Evaluating LLMs' Mathematical and Coding Competency through Ontology-guided Interventions

Anonymous ACL submission

Abstract

Recent advancements in Large Language Models (LLMs) have showcased striking results on existing logical reasoning benchmarks, with some models even surpassing human performance. However, the true depth of their competencies and robustness in reasoning tasks remains an open question. To this end, in this paper, we focus on two popular reasoning tasks: arithmetic reasoning and code generation. Particularly, we introduce: (i) a general ontology of perturbations for maths and coding questions, (ii) a semi-automatic method to apply these perturbations, and (iii) two datasets, MORE and CORE, respectively, of perturbed maths and coding problems to probe LLM capabilities in numeric reasoning and coding tasks. Through comprehensive evaluations of both closed-source and open-source LLMs, we show a significant performance drop across all the models against the perturbed questions, suggesting that the current LLMs lack robust problem solving skills and structured reasoning abilities in many areas, as defined by our ontology.

1 Introduction

Logical reasoning in a structured and well-defined domain, such as mathematics and programming, becomes increasingly harder with the increasing presence of interspersed and diverse situations, events, and contexts formulated through natural language queries. Current state-of-the-art Large Language Models (LLM) have shown impressive performance on mathematical problems (Cobbe et al., 2021a) and reasonable performance on coding problems (Chen et al., 2021a) expressed in natural language. However, these evaluations barely test the depth of LLMs' expertise, and thus we do not currently have clear insights into the LLM capabilities in these domains. For example, in mathematics, GPT-4's performance monotonically decreases from GSM-8k (Cobbe et al., 2021a) (92%; 5-shot CoT) on grade school mathematical problems demanding rigorous arithmetic and logical reasoning to solve; to MMLU-Math (87.5%) (Hendrycks et al., 2020) on a collection of mathematical problems, ranging in difficulty from elementary to advanced levels; and to MATH (50.36%) (Hendrycks et al., 2021) on challenging competition mathematics problems. Similar variance in LLM performance can also be observed for coding challenges (Chen et al., 2021a). Such shallow evaluations are unfit for an objective measure of the finer LLM capabilities as (i) many LLMs like GPT-4 (OpenAI, 2023) are exposed to publicly available math and coding datasets during pre-training; and ii) many datasets focus on advanced branches of mathematics and problems without bolstering the fundamentals. Hence, before testing the LLMs' breadth of capabilities by delving into higher mathematics and evaluating competitive coding questions, we instead focus on depth through one fundamental question:

> How robust are the capabilities of LLMs in terms of reasoning and understanding of the problem-solving process?

In this work, our goal is to provide an evaluation mechanism that provides clear insights into the robustness of the reasoning abilities of LLMs in the context of maths and coding. Following previous work towards probing language models (Ribeiro et al., 2020; Wu et al., 2023; Li et al., 2024a; Wang et al., 2024), we evaluate the robustness of LLMs' understanding of interesting linguistic and logical structures and derive insights based on them.

Specifically, we design an adaptive dynamic evaluation benchmark through novel ontology-guided perturbations on existing problems. We introduce a novel ontology of perturbation operations that lists various changes across a diverse set of factors, which we apply to previously introduced arithmetic and coding problems. These perturbations allow

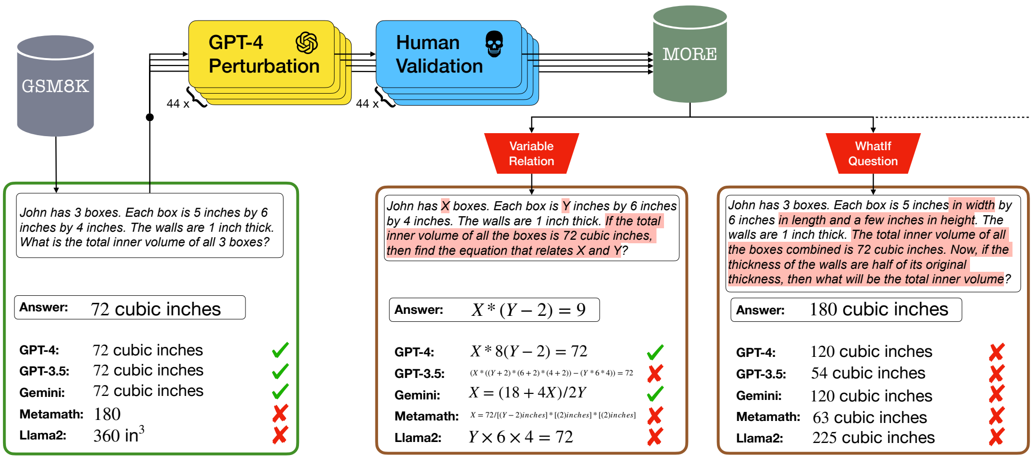

Figure 1: A semi-automated pipeline of creating MORE, from five simple questions from GSM8k. An analogous pipeline is used to create the perturbations of the coding questions from HumanEval, named CORE.

us to assess whether the model comprehends underlying concepts. For instance, while a model may correctly answer questions in a dataset like GSM8k, it might struggle when presented with a simple perturbation to the question, such as replacing numerical values in maths questions with variables, which challenges the model to establish relationships among the variables, revealing its deeper understanding (or lack thereof). By introducing these ontological perturbations, (1) we gain insights into the models' reasoning abilities and (2) uncover strategies for future data augmentation that can then be utilized to enhance LLMs through weakly supervised fine-tuning methodologies.

Our ontology consists of 44 types of perturbations, which we apply to sample questions from GSM8K and coding questions from HumanEval, resulting in 216 and 219 perturbed questions respectively. Our evaluation of GPT-4, GPT-3.5, Metamath, Llama-code, Llama3-Instruct, and Gemini 1.5 shows that most of these models very quickly degrade under different perturbation types. Our contributions are as follows:

1. We propose a novel, extensive, and extensible ontology of perturbation operations for basic-math- and coding-based reasoning tasks expressed in natural language.
2. We present a semi-automatic method to exercise such perturbations first through GPT-4, followed by manual filtering. We generate two datasets MORE and CORE—**M**athematics- and **C**ode-**O**riented **R**obustness **E**valuation, respectively—consisting of 216 maths and 219 coding questions.
3. We gain insights into the range of capabilities and limitations on such math and coding tasks for several LLMs.

2 Related Work

A variety of datasets have been developed to assess AI reasoning capabilities across multiple domains. In causal reasoning, significant datasets include those by (Huang et al., 2023; Bondarenko et al., 2022). For coding, notable contributions have been made by (Chen et al., 2021b; Austin et al., 2021). Additionally, mathematical reasoning has been addressed through datasets designed for different educational levels: grade-school (Cobbe et al., 2021b,a), high school (Hendrycks et al., 2021), and college level (Sawada et al., 2023; Zheng et al., 2021). Despite the advancements shown by large language models (Ahn et al., 2024), recent studies (Mondorf and Plank, 2024) contend that these models more closely resemble stochastic parrots (Bender et al., 2021) than true systematic reasoners, exhibiting significant limitations particularly in scenarios not covered by their training data (Bender et al., 2021; Wan et al., 2024).

Therefore, Recent work has focused on the robustness of reasoning under various perturbations that alter reasoning question. Different domain-specific methods have been proposed for generating test cases for reasoning tasks (Yu et al., 2023a; Wu et al., 2023), as summarized in Table 1. In the field of mathematics, contemporary works have employed techniques such as numerical or symbolic substitutions (Li et al., 2024a; Zhou et al., 2023; Meadows et al., 2023; Wang et al., 2024; Patel et al., 2021), the insertion of irrelevant distractors (Shi et al., 2023; Li et al., 2023), functional equivalence

Variant Name	Parent Domain(Dataset)	Type	Annotation	Dimension	Categories
SVAMP (Patel et al., 2021) ⋆	math(ASDiv-A)	Equation-formed list	Human (Q,A)	V L	3
MetaMathQA (Yu et al., 2023a)	math(GSM8K, MATH)	Dynamic CheckList	GPT-3.5-Turbo	V R	4
GSM-HARD (Gao et al., 2022)	math(GSM8K)	Program-formed CheckList	Codex (Q,A), Human (A)	V	1
GSM-IC (Shi et al., 2023) ⋆	math(GSM8K)	Static Checklist	Human (Q)	L	3
GSM-PLUS (Li et al., 2024a) ⋆	math(GSM8K)	Dynamic CheckList	GPT-4, Human (Q,A)	R L T C	8
MORE-CORE (Our) ⋆	math(GSM8K), code(HumanEval)	Dynamic Ontology	GPT-4, Human (Q,A)	R L C T F S V	44

Table 1: Overview of variants in reasoning datasets arising from perturbation types. ⋆ refers to datasets specifically designed to evaluate the robustness of model performance. Different letters represent different perturbation types: [R]epresentational Change, [L]ogic Alteration, [C]oncept Analysis, Critical [T]hinking, [F]ormulation Adjustment, [S]caling, [V]alue Replacement

(Srivastava et al., 2024), and reverse prediction (Yu et al., 2023b; Berglund et al., 2023; Deb et al., 2023) to uncover conceptual errors (Sanyal et al., 2022), cognitive biases (Dasgupta et al., 2022), or sensitivity to reasoning context (Wu et al., 2023). To our knowledge, well-established perturbation methods beyond the domain of mathematics are lacking. In this work, we consolidate and develop a broader underlying ontology that connects and expands upon previous methods for perturbing reasoning datasets. This new framework is both systematic and hierarchical, and it is readily adaptable to various domains, including mathematics and coding.

A distinct line of research focuses on evaluating reasoning through non-conclusion-based assessments, which provide deeper insights into models' reasoning behaviors. For example, ReasonEval (Xia et al., 2024) analyzes the *reliability* and *redundancy* of generated reasoning steps, highlighting the qualitative aspects of reasoning. Similarly, Li et al. (2024b) target at error identification within the reasoning path rather than simply identifying the correct answer. Furthermore, Zeng et al. (2023) explore the robustness of models across varied potential reasoning paths, reinforcing the idea that higher accuracy does not necessarily improve reasoning quality. Our ontology extends these approaches by including perturbations on various concepts related to reasoning path and question understanding, thereby enriching the framework for assessing reasoning capabilities.

3 The Ontology of Perturbations

3.1 The Need for Ontology-based Perturbations

We plan to first identify a set of factors upon which the solution of a structured reasoning problem (expressed in natural language) may depend on (similar to Kaushik et al. (2021)); and perturb a seed question under these set of factors semi-automatically in a model-agnostic way (i.e., not necessarily adversarial to a target model). In the NLI context, Kaushik et al. (2021) utilized human workers to directly perturb a hypothesis, keeping the premise constant; and in a post-hoc way, identifies the categories (or factors) which such revisions pertain to. Previous works (Xu et al., 2023; Li et al., 2024a; Wang et al., 2024) discusses ways of perturbation, by identifying a set of factors which is specifically designed to increase the complexity of a seed questions in limited ways. The categories are broad and do not exploit the *logical* nature of the underlying domain (along with the *linguistic* dimensions of the instruction). This is where, we believed, an ontological approach may help, where broader categories can help us generalize, while fine-grained sub-categories exploit the domain-specific characteristics.

Let's take mathematics for example. The solution to a reasoning problem can depend on the number and complexity of operations, variables, functions, and possible existing theorems (external knowledge). Similarly, code generation problems can depend on the data structures, variables, functions, and libraries it needs access to. On top of this well-defined set of factors existing in structured reasoning problems, the list of factors expands as the problem is expressed in natural language. Entities and relations expressed in the text need to be mapped to variables and constants (in both). Physical actions (giving and taking apples) may need to be mapped to mathematical operations (or code). It is clear that the set of *logical* and *linguistic* factors co-exist in these reasoning problems, detailed in Appendix I.1. Therefore we come up with an extensible ontology, capturing the above nuances. We believe it will capture and categorize the factors where LLMs fail over multiple domains. As others have shown, the same process can be enabled to perform data augmentations.

3.2 The Ontology

Extending SVAMP (Patel et al., 2021)-like perturbations, we propose a set of high-level categories that are applicable to a broad class of reasoning tasks, expressed in natural language. We primarily identified the following hierarchy (see Table 2):

Level I: Aspect. There are two aspects to these perturbations: (i) *structural perturbation* and (ii) *representational perturbation*. *Structural perturbation* covers all perturbations that probe the underlying reasoning path (or structure) in different ways, by slightly varying the logic behind the question or probing intermediate steps, seeking explanations. *Representational perturbations* involves modification of the encoding of the question or solution while preserving the underlying logic of the original question.

Level II: Target. The subject of change in each *aspect* is gradually refined into multiple *Targets*. For example, the target of *logic*, under *structural perturbations*, deals with perturbations that alter the reasoning path in different controlled ways.

Level III: Dimension. This is a further refinement that defines the exact target dimensions (the WHAT) in the reasoning process (question, reasoning, computation, answer expression etc.) to which the perturbations are applied.

Level IV: Category. This level captures the method (the HOW) through which the higher-level *Dimension* perturbation is achieved. These methods are domain dependent and, thus, their implementations vary from maths to coding problems.

4 Curation of MORE and CORE

Our objective is to assess the resilience of LLMs to perturbations of maths and coding questions along various dimensions. Thus, as seed datasets, we use GSM8K (Cobbe et al., 2021b)—a collection of mathematical problems demanding rigorous arithmetic and logical reasoning—and HumanEval (Chen et al., 2021a) for coding. Five questions [1] from GSM8K are perturbed using our ontological framework (see Appendix I) to generate MORE. On the other hand, we sampled five coding problems from HumanEval dataset (Chen et al., 2021a) that were perturbed using the ontology explained in Appendix I. These perturbations are aimed at modifying the problems in terms of complexity and representation to assess the robustness of the LLMs to these ontological categories of perturbations. Fig. 2 shows examples of three perturbed questions and answers from MORE and CORE. Examples and definitions of all the remaining perturbations are present in Appendix I. We use a three-staged combination of automatic generation from GPT-4 (OpenAI, 2023) with human verification and annotation to create MORE and CORE: (i) perturbed question generation (§4.1), (ii) filtering and validation of generated questions (§4.2), and (iii) annotating final answers (§4.3).

4.1 Perturbed Question Generation

In the first stage, our objective is to create perturbed questions from the source GSM8K/HumanEval questions for each perturbation type. We write prompt templates for each perturbation type and fill them with a source question to create the input prompt to GPT-4. Each template captures the essence of the respective perturbation type (Appendix I.2, Appendix I.3, Appendix I.4) to instruct GPT-4 on how to perturb the source question.

For example, the prompt for *Remove Constraint* (**G1.**) for our running example is as follows:

> Instruction: Rewrite the original mathematical context below based on the #Rewrite Requirement#.
>
> Your output should only be #Rewritten Context#.
>
> #Original Context#: John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick.
>
> #Original Query#: What is the total inner volume of all 3 boxes?
>
> #Rewrite Requirement#: 1. Remove some constraints or information from the original context. 2. Make sure the rewritten question can still be solved, but the answer is simpler.
>
> #Rewritten Context#:

This prompt to GPT-4 generated: *John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. What is the total volume of all 3 boxes?*. The black text in this prompt marks the static template components to enforce the intended perturbation, while the blue text indicates the source question. These templates are used iteratively to generate perturbed questions for GPT-4.

[1] Maths questions in the GSM8K dataset take between two and eight steps to solve. We randomly chose five questions that take three to seven steps to solve. We cover various topics involving algebraic questions, physical application questions, and decision-based application questions

Aspect (Level I)	Target (Level II)	Dimension (Level III)	Category (Level IV)	Math	Code
Structural Perturbation *Def*: Modification on specific aspects of logic or concepts to alter the reasoning process required to reach the answer	**Logic** *Def*: Modifications to the reasoning framework or logic underpinning a problem.	**Granularity Adjustment** *Def*: Fine-grained sub-tasks of the original question	**G1**. Remove Constraint **G2**. Partial Solution **G3**. Solution Plan **G4**. Detail Expansion	Remove Constraint Median Inquiry Solution Plan Detail Elaboration	Remove Constraint Helper Function Solution Plan Example Detail
		Reasoning Adjustment *Def*: Target at logical structure of the original	**G5**. Add Restriction **G6**. Subsequent Question **G7**. Concurrent Question **G8**. Change Question **G9**. Info Recombination **G10**. Domain Knowledge **G11**. Complex Reality **G12**. General Solution	Restrict Question Further Question Parallel Question Change Query Info Recombination Theoretical Challenge Value Probability Code Implementation	Restrict Requirement Further Requirement Parallel Requirement Change Docstring Info Recombination Code Import Example Boundary Higher Order
		Computation Adjustment *Def*: Target at values or entities	**G13**. Computation Demand **G14**. Change Value **G15**. Change Operation	Value Big Change Subject Change Calculation	Generalize Parameter Parameter Content Variable Type
		Formulation Adjustment *Def*: Reformulate question for solution form to be an abstract expression.	**G16**. Symbolic Response **G17**. Value Relationship **G18**. Variable Group **G19**. Backward Reasoning **G20**. Counterfactual **G21**. Solve Value **G22**. Identify Range	Variable Response Variable Relation Variable Scaling Variable Adaptation WhatIf Question Solve X Variable Range	Code Execution Parameter Relationship Variable Substitution Reverse Engineering WhatIf Code Solve Input Variable Range
	Concept *Def*: Examination and Analysis of the underlying concepts and principles of a problem	**Question Understanding** *Def*: Interpretation of the information inside the question	**G23**. Inherent Premise **G24**. Complete Missing **G25**. Question Formulation **G26**. Add Misinformation	Identify Assumption Info Sufficiency Question Formulation Introduce Distraction	Test Case Incomplete Answer Question Formulation Introduce Bias
		Solution Understanding *Def*: Assessment of the problem-solving processes	**G27**. Optimize Solution **G28**. Step Functionality **G29**. Theoretical Basis **G30**. Cost Analysis	Info Necessity Step Necessity Theoretical Basis Solution Efficiency	Reduce Complexity Step Necessity Theoretical Basis Code Complexity
		Critical Thinking *Def*: Identification of noise, inaccuracies and inconsistencies	**G31**. Seek Clarification **G32**. Conditional Analysis **G33**. Conflicting Information **G34**. Surface Error **G35**. Hidden Error	Introduce Ambiguity Discuss Separately Introduce Contradiction Value Uncommon Value Error	Example Requirement Incomplete Requirement Wrong Example Runtime Error Logical Error
Representational Perturbation *Def*: Preservation of the underlying logic and conceptual framework, but modification of the encoding or representation	**Question Format** *Def*: Direct modification on the encoding of the question while keeping the logical structure intact	**Format Change** *Def*: Rephrasing the question in a different format	**G36**. Setting Rephrase **G37**. Change Sequence **G38**. Close Format **G39**. Data Restructuring	Change Setting Change Sequence True False Value Structuring	Realworld Usecase Parameter Sequence True False Complex Docstring
		Format Comparison *Def*: Comparing two problem of different forms	**G40**. Identical Problem	Identical Question	Identical Code
	Answer Format Def: Indirect modification on the output form	**Format Constraint** *Def*: Add constraint on the solution	**G41**. Reasoning Format **G42**. Reasoning Style **G43**. Alternative Answer **G44**. New Rule	Binary Coded X Language Alternative Answer Define Rules	No Keyword X Language Alternative Answer Simple Name

Table 2: Our proposed ontology framework with domain, dimension, mathematical and code realization categories.

4.2 Filtering and Validation

Unfortunately, GPT-4-generated perturbed questions sometimes lack meaning and suitability for robustness testing due to complex and open-ended perturbation types, leading to errors in generation. As noted in Li et al. (2024a), GPT-4 may i) fail to incorporate perturbations, such as missing values in *Data Restructuring*, ii) introduce unintended changes. We aim to maintain Human Understandability, Logical Coherence, and Instruction Adherence, as detailed in Appendix C.2.

To ensure these qualities and relevance, we use a semi-automatic filtering process. Initially, GPT-4 performs an automated check against the three criteria, discarding any questions that do not meet them. Failed questions are regenerated and re-evaluated, with persistent failures handled by a human annotator.

Human Verification. Despite automatic verification, perturbed questions still have limitations, so we conduct a final human verification to refine them. Our findings show that 36% of the filtered questions needed minor rewording, 31% contained significant inaccuracies or failed the filtering, and 33% were correct as is. Thus, the final questions in MORE are high-quality, understandable, logically coherent, and aligned with the intended perturbation method. Human verification is performed by five PhD computer science students, with each question revised by two annotator and verified by two others.

4.3 Obtaining Final Answers of the Perturbed Questions

Finally, we also annotate the gold answer for the perturbed questions. We engaged the same five annotators for this process. Each gold answer was initially annotated by one annotator. Subsequently, the annotated responses underwent verification by the other two annotators.

4.4 Statistics of MORE and CORE

We sampled five questions from GSM8K and HumanEval and perturbed them using GPT-4 in 44 distinct perturbation categories. Following a rigorous process of filtering and validation, we retained a total of 216 and 219 perturbed questions in MORE and CORE, respectively. We specify the detailed statistics in Appendix B and the details of the five

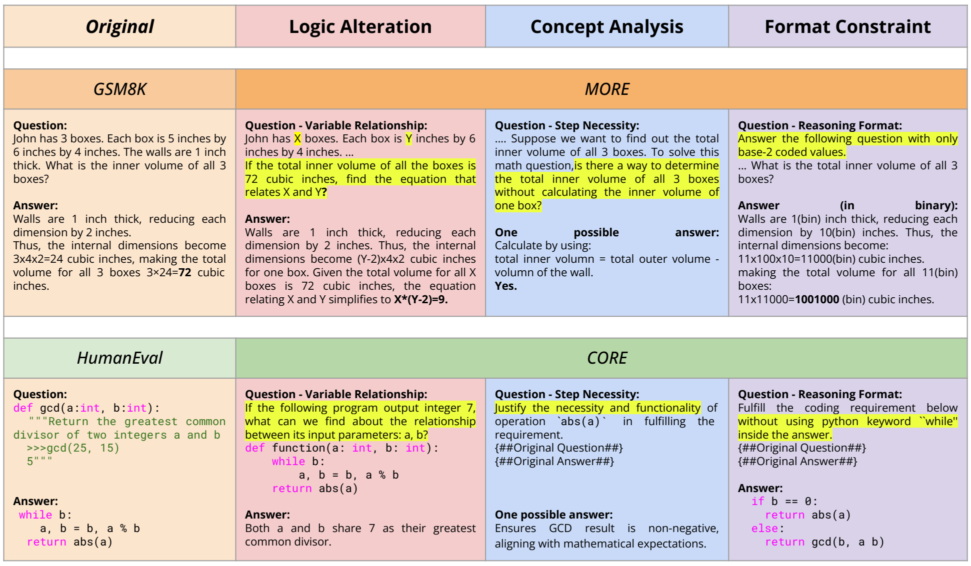

Figure 2: Examples of the original questions and perturbed questions with *Logic, Concept* and *Format* as Targets. The targeted change for each question is highlighted in yellow background

selected question from each dataset in Appendix F and Appendix G respectively.

5 Experiments

5.1 Evaluation Protocol

Owing to the loosely controlled format of the LLM responses to the majority of the questions, calculating accuracy through direct string matching with the annotated answer may not always be reliable. Additionally, in the context of *concept analysis*, curating an exhaustive list of correct answers could be intractable. For instance, the category *optimize solution* (**G27.**) asks to further optimize the provided solution. There could be numerous distinct valid ways to optimize the given solution. To address these challenges, manual evaluation is necessary. To empirically justify this, we prompted GPT-4 for automated answer evaluation, yielding an agreement of 88.76% with human annotation on the answers of GPT-4 to MORE questions.

5.2 Experimental Setup

We evaluated five prominent closed- and open-sourced LLMs on our benchmark. The closed-sourced LLMs are GPT-4, GPT-3.5, and Gemini 1.5. The remaining open-sourced LLMs include one general-purpose LLM and one LLM finetuned on task-specific datasets. The general-purpose LLM is Llama3-8B-Instruct and task-specific LLMs are MetaMath-70B-V1.0 and CodeLlama-70B-Instruct for coding and maths, respectively. MetaMath-70B-V1.0 is finetuned on a mixture of datasets from Metamath (Yu et al., 2023b) and Mistral (Jiang et al., 2023) and CodeLlama-70B-Instruct is finetuned on publicly available coding and coding-related instructions (Rozière et al., 2023). Model Details are specified in Appendix C.1. We listed the prompts used for these models in Appendix J. Each question is evaluated with `pass@1` metric under zero-shot setting. More details in Appendix J on the evaluation settings.

5.3 Experimental Results and Analyses

General Performance Analysis. The results show that perturbed questions significantly challenge all models in both math and coding contexts. GPT-4's accuracy decreased notably, as did other LLMs, with all showing a performance decline over 30 points. Notably, closed-source models outperformed open-source ones in every tested aspect. Models like CodeLlama and Metamath, fine-tuned on specific tasks, performed better in logic alteration and representational perturbations but worse in concept analysis. This suggests fine-tuning may restrict broader reasoning capacities. In general, LLMs handled logic alteration better than concept analysis, indicating their robustness in abstract reasoning yet limitations in understanding deeper mathematical concepts. GPT-4 demonstrated resilience across various question types, outshining others especially in handling different problem-solving frameworks, although it still struggled more in math than in coding in concept analysis. We include Target-wise(Level II) performance

	Aspect		Structural									Representational				
	Target	Original	Logic					Concept				Q. Format			A. Format	*Weighted*
	Dimension		Gran. Adjust.	Reason Adjust.	Compute. Adjust.	Formul Adjust.	*Avg. Perf.*	Quest. Under.	Sol. Under.	Crit Think.	*Avg. Perf.*	Form. Change.	Form. Comp.	*Avg. Perf.*	Form. Constraint	*Avg.*
Maths (MORE)	GPT-4	100	100	80	90.91	60	78.30	85	65	48	64.62	90	60	84.00	65	74.21
	GPT-3.5	80	75	27.5	54.55	25.71	38.68	55	45	12	35.38	35	40	36.00	5	35.75
	Gemini	80	90	50	81.82	37.14	56.60	60	20	16	30.77	55	20	48.00	30	46.15
	Llama	60	50	12.5	18.18	5.71	17.92	35	60	4	30.77	5	60	16.00	5	26.24
	Metamath	80	70	15	27.27	11.43	25.47	30	25	4	18.46	35	80	44.00	20	21.27
	Average	80	77	37	54.55	27.90	43.39	53	43	16.8	36.00	44	52	45.60	25	40.72
Coding (CORE)	GPT-4	80	90	37.5	46.67	50	52.29	65	80	44	61.54	65	40	60.00	55	56.7
	GPT-3.5	80	73.68	35	40	29.41	40.74	60	75	40	56.92	50	40	48.00	45	47.09
	Gemini	80	80	32.5	53.33	23.53	41.28	65	75	44	60.00	45	40	44.00	35	47.32
	Llama	60	45	12.5	33.33	11.76	21.10	50	50	8	33.85	25	40	28.00	20	36.61
	CodeLlama	60	80	40	40	11.76	38.53	35	35	28	32.31	40	0	32.00	40	26.34
	Average	72	73.74	31.5	42.67	25.29	38.79	55	63	32.8	48.92	45	32	42.40	39.00	42.81

Table 3: Model performance on maths and coding across various *Dimensions* (Level III of ontology). All the average reported is weighted average.

	Mod.	Q. Simp.	R. Adj.	C. Adj.	S. Man.	*Avg.*
MORE	GPT-4	100	77.5	90.91	71.43	81.13
	GPT-3.5	90	50	90.91	40	58.49
	Gemini	95	57.50	63.64	45.71	61.32
CORE	GPT-4	100	50	46.67	55.88	59.43
	GPT-3.5	82.35	42.50	40	26.47	43.40
	Gemini	70.59	25	53.33	26.47	36.79

Table 4: The impact of incorporating the original question and answer into the prompt on the performance of *logic* Target within the MORE and CORE. The reported average is weighted average.

	Mod.	Q. Simp.	R. Adj.	C. Adj.	S. Man.	*Avg.*
Self-C	GPT-4	95	87.5	90.91	65.71	82.08
	GPT-3.5	60	45	45.45	25.71	41.51
	Gemini	75	45	81.82	40	52.83
POT	GPT-4	95	90	81.82	68.57	83.02
	GPT-3.5	75	57.5	54.55	25.71	50
	Gemini	90	60	63.64	45.71	61.32

Table 5: The impact of using prompting techniques on the performance of *Logic* Target within the MORE and CORE. Self-C stands for Self-Consistency prompting (Wang et al., 2022) and POT stands for Program of Thought (Chen et al., 2022)

analysis in Appendix D

Incorporation of Original Answer in Prompt. In Table 4, providing models with the correct answer to the original question along with the prompt significantly improves their ability to solve perturbed questions, particularly in the *Computational Adjustment* dimension. However, performance remains weak in *Symbolic Manipulation*, highlighting challenges in abstract reasoning despite access to solutions. Notably, even equipped with correct answers, some models like Gemini and GPT-3.5 still fail on simpler question variants, underscoring their low sensitivity to semantic perturbations.

Prompting Techniques. In Table 5, different prompting techniques greatly influence model performance in *logic* alteration tasks. The Program-of-Thoughts technique notably boosts reasoning capabilities in closed-source models by reducing logical errors, leading to better performance in symbolic manipulation for GPT-4. Conversely, the Self-Consistency method shows only minor improvements and even a performance decline in the Gemini model, suggesting difficulties in effective in-context learning for new unseen tasks.

Identified Vulnerabilities in Reasoning. The *Formulation Adjustment* dimension presents a significant challenge to both closed-source and open-source models, largely due to the demands of abstract reasoning. Instead of reasoning an number or code as answer, this involves manipulation of abstract maths and coding concepts in the logical space behind the surface of the problem. For example, in the *WhatIf* category, models must hypothesize outcomes by changing certain events under consistent conditions, which requires a nuanced grasp of the problem-solving framework. The *Critical Thinking* dimension tests a model's ability to scrutinize relationships between pieces of information, demanding a comprehensive analysis to identify inconsistencies without a predefined solution path. This emphasizes the necessity for models to thoroughly understand and navigate through all possible avenues to effectively resolve conflicts or discrepancies. Furthermore, the *Format Change* dimension poses difficulties to models like ChatGPT attempt to follow these constraints but often fail to maintain the integrity of their reasoning when adapting to new formats, highlighting a lack of flexibility in handling varied task demands.

6 Discussions

6.1 Difficulty Change by Perturbations

The performance drop may stem from an increased scale of reasoning or a higher level of abstract reasoning required. To explore this, experiments measured changes in the scale and depth of reasoning by comparing the number of reasoning steps and the depth required for each perturbed question against its original version. Difficulty was also evaluated through A/B testing and by recording human performance and response times across various perturbation categories as detailed in Appendix E. Table 7 conducted human evaluation on 44 perturbation types, 11 increased the number of reasoning steps needed and 10 required deeper reasoning compared to original questions. Although more complex questions increased the time humans needed to respond, human performance remained almost the same. However, LLMs showed a notable decrease in performance—11.6% for increased reasoning steps and 3.9% for deeper reasoning. Further, There was also a more than thirty percent change in model performance for perturbed questions of equal difficulty, indicating that increased complexity have minor impacts on model performance, the major performance gap may still come from lack of robustness of LLMs.

Category	Human Acc(Δ)	Model Acc(Δ)	Time Consumption
	Number of Reasoning Steps		
⬆	95.2(-4.8)	32.0(-44.0)	178%
⊜	98.4(-1.6)	44.4(-31.6)	39%
	Reasoning Depth		
⬆	97.3(-2.7)	38.4(-37.6)	113%
⊜	97.7(-2.3)	42.1(-33.9)	62%

Table 6: Summary of Human and Model Accuracy, and Time Consumption by Number of Steps and Conceptual Depth of Questions. ⬆ indicates an increase, ⊜ indicates no change in reasoning steps or depth. Δ stands for the performance change relative to original question

6.2 Design Choices behind Ontology

An effective perturbation type maintains control over most variables while introducing only unidirectional changes to the original questions. Ideally, these perturbations should be noticeable to humans yet subtle enough that the required changes in skills for solving these variant questions do not significantly alter human reasoning, due to inherent human cognitive priors. Any data perturbation ontology necessitates predefined assumptions about which aspects of the data are mutable and how these changes might influence the outcomes. Therefore, recognizing and understanding these assumptions is crucial for enhancing future data augmentation efforts. We document the aspects we have modified, the rationale behind these changes in Appendix I.1.

6.3 Scaling to More Instances

Our human-in-the-loop approach may restrict scaling to more instances; however, our primary focus is on evenly evaluating performance across various perturbation categories, rather than on scaling. Nonetheless, it is feasible to expand the dataset through a multi-agent approach (Wang et al., 2024), which selectively filters out the more challenging samples. Our initial experiments, as detailed in Table 7, indicate that GPT-4 can successfully filter out challenging perturbation categories, achieving a perturbation success rate of over 90%.

7 Conclusion

Our study evaluated the robustness of several prominent Large Language Models (LLMs) in handling mathematical and coding problems. By employing an ontology for random perturbations on questions from the GSM8K and HumanEval datasets, we crafted two specialized datasets, MORE and CORE, containing 216 and 219 questions respectively. These datasets target a broad variations of mathematical and coding problem-solving and analytical skills, resulting in notable performance drops in LLMs upon evaluation. The introduction of MORE and CORE provides a new framework for assessing LLMs' abilities in mathematics and coding, while also revealing their vulnerabilities in consistent reasoning across different formats. This research highlights the complex challenges that LLMs face, stressing the importance of continued exploration into their strengths and weaknesses in logical reasoning tasks. Our dataset MORE and CORE will be publicly available online.

8 Limitations

Despite our attempt to construct a novel systematic ontology to evaluate an LM's "real" robustness and reasoning capabilities in structured reasoning tasks, it may not precisely reflect LLM's true ability due to several factors.

Incompleteness In our endeavor to develop a comprehensive ontology for evaluating Language Models' (LMs) responses to perturbed questions across various reasoning scenarios, we recognize significant limitations. Firstly, despite our efforts, the ontology may not fully capture all essential aspects of reasoning abilities, lacking in breadth and depth. Secondly, the complexity within each reasoning category can vary significantly. For instance, within the *Computation Demand* category, adjusting the number of digits in mathematical operations allows us to modulate the reasoning challenge. However, creating a benchmark that exhaustively encompasses all facets of reasoning behavior is an unattainable goal. Such an exhaustive compilation is beyond the scope of any single study and necessitates collective efforts from the broader research community.

Scalability The size of our dataset is constrained due to the human in the loop required for its preparation. Each question generated by GPT-4 needs to be meticulously reviewed to ensure it is solvable and accurately reflects the intended perturbation specific to its category, without introducing unintended modifications. Furthermore, confirming the accuracy of answers is a critical step, as many questions do not yield answers that exactly match a predefined format. This verification process limits our ability to expand the dataset on a large scale, as it relies on manual effort.

9 Potential Risks

Not applicable.

10 Ethical Considerations

Not applicable.

References

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *ArXiv*, abs/2108.07732.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? . *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*.

Lukas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The reversal curse: Llms trained on "a is b" fail to learn "b is a". *ArXiv*, abs/2309.12288.

Alexander Bondarenko, Magdalena Wolska, Stefan Heindorf, Lukas Blübaum, Axel-Cyrille Ngonga Ngomo, Benno Stein, Pavel Braslavski, Matthias Hagen, and Martin Potthast. 2022. Causalqa: A benchmark for causal question answering. In *International Conference on Computational Linguistics*.

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. 2021a. Evaluating large language models trained on code.

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. 2021b. Evaluating large language models trained on code.

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

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. 2021a. Training verifiers to solve math word problems. *ArXiv*, abs/2110.14168.

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. 2021b. Training verifiers to solve math word problems. *ArXiv*, abs/2110.14168.

Ishita Dasgupta, Andrew Kyle Lampinen, Stephanie C. Y. Chan, Antonia Creswell, Dharshan Kumaran, James L. McClelland, and Felix Hill. 2022. Language models show human-like content effects on reasoning. *ArXiv*, abs/2207.07051.

Aniruddha Deb, Neeva Oza, Sarthak Singla, Dinesh Khandelwal, Dinesh Garg, and Parag Singla. 2023. Fill in the blank: Exploring and enhancing llm capabilities for backward reasoning in math word problems. *ArXiv*, abs/2310.01991.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *ArXiv*, abs/2211.10435.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Xiaodong Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *ArXiv*, abs/2009.03300.

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

Yinya Huang, Ruixin Hong, Hongming Zhang, Wei Shao, Zhicheng YANG, Dong Yu, Changshui Zhang, Xiaodan Liang, and Linqi Song. 2023. Clomo: Counterfactual logical modification with large language models. *ArXiv*, abs/2311.17438.

Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L'elio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *ArXiv*, abs/2310.06825.

Divyansh Kaushik, Amrith Setlur, Eduard Hovy, and Zachary C Lipton. 2021. Explaining the efficacy of counterfactually augmented data. *International Conference on Learning Representations (ICLR)*.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. 2024a. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. *ArXiv*, abs/2402.19255.

Xiaoyuan Li, Wenjie Wang, Moxin Li, Junrong Guo, Yang Zhang, and Fuli Feng. 2024b. Evaluating mathematical reasoning of large language models: A focus on error identification and correction.

Zekun Li, Baolin Peng, Pengcheng He, and Xifeng Yan. 2023. Do you really follow me? adversarial instructions for evaluating the robustness of large language models. *ArXiv*, abs/2308.10819.

Jordan Meadows, Marco Valentino, Damien Teney, and André Freitas. 2023. A symbolic framework for evaluating mathematical reasoning and generalisation with transformers.

Philipp Mondorf and Barbara Plank. 2024. Beyond accuracy: Evaluating the reasoning behavior of large language models - a survey. *ArXiv*, abs/2404.01869.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Arkil Patel, S. Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *North American Chapter of the Association for Computational Linguistics*.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of nlp models with checklist. *arXiv preprint arXiv:2005.04118*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, I. Evtimov, Joanna Bitton, Manish P Bhatt, Cristian Cantón Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre D'efossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *ArXiv*, abs/2308.12950.

Soumya Sanyal, Zeyi Liao, and Xiang Ren. 2022. Robustlr: A diagnostic benchmark for evaluating logical robustness of deductive reasoners.

Tomohiro Sawada, Daniel Paleka, Alexander Havrilla, Pranav Tadepalli, Paula Vidas, Alexander Kranias, John J. Nay, Kshitij Gupta, and Aran Komatsuzaki. 2023. Arb: Advanced reasoning benchmark for large language models. *ArXiv*, abs/2307.13692.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Huai hsin Chi, Nathanael Scharli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*.

Saurabh Srivastava, B AnnaroseM, V AntoP, Shashank Menon, Ajay Sukumar, T AdwaithSamod, Alan Philipose, Stevin Prince, and Sooraj Thomas. 2024. Functional benchmarks for robust evaluation of reasoning performance, and the reasoning gap. *ArXiv*, abs/2402.19450.

Yuxuan Wan, Wenxuan Wang, Yiliu Yang, Youliang Yuan, Jen tse Huang, Pinjia He, Wenxiang Jiao, and Michael R. Lyu. 2024. A & b == b & a: Triggering logical reasoning failures in large language models. *ArXiv*, abs/2401.00757.

Siyuan Wang, Zhuohan Long, Zhihao Fan, Zhongyu Wei, and Xuanjing Huang. 2024. Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation. *ArXiv*, abs/2402.11443.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai hsin Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2023. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. 2024. Evaluating mathematical reasoning beyond accuracy. *arXiv preprint arXiv:2404.05692*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Long Long Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zheng Li, Adrian Weller, and Weiyang Liu. 2023a. Metamath: Bootstrap your own mathematical questions for large language models. *ArXiv*, abs/2309.12284.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models.

Zhongshen Zeng, Pengguang Chen, Shu Liu, Haiyun Jiang, and Jiaya Jia. 2023. Mr-gsm8k: A meta-reasoning benchmark for large language model evaluation.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2021. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *ArXiv*, abs/2109.00110.

Zihao Zhou, Qiufeng Wang, Mingyu Jin, Jie Yao, Jianan Ye, Wei Liu, Wei Wang, Xiaowei Huang, and Kaizhu Huang. 2023. Mathattack: Attacking large language models towards math solving ability. In *AAAI Conference on Artificial Intelligence*.

A Recommendations

Based on our findings, we make the following recommendations as strategies to address the weaknesses we identified in the logical reasoning competencies of LLMs.

Diversify the Datasets and Formats Used in Fine-tuning. If a model is trained exclusively on a single problem-solving method, its capability to adapt to questions presented in different formats and solve a diverse array of problems diminishes. To counter this, we suggest boosting the model's resilience to perturbations by fine-tuning it with datasets in a variety of formats and adding augmented instructions.

Include More Complex Open-Ended Questions. It is also crucial to move beyond simple multiple-choice questions, and include open-ended questions that test the model's comprehension of mathematical concepts in the fine-tuning dataset, as this enhances its overall understanding and interpretation of questions.

B Dataset Details

In particular, there are a total of 5 maths questions for each category except *Change Subject* and *Reverse Engineering*, which have 3 and 4 questions, respectively, in MORE. Likewise, all but *Reverse Engineering* perturbation—with 4 questions—have 5 coding questions in CORE.

C Experiment Details

C.1 Model Details

- we use version "2023-09-01-preview" for both GPT-4 and GPT-3.5.
- Llama3-Instruct `https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct`
- MetaMath-70B-V1.0 `https://huggingface.co/meta-math/MetaMath-70B-V1.0`
- CodeLlama-70B-Instruct `https://huggingface.co/codellama/CodeLlama-70b-Instruct-hf`

C.2 Filtering Criteria

(i) **Human Understandability**: The generated questions should be comprehensible to humans. The language, structure, and presentation of the questions should be clear and easy to understand. Vague or confusing questions should be rejected.

(ii) **Logical Coherence**: The questions must make logical sense. They should not contain contradictions[2], nonsensical premises, or incoherent elements.

(iii) **Instruction Adherence**: The generated questions should closely adhere to the instructions in the prompt for the specific perturbation type. The question should not deviate from the intended method of perturbation.

D Fine-Grained Analysis

As illustrated in Table 3, the introduction of perturbed questions poses significant challenges to all models in both maths and coding contexts. Specifically, GPT-4's accuracy decreased from 100% to 74.2% and from 80% to 56.7% in math and coding scenarios respectively. This trend of performance degradation is even more pronounced in other LLMs, with all experiencing a decline exceeding 30 points in their weighted average performance across both the mathematical and coding datasets. For instance, GPT-3.5 witnessed a dramatic performance reduction from 80% to 35.75% on the mathematical dataset and from 80% to 47.09% for the coding dataset.

Notably, closed-source models consistently outperform open-source models in every tested dimension. Additionally, it has been observed that models which have undergone fine-tuning on task-specific data—such as, CodeLlama for coding problems and Metamath for math problems—show enhanced performance in the areas of *logic alteration* and *representational perturbations* as compared to the Llama2-Chat model. However, this fine-tuning process appears to compromise Llama2's capabilities within the *concept analysis* domain. This observation suggests that the focus of fine-tuned, task-specific data on deriving a fixed solution might limit a model's broader capacity for reasoning, thereby affecting its ability to analyze and comprehend the underlying problem-solving process.

(Level II) Target-wise Performance. Following Table 3, LLMs generally showed better results on *logic alteration* questions, which involve concrete reasoning steps in problem-solving. Despite this, even the state-of-the-art models struggled with certain perturbed versions of these questions. This indicates that while current models may possess general task-solving skills and abstract reasoning ability, there is still a limitation in their reasoning robustness when faced with altered logic. On the other hand, *concept analysis* questions, which demand a deeper understanding of mathematical concepts and problem-solving frameworks, resulted in lower success rates. This suggests that while current models can find correct answers, they may lack a systematic logical framework for problem-solving and struggle with analyzing and understanding different concepts necessary to answer the question.

GPT-4, in particular, demonstrated superior performance across all categories, showing increased resilience to changes in question format and expected responses. This contrasts with other models, which performed poorly on tasks involving representational perturbations, hinting at a limitation in transferring their reasoning processes to different formats. Interestingly, the average performance decline across domains was similar for both math and coding contexts, with the notable exception of the *concept analysis* domain, where the drop in math performance was 21% greater than in coding. This discrepancy suggests that LLMs may possess a more profound understanding of problem-solving frameworks in coding contexts compared to mathematical ones.

E Benchmark Difficulty Evaluation

The evaluation of difficulty was conducted by three undergraduate students. Each participant was presented with questions to solve on paper, without access to calculators or computers. Their task completion time for each question was recorded. The students also documented changes in the number of steps required to solve perturbed questions compared to the original, noting whether the number of steps increased, or remained roughly the same. Additionally, they assessed whether the perturbed variants demanded more higher level mathematical concepts or skills.

F Original Questions from GSM8K

The following selected questions are from the GSM8K dataset, specifically chosen for their variations in complexity. Each of the five questions requires between 3 to 7 steps to solve, illustrating the range of reasoning complexity present in the GSM8K dataset. These questions span a wide array of everyday topics that involve the application of mathematical principles, including physi-

[2]Except for the *conflicting information* (**G33**.) type, where we intentionally introduce contradictions.

Dimension	Category	Human Acc	Model Acc	Time Consump	Steps	Reasoning Depth
Granularity Adjustment	Remove Constraint	100	84	-70	⊜	⊜
	Partial Solution	100	70	-40	⊜	⊜
	Solution Plan	100	76	-50	⊜	⊜
	Detail Expansion	100	70	-50	⊜	⊜
Reasoning Adjustment	Add Restriction	100	22	+100	⬆	⊜
	Subsequent Question	100	34	+50	⊜	⊜
	Concurrent Question	100	36	+150	⊜	⊜
	Change Question	100	42	-70	⊜	⬆
	Info Recombination	87	28	+40	⬆	⊜
	Domain Knowledge	80	56	+450	⬆	⬆
	Complex Reality	100	32	+100	⬆	⊜
	General Solution	100	24	+0	⊜	⬆
Computation Adjustment	Computation Demand	100	36	+20	⊜	⊜
	Change Value	100	56	-10	⊜	⊜
	Change Operation	100	66	+0	⊜	⊜
Formulation Adjustment	Symbolic Response	100	42	+100	⊜	⬆
	Value Relationship	93	20	+100	⊜	⬆
	Variable Group	100	24	+140	⬆	⬆
	Backward Reasoning	100	26	+100	⊜	⬆
	Counterfactual	100	18	+160	⬆	⊜
	Solve Value	100	28	+140	⊜	⊜
	Identify Range	93	26	-40	⊜	⊜
Question Understanding	Inherent Premise	100	38	+160	⊜	⊜
	Complete Missing	100	60	-50	⊜	⊜
	Question Formulation	93	50	+200	⊜	⊜
	Add Misinformation	100	68	+50	⊜	⊜
Solution Understanding	Optimize Solution	100	50	+160	⬆	⬆
	Step Functionality	100	42	+100	⊜	⬆
	Theoretical Basis	100	62	-50	⊜	⊜
	Cost Analysis	100	58	+50	⊜	⬆
Critical Thinking	Seek Clarification	80	26	-50	⊜	⊜
	Conditional Analysis	93	16	+200	⬆	⊜
	Conflicting Information	100	8	+50	⊜	⊜
	Surface Error	100	44	+50	⊜	⊜
	Hidden Error	93	30	+200	⊜	⊜
Format Change	Setting Rephrase	100	50	+0	⊜	⊜
	Change Sequence	100	52	+0	⊜	⊜
	Close Format	93	36	+20	⊜	⊜
	Data Restructuring	100	40	+160	⬆	⊜
Format Comparison	Identical Problem	87	42	+20	⊜	⊜
Format Constraint	Reasoning Format	100	30	+200	⬆	⊜
	Reasoning Style	100	34	+170	⊜	⊜
	Alternative Answer	100	28	+60	⊜	⊜
	New Rule	87	36	+250	⬆	⊜
Average		97.7	41.2	+74.3	N/A	N/A

Table 7: Comparison of Average Baselines: Human vs. Models. Displays accuracy rates for participants and models, and time change percentage for solving perturbed vs. original questions. ⬆ indicates an increase; ⊜ signifies equal reasoning depth.

cal dimensions, profit maximization, purchasing decisions, time management, and solving multi-variable equations. Those 5 questions demands diversity of mathematical problem-solving skills in different situations.

F.1 Question 1

A merchant wants to make a choice of

purchase between 2 purchase plans: jewelry worth $5,000 or electronic gadgets worth $8,000. His financial advisor speculates that the jewelry market will go up 2.5% while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much profit would this be?

Answer: If he purchases jewelry, he will make a profit of 2.5% which is 5000*(2.5/100) = 125. If he purchases electronic gadgets, he will make a profit of 1.2% which is 8000*(1.2/100) = 96. If he wants to maximize profit, since 125 > 96, he will choose to purchase jewelry, thereby making a profit of 125

F.2 Question 2

Question 2: John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes?

Answer: The walls subtract 2*1=2 inches from each dimension. So each box has 5-2=3 inch width It also has a 6-2=4 inch height. Finally, it has a 4-2=2 inch depth. So the inner volume of one box is 4*3*2=24 cubic inches. So in total the inner volume of the 3 boxes is 3*24=72 cubic inches

F.3 Question 3

Question 3: Kylar went to the store to buy glasses for his new apartment. One glass costs $5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?

Answer: The discount price of one glass is 60/100 * 5=3. If every second glass is cheaper, that means Kylar is going to buy 16 / 2 = 8 cheaper glasses. So for the cheaper glasses, Kylar is going to pay 8 * 3 = 24. And for the regular-priced glasses, Kylar will pay 8 * 5 = 40. So in total Kylar needs to pay 24 + 40 = 64 for the glasses he wants to buy.

F.4 Question 4

Question 4: Vicki is planning a pop concert at her high school. The show will be 2 hours. She is allowing each group 2 minutes to get on stage, 6 minutes to perform, and then 2 minutes to exit the stage. If she allows a 10-minute intermission, how many groups can perform in the concert?

Answer: First, we should convert the 2 hours of showtime into minutes for our calculations. Since there are 60 minutes in 1 hour, the show will be 2 x 60 = 120 minutes. Of those 120 minutes, 10 will be used for intermission, so 120 – 10 = 110 minutes for performances. Each group will use 2 minutes to get on stage + 6 minutes to perform + 2 minutes to exit the stage = 10 minutes of show time. Of the 110 minutes of performances, 10 are used per group, so 110 minutes / 10 = 11 groups can perform.

F.5 Question 5

Question 5: Together Lily, David, and Bodhi collected 43 insects. Lily found 7 more than David. David found half of what Bodhi found. How many insects did Lily find?

Answer: Let B = the number of insects Bodhi collected. David = B/2, Lily = B/2 + 7. B + B + 7 = 43. Simplify: 2B = 36. Simplify B = 18 insects. David = 18/2 =9 insects. Lily = 9 + 7 = 16 insects. Lily found 16 insects.

G Original Questions from HumanEval

The following selected questions are from the HumanEval dataset, specifically chosen for their variations in complexity. Each of the five questions requires different number of lines code to solve, illustrating the range of reasoning complexity present in the HumanEval dataset. These questions includes basic programming concepts such as string manipulation, list indexing, classic algorithm, math problem and state conditions. Those 5 questions demands diversity of programming skills and concepts in different situations.

G.1 Question 1

```
def flip_case(string: str) -> str:

```

```
    """For a given string, flip lowercase characters to uppercase and uppercase to lowercase.

    >>> flip_case('Hello')
    'hELLO'
    """
    return string.swapcase()
```

G.2 Question 2

```
def greatest_common_divisor(a: int, b: int) -> int:

    """ Return a greatest common divisor of two integers a and b

    >>> greatest_common_divisor(3, 5)
    1
    >>> greatest_common_divisor(25, 15)
    5
    """

    while b:
        a, b = b, a \% b
    return abs(a)
```

G.3 Question 3

```
def derivative(xs: list):

    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
    Return derivative of this polynomial in the same form.

    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
    if len(xs) == 1: return [0]
    if len(xs) == 0: return []
    return [(i * x) for i, x in enumerate(xs)][1:]
```

G.4 Question 4

```
def sum_squares(lst):

    """
    This function will take a list of integers. For all entries in the list, the function shall square the integer entry if its index is a multiple of 3 and will cube the integer entry if its index is a multiple of 4 and not a multiple of 3. The function will not
    change the entries in the list whose indexes are not a multiple of 3 or 4. The function shall then return the sum of all entries.

    Examples:
    For lst = [1,2,3] the output should be 6
    For lst = []  the output should be 0
    For lst = [-1,-5,2,-1,-5]  the output should be -126
    """

    result =[]
    for i in range(len(lst)):
        if i%3 == 0:
            result.append(lst[i]**2)
        elif i% 4 == 0 and i%3 != 0:
            result.append(lst[i]**3)
        else:
            result.append(lst[i])
    return sum(result)
```

G.5 Question 5

```
def is_nested(string):

    """
    Create a function that takes a string as input which contains only square brackets.
    The function should return True if and only if there is a valid subsequence of brackets
    where at least one bracket in the subsequence is nested.
    Examples:
    [[]] output: True
    []][ output: False
    [] output: False
    [[][]] output: True
    [[]][[ output: True
    """

    stack = []
    depth = 0
    for i, char in enumerate(string):
        if char == '[':
            stack.append('[')
            if depth > 0:
                depth -= 1
        elif char == ']':
            if len(stack) > 0:
                stack.pop()
                depth += 1
            if depth >= 2:
                return True
            if len(stack) == 0:
                depth = 0
    return False
```

H Ontology

The summary of our proposed ontological categories is shown in Table 2.

I Ontology of Perturbations

I.1 Principles behind Ontology

Consider a maths question:

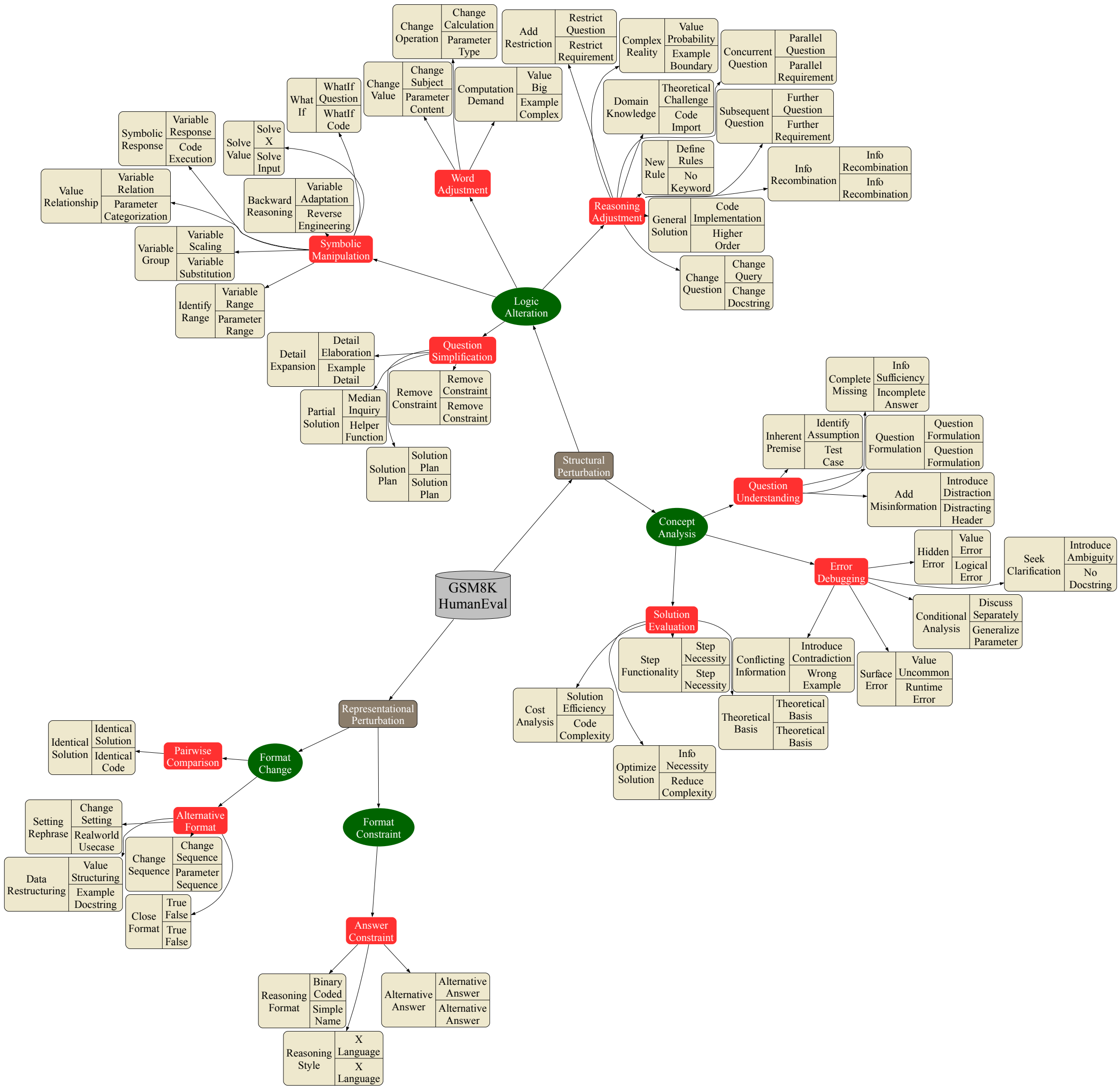

Figure 3: The ontology of the perturbations.

Question: John has 3 boxes, each of which is externally measured as 5 inches by 6 inches by 4 inches. The boxes have walls that are 1 inch thick. What is the total inner volume of all the boxes?

We consider the following eight aspects of such questions:

(i) **Information**: Each sentence clause that is mentioned inside the question. For example: `Each box is 5 inches by 6 inches by 4 inches.`

(ii) **Query**: What is being asked by the Question that can be calculated with the given Information? For example: `What is the total inner volume of all the boxes?`

(iii) **Values:** Values inside the Information. For example, 3 boxes in this particular instance.

(iv) **ToolBox**: Mathematical concepts, formulas, and operations that are relevant to solving a specific problem. For example: `Multiplication is used to calculate the volume of a rectangular prism (box) as length × width × height and Subtraction is used to adjust the external dimensions to account for the wall thickness.`

(v) **Mathematical Structure**: Chain of thought and problem-solving strategies that outline how the 'Tools' in Toolbox are organized to transition from the given data to the final answer. For example, to solve the question above: `first, Subtract the` thickness of the walls; second, calculate the volume of one box; third, multiply the volume of one box by the number of boxes.

(vi) **Query Representation**: The Format of how Information and Values are presented. For example: the sequence of Information presented.

(vii) **Final Answer**: Final answer to the Query. For example: 72

(viii) **Answer Representation**: The Format of the answer presented.

In a similar vein, consider a coding question:

```
def greatest_common_divisor(a: int, b:
    int) -> int:
    """ Return the greatest common
    divisor of two integers a and b
    Example:
    >>> greatest_common_divisor(3, 5)
    1
    >>> greatest_common_divisor(25, 15)
    5
    """
```

We can decompose the coding question into the following aspects:

(i) **Question Header**: The name of the function, in the case above, `greatest_common_divisor`

(ii) **Docstring**: Defines the requirement for the final output. For example, `Return the greatest common divisor of two integers`

(iii) **Values**: The type and structure of input arguments. In the above example, `a (integer type)` and `b (integer type)`

(iv) **Examples**: Demonstrations of how the function is used. In the case above,

```
"""
>>> greatest\_common\_divisor(3, 5)
1
>>> greatest\_common\_divisor(25, 15)
5
"""
```

(v) **Toolbox**: Libraries and operations that can be used to achieve a function.

(vi) **Code Structure**: Sequence of steps of code to fulfill the requirement specified in **Docstring**

(vii) **Question Representation**: Format of how the **Question header** and **Docstring** is presented

(viii) **Answer Representation**: Format of how the **Code Structure** is presented.

The perturbations in the ontology we introduce (Fig. 3) operate on these eight aspects of a maths or coding question. Each perturbation changes only one or two aspects of the original question.

We broadly group these perturbations into two main categories: *Structural Perturbation* and *Representational Perturbation*. *Structural Perturbations* generate new questions by modifying the specific targeted aspects of inherent logic, framework, or concepts in the original question. *Structural Perturbation* is further categorized into *Logic Alteration* and *Concept Analysis*. *Logic-Alteration* perturbations changes the logic underpinning a problem through addition or removal of information, or it changes the reasoning framework of the original problem. The *Concept Analysis* questions, however, examines the underlying concepts and principles of the problem. Rather than solving a specific problem, these questions focus on analyzing the process of problem solving, and how it get the solutions, which may require a deeper understanding of the question and problem solving framework. Details and examples for each of these perturbation types are presented below.

Unlike *Structural Perturbations*, *Representational Perturbations* retain the logical structure of the original solution, only to exclusively change the representation or encoding of the information present in the question or in the answer. In our ontology, *Representational Perturbation* has only two manifestations, *Format Change*, which directly alters the representation of the questions and answers. *Format Constraint*, which add constraint that indirectly alters the format of the answers. More details and examples are below.

For each of the above broad perturbation types, we further define many dimensions of perturbations. We apply specific methods to introduce variations or *perturbations* to the questions along these dimensions. Each dimension can further manifest in various ways that correspond to some method of perturbation. For example, a dimension such as "simplify question" can be realized in different ways for the "logic alteration" perturbation type. These perturbations can affect the difficulty level of the questions, making them either more challenging or simpler. Additionally, some perturbations may result in questions that do not have a definitive answer.

I.2 Logic Alteration

This category groups all the perturbations that have a definitive final answer. The final answer can be in the format of a value (Math) or code(HumanEval) (for dimension "Question Simplification", "Reasoning Adjustment", "Computation Adjustment") or a mathematical expression (Math) or Natural Language (Code) (for dimension "Symbolic Reasoning"). For logic alteration questions, if the final answer is normalized to the most simplified form. The generated answer can be deemed correct only if it can also normalize to the same form.

(i) **Question Simplification**: This dimension aims to make the question easier to solve. It can achieve this by using four ways:

G1. *Remove Constraint*: Remove one piece of constraint that make the question easier to solve
Remove Constraint (Math): Delete one piece of **information** from the original question that does not make the question unsolvable. The aim is to simplify the question. Example:
Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. What is the total volume of all 3 boxes?

Remove Constraint (Code): Simplify the coding requirement by removing one constraint or transformation in the **Docstring**
Generate a python function that fulfills the requirement in docstring and examples usages below.
Changed from G.1:

```python
def change_case(string: str) -> str:

    """For a given string, convert
    all uppercase characters to
    lowercase.

    >>> change_case('Hello')
    'hello'
    """
```

G2. *Partial Solution*: The answer only need to solve parts of the original question
Median Inquiry: Change the original **query** to ask one of the intermediate values that is used to solve for the final answer of the original query. The aim is to simplify the question. Example:
Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the inner volume of one box?

Helper Function: Provide a helper function alongside the coding question that achieves partial function in **Code Structure**
Changed from Appendix G.1: Generate a python function that fulfills the requirement in docstring and examples usages below. You should complete the function using helper function.

```python
def helper_function(char: str) ->
    str:
    """Checks if a given character
    is uppercase or lowercase, and
    flips its case."""

    if char.isupper():
        return char.lower()
    elif char.islower():
        return char.upper()
    else:
        return char

def flip_case(string: str) -> str:

    """For a given string, flip
    lowercase characters to
    uppercase and uppercase to
    lowercase by using the helper
    function above to achieve the
    requirement
    >>> flip_case('Hello')
    'hELLO'
    """

"""
```

G3. *Solution Plan*: Besides the original question, provide a high level plan of how the question should be answered, the solution will only need to execute the abstract plan.
Solution Plan (Math): Provide the original question along with its **mathematical structure** (problem strategy) to the question, ask the model to solve the question by following the strategy.
Changed from F.2:

> John has 3 boxes. Each box is 5 inches

by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes? Follow this plan to solve the question: [#Solution Plan#] Subtract the thickness of the walls from each dimension of the box to get the inner dimensions. Determine the width, height, and depth of the inner box. Calculate the inner volume of one box by multiplying the width, height, and depth. Calculate the total inner volume by multiplying the inner volume of one box by the number of boxes.

Solution Plan (Code): Provide the high level plan of how the code need to be written along with the question. **Changed from G.1:** Generate a python function that fulfills the requirement in docstring and examples usages below. You should follow the solution plan when solving the problem.

```
def flip_case(string: str) -> str:
    """
    Inverts the case of each character in the provided string.

    This function takes a string as an argument and returns a new string with each character's case inverted.
    Uppercase letters are converted to lowercase, and lowercase letters are converted to uppercase.

    Solution Plan:
    1. Create a result variable to hold the updated string.
    2. Iterate through each character in the string.
    3. Check if the character is uppercase; if so, convert it to lowercase and add it to the result.
    4. If the character is lowercase, convert it to uppercase and add it to the result.
    5. After iterating through all characters, return the result.
    """
```

G4. *Detail Expansion*: Besides the original question, provide a few key important details or explanations without which is hard to solve the question.
Detail Elaboration: Provide original question along with the **toolbox** (commonsense knowledge) to solve the question.
Changed from F.2:

John has 3 boxes. Each box has outside dimensions of 5 inches by 6 inches by 4 inches. The walls of each box are 1 inch thick, uniformly throughout each face of the box, thereby reducing the inner dimensions of each box. The material of the boxes is uniformly distributed and does not bulge or cave in thereby affecting the inner volume. There are no internal structures or partitions inside the boxes that could further reduce the inner volume. What is the total inner volume of all 3 boxes?

Example Detail: Besides providing the input and output of each **example**, it also provide a step by step explanation of how the input is transformed to the output. **Changed from G.3:** Generate a python function that fulfills the requirement in docstring and examples usages below.

```
def derivative(xs: list):
    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
    Return derivative of this polynomial in the same form.

    >>> derivative([3, 1, 2, 4, 5]) calculates the derivative as [1*1, 2*2, 3*4, 4*5] resulting in [1, 4, 12, 20].

    >>> derivative([1, 2, 3]) calculates the derivative as [1*2, 2*3] resulting in [2, 6].
    """
```

(ii) **Reasoning Adjustment**: This dimension targets to partially change the logical structure of the original problem. It can be achieved through eight ways:

G5. *Add Restriction*: Add a new piece of condition or requirement to the answer of the question.
Restrict Question: Adding a new piece of **information** that serves as a constraint or modifier on the query. Example:
Changed from F.2:

John has 3 boxes, each of which is externally measured as 5 inches by 6 inches by 4 inches. The boxes have walls that are 1 inch thick. There is also an added wooden board divider in the middle across the smallest dimension which is also 1 inch thick. What is the total inner volume of all the boxes?

Restrict Requirement: Add a piece of information that serves as a constraint or modifier on the function.

Changed from G.1

```
def flip_case(string: str, index: int) -> str:

    """For a given string, flip lowercase characters to uppercase and uppercase to lowercase. Only flip the case for characters at indices which are multiples of the provided index.
    Note: If the index provided is 2, only the characters at the 2nd, 4th, 6th positions and so on will have their cases flipped.

    >>> flip_case('Hello', 2)
    'HeLlO'
    """
```

G6. *Subsequent Question*: Adding an additional query or requirement based on the answer of of the original question.

Further Question: Adding an additional **query** that will need extra steps of calculation based on the final answer of the original query.

> **Changed from F.2**: John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes? **If John wants to entirely fill these boxes with small cubes each measuring 0.5 inches on all sides, then how many cubes will he need?**

Further Requirement: Adding an additional requirement of transformation based on the output of the original function.

```
def flip_case_count(string: str) -> Tuple[str, int]:

    """
    For a given string, flip lowercase characters to uppercase and uppercase to lowercase. Additionally, return the number of case flips performed.

    >>> flip_case_count('Hello')
    ('hELLO', 5)
    """
```

G7. *Concurrent Question*: Adding an additional query or requirement that is independent from the original question.

Parallel Question: Adding an additional **query** along with the original query based on the information given in the question, the added **query** should inquiry a value that is irrelevant of the original answer. Example:

Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes? **What is the total volume of the material used to build the boxes?**

Parallel Requirement: Adding an additional requirement in **Docstring** that does not rely on the output of the original question.

Changed from G.1:

```
def flip_case_and_count(string: str) -> Tuple[str, int]:

    """For a given string, not only should you flip lowercase characters to uppercase and uppercase to lowercase. You should also output another Title case where only the first letter of each word is capitalized"""

    """>>> flip_case_and_count('Hello')
    ('hELLO', 'Hello')
    """
```

G8. *Change Question:* Change the current query or requirement to a different but similar one based on the existing information provided inside the question.

Change Query: Change the **query** to ask for another value that requires more computation based on the information given in the question.

Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total **outer volume** of all 3 boxes?

Change Docstring: Change the **Docstring** to another requirement based on the input given in the **question header**.

Changed from G.3:

```python
def calc_derivative(xs: list):

    """ xs represent coefficients of a polynomial.
    xs[0] * (exp (x))^0 + xs[1] * (exp(x))^1 + xs[2] * (exp(x))^2 + ....
    Return derivative of this polynomial in the same form.
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    >>> derivative([1, 2, 3])
    [2, 6]
    """
```

G9. *Info Recombination*: Combine the fundamental concepts or frameworks from another question with the original question.

Info Recombination (Math): Graft **mathematical structure** from another question and combine with the original question.

Changed from F.4:

> Vicki and James are planning an event at their high school that combines a pop singing concert and dance events. The whole event will be 2 hours long. Vicki is allowing each musical group 2 minutes to get on stage, 6 minutes to perform, and then 2 minutes to exit the stage. James will also perform two solo dance routines, each lasting five minutes. Considering a 10-minute intermission during the show, how many musical groups can perform at the concert?

Info Recombination (Code): Merge the requirement from another coding question with existing question. **Changed from G.1**:

```python
def flip_case_and_odd_sum(string: str) -> tuple:
    """
    Given a string, flip lowercase characters to uppercase and uppercase to lowercase.
    Also return the odd letters that are in even positions of the original string.
    string Index starts from 0, alphabet index start from 1. Aa is 1, Bb is 2..
    Examples:
    >>> flip_case_and_odd_sum('Hello')
    ('hELLO', 'o')
    """
```

G10. *Domain Knowledge*: Introduce a specific knowledge in math or code and merge it with the question.

Theoretical Challenge: Incorporate a specific theorem into the question so that perturbed question requires a new **toolbox** to solve.

Changed from F.2:

> John has an infinite number of boxes numbered as first, second, third, and so on. The first box is 5 inches by 6 inches by 7 inches in size. Starting from the second box each box is half the size of the previous box in each dimension. What is the total volume all the boxes combined?

Code Import: The requirement requires to use a specific python library to solve the problem. **Changed from G.1**: Rewrite the function below to take in batch input parameters and use the multicore cpu for efficiency.

G11. *Complex Reality*: Add an aspect of complexity in the real world scenario.

Value Probability:Introduce concept of uncertainty to deterministic values and calculate the estimation. The perturbed question will require **toolbox** (knowledge) of probability.

Changed from F.1:

> A merchant wants to make a choice of purchase between 2 purchase plans: jewelry worth \$5,000 or electronic gadgets worth \$8,000. His financial advisor speculates that the jewelry market has a 70% chance to go up 2.5% and a 30% chance to remain the same, while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much estimated profit would this be?

Example Boundary: Add boundary examples along with the existing **examples**. The boundary examples contains input that does not met requirement specified in the docstring. **Changed from G.3**: Write a function to fulfill the requirement and all the examples inside the docstring

```python
def derivative(xs: list):

    """ xs represent coefficients of a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2 + ....
    Return derivative of this polynomial in the same form. The solution should pass all the test cases specified below

```

```
    # Regular case
    >>> derivative([3, 1, 2, 4, 5])
    [1, 4, 12, 20]
    # Smaller case
    >>> derivative([1, 2, 3])
    [2, 6]
    # Special case with empty list
    >>> derivative([])
    []
    # Boundary case, the shortest polynomial
    >>> derivative([1])
    [0]
    # Boundary case, all-zero polynomial
    >>> derivative([0.0, 0.0, 0.0])
    [0, 0]
    """
```

G12. *General Solution*: Provide the solution in a more general scenario.
Code Implementation: Develop a code function to solve the question in general.
Changed from F.2:

> # Original Examples # Can you write a Python code to find out what is the total inner volume of all 3 boxes?

Higher Order: Write a higher order function that can solve the coding problem in general.
Changed from G.2

```
def greatest_common_divisor(numbers: list[int]) -> int:
    """
    Calculates the greatest common divisor (GCD) of a list of integers.
    Returns the GCD as an integer.

    Examples:
    - For numbers = [20, 40, 60], the function returns 20.
    - For numbers = [35, 14], the function returns 7.
    """
```

(iii) **Computation Adjustment**: While retaining the **Logical Structure**, this type aims to change one single reasoning step of the original question.

G13. *Computation Demand*: Change the value to complex values that put a high demand on computation.
Value Big: Significantly increasing the magnitude of values that pose a challenge for calculations.
Changed from F.2:

> John has 3000 boxes. Each box is 500 inches by 600 inches by 400 inches. The walls are 100 inches thick. What is the total inner volume of all the boxes?

Generalize Parameter: Extend the current parameter into different python object types
Changed from G.2:

```
def find_common_divisor(value1: Union[int, float, str], value2: Union[int, float, str]) -> float:
    """
    Takes two values (int, float, or float in string format) and finds the largest float that divides both into integers.
    Inputs can be a mix of types. Returns the divisor as a float.

    Examples:
    print(find_common_divisor("0.5", 1))  # 0.5
    print(find_common_divisor(0.25, "1.25"))  # 0.25
    """
```

G14. *Change Value*: Change the content of the value to a different one.
Change Subject: If there are multiple mentions in the question, Exchange **values** of names or references in the question.
Changed from F.5:

> Together David, Bodhi, and Lily collected 43 insects. **David** found 7 more than **Bodhi**. **Bodhi** found half of what **Lily** found. How many insects did Lily find?

Parameter Content: Change the format or meaning of the input parameter.
Changed from G.3:

```
def derivative(polynomial: str):

    """ 'polynomial' is a string that stands for polynomial for form
    coefficients_0 + coefficients_1 * x + coefficients_2 * x^2 + ....
    This function will return the derivative of the aforementioned polynomial in the same format.

    >>> derivative('3 +1x + 2x^2 + 4x^3 + 5x^4')
    '1 + 4x + 12x^2 + 20x^3'
    >>> derivative('1 - 2x + 3x^2')
    '-2 + 6x'
    """
```

G15. *Change Operation*: Change one operation regarding how the **Values** are processed.

Change Calculation: Change no more than 3 words in original question so that the **toolbox** (mathematical operations) involved in the calculation are changed.
Changed from F.2:

> John has 3 boxes. The inner dimension of each box is 3 inches by 4 inches by 2 inches. The walls are 0.5 inches thick. What is the total outer volume of all 3 boxes?

Variable Type: Change the python object type of the original parameter while keep its content the same, also specify the return variable to be in a specific type.
Changed from G.3:

```python
def derivative(xs: list[str]) ->
    list[str]:

    """ xs represent coefficients of
     a polynomial.
    xs[0] + xs[1] * x + xs[2] * x^2
    + ....
    Return derivative of this
    polynomial in the same form.
    """
```

(iv) **Symbolic Manipulation**: This dimension test the abstract reasoning ability of under the same logical structure of the original question. This dimension focus on solving the general version of the original reasoning problem, rather than focus on to get a standard solution. For math, We change the context to include one or more symbolic variables to replace its original **values**.

G16. *Symbolic Response*: Use logic to infer the final output after a sequence of steps.
Variable Response: Replace one **value** inside the question with a variable and answer with the variable included.
Changed from F.2:

> John has X boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all the boxes as a function of X?

Code Execution: Given Docstring requirement, and specific input parameter, find the output for the function without writing any code. **Changed from G.1**: Find the output of the following function description, if the input is:string = "Hello World&7"

```python
def flip_case(string: str) -> str:

    """For a given string, flip
    lowercase characters to
    uppercase and uppercase to
    lowercase."""
```

G17. *Values Relationship*: Identify the relationship between input values or parameters if the output or the final answer is given.
Variable Relationship: Replace a pair of **values** inside the question with variables. After answering the original question, the variable forms a relationship. Query that relationship.
Changed from F.2:

> John has X boxes. Each box is Y inches by 6 inches by 4 inches. The walls are 1 inch thick. If the total inner volume of all the boxes is 72 cubic inches, then find the equation that relates X and Y?

Parameter Relationship: Given the output of the function, categorize the possible groups of inputs parameters into the question. **Changed from G.2:** If the below program output integer 7. What is the relationship between a and b

```python
def function(a: int, b: int) -> int:
    while b:
        a, b = b, a % b
    return a
```

G18. *Variable Group*: Change a group of several input values or parameters to variables. *Variable Scaling*: After answering the question, change the **query** to: if certain factual numbers in the question is scaled up by x, how will the final answer change?
Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. Now, the number of boxes, the box outer dimensions, and the wall thickness are all scaled up by a factor of X. What is the total inner volume of all the boxes as a function of X?

Variable Substitution: Change one or more variables inside the docstring to input parameters. **Changed from G.1**:

```python
def flip_case(string: str,
    specific_value: str) -> str:

    """"""For a given string and
    specific value, flip the
    specific value from lowercase to
     uppercase or uppercase to
    lowercase.  The function will
    only flip the case of the
    specific value in the string.
```

```
    >>> flip_case('Hello', 'h')
    'hello'
    """
```

G19. *Backward Reasoning*: Reverse the reasoning process, reason from how to reach input from output.

Variable Adaptation: If the answer to the question add or subtract by a certain amount x, pick one **value** inside the **Information** and ask how it should change if other **values** are kept the same.

Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. If the total inner volume of all 3 boxes increases by a certain variable X, how should the thickness of the walls adjust correspondingly if the number of boxes and the external dimensions of each box stay the same? Write the answer as a function of X.

Reverse Engineering: Change the **Docstring, Function Header, and Examples** to find the function that can reverse engineer the original function. Specifically, mapping the output back to its original inupt. **Changed from G.1**: Create a function that reverses the following function's process, effectively transforming its output back into the original input

```
def function(string: str) -> str:
    return string.swapcase()
```

G20. *What If*: What would the outcome be if X had happened instead of Y, given the same initial conditions and context.

WhatIf Question: First mask some number of **values** inside the question and answer the original question. What if we change one value inside the question, how will the final answer change? (The final answer should not have variables included as the masked value could be solved given the final answer.)

Changed from F.2:

> John has 3 boxes. Each box is 5 inches in width by 6 inches in length and a few inches in height. The walls are 1 inch thick. The total inner volume of all the boxes combined is 72 cubic inches. Now, if the thickness of the walls is half of its original thickness, then what will be the total inner volume?

WhatIf Code: WhatIf the **code structure** or **input value** is changed, and some condition is masked. **Changed from G.1**: Find the output of the 'changed_function', if the input is the same.

```
We know that if we input
    masked_input to the `
    original_function`, the output
    is following:
>>> original_function(masked_input)
'hELLO'

Here is the `original_function`:
def original_function(string: str)
    -> str:
    return string.swapcase()

Here is the `changed_function`:
def changed_function(string: str) ->
     str:
    return string.swapcase()[::-1]

What will be the output for `
    changed_function(masked_input)`"
```

G21. *Solve Value*: Mask one variable's **value** inside question, given answer, infer the masked value.

Solve X: Replace one value inside the question with X and solve for X.

Changed from F.2:

> John has X boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. If the total inner volume of all 3 boxes is 72 cubic inches what is the value for X?

Solve Input: Determine the input value of the function, based on the known output value. **Changed from G.1** What is input to the following function, if the output is: "hELLO 9"

```
def function(string: str) -> str:
    return string.swapcase()
```

G22. *Identify Range*: Find what are possible constraint on the values. *Variable Range*: Replace one **value** with variable, and change the **query** to find the possible range of values based on the question.

Changed from F.2:

> John has 3 boxes. Each box is X inches by 6 inches by 4 inches. The walls are 1 inch thick. Suppose we want to find out the total inner volume of all the boxes. What are the possible ranges of values of variable X based on the given information?

Parameter Range: Identify what are the constraint on the input parameter, or what is the range of output parameter if input parameter is contraint to take certain value.
Changed from G.3: If all the item inside the input list is smaller than 1, what are the constraints on the output from this function below?

```
def function(xs: list):
    return [(i * x) for i, x in enumerate(xs)][1:]
```

I.3 Concept Analysis

This perturbation type encompasses questions that concentrate on the model's capabilities beyond mere problem-solving accuracy. The responses to these questions should be in natural language format. Instead of just assessing whether the model can correctly predict answers to new questions, we aim to examine the depth of knowledge the models possess and understanding of important concepts and rationales in the process of solving the original questions. Essentially, we are asking: *Does the model predict correctly because it truly understands the question?* To test this, we observe how the model behaves in different or unusual scenarios that are not typically presented in standard questions.

(i) **Question Understanding**: This dimension examines how model decompose, interpret and analyze the information inside the question.

G23. *Inherent Premise*: Identify the underlying premise of the question.
Identify Assumption: Identify one hidden commonsense assumption in the question that requires the answer to be answerable.
Changed from F.3:

> You do not need to solve the question below, just identify one important hidden assumption that is required for the question to be answerable. Kylar went to the store to buy glasses for his new apartment. One glass costs $5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?

Test Case: List different boundary test cases that is valid for the input of the question.
Changed from G.1: Provide input parameters for the test cases of the specified coding problem. These parameters should encompass boundary conditions within the scope defined by the function's requirements specification, and avoid scenarios that fall outside of these requirements.

```
def flip_case(string: str) -> str:
    """For a given string, flip lowercase characters to uppercase and uppercase to lowercase.
    """
```

G24. *Complete Missing*: Fulfill the missing information in the question by analyze how the information is structured and presented inside the question.

Missing Info: Mask or delete an important piece of **information** and ask what additional information is needed to make the question answerable.
Changed from F.2:

> John owns 3 boxes, each measuring 5 inches by 6 inches by 4 inches. Each box also had inner walls with certain non-zero thicknesses. Suppose you want to find out the total inner volume of all the boxes. What information is missing to calculate that?

Incomplete Answer: Given the question, mask partial answer of the original, the model need to infer the missing lines based on the context.
Changed from G.1: Complete the function below by predicting what is inside the masked code paragraph

```
def flip_case(string: str) -> str:
    """For a given string, flip lowercase characters to uppercase and uppercase to lowercase.
    >>> flip_case('Hello')
    'hELLO'
    """
    [masked code paragraph]
        if char.isupper():
            result += char.lower()
        else:
            result += char.upper()
    return result
```

G25. *Question Formulation*: Formulate the question based on its answer. *Question Formulation - (Math)*: Formulate a **question** to the chain of thought **gold answer**.
Changed from F.1:

Formulate a math application **question** that requires the following **mathematical structure** (calculations): 5000*(2.5/100) = $125 8000*(1.2/100) = $96 $125 > $96 $125 Math Question: Ask potential structures of math application.

Question Formulation - (Code): Formulate a concise coding requirement by looking at the function code.
Changed from G.2: Write a concise code description for the following code of its functionality no more than 1 sentence.

```
def function(a,b):
    while b:
        a, b = b, a % b
    return a
```

G26. *Add Misinformation*: Add a piece of distracting information that can mislead the answer.
Introduce Distraction: Add a Potentially Distracting **information** that will not affect the answer to the question.
Changed from F.1:

A merchant is considering a decision between the following purchase plans: jewelry with a value of $5,000, a trip to Europe costing $7,000, or electronic gadgets worth $8,000. His financial advisor predicts that the jewelry market will increase by 2.5%, the travel market will stay relatively stable with little to no change, and the electronic gadgets market will rise by 1.2% within the same month. He recently also came into an inheritance of $20,000 that he doesn't need to use right away. If the merchant's goal is to maximize profit at the end of this month by making a purchase choice, how much profit would this be?

Introduce Bias: Change the python header to describe another function requirement, and change all the examples demonstrations bias towards a specific output **Changed from G.1**

```
def uppercase(string: str) -> str:
    """For a given string, flip lowercase characters to uppercase and uppercase to lowercase.
    >>> flip_case('hello')
    'HELLO'
    """
```

(ii) **Solution Evaluation**: This dimension focuses on the problem-solving process to get to the final answer and how to optimize it.

G27. *Optimize Solution*: Assess whether the current state is optimal or if improvements are necessary.
Info Necessity: Check If there is redundant **information** given in the question, if yes, identify the redundant information.
Changed from F.2:

John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. Suppose we want to find out the total inner volume of all 3 boxes. To solve this math question, is there a way to determine the total inner volume of all 3 boxes without calculating the inner volume of one box?

Reduce Complexity: Assess whether the complexity of the current code be further reduced.
Changed from G.3: Optimize the code below to more efficiently achive the same requirement specified in the docstring

```
def derivative_polynomial(coefficients, derivative=None, index=0):
    """
    This function calculates the derivative of a polynomial using recursion.
    coefficients: List of coefficients of the polynomial.
    derivative: List to store the coefficients of the derivative. Initially None.
    index: Current index in the coefficients list.

    The base case of the recursion is when the index is equal to the length of the coefficients list.
    """
    if index > 0:
        derivative_coefficient = index * coefficients[index]
        derivative.append(derivative_coefficient)
    return derivative_polynomial(coefficients, derivative, index + 1)
```

G28. *Step Functionality*: Whether there are alternative answers that follow the constraint.
Step Necessity: Whether there are any alternative solutions **reasoning steps** without calculating an specific intermediate value.
Changed from F.2:

John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. Suppose we want to find out the total inner volume of all 3 boxes. To solve this math question, is there a way to determine the total inner volume of all 3 boxes without calculating the inner volume of one box?

Step Necessity: Provide one line of code inside the Python function, and explain the functionality of that line of code in the context of the whole solution.

Changed from G.2: Explain what is the the line below the comment functionality?

```python
def greatest_common_divisor(a: int,
    b: int) -> int:

    """ Return a greatest common
    divisor of two integers a and b
    >>> greatest_common_divisor(3,
    5)
    1
    >>> greatest_common_divisor(25,
    15)
    5
    """
    while b:
        a, b = b, a % b
    # What is the functionality of `
    abs()`
    return abs(a)
```

G29. *Theoretical Basis*: Identify the theory or principles in solving the question in general.

Theoretical Basis (Math): Identify the underlying arithmetic or algebraic rules (**toolbox**) that govern the solution to the question.

Changed from F.2:

John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. Assume you want to find out the total inner volume of all 3 boxes. Can you identify one underlying mathematical theory which is required to do that?

Theoretical Basis (Code): Request explanation on essential python concepts required to solve the question, for example, related to python objects and programming skills.

Changed from G.1: Please describe to me in simple terms, assuming I have no knowledge of programming. Your task isn't to solve the coding problem itself, but rather to identify the programming concepts in Python that would be necessary to address the problem presented below.

```python
def flip_case(string: str) -> str:
    """For a given string, flip
    lowercase characters to
    uppercase and uppercase to
    lowercase.
    >>> flip_case('Hello')
    'hELLO'
    """
```

G30. *Cost Analysis*: Analyze the computational cost regarding the solution.

Solution Efficiency: Compare two solution plans on solving the question and evaluate which one uses less computation.

Changed from F.2:

Evaluate which solution plan is more efficient in solving the question?
Plan 1: Calculate the volume of the outer dimensions for one box, calculate the volume of the material used for the walls for one box, subtract the latter from the former to find the inner volume of one box, and then multiply this by 3 for all boxes.
Plan 2: Calculate the inner dimensions of a single box by subtracting twice the thickness of the walls from each outer dimension, then find the volume of this inner space and multiply by 3 for all boxes.

Code Complexity: Analyze the time complexity and space complexity of the provided code solution.

Changed from G.1 Analyze the time and space complexity regarding to input parameter string of the following function.

```python
def flip_case(string: str) -> str:
    """For a given string, flip
    lowercase characters to
    uppercase and uppercase to
    lowercase.
    >>> flip_case('Hello')
    'hELLO'
    """
```

(iii) **Spot Error**: In this dimension, deliberate errors are introduced into the question or in a provided example answer. The purpose is to see if the LLM can identify and rectify these errors. This tests the LLM's error detection capabilities, which is crucial for reliability in practical applications.

G31. *Seek Clarification*: The question requires to be clarified first before answering.

Introduce Ambiguity: Introduce Ambiguity to the question implicitly by changing the original **information**, so that the question cannot be solved without clarification.

Changed from F.2:

> John has three 5x6x4 inch boxes. A particular side of each box have 1 inch thick walls. What does the total inner capacity of these boxes amount to?

Example Requirement: Remove the coding requirement in the docstring, instead only provide examples as a coding requirement. The provided examples will define and demonstrate the expected behavior in various scenarios. **Changed from G.1**: Begin by analyzing the function's behavior specified in the docstring to understand its pattern, and then proceed to code the function accordingly.

```
def flip_case(string: str) -> str:
    """
    function('Hello') == 'hELLO'
    function('Python 3.8') == '
   pYTHON 3.8'
    function('123abcXYZ') == '123
   ABCxyz'
    function('MixedCASE123') == '
   mIXEDcase123'
    function('ALLUPPERCASE') == '
   alluppercase'
    """
```

G32. *Conditional Analysis*: Based on different possible situations of the question, the answer should separately presented.

Discuss Separately: Introduce new **information** containing variables or conditions that require the answer to be discussed separately based on conditions or variables.

Changed from F.1:

> A merchant wants to make a choice of purchase between 2 purchase plans: jewelry worth \$5,000 or electronic gadgets worth \$8,000. His financial advisor speculates that the jewelry market will go up x% while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much profit would this be?

Incomplete requirement: Left some condition unspecified in the docstring. **Changed from G.1**:

```
def flip_case(ch: str) -> str:

    """For a given string, all the
   letters inside the string should
    be changed. flip lowercase
   characters to uppercase.
    >>> flip_case('h')
    'H'
    """
```

G33. *Conflicting Information*: Introduce a new piece of information that is conflicting with existing information. This will make the question unanswerable, so the if the LLM can spot the error without mentioning. *Introduce Contradiction*: Add a piece of contradicting **information** to the question and check if LLM can spot the problem.

Changed from F.2:

> John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. Each box is also 8 inches in width. What is the total inner volume of all 3 boxes?

Wrong Example: Include an example that is conflicting with the requirement specified in the docstring. **Changed from G.1**:

```
def flip_case(string: str) -> str:
    """"""For a given string, flip
   lowercase characters to
   uppercase and uppercase to
   lowercase.
    >>> flip_case('Hello')
    'hello'
    """
```

G34. *surface Error*: Introduce an obvious error that can be spot without reasoning. *Value Uncommon*: Change the **values** so that it seems wired or unusual by commonsense knowledge standards.

Changed from F.2:

> Can you spot anything unusual for the following question? John has 3 boxes. Each box measures 50000 miles by 60000 miles by 40000 miles. The walls of the boxes are 100 miles thick. What is the total inner volume of all 3 boxes?

Runtime Error: Introduce a piece of error that will cause a runtime error or syntax error in python. **Changed from G.1**: Debug the error in the following code

```
def flip_case(string, str) -> str:
    """For a given string, flip
   lowercase characters to
   uppercase and uppercase to
   lowercase.
    >>> flip_case('Hello')
    'hELLO'
    """
    return string.swapcase()
```

G35. *Hidden Error*: Introduce a hidden error that need logical reasoning to spot. *Value Error*: Change the **values** so that the question does not make sense.
Changed from F.4:

> Vicki is planning a pop concert at her high school. The show will be 2 minutes. She is allowing each group 2 hours to get on stage, 6 hours to perform, and then 2 hours to exit the stage. If she allows a 10-hour intermission, how many groups can perform in the concert?

Value Error: Introduce the change in the code that will cause a Value Error in python.
Changed from G.1

```python
def flip_case(string: str) -> str:
    """For a given string, flip lowercase characters to uppercase and uppercase to lowercase.
    >>> flip_case('Hello')
    'hELLO'
    """
    string = list(string.swapcase())
    return string
```

I.4 Representational Perturbation – Format Change

This dimension is inspired by metamath (Yu et al., 2023b). It involves changing the **question representation** by modifying the question encoding or specify the representation of the answer in different ways while keeping the underlying logical structure and conceptual framework of the original question intact. The objective is to verify whether the LLM can still provide correct answers even when the format or presentation of the question changes. This tests the model's ability to reason irrespective of how it's presented. It also tests models' instruction following ability where the **answer representation** must follow a certain format.

(i) **Alternative Format**:

G36. *Setting Rephrase*: Rephrase the question in another setting. *Change Setting*: Rephrase by changing the application setting and values inside the information, while keeping the core mathematical structure intact.
Changed from F.2:

> Maria has 4 cuboids. Each cuboid is 7 feet by 9 feet by 6 feet. The walls are 2 feet thick. What is the total volume of all the cuboids?

Realworld Usecase: Frame the requirement in docstring into a problem that will happen in a realworld scenario. **Changed from G.1**:

```python
def switch_text_case(text: str) -> str:
    """
    Imagine you're working on a document and you've mistaken the case in the text you write. You wrote all the lower case letters in uppercase and vice versa, suppose you want to correct all of them using python.
    """
```

G37. *Change Sequence*: Change the order of the information and names of the variables that is originally presented in the question.
Change Sequence: Change the sequence of information given in the question without affecting the solvability of the question.
Changed from F.2:

> The walls of John's boxes are 1 inch thick. Each of these boxes measures 5 inches by 6 inches by 4 inches. John has 3 boxes. What is the total inner volume of all 3 boxes?

Parameter Sequence: Change the sequence of the input parameter and change the input parameter names.
Changed from G.2

```python
def munchee_bunchee(xray: int, yoyo: int) -> int:

    """ Return a common divisor that is the largest of two integers xray and yoyo
    >>> munchee_bunchee(3, 5)
    1
    >>> munchee_bunchee(25, 15)
    5
    """
```

G38. *Close Format*: Rewrite the sentence as a closed-format question that evaluates the correctness of possible answers.
True False: Evaluate a potentially misleading answer and check the correctness of the answer.
Changed from F.1:

> A merchant wants to make a choice of

purchase between 2 purchase plans: jewelry worth \$5,000 or electronic gadgets worth \$8,000. His financial advisor speculates that the jewelry market will go up 2.5% while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much profit would this be? Evaluate the correctness of this answer with respect to the above question: \$96.

True False: Check if a given code answer can solve the requirement in docstring. **Changed from G.2**: Evaluate whether the solution below is the correct solution for the coding question, True or False?

```
Function:

def greatest_common_divisor(a: int,
    b: int) -> int:

    """ Return a greatest common
    divisor of two integers a and b
    >>> greatest_common_divisor(3,
    5)
    1
    >>> greatest_common_divisor(25,
    15)
    5
    """


Solution:

    while a:
        a, b = a % b, a
    return b
```

G39. *Data Restructuring*: Change the layout, organization of the data presented in the question.
Value Structuring: Arrange the variables inside the question in a tabular format.
Changed from F.2:

```
| Variable | Value |
|----------|-------|
| a        | 3     |
| b        | 5     |
| c        | 6     |
| d        | 4     |
| e        | 1     |
```

John has 'a' boxes. Each box is 'b' inches by 'c' inches by 'd' inches in dimensions. The walls are 'e' inch thick. What is the total inner volume of all the 'a' boxes?
Complex Docstring: Elaborate the documentation string by exhaustively detailing more conditional pathway within the code.
Changed from G.1:

```
def function(string: str = None) ->
    str:
    """
    For any specified sequence of
    alphabetical characters,
    interspersed with spaces,
    numerical digits, and various
    symbols, implement a
    sophisticated transformation
    algorithm designed to
    selectively convert  each
    alphabetical character from its
    current case representation,
    either lowercase or uppercase,
    to its diametrically opposite
    case representation. This
    algorithm ensures that every
    character initially presented in
     lowercase is meticulously
    transmuted to uppercase, and
    conversely, every character
    originally in uppercase is
    converted to lowercase, while
    meticulously preserving the
    integrity and original
    positioning of spaces, numerical
     digits, and any other non-
    alphabetical symbols, leaving
    these elements unaltered within
    the sequence.
    >>> function('Hello')
    'hELLO'
    """
```

G40. *Identical Problem*: Check if the two question or code are identical in describing or solving the same problem.
Identical Question: If two questions requires exactly the same framework or thinking procedure to solve.
Changed from F.1:

Question 1: A merchant wants to make

a choice of purchase between 2 purchase plans: jewelry worth $5,000 or electronic gadgets worth $8,000. His financial advisor speculates that the jewelry market will go up 2.5% while the electronic gadgets market will rise 1.2% within the same month. If the merchant is looking to maximize profit at the end of this month by making a choice, how much profit would this be? Question 2: An investor is unsure of which investment to make: gold valued at $10,000 or stocks valued at $15,000. His financial consultant predicts that the gold market will inflate by 3.5% while the stock market will increase by 2.2% over the next quarter. If the investor wants to achieve the highest return on his investment at the end of this quarter, how much would his initial investment be? Does Question 1 and Question 2 require identical steps to answer?

Identical Code: Are the two solutions to the question identical in terms of their functionality?
Changed from G.3 Is function_1 and function_2 identical in terms of its functionality?

```
Code 1:
def function(xs: list):
    return [(i * x) for i, x in
   enumerate(xs)][1:]
Code 2:
def function(xs: list):
    derivative = [i * xs[i] for i in
     range(1, len(xs))]
```

(ii) **Answer Constraint**: This dimension add a constraint on the solution so that it should conduct reasoning under the constraint

G41. *Reasoning Format*: The format for the final answer should be converted in a certain way.
Binary Coded: Answer the final question in base-n.
Changed from F.2:

Answer the following question with only base-2 coded values. Question: John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes?

No Keyword: The solution should not use a specific python keyword. For example, "for" or "while". **Changed from G.2**: Answer the coding function below without using python keywords: "while", "for" in the solution

```
def greatest_common_divisor(a: int,
   b: int) -> int:

    """ Return a greatest common
   divisor of two integers a and b
    >>> greatest_common_divisor(3,
   5)
    1
    >>> greatest_common_divisor(25,
   15)
    5
    """
```

G42. *Reasoning Style*: The reasoning steps should be performed in a certain style.
X Language (Math): Give the answer in certain language from Spanish, Chinese, Bengali, English, French
Changed from F.2: Answer the following question with only Chinese language, because I do not understand English.

Question: John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes?

X Language (Code): Give the code answer in another coding language.
Changed from G.1: Answer the coding question below;

```
func flipCase(str string) string {
// flipCase takes a string and flips
     the case of each character:
   lowercase to uppercase and
   uppercase to lowercase.

}
```

G43. *Alternative Answer*: Find the alternative solutions to existing solution. *Alternative Answer (Math)* : Give an alternative solution that is different from the standard **reasoning steps**, but arrives at the same correct final answer.
Changed from F.2:

Give an different step-by-step solution

to calculate the answer to the following question. Make sure the solution is different from the solution below. Question: John has 3 boxes. Each box is 5 inches by 6 inches by 4 inches. The walls are 1 inch thick. What is the total inner volume of all 3 boxes? Solution: The walls subtract a (1 + 1) = 2 inches from each dimension. So, each box has a reduced width of (5 - 2) = 3 inches, reduced length of (6 - 2) = 4 inches and reduced height of (4 - 2) = 2 inches. So the inner volume of each box is 3 * 4 * 2 = 24 cubic inches. The total inner volume of 3 boxes are 3 * 24 = 72 cubic inches. Alternative Step by Step Solution:

Alternative Answer (Code): Find an alternative solution to existing coding solution. **Changed from G.1**:

```
Find a different solution other than
    :
def flip_case(string: str) -> str:

    return string.swapcase()
```

G44. *New Rule:* Integrate a new rule into the original question that requires the solution follow the new rule. This type tests the model's ability to adapt to new ruels and knowledge and use it inside the solution.

Define Rules:Define a new mathematical rule that will change how the **toolbox** (commonsense knowledge) is applied during calculation.

Changed from F.2:

In a parallel universe, John has 3 boxes. Each box has peculiar dimensions: 5 quarks by 6 quarks by 4 quarks with walls that are 1 quark thick. In this universe, the total inner volume of a box is calculated by using the Illusory Volume operation, represented as IV. The IV operation is defined as: (length * width * height) - (number_of_walls * thickness_of_each_wall). What is the total inner volume of all 3 boxes?

Simple Name: The generated code should only have variables names in a certain format.

Changed from G.1: Answer the coding question below and only use 6 letter word for each variable names inside the solution

```
def flip_case(string: str) -> str:
    """For a given string, flip
    lowercase characters to
    uppercase and uppercase to
    lowercase.
    >>> flip_case('Hello')
    'hELLO'
    """
```

Overall, these dimensions in the "Format Change" and "Format Constraint" Domain are designed to challenge the LLMs in ways that reveal their limitations and strengths in maintaining accuracy and functionality under modified or challenging conditions.

J Evaluation Details

We used separate prompt templates for open source and close source models because close source models sometimes give the final answer directly and omit reasoning steps even if prompted with "Let's think step by step". To ensure the model performs Chain of Thought Reasoning, we use the following prompt template for GPT-4, GPT-3.5, and Gemini to generate the answer:

> Solve the question step by step before giving the final answer. Do not directly give the final answer.
> Question
> Reasoning Step:

For Metamath, CodeLlama and Llama2-Chat, we use the following:

> Below is an instruction that describes a task. Write a response that appropriately completes the request.
> ### Instruction: Question
> ### Response: Let's think step by step.

The temperature of GPT-4 and GPT-3.5 was set to 0.7 (the default setting in OpenAI playground) for *Concept Analysis question* and 0.1 for *Logic Alteration questions and Format Change questions*. Similarly, the temperature for Llama, ChatGPT, and Gemini were set to 0.8 and 0.1 for *Concept Analysis* and *Logic Alteration questions and Format Change questions*, respectively.

K Experiment Details

Prompt to incorporate original answer:

> Given the original question and its answer,

> Solve the question that is a perturbed variant of the original question. Solve the #perturbed question# step by step before giving the final answer. Do not directly give the final answer.
> #Original Question#: original question
> #Original Answer#: original answer
> #Perturbed Question#: perturbed question

Self Consistency Prompting: We randomly picked one question answer pair in the same category from our maths dataset MORE and prepend it to the front of the perturbed question as a one shot demonstration. Then we use the below prompt template for Self Consistency prompting. We sample the generation three times and get the final answer by majority voting. In case of tie, we randomly pick an answer.

> Given the oneshot demonstration of a question and its final answer, Solve the #question# step by step before giving the final answer. Do not directly give the final answer.
>
> #Demonstration Question#: demonstration question
> #Demonstration Final Answer#: demonstration answer
>
> #Question#: question Reasoning Step: [Reasoning Steps] Final answer: [Final answer]

Program of Thought Prompting: We use the following prompt template for the program of thought experiments:

> Instruction: You are an experienced professional skilled in using python programs to solve math related problems. Solve the question below using python programs, You will only write code blocks.
>
> Problem: Question

L Detailed Results

The detailed results across the perturbation categories for all the models are illustrated in Tables 8 and 9.

M Inclusivity of skill set

Dependence between Perturbation Types. In our ontology, some specialized perturbation types, which we refer to as *Enhanced types*, require skill in solving some other primary perturbation types which we call *Primary types*. For instance, consider the process of solving perturbed questions generated as outlined in **G20**. The initial step for the model involves identifying the value of an unknown variable from its answer. Subsequently, the model calculates how this value alters the final answer. This initial step demands skills similar to those described in **G21**. Consequently, we anticipate that enhanced perturbation types will be challenging to answer. Following Table 10, across all models, *primary types* exhibit higher overall performance as compared to *enhanced types*. Furthermore, it is observed that open-source models do not experience as significant a performance drop as closed-source models when handling enhanced types. This can be attributed to the fact that open-source models already demonstrate near-zero performance in answering primary type questions. Therefore, their inability to answer enhanced questions does not result in a notable decrease in performance.

Performance across Question Difficulty. In our experimentation with various LLMs, we consistently employed the Chain of Thought (CoT) methodology to derive the ultimate answer. This prompts a natural inquiry: *Does the performance of LLMs exhibit any correlation with the number of steps needed to arrive at the final answer?* Surprisingly, in our extensive experiments (as illustrated in Figure 4), we did not discern any definitive correlation or discernible trend. Instead, performance appears to diminish based on the inherent difficulty of the original question in GSM8K. Put differently, if an LLM fails to provide an accurate response to the initial question, its performance similarly falters when confronted with perturbed questions.

Dimension	Category	GPT-4	GPT-3.5	Gemini	Metamath	Llama2-Chat
Original		5	4	4	4	3
Question Simplification	Remove Constraint	5	5	5	5	4
	Partial Solution	5	3	3	3	2
	Solution Plan	5	4	5	2	4
	Detail Expansion	5	3	5	4	0
Reasoning Adjustment	Add Restriction	3	1	2	1	0
	Subsequent Question	4	1	3	0	0
	Concurrent Question	4	2	4	1	1
	Change Question	5	2	3	1	1
	Info Recombination	4	1	3	0	1
	Domain Knowledge	4	2	4	2	2
	Complex Reality	3	0	1	1	0
	General Solution	5	2	0	0	0
Computation Adjustment	Computation Demand	4	2	4	1	0
	Change Value	1	1	1	1	0
	Change Operation	5	3	4	1	2
Symbolic Manipulation	Symbolic Response	4	3	3	0	0
	Value Relationship	3	1	2	0	0
	Variable Group	3	1	2	0	0
	Backward Reasoning	2	1	1	1	0
	WhatIf	3	1	0	1	0
	Solve Value	5	2	4	1	0
	Identify Range	1	0	1	1	2
Question Understanding	Inherent Premise	5	2	2	0	1
	Complete Missing	5	4	5	2	4
	Question Formulation	3	1	2	1	1
	Add Misinformation	4	4	3	3	1
Solution Evaluation	Optimize Solution	3	3	2	2	4
	Step Functionality	1	0	0	0	2
	Theoretical Basis	4	4	1	2	4
	Cost Analysis	5	2	1	1	2
Error Debugging	Seek Clarification	1	2	1	0	0
	Conditional Analysis	3	0	2	0	0
	Conflicting Information	2	0	1	0	0
	Surface Error	4	1	0	1	1
	Hidden Error	2	0	0	0	0
Alternative Format	Setting Rephrase	4	3	2	4	1
	Change Sequence	5	2	3	3	0
	Close Format	4	2	4	0	0
	Data Restructuring	5	0	2	0	0
Pairwise Comparison	Identical Problem	3	2	1	4	3
Answer Constraint	Reasoning Format	4	0	2	0	0
	Reasoning Style	4	0	2	0	0
	Alternative Answer	2	0	0	2	0
	New Rule	3	1	2	2	1

Table 8: Number of examples correctly predicted by each model on MORE. There are a total of 5 questions for each category except "Change Value", which only has 2 questions.

Dimension	Category	GPT-4	ChatGPT	Gemini	CodeLlama	Llama2-Chat
Original		4	4	4	3	3
Question Simplification	Remove Constraint	4	4	4	4	2
	Partial Solution	5	3	4	5	2
	Solution Plan	5	4	4	3	2
	Detail Expansion	4	3	4	4	3
Reasoning Adjustment	Add Restriction	0	0	2	2	0
	Subsequent Question	2	2	1	1	3
	Concurrent Question	3	1	0	2	0
	Change Question	2	2	2	2	1
	Info Recombination	2	1	1	1	0
	Domain Knowledge	3	4	3	4	0
	Complex Reality	3	2	3	3	0
	General Solution	0	2	1	1	1
Computation Adjustment	Computation Demand	1	1	2	2	1
	Change Value	2	1	1	2	1
	Change Operation	4	4	5	2	3
Symbolic Manipulation	Symbolic Response	4	3	1	2	1
	Value Relationship	1	1	1	1	0
	Variable Group	3	1	1	0	1
	Backward Reasoning	2	2	3	1	0
	WhatIf	3	1	0	0	0
	Solve Value	1	1	0	0	0
	Identify Range	3	1	2	0	2
Question Understanding	Inherent Premise	2	3	2	1	1
	Complete Missing	3	1	3	1	2
	Question Formulation	4	4	4	2	3
	Add Misinformation	4	4	4	3	4
Solution Evaluation	Optimize Solution	2	2	3	2	2
	Step Functionality	5	5	4	2	2
	Theoretical Basis	5	3	4	0	4
	Cost Analysis	4	5	4	3	2
Error Debugging	Seek Clarification	2	2	2	3	0
	Conditional Analysis	1	1	1	0	0
	Conflicting Information	1	0	0	0	0
	Surface Error	4	4	4	2	1
	Hidden Error	3	3	4	2	1
Alternative Format	Setting Rephrase	3	2	2	1	3
	Change Sequence	4	3	2	3	1
	Close Format	3	1	2	2	0
	Data Restructuring	3	4	3	2	1
Pairwise Comparison	Identical Problem	2	2	2	0	2
Answer Constraint	Reasoning Format	2	2	2	2	1
	Reasoning Style	3	2	2	3	1
	Alternative Answer	3	3	2	2	0
	New Rule	3	2	1	1	2

Table 9: Number of examples correctly predicted by each model on CORE. There are a total of 5 questions for each category.

Domain	Enhanced Type	Primary Type	GPT-4	GPT-3.5	Gemini	Metamath	Llama2-Chat
Logic Alteration	Backward Reasoning	Solve Value	20	0	20	-20	0
	Value Relationship	Symbolic Response	20	40	40	0	0
	Variable Group	Symbolic Response	20	40	20	0	0
	Identify Range	Symbolic Response	60	60	40	-20	-40
	What If	Solve Value	40	20	80	0	0
	Solution Plan	Detail Expansion	0	20	0	-40	80
Concept Analysis	Seek Clarification	Conditional Analysis	40	-40	20	0	0
	Optimize Solution	Cost Analysis	20	20	20	20	-60
	Conflicting Information	Complete Missing	20	60	20	40	80
	Optimize Solution	Step Functionality	20	0	0	40	-40
	Hidden Error	Step Functionality	0	0	20	0	0
Average			25.45	16.36	30.91	0	12.73

Table 10: Performance drop in Enhanced vs. Primary Type questions on MORE. The value equals (accuracy of Primary - accuracy of Enhanced), so positive entries indicate higher performance for Primary Type questions.

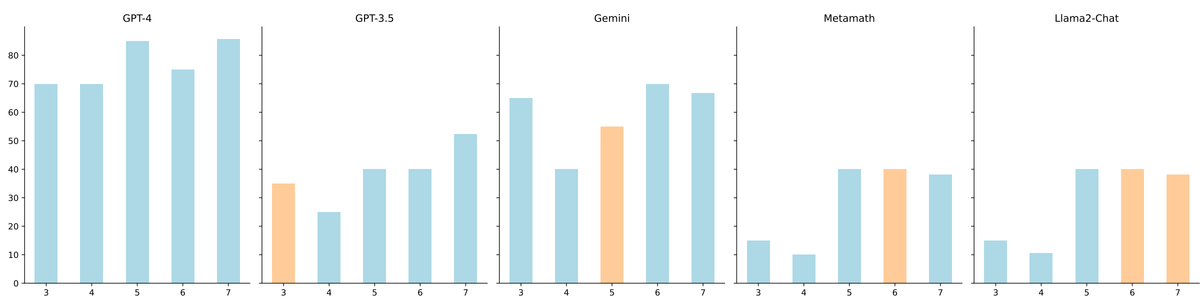

Figure 4: Model performance for each question. The blue color indicates the model predicted correctly for the original question, and orange means the opposite. '3', '4', '5', '7', '8' stands for the number of steps in the gold answer for the perturbed question.