https://www.anthropic.com/news/claude-3-family>, 2024a. (Cited on page 8)
- Anthropic. Claude 3.5 sonnet. <https://www.anthropic.com/news/claude-3-5-sonnet>, 2024b. (Cited on page 8)
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann,

Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022. URL <https://arxiv.org/abs/2212.08073>. (Cited on pages 3 and 27)

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL <https://aclanthology.org/2024.acl-long.172>. (Cited on page 28)

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Cosmopedia, 2024. URL <https://huggingface.co/datasets/HuggingFaceTB/cosmopedia>. (Cited on pages 3 and 27)

Bernd Bohnet, Kevin Swersky, Rosanne Liu, Pranjal Awasthi, Azade Nova, Javier Snaider, Hanie Sedghi, Aaron T Parisi, Michael Collins, Angeliki Lazaridou, Orhan Firat, and Noah Fiedel. Long-span question-answering: Automatic question generation and qa-system ranking via side-by-side evaluation, 2024. URL <https://arxiv.org/abs/2406.00179>. (Cited on page 3)

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf. (Cited on page 26)

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=ktrw68Cmu9c>. (Cited on page 27)

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>. (Cited on pages 1, 3, 9, and 24)

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024. URL <http://jmlr.org/papers/v25/23-0870.html>. (Cited on page 26)

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training

- verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>. (Cited on pages 1, 3, 8, and 24)
- Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1): 37–46, 1960. (Cited on page 10)
- Cohere. Cohere command r+, August 2024. URL <https://docs.cohere.com/docs/command-r-plus>. (Cited on page 8)
- Zhuyun Dai, Arun Tejasvi Chaganty, Vincent Y Zhao, Aida Amini, Qazi Mamunur Rashid, Mike Green, and Kelvin Guu. Dialog inpainting: Turning documents into dialogs. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4558–4586. PMLR, 17–23 Jul 2022. URL <https://proceedings.mlr.press/v162/dai22a.html>. (Cited on page 4)
- Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL <https://aclanthology.org/2021.naacl-main.365>. (Cited on page 28)
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423>. (Cited on page 26)
- Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL <https://aclanthology.org/N19-1246>. (Cited on page 28)
- Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. Cyclomatic complexity. *IEEE software*, 33(6):27–29, 2016. (Cited on page 24)
- Gemini Team et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024. URL <https://arxiv.org/abs/2403.05530>. (Cited on pages 1 and 8)
- Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL <https://arxiv.org/abs/2401.03065>. (Cited on page 3)
- Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2023. URL <https://arxiv.org/abs/2306.11644>. (Cited on pages 1, 3, and 27)
- T Hariprasad, G Vidhyagaran, K Seenu, and Chandrasegar Thirumalai. Software complexity analysis using halstead metrics. In *2017 International Conference on Trends in Electronics and Informatics (ICELI)*, pp. 1109–1113. IEEE, 2017. (Cited on page 24)

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=d7KBjmI3GmQ>. (Cited on pages 3 and 9)

Pengfei Hong, Navonil Majumder, Deepanway Ghosal, Somak Aditya, Rada Mihalcea, and Soujanya Poria. Evaluating llms' mathematical and coding competency through ontology-guided interventions, 2024. URL <https://arxiv.org/abs/2401.09395>. (Cited on page 28)

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. Unnatural instructions: Tuning language models with (almost) no human labor. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14409–14428, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.806. URL <https://aclanthology.org/2023.acl-long.806>. (Cited on pages 10 and 26)

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL <https://arxiv.org/abs/2404.06654>. (Cited on page 28)

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeFYf9>. (Cited on page 21)

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>. (Cited on page 24)

Greg Kamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023. (Cited on page 28)

Varun Kumar, Ashutosh Choudhary, and Eunah Cho. Data augmentation using pre-trained transformer models. In William M. Campbell, Alex Waibel, Dilek Hakkani-Tur, Timothy J. Hazen, Kevin Kilgour, Eunah Cho, Varun Kumar, and Hadrien Glaude (eds.), *Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems*, pp. 18–26, Suzhou, China, December 2020. Association for Computational Linguistics. URL <https://aclanthology.org/2020.lifelongnlp-1.3>. (Cited on page 26)

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. (Cited on page 21)

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Ren Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. In *Forty-first International Conference on Machine Learning*, 2024. (Cited on pages 3 and 27)

Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and Yongbin Li. Evocodebench: An evolving code generation benchmark with domain-specific evaluations. In *The Thirty-eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. URL <https://openreview.net/forum?id=kvjbFVHpnny>. (Cited on page 24)

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context language models understand long contexts? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.859. URL <https://aclanthology.org/2024.acl-long.859>. (Cited on pages 24 and 28)

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving ≈ 80 URL <https://arxiv.org/abs/2312.09241>. (Cited on page 4)

Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. Augmenting math word problems via iterative question composing, 2024a. URL <https://arxiv.org/abs/2401.09003>. (Cited on page 4)

Ruibo Liu, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, Daiyi Peng, Diyi Yang, Denny Zhou, and Andrew M. Dai. Best practices and lessons learned on synthetic data, 2024b. URL <https://arxiv.org/abs/2404.07503>. (Cited on page 27)

Llama Team et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>. (Cited on pages 1, 8, and 27)

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>. (Cited on page 21)

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. MathGenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2732–2747, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL <https://aclanthology.org/2024.acl-long.151>. (Cited on page 4)

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct, 2023. URL <https://arxiv.org/abs/2308.09583>. (Cited on page 28)

Mistral. Mistral large 2. <https://mistral.ai/news/mistral-large-2407>, 2024. (Cited on page 8)

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweta Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. Agentinstruct: Toward generative teaching with agentic flows, 2024. URL <https://arxiv.org/abs/2407.03502>. (Cited on page 26)

Joao Monteiro, Pierre-Andre Noel, Etienne Marcotte, Sai Rajeswar, Valentina Zantedeschi, David Vazquez, Nicolas Chapados, Christopher Pal, and Perouz Taslakian. Replika: A question-answering dataset for benchmarking llms on unseen reference content, 2024. URL <https://arxiv.org/abs/2406.11811>. (Cited on page 28)

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023. URL <https://arxiv.org/abs/2306.02707>. (Cited on page 26)

OpenAI Team et al. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>. (Cited on pages 1, 7, 8, and 26)

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf. (Cited on page 26)

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Lixin Su, and Xueqi Cheng. Has-qa: Hierarchical answer spans model for open-domain question answering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6875–6882, Jul. 2019. doi: 10.1609/aaai.v33i01.33016875. URL <https://ojs.aaai.org/index.php/AAAI/article/view/4664>. (Cited on page 28)

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf>. (Cited on page 21)

Arkil Patel, Satwik Bhattacharya, and Navin Goyal. Are NLP models really able to solve simple math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL <https://aclanthology.org/2021.naacl-main.168>. (Cited on page 8)

Arkil Patel, Satwik Bhattacharya, Siva Reddy, and Dzmitry Bahdanau. MAGNIFICo: Evaluating the in-context learning ability of large language models to generalize to novel interpretations. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2167–2189, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.134. URL <https://aclanthology.org/2023.emnlp-main.134/>. (Cited on page 27)

Arkil Patel, Siva Reddy, Dzmitry Bahdanau, and Pradeep Dasigi. Evaluating in-context learning of libraries for code generation. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2908–2926, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.161. URL <https://aclanthology.org/2024.naacl-long.161>. (Cited on page 27)

Archiki Prasad, Elias Stengel-Eskin, Justin Chih-Yao Chen, Zaid Khan, and Mohit Bansal. Learning to generate unit tests for automated debugging, 2025. URL <https://arxiv.org/abs/2502.01619>. (Cited on page 27)

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. (Cited on page 26)

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions 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, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL <https://aclanthology.org/D16-1264>. (Cited on page 28)

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions for squad, 2018. URL <https://arxiv.org/abs/1806.03822>. (Cited on page 28)

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémie Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL <https://arxiv.org/abs/2308.12950>. (Cited on page 3)

Timo Schick and Hinrich Schütze. Generating datasets with pretrained language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6943–6951, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.555. URL <https://aclanthology.org/2021.emnlp-main.555>. (Cited on page 26)

Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, and Anirudh Goyal. Ai-assisted generation of difficult math questions, 2024. URL <https://arxiv.org/abs/2407.21009>. (Cited on page 4)

Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, and Omer Levy. SCROLLS: Standardized CompaRison over long language sequences. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 12007–12021, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.823. URL <https://aclanthology.org/2022.emnlp-main.823>. (Cited on page 28)

Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. ZeroSCROLLS: A zero-shot benchmark for long text understanding. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7977–7989, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.536. URL <https://aclanthology.org/2023.findings-emnlp.536>. (Cited on page 28)

Zayne Rea Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=jenyYQzue1>. (Cited on page 3)

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. Mathscale: Scaling instruction tuning for mathematical reasoning, 2024. URL <https://arxiv.org/abs/2403.02884>. (Cited on page 28)

Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL <https://qwenlm.github.io/blog/qwen2.5/>. (Cited on page 8)

The Mosaic Research Team. Introducing dbrx: A new state-of-the-art open llm, March 2024b. URL <https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm>. (Cited on page 8)

Shubham Toshniwal, Ivan Moshkov, Sean Narendhiran, Daria Gitman, Fei Jia, and Igor Gitman. Openmathinstruct-1: A 1.8 million math instruction tuning dataset, 2024. URL <https://arxiv.org/abs/2402.10176>. (Cited on page 28)

Cunxiang Wang, Ruoxi Ning, Boqi Pan, Tonghui Wu, Qipeng Guo, Cheng Deng, Guangsheng Bao, Xiangkun Hu, Zheng Zhang, Qian Wang, and Yue Zhang. Novelqa: Benchmarking question answering on documents exceeding 200k tokens, 2024a. URL <https://arxiv.org/abs/2403.12766>. (Cited on page 28)

Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa, 2024b. URL <https://arxiv.org/abs/2406.17419>. (Cited on pages 24 and 28)

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL <https://aclanthology.org/2023.acl-long.754>. (Cited on pages 1, 2, 3, 10, and 26)

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022a. URL <https://openreview.net/forum?id=gEZrGCozdqR>. (Cited on page 26)

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf. (Cited on page 8)

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 52632–52657. PMLR, 21–27 Jul 2024. URL <https://proceedings.mlr.press/v235/wei24h.html>. (Cited on pages 3 and 4)

Yeming Wen, Pengcheng Yin, Kensen Shi, Henryk Michalewski, Swarat Chaudhuri, and Alex Polozov. Grounding data science code generation with input-output specifications, 2024. URL <https://arxiv.org/abs/2402.08073>. (Cited on page 27)

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierrick Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL <https://aclanthology.org/2020.emnlp-demos.6>. (Cited on page 21)

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=CfXh93NDgH>. (Cited on pages 1, 2, 3, 4, 10, and 26)

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a. (Cited on page 8)

Zhicheng Yang, Yinya Huang, Wei Shi, Liang Feng, Linqi Song, Yiwei Wang, Xiaodan Liang, and Jing Tang. Benchmarking llms for optimization modeling and enhancing reasoning via reverse socratic synthesis, 2024b. URL <https://arxiv.org/abs/2407.09887>. (Cited on page 28)

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL <https://aclanthology.org/D18-1259>. (Cited on page 28)

Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. ZeroGen: Efficient zero-shot learning via dataset generation. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11653–11669, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.801. URL <https://aclanthology.org/2022.emnlp-main.801>. (Cited on page 26)

- Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process, 2024. URL <https://arxiv.org/abs/2407.20311>. (Cited on page 4)
- Asaf Yehudai, Boaz Carmeli, Yosi Mass, Ofir Ariv, Nathaniel Mills, Eyal Shnarch, and Leshem Choshen. Achieving human parity in content-grounded datasets generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=RjYKTQOLW>. (Cited on page 4)
- Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woomyoung Park. GPT3Mix: Leveraging large-scale language models for text augmentation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 2225–2239, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.192. URL <https://aclanthology.org/2021.findings-emnlp.192>. (Cited on page 26)
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024a. URL <https://openreview.net/forum?id=N8N0hgNDRt>. (Cited on pages 3, 4, and 28)
- Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and Qiufeng Yin. WaveCoder: Widespread and versatile enhancement for code large language models by instruction tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5140–5153, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. URL <https://aclanthology.org/2024.acl-long.280>. (Cited on page 4)
- Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhua Chen. MAMMOOTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=yLC1Gs770I>. (Cited on page 28)
- Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Dylan Slack, Qin Lyu, et al. A careful examination of large language model performance on grade school arithmetic. *arXiv preprint arXiv:2405.00332*, 2024a. (Cited on page 10)
- Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞ bench: Extending long context evaluation beyond 100k tokens, 2024b. URL <https://arxiv.org/abs/2402.13718>. (Cited on page 28)
- Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. Gsm-infinite: How do your llms behave over infinitely increasing context length and reasoning complexity?, 2025. URL <https://arxiv.org/abs/2502.05252>. (Cited on page 4)

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.

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.

²<https://platform.openai.com>

³<https://aistudio.google.com>

⁴<https://console.anthropic.com>

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

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

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.

C Additional Results and Discussion

C.1 Manual Verification

We manually reviewed 100 examples from each of the 3 benchmarks. For CHASE-QA, we verify whether the question and the corresponding ground-truth answer are correct based on the context. For CHASE-CODE, we verify whether the answer code seeks to implement the objectives stated in the problem statement. For

Constructing CHASE-Code

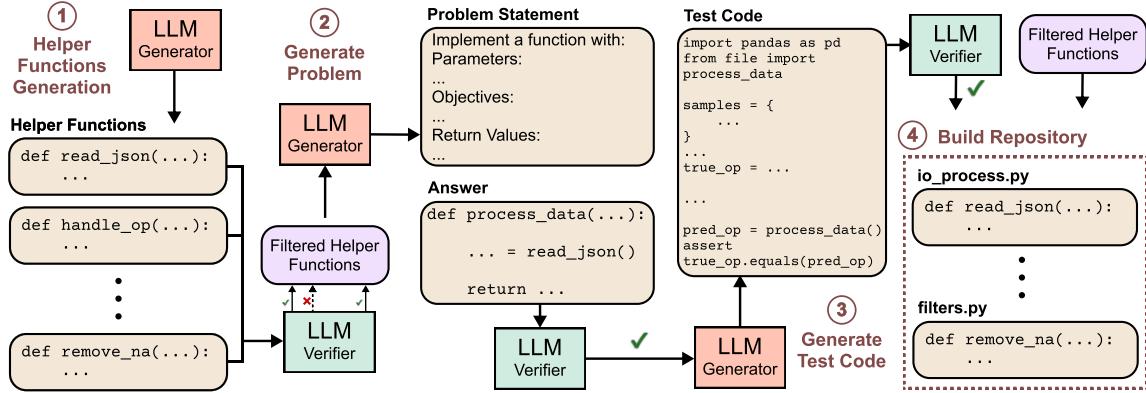


Figure 4: Pipeline for creating an example in CHASE-CODE.

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

CRITERIA	
Number of examples	671
Average context size	6000
Average number of answer points	3.6
Average number of relevant documents	3.3
Average number of irrelevant documents	7.7

CHASE-MATH, we verify whether the ground-truth answer and reasoning are correct for the problem. We found 6 errors in CHASE-QA, 3 errors in CHASE-CODE and 7 errors in CHASE-MATH. Examples of such errors in generation are discussed in Appendix E. We believe these error rates are sufficiently low, thereby encouraging reliance on CHASE-generated benchmarks for evaluation.

C.2 Post-hoc Automatic Verification for CHASE-MATH

In our experiments, we found that some post-hoc filtration techniques work well for weeding out the incorrect examples for CHASE-MATH. We outline a general recipe here. Given a set of *held-out* verifier LLMs (not being evaluated) $\{V_1, V_2, \dots, V_n\}$, we can obtain predictions for all the problems generated by applying the CHASE procedure. Problems for which the predictions of at least k verifiers ($k > 1$) matches the ground-truth annotation can be included in the dataset provided that the predictions of the remaining $n - k$ verifiers do not agree on an answer different from the ground-truth. The intuition here is that if multiple LLMs that did not participate in generating the problem-answer pair also reach the same answer when provided with the problem, then there is a high chance that the problem-answer pair is correct. Problems for which a majority of the verifiers agree on an answer different from the ground-truth should be discarded. Finally, the small proportion of problems for which different verifiers make different predictions, none of which match the ground-truth answer can be manually examined and rectified.

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

CRITERIA	ALGO	DP
Number of examples	250	250
Average context size	17000	17000
Average number of statements	22.0	11.4
Average Cyclomatic Complexity	15.4	8.2
Average Halstead Difficulty	19.6	12.2

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

CRITERIA
Number of examples
Average reasoning depth
Average number of words in question

C.3 Intrinsic Complexities of CHASE Benchmarks

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

C.4 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 9. 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

Table 9: Comparison of model performances (pass@1) on CHASE-CODE and HumanEval, a widely-used benchmark for code generation.

MODEL	CHASE-CODE	HUMAN EVAL
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

performances of all models on both datasets in Table 10. 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.

C.5 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 11.

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

Table 10: 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.6 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 12. While there is a clear increase in performance for all models, the task still remains difficult to solve, in general. Examples of errors made by models even with this better prompting technique are provided in Figure 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 (OpenAI Team et al., 2024, Ouyang et al., 2022) being released in recent years combined with the benefits obtained from instruction fine-tuning (Chung et al., 2024, Wei et al., 2022a), 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 fine-tuning smaller LLMs. Mitra et al. (2024) design an agentic framework that uses raw text and

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

MODEL	ACCURACY	K-PRECISION	RECALL
Gemini-1.5-Pro	63.2	85.1	68.6
GPT-4o	55.3	86.7	58.3
Claude-3.5-Sonnet	36.1	77.6	49.0
Gemini-1.5-Flash	55.1	82.3	61.7
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	25.2	61.3	32.0
Qwen2.5-7B	22.2	56.9	30.3

Table 12: 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

code data as seeds to generate large scale synthetic data of prompts and responses.

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

D.2 Synthetic Data for Code Generation and Math Reasoning

There has been significant recent interest in generating synthetic data for code. [Wen et al. \(2024\)](#) utilize I/O specification apart from synthetic intents to generate data science code instructions. [Patel et al. \(2024\)](#) generate code library specifications using GPT-4 to evaluate whether LLMs can learn new code libraries in-context. [Chen et al. \(2023\)](#) and [Prasad et al. \(2025\)](#) focus on generating unit tests for code generation. [Patel et al. \(2023\)](#) implement an LLM-based procedure to paraphrase problems for SQL code generation. In this

work, we automatically generate all parts of a code generation problem: the repository context, problem specification, ground truth answer code, and tests.

Generating synthetic data to improve math reasoning has also been a very active area of research. Tang et al. (2024) explored extracting topics and knowledge from math problems to prompt an LLM to generate new data. Hong et al. (2024) create math problems by using GPT-4 to perturb existing problems in GSM8k. Toshniwal et al. (2024) create an instruction-tuning dataset by synthesizing code-interpreter style solutions for existing math problems using open-source LLMs. Yang et al. (2024b) create a reverse socratic approach to synthesize math problems from optimization scenarios. Luo et al. (2023) generate diverse math instructions data using *Evol-Instruct* and then train LLMs using reinforcement learning. Yue et al. (2024) build an instruction-tuning dataset by using GPT-4 to generate hybrid CoT and PoT rationales for examples from diverse math datasets. Yu et al. (2024a) bootstrap problems from existing math datasets by using an LLM to rephrase the question text.

D.3 Human-curated Evaluation Benchmarks

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

Long Context Reasoning. Kamradt (2023) introduced the Needle-In-a-Haystack (NIH) task in which a model is asked to retrieve a fact placed in a long context. However, this only assessed surface-level long-context retrieval abilities without much reasoning. In the past few months, many more long-context understanding benchmarks have been created. SCROLLS Shaham et al. (2022) and ZeroScrolls (Shaham et al., 2023) are long-context benchmarks created by aggregating and repurposing existing datasets. Compared to the QA datasets in ZeroScrolls which consist of contexts based on science and literature, CHASE-QA focuses on real-world scenarios where a user is looking for information across multiple documents. Many benchmarks such as LongBench (Bai et al., 2024), RULER (Hsieh et al., 2024), L-Eval (An et al., 2024), LooGLE (Li et al., 2024b) and InfiniteBench (Zhang et al., 2024b) consist of long-context QA tasks. However, the relevant information for answering the question is comparatively much more centralized in the context, surrounded by distracting information. In contrast, the relevant information in CHASE-QA is spread across multiple documents. Moreover, the irrelevant information in the other documents is still very closely related to the topics of the question, thereby making the task very difficult. Perhaps, the benchmark most similar to our work is Loong (Wang et al., 2024b). Like CHASE-QA, Loong also 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. In contrast CHASE-QA models the more realistic scenario of searching a bunch of documents that are on very closely related topics, only some of which contain the answer. Moreover, unlike all these prior works, we create long-context reasoning benchmarks completely from scratch in which the long text contexts are also generated using LLMs. Further note that in contrast to these prior works that use manual annotation, our approach is highly scalable, capable of creating thousands of examples across diverse domains automatically in a much more cost-efficient manner.

E Error Analysis

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

E.1 Errors made while solving CHASE benchmarks.

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

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

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

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

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

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

F Prompts

F.1 Prompts for CHASE-QA

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

- Generating scenarios: Figure 12
- Generating QA pairs: Figure 13
- Generating irrelevant information: Figure 14
- Generating documents: Figure 15
- Verifying irrelevant information: Figure 16
- Verifying absence of relevant information: Figure 17
- Verifying presence of answer: Figure 18
- Solving the task: Figure 19
- LLM-as-a-judge: Figure 20

F.2 Prompts for CHASE-CODE

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

- Generating helper functions: Figure 23
- Generating problem statement and answer code: Figure 24
- Generating test code: Figure 25
- Generating test for verifying if function executes: Figure 26
- Verify correctness of problem statement: Figure 27
- Solving the task: Figure 28

F.3 Prompts for CHASE-MATH

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

- Breaking down seed MWPs: Figure 29
- Creating continuation of MWP: Figure 30
- Solving MWP: Figure 31
- Solving MWP sentence-by-sentence: Figure 32

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.

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, threshold):
8     reconstructed_matrix = svd_reconstruction(matrix, k)
9     extracted_entities = extract_entities(text, entities)
10    can_schedule = can_schedule_meetings(intervals, 0)
11    interesting_cells = high_value_coordinates(reconstructed_matrix,
12                                                 threshold)
12    interesting_cells_with_distances = [((i, j), abs(i) + abs(j)) for ((i, j),
13                                         _) in interesting_cells]
13    return reconstructed_matrix, extracted_entities, can_schedule,
           interesting_cells_with_distances
```

Predicted Code:

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

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 $2/3$ 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 $1/3$ 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 * $2/3$ = 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 $1/3$ * 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 $2/3$ of Harry's 9 hours, which is $(2/3) * 9 = 6$ 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 $1/3$ 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.

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 $3 * 3 = << 3 * 3 = 9 >>$ 9 times So he runs $9 * 60 = << 9 * 60 = 540 >>$ 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 meters/day * 7 days = 840 meters.

Sentence 9: Nothing to solve.

Sentence 10: He reduces his total running distance by 420 meters, meaning he'll run 840 meters - 420 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.

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.

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.

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

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

Ground-truth Answer:

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

Figure 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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

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

CHASE-CODE: Verifying if Function Executes

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

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

{FUNCTION}

Follow these instructions:

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

Give output in the following format:

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

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

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.

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.

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.

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 $5 \times 3 = 15$ 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.

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. 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.