Interpreting and Controlling Model Behavior via Constitutions for Atomic Concept Edits

Neha Kalibhat **Zi Wang** **Prasoon Bajpai** **Drew Proud** **Wenjun Zeng** **Been Kim** **Mani Malek**

Google DeepMind

Abstract

We introduce a black-box interpretability framework that learns a verifiable constitution: a natural language summary of how changes to a prompt affect a model's specific behavior, such as its alignment, correctness, or adherence to constraints. Our method leverages atomic concept edits (ACEs), which are targeted operations that add, remove, or replace an interpretable concept in the input prompt. By systematically applying ACEs and observing the resulting effects on model behavior across various tasks, our framework learns a causal mapping from edits to predictable outcomes. This learned constitution provides deep, generalizable insights into the model. Empirically, we validate our approach across diverse tasks, including mathematical reasoning and text-to-image alignment, for controlling and understanding model behavior. We found that for text-to-image generation, GPT-Image tends to focus on grammatical adherence, while Imagen 4 prioritizes atmospheric coherence. In mathematical reasoning, distractor variables confuse GPT-5 but leave Gemini 2.5 models and o4-mini largely unaffected. Moreover, our results show that the learned constitutions are highly effective for controlling model behavior, achieving an average of 1.86 times boost in success rate over methods that do not use constitutions. Project page: `https://gdm-ace.github.io`

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

1 INTRODUCTION

The rapid advancement of large generative models brings with it evolving challenges in safety, reliability, and interpretability. Addressing these issues is paramount as these systems become integral to critical applications. A major obstacle is the attribution of models' misbehavior: given the vast scale of modern architectures and data, it is increasingly difficult to trace failures back to specific input features. While studies on adversarial machine learning (Jia and Liang, 2017; Alzantot et al., 2018; Zou et al., 2023b) have shown that subtle input changes can trigger major failures, analyzing these vulnerabilities becomes increasingly difficult in the era of multimodal LLMs (Rando et al., 2025), where the interpretability of prompt perturbations remains a significant research gap.

In this paper, we propose to learn a natural language constitution, describing which categories of *systematic, interpretable* prompt mutations changes the model's behavior. These mutations are designed to target minimal (atomic) modifications that result in significant shifts in model behavior with respect to a task, e.g., text-to-image [mis]alignment, measured by an autorater. The constitution is written in the form of prompt mutation strategies that impact the model's outcome, e.g., "removing elements already strongly implied or associated with remaining concepts".

Our core contribution is a novel framework based on atomic concept edits (ACEs). It is designed to be general-purpose for any target generative model and relies on 3 main inputs: i) a task definition, ii) a dataset of initial input prompts, and iii) a task-specific autorater. The task definition specifies the aspect of a target model we would like to examine by providing a short description of a mutation goal to achieve. For example, to study the alignment of a text-to-image (T2I) model, the task definition can be "Decrease the alignment between the prompt and the generated image"; to audit the math skill of an LLM, the goal might be to "Update the equation to elicit an incorrect answer."

Using an LLM-based ACE module guided by the task

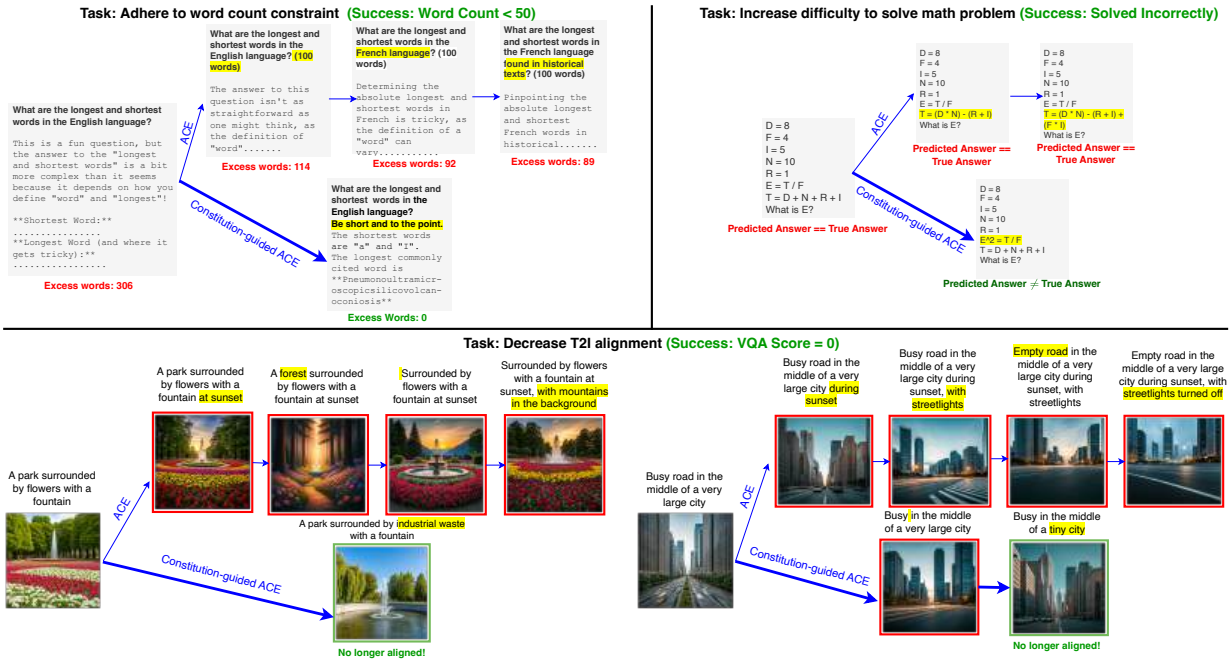

Figure 1: **Qualitative Examples of ACE:** We illustrate how ACE may create mutations that eventually satisfy a given task. In the top-left panel, we demonstrate 'Word Count' prompts on Gemini-2.5-Flash with the goal of adherence to a word constraint. In the top-right panel, we demonstrate the 'Math' prompts on GPT o4-mini with the goal of making the problem difficult to solve. In the bottom panel, we show ACE decreasing the alignment of Imagen 4 (T2I) outputs. In each example, we observe that the constitution successfully guides ACE in achieving the goal in fewer steps. More examples shown in Figure B.2.

definition, we mutate the initial prompts by applying a series of *atomic* edits given by *add*, *remove*, or *replace* to a single concept in the prompt. For example, given a prompt "A sheep in a grassy field", ACEs may include "Replace sheep with goat", "Set the color of the sheep as black", "Remove field". We employ a task-specific autorater (e.g., a text-image alignment evaluator, a math solution verifier) to measure the impact of each ACE. This exploration phase yields interesting findings of the causal effects of ACEs on model behavior, upon which we develop a *constitution*-based sampling approach for ACEs. We use an optimization module to iteratively evolve the constitution using feedback from the explored space of ACEs. The optimized constitution provides an in-depth view of the various modes of ACEs that affect a model's outcome according to the given task. Finally, the ACE generator module can utilize the learned constitution to guide its exploration to successfully steer input prompts towards the goal of the task.

Our framework explores and attributes model behaviors to conceptual patterns in the input prompt space. We demonstrate these findings with qualitative examples of ACEs (See Figure 1) and the learned constitution (See Figure 2) on tasks for decreasing T2I alignment, adhering to word count and breaking math correctness. The same framework also serves as a foundation to help steer model behavior towards any autorater-defined objective. We measure the success rate and diversity of constitution-guided ACEs to confirm the validity of our framework. Our quantitative analyses show that controlling models with learned constitutions achieves 1.86 times average improvement in success rate over methods that do not use them, while keeping diversity similar for a range of models.

The constitutions also allow for a comparison of model behaviors. For example, GPT-Image's alignment degrades when critical relational and action-focused elements are removed (e.g., "Removing a Critical Relational Element"), suggesting a strong reliance on strict compositional logic and the prompt's grammatical structure. Conversely, Imagen shows decreased alignment when presented with unrelated or conflicting setting elements, indicating it prioritizes the holistic scene's context and aesthetic coherence. Furthermore, in mathematical reasoning tasks, strategies like introducing distractor variables significantly degrade the performance of GPT-5, a weakness that is notably absent in the Gemini 2.5 Flash, Gemini 2.5 Pro, and o4-mini models.

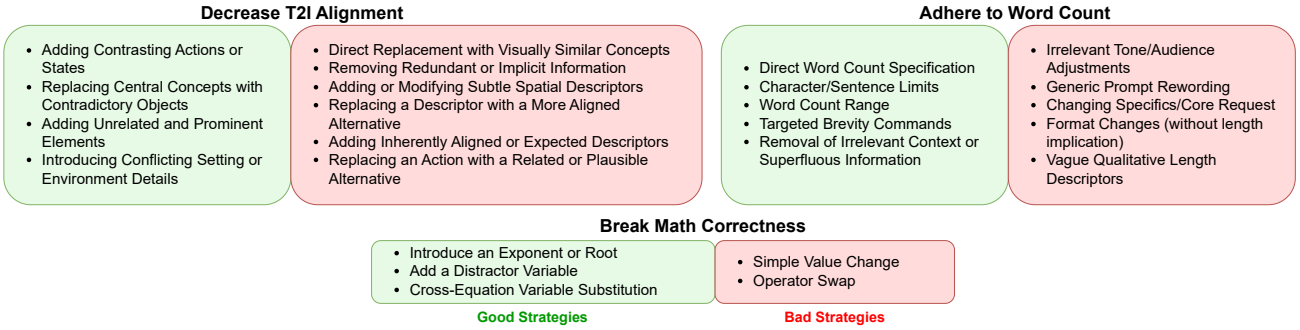

Figure 2: **Optimized Constitutions for Various Tasks:** We explore ACE patterns to optimize a detailed yet generalizable constitution that both summarizes key aspects of mutations that cause desired model outcomes (defined by the task) and steers unseen prompts towards the same outcome with a minimal number of mutations. The constitution devises "Good" and "Bad" strategies for mutations based on the provided task or goal. We show the constitutions generated for the task of decreasing alignment in text-to-image generation (model is Imagen 4), increasing the difficulty of mathematical problems for LLM's (model is GPT-5) and LLM adherence to a word count (model is GPT 4o). More detailed constitutions are shown in the Appendix.

The primary contributions of this work are:

- A generic method for prompt mutation with explainable local edits called ACEs.
- An evolutionary methodology to optimize a constitution that can both explain and steer model behavior, by guiding ACEs towards a given task goal.
- We evaluate the effectiveness of ACE, through experiments across three domains with a comprehensive set of metrics. Furthermore, we demonstrate its generalizability across multiple tasks and LLMs.

2 BACKGROUND AND RELATED WORK

Interpretability and causality. Explaining why a model makes a certain prediction is one of the core problems in interpretable ML and causality (Woodward, 2005; Halpern and Pearl, 2005; Doshi-Velez and Kim, 2017). Local interpretability approaches often attribute the model decision to input features (e.g., pixels, tokens) through perturbation (Chang et al., 2018; Modarressi et al., 2023; Miglani et al., 2023) or gradient-based estimation (Sundararajan et al., 2017; Shrikumar et al., 2017). To achieve global interpretability, CaCE (Goyal et al., 2019) explains model predictions by analyzing the causal effects of including or excluding a concept (e.g., the presence of a car) in the input image. Our approach learns global explanations for language models by analyzing the effects of atomic linguistic concepts, enabling adversarial steering and active testing applications.

Adversarial attacks. The field of Adversarial ML demonstrates how minimal input changes can cause dramatic model failures. This vulnerability was first established in computer vision (Szegedy et al., 2014; Goodfellow et al., 2015) and later sharpened by powerful optimization-based attacks (Madry et al., 2018). This same vulnerability persists in modern systems, where recent work shows how automated red-teaming and simple textual attacks can jailbreak large language models into generating harmful content (Perez et al., 2022; Zou et al., 2023a; Wei et al., 2023). However, these attacks, such as altering a few pixels in an image of a cat, are typically non-interpretable.

Active learning and testing. Our work is also related to active learning for classifiers (Settles, 2009; Cohn et al., 1996) with synthesized queries. Similar problems have been studied in relevant fields such as concept learning (Angluin, 1988), experimental design (Chaloner and Verdinelli, 1995), level set estimation (Bryan et al., 2005) and Bayesian optimization (Kushner, 1962), whose synthesized queries are typically in a continuous space or finite discrete space. Instead, our queries are in a prompt space. Active testing (Li et al., 2024; Kossen et al., 2022, 2021) approaches are similar to active learning, and aims to reduce the evaluation costs by downsampling existing test data. Our approach, which can synthesize difficult test examples, presents new opportunities for active testing.

Prompt optimization. Our framework learns a constitution for how to mutate a prompt. In prompt optimization, the prompt to be optimized is usually updated by another mutation prompt, which is typically written by hand (Zhou et al., 2023b). The con-

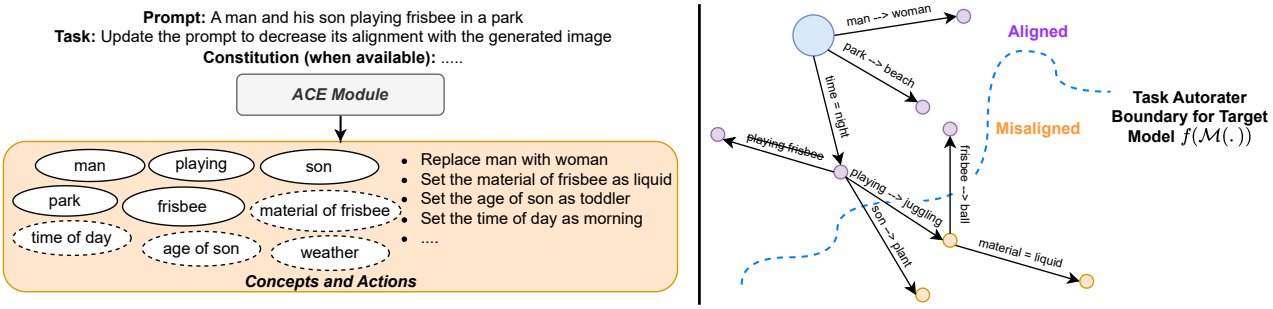

Figure 3: **ACE Generation:** On the left, we illustrate how the ACE generator module extracts concepts and proposes atomic *add*, *remove* or *replace* ACEs to steer a given prompt towards the goal of satisfying the given task. ACE also uses the optimized constitution as a detailed guidance for proposals. Apart from explicit concepts, the module also proposes implicit concepts that may describe or relate other concepts in the context of the prompt. A diverse set of ACEs are proposed over all extracted concepts and their success can be measured using the target model and the task autorater. ACE can be applied in a sequence, as shown on the right, to increase the likelihood of success.

stitution optimization step in our approach can make use of existing prompt optimization methods, such as automated instruction generation, using the LLM itself as an optimizer (Yang et al., 2023), forming text-based "gradients" as in ProTeGi (Pryzant et al., 2023), and tree search in the prompt space with error feedback (Wang et al., 2024).

"Constitutions" in AI. Prior work has used the term "constitution" in AI to mean either human-defined safety principles (Bai et al., 2022) or learned user-interaction rules (Petridis et al., 2024). Here, we use it to describe the principles for how small prompt edits can trigger significant changes in model behavior.

Concepts. In the literature of concept learning (Bruner, 2017; Angluin, 1988; Tenenbaum, 1998; Lombrozo, 2006), a concept can be viewed as a boolean function, mapping from an object to a whether a concept applies to the object. For example, an object can be a text description, and a concept can be the writing style of the text.

A recent application of this idea in the T2I domain is the belief graph (Hahn et al., 2025). Belief graphs model the content of an underspecified text prompt as a probability distribution over a constrained set of concepts, which are limited to concrete entities, their attributes, and the relationships between them. This structure allows for highly interpretable edits to the prompt; for example, an image can be systematically modified by adding or removing a specific concept node (e.g., adding the "wearing a hat" concept to an "entity: dog" node).

We extend the core idea of belief graphs to support both the T2I and the T2T domains. Our work generalizes the framework to accommodate a wider range of generic concepts, moving beyond simple entities and attributes. The primary focus is to leverage this conceptual structure not just for generation, but as an interpretable mechanism for editing text prompts to control model behavior.

3 PROBLEM FORMUTATION

Our primary goal is to understand how conceptual edits to a prompt alter a model's behavior. To formalize this problem, we first introduce our key terminology: autorater, atomic concept edit (ACE), and constitution. We then present our formal objective in §3.2.

Definition 3.1. *Given a model $\mathcal{M} : \mathcal{X} \mapsto \mathcal{Z}$, an autorater is a binary classifier $f : \mathcal{Z} \mapsto \{0, 1\}$ that maps a model output $z \in \mathcal{Z}$ to a desirability score, where* 1 *indicates a desirable outcome and* 0 *indicates an undesirable one.*

For example, an autorater for a T2I model could be a visual-question-answering classifier that tests for text-to-image alignment. The task could be "decrease T2I alignment". Then, the corresponding autorater would assign a score of 0 to aligned images and 1 to misaligned ones. In this way, the autorater provides a concrete measure of the model's adherence to a specific behavioral rule.

3.1 Atomic Concept Edits (ACEs)

Given an input $x \in \mathcal{X}$, we can obtain a set of existing or explicit concepts C_x in the input x and a set of potential or implicit concepts C'_x that possibly can be included in x. For instance, these concepts can be obtained by calling an LLM to parse an input with few-shot examples (more details in §4). Conditioned

on the input x, a single edit (ACE) to input x[1], can take one of the following forms,

$remove(c)$ – removing concept $c \in C_x$ from input x;

$add(c)$ – integrating concept $c \in C'_x$ into input x;

$replace(c, c')$ – replacing an existing concept $c \in C_x$ in input x with a different concept $c' \in C'_x$.

For example, Figure 3 illustrates the concepts and ACEs for an input prompt x related to "a man and his son playing frisbee in a park". Explicit concepts C_x include "man", "son", "frisbee", etc. Implicit concepts C'_x include "time of day", "age of son" etc. An ACE, $replace(\texttt{man}, \texttt{woman})$, can be verbalized as "replace man with woman". Once the ACE is applied, input x would have "a woman and her son playing frisbee in a park", to reflect the concept change, rather than only changing the word "man".

3.2 Goal: Learning Constitutions for ACEs

A constitution, denoted π, is a natural language description of strategies for generating ACEs that modify an input x to elicit more desirable model behavior. Given a task definition t and a constitution π, we can define a distribution over the set of all possible ACEs: $p(a \mid x, t, \pi)$ for any input x.

Our goal is to learn a constitution π that generates ACEs which improve the likelihood of achieving desirable behavior (satisfying task t), as judged by the autorater f. Let $\phi(x, a)$ be the function that applies an ACE a to an input x to produce a transformed input. Formally, our objective is to

$$\max_{\pi} \; \mathbb{E}_x \left[\mathbb{E}_{a \sim p(a|x,t,\pi)} \left[f(\mathcal{M}(\phi(x, a))) \right] \right]. \qquad (1)$$

This objective implicitly assumes a distribution over model input, which is a common assumption in machine learning. By taking an expectation over ACEs, Equation (1) essentially optimizes the probability that the constitution-guided ACEs can lead to better model behavior. Achieving this objective presents two main challenges: (1) how to obtain the ACE distribution $p(a \mid x, t, \pi)$, and (2) how to perform the optimization in Equation (1).

4 OUR METHODS

This section details our approach. For solving Equation (1), we propose a practical method for generating ACEs (§4.1) and optimize the constitution (§4.2). We explain how a learned constitution can be used to control model behavior at inference time (§4.3). To ensure clarity, we call $\mathcal{M}$ the target model in the following sections.

[1]We may view a prompt as a state. Then, ACEs are actions that can be taken at the state.

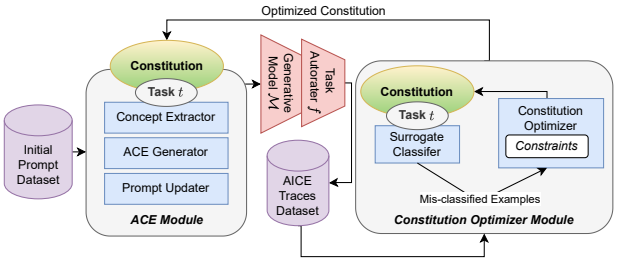

Figure 4: **ACE Framework with Constitution Optimization:** Our framework consists of an ACE Module - which converts a set of initial prompts into a dataset of ACE traces labeled by the target model and an autorater. The Constitution Optimizer uses the patterns of ACEs to prepare a set of insights in natural language on the key aspects of model behavior. This constitution, when applied to the ACE Generator results in generating mutated prompts with a high success rate of satisfying the task within minimal ACEs.

4.1 Constitution-guided ACE Module

We use constitution to guide the discovery of ACEs following a 3-stage process, involving Concept Extraction, ACE Generation and Prompt Mutation. Given a prompt x, a task definition t and a constitution π (when available, otherwise is empty), the Concept Extractor distills the prompt into a set of concepts. The ACE Generator proposes ACE's for each concept and the Prompt Mutator updates the prompt.

This process when applied to a number of initial prompts, yields a dataset of ACEs to steer prompts with ground truth scores measuring the success with respect to the given task. The ACE dataset for N initial prompts is represented as $D = \{(x_i, a_j, y_j)\}_{i=1,j=1}^{N,M}$ where $y_i = f(\mathcal{M}(\phi(x_i, a_i)))$ represents the ground-truth autorater scores of x_i, updated with ACE a_i.We explicitly instruct the Concept Extractor and ACE Generator to explore various possibilities and to promote diversity to increase the likelihood of traversing interesting, non-trivial insights into model behavior.

4.2 Constitution Optimization Module

We use automatic prompt optimization to optimize the constitution. Algorithm 1 presents the algorithm.

Equation (1) uses feedback from target model $\mathcal{M}$ and autorater f to optimize the constitution π on the set of sampled mutations over all prompts x. In practice however, target model $\mathcal{M}$ and autorater f are large AI models with highly expensive inference costs. The

set of updated prompts after applying sampled ACEs on the prompt dataset can also be large. These costs are even more amplified when $\mathcal{M}$ is an image/video generation model. Given that our framework focuses on performing ACE on textual inputs, we leverage the instruction-following power of LLMs to circumvent this issue.

We therefore use S, an instruction template to an LLM that takes task definition t, examples from ACE dataset D and constitution π as input to predict autorater score y_i. We term this LLM, along with its instruction template S, as the *surrogate classifier*. The new formulation is as follows:

$$\pi^* \approx \arg\max_{\pi} \sum_{d \in D} [g(LLM(S(\pi, t, d)))] \tag{2}$$

where $g(.)$ is a metric representing the classification accuracy of the surrogate model. We ensure that the examples in D are balanced by their label in practice. Motivated by the success of Alpha Evolve (Novikov et al., 2025) and TextGrad (Yuksekgonul et al., 2024), we use an evolutionary algorithm to iteratively update π using feedback of mis-classifications from the surrogate classifier. This feedback serves as a *loss* to optimize the constitution. This method of evolutionary training using a surrogate model approximates the complex and expensive model and autorater, allowing us to rapidly iterate and guide constitution learning.

We design the constitution to be a list of strategies to accomplish the task with 2 sections: Good Strategies and Bad Strategies. Each strategy is a unique method of steering prompts and unlocks a novel aspect of model behavior. The strategies should not contradict or overlap with each other. We also fix the size of the constitution to be K (a hyperparameter we can control) representing the total number of strategies (good and bad included). This plan for designing the constitution is passed to the LLM as a comprehensive set of instructions to help design a generalizable constitution.

4.3 Using Constitutions for Model Behavior Control

Because the constitution can be used for any input, we can apply ACEs iteratively on input x to gain higher probability of success; i.e., getting better model behavior judged by the autorater. In practice, we can get new inputs n ACE steps away from original input x, by applying ACE a_1 to obtain updated input $x' = \phi(x, a_1)$, applying ACE a_2 to obtain $x'' = \phi(x', a_2)$, so on and so forth until the n-th ACE.

Applying a sequence of ACEs enables greater exploration of the input space. If there exists non-zero probability of success at each step, we can show that the probability of success approaches 1 as the number of ACE steps goes to infinity.

Algorithm 1 Constitution Optimization Algorithm

Input: Training set D_{train}, Validation set D_{val}, Test set D_{test}, Language Model LLM, Number of epochs E, task definition t
SC (Surrogate Classifier) is the LLM along with the instruction template S to predict the autorater score for each given example.
CO (Constitution Optimizer) is the LLM along with an instruction template to update a given constitution using the classification outcome res of the SC
Initialize:
Generate an initial constitution π^* using the LLM and D_{train}
Set candidate list $\mathcal{L}_{candidates} \leftarrow \{\pi^*\}$
for epoch $e = 1$ to E **do**
  for π_i in $\mathcal{L}_{candidates}$ **do**
    $res_{val}^{(i)} = SC(LLM, \pi_i, t, D_{val})$
    $acc_{val}^{(i)} = g(res_{val}^{(i)})$ ▷ $g(.)$ measures the classification accuracy using predictions in res
  end for
  $i^* = \arg\max_i acc_{val}^{(i)}$ ▷ Find the best candidate using D_{val}
  $\pi^* \leftarrow \pi_{i^*}$
  $res_{train} \leftarrow SC(LLM, \pi^*, D_{train})$
  $res_{test} \leftarrow SC(LLM, \pi^*, D_{test})$

  Shuffle D_{train}
  for batch $b_j = b_1$ to b_B in D_{train} **do** ▷ Optimize on batches of D_{train}
    $\pi_j \leftarrow CO(LLM, \pi^*, b_j, res_{train,j})$
  end for
  $\mathcal{L}_{candidates} \leftarrow \{\pi_1, \ldots, \pi_B\}$
end for

5 EXPERIMENTS

In this section, we use ACE and Constitution-guided ACE (CACE) to understand and steer LLMs and text-to-image (T2I) models on 3 different downstream tasks described in § 5.1. We show the qualitative and quantitative results on the learned constitutions, compared with ACE (without constitution) as a baseline. Across our experiments, Gemini 2.5 Flash (Gemini Team Google, 2024) was used as the ACE Generator and for Constitution Optimization, and target models include Gemini 2.5 Flash and Pro (Gemini Team Google, 2024), GPT-5, 5-mini, 5-nano, 4o and OpenAI o4-mini (OpenAI, 2024). See § A for more details.

5.1 Downstream tasks and metrics

Each downstream task has a high-level task description, a dataset of prompts and an autorater that determines whether a model response is desirable. The metrics for all tasks include the surrogate accuracy (i.e., the accuracy of autorater scores predicted by the

Table 1: **Measuring the Impact of ACE and Constitution Optimization:** Using 3 diverse tasks, we measure the performance of various generative models (text-to-image and text-to-text) against 3 metrics. We compare the accuracy of the surrogate classifier using an un-optimized constitution (Epoch 0) with an optimized constitution (Epoch E). The success rate measures the percentage of initial prompts that were successfully steered towards satisfying the task within a maximum ACE sequence length of 4. We also compute the relative increase (r_1/r_0-1, where r_1 is the success rate for ACE with constitution and r_0 is the baseline ACE) in success rate noted in ($\uparrow \cdot$). We use Self-BLEU to verify if ACE maintains the diversity of prompts after mutations. Across every task and target model tested, using optimized constitutions to modify prompts achieves a 1.86 times average relative increase in success rate over the baseline.

Target Model	**Task**	*Surrogate Accuracy* ↑		*Success Rate* ↑		*Self-BLEU* ↓		
		Initial Constitution	**Final Constitution**	**ACE**	**ACE Constitution**	**Starter Prompts**	**ACE**	**ACE Constitution**
Gemini 2.5 Flash	Word Count	87.46	87.46	33.32 ± 3.55	**63.38 ± 3.30** (↑ 90%)	0.04	0.03 ± 0.02	0.15 ± 0.02
Gemini 2.5 Pro	Word Count	94.70	94.70	25.50 ± 3.37	**74.70 ± 3.65** (↑ 193%)	0.04	0.03 ± 0.02	0.18 ± 0.01
GPT 5	Word Count	93.85	**96.08**	32.67 ± 3.26	**86.02 ± 2.50** (↑ 163%)	0.04	0.04 ± 0.02	0.19 ± 0.01
GPT 5-mini	Word Count	97.49	97.49	29.74 ± 3.68	**93.47 ± 2.41** (↑ 214%)	0.04	0.03 ± 0.02	0.31 ± 0.01
GPT 5-nano	Word Count	93.30	**95.81**	31.37 ± 3.28	**87.11 ± 2.69** (↑ 177%)	0.04	0.03 ± 0.02	0.35 ± 0.01
GPT 4o	Word Count	93.29	**95.81**	29.92 ± 3.60	**49.24 ± 4.38** (↑ 65%)	0.04	0.02 ± 0.01	0.02 ± 0.01
o4-mini	Word Count	91.92	**92.47**	33.84 ± 3.40	**67.54 ± 4.00** (↑ 100%)	0.04	0.03 ± 0.02	0.09 ± 0.02
Gemini 2.5 Flash	Math	68.23	**78.45**	17.46 ± 4.92	**46.62 ± 7.98** (↑ 167%)	0.35	0.07 ± 0.06	0.10 ± 0.02
Gemini 2.5 Pro	Math	62.29	**66.92**	22.53 ± 5.83	**82.33 ± 4.95** (↑ 265%)	0.35	0.07 ± 0.05	0.24 ± 0.01
GPT 5	Math	68.76	**76.20**	13.40 ± 4.74	**61.82 ± 8.28** (↑ 361%)	0.35	0.02 ± 0.03	0.23 ± 0.01
o4-mini	Math	55.40	**69.45**	10.07 ± 4.83	**87.40 ± 5.09** (↑ 768%)	0.35	0.01 ± 0.03	0.17 ± 0.04
Imagen 4	T2I Align	82.14	**92.08**	83.61 ± 5.12	**97.00 ± 2.72** (↑ 16%)	0.04	0.03 ± 0.01	0.03 ± 0.01
DALL-E 3	T2I Align	79.82	**91.34**	70.83 ± 5.93	**87.13 ± 4.30** (↑ 23%)	0.04	0.02 ± 0.02	0.01 ± 0.01
GPT-Image	T2I Align	82.47	**91.25**	91.06 ± 4.20	**98.97 ± 1.02** (↑ 9%)	0.04	0.03 ± 0.01	0.03 ± 0.00

surrogate), the ACE success rate (i.e., whether the desired model behavior can be achieved by taking ACEs), and the Self-BLEU score (Papineni et al., 2002) for measuring the diversity of a set of prompts. A higher Self-BLEU score implies more repetition in the generated text and therefore less diversity.

Three tasks we tested are below.

Word Count: The high level description is "force the model to adhere to a word count constraint". The autorater counts the number of words in the model response, and classify whether there are fewer than 50 words (this exact constraint description is not revealed to ACE). For generating ACEs and learning a constitution, we sample 100 questions from the wikiHow dataset of LIMA (Zhou et al., 2023a) as the initial prompts, such as "How to cook an omelette?". We then evaluate the constitution on 100 held-out questions in LIMA.

Math: The task is to steer model to incorrectly answer math questions. We sample and 30 test questions as initial prompts from the GSME dataset in (Li et al., 2025), where each math problem is presented as a constraint satisfaction problem (CSP), such as "a=1, b=2, c = a+b. What is c?" The high-level task description is "Modify the math problem to make it more difficult to solve correctly while ensuring it remains a valid CSP, solvable by a symbolic math solver." The autorater uses SymPy (Meurer et al., 2017) to parse the CSP and solve the equations to obtain the ground truth answer.

T2I Align: The high-level description is "Update the prompt to decrease its alignment with its generated image". We use 150 starter text-to-image captions from the COCO dataset (Lin et al., 2015) as initial prompts. We evaluate three target T2I models: Imagen 4 (Saharia et al., 2022), DALL-E 3 (Ramesh et al., 2021), and GPT-Image 1 (OpenAI, 2024). For any given prompt, we generate 3 images and use the VQA Score as judged by Gemini (DeepMind, 2025) as the autorater, selecting the lowest of the 3 scores as the overall score for the prompt.

5.2 Qualitative Results: Understanding Models

For Word Count, since the task is very simple, we know the constitution should include concepts such as limiting the word count to a certain number, being concise etc. The constitution learned for this task is shown in Figure 2, which verifies that ACE is indeed capable of discovering the right kind of insights, even with very limited information of the task and mutation utilities. In the example illustrated in Figure 1, we observe that baseline ACEs contain long responses to the input until the third step in the sequence which gets a lower score due to a word constraint. In constitution-guided ACE however, the guidance provided helps mutate the prompt to quickly satisfy the word count limit within

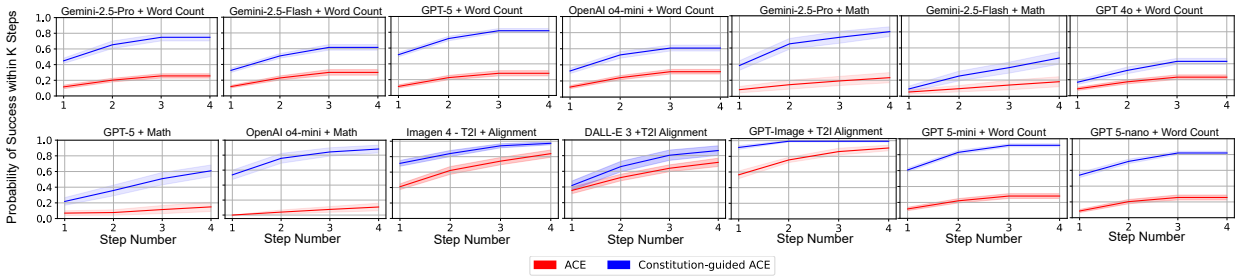

Figure 5: **Probability of ACEs Succeeding:** We measure the probability of ACE satisfying the goal task within the given number of steps (or length of ACE sequences), represented in the x-axes. We observe that using an optimized constitution with ACE helps achieve the goal faster (blue curves). Within 4 steps, we observe that ACE achieves a high success rate ($> 0.8\%$) across all models and tasks.

the first step.

Figure 2 shows the optimized constitution for T2I Align, Math and Word count tasks. Based on the task and target model, the optimized constitution summarizes unique aspects of conceptual perturbations and groups them as good or bad strategies. We observe that each insight on model behavior is described in a generalizable manner and does not contain any specific examples from the data. It is also important to note that these are actionable strategies that discover non-trivial dimensions and can directly be used to prepare relevant data and debug models. We observe that text-to-image models can struggle with alignment when objects, actions or the environment is either unrealistic or contradictory. This is illustrated in Figure 1 where constitution-guided ACEs like "adding industrial waste" to a scene with a park and "replacing very large with tiny" for a busy city, lead to successfully decreasing the text-to-image alignment. In math tasks, we observe that models sometimes cannot solve equations when complex operations like exponents are added (See Figure 1). Such mutations make the problem more complex, exposing LLM difficulty in solving involved mathematical problems.

We also observe unique behavior across models (See Appendix C) and discover non-trivial strategies. For word count constitutions, we do not explicitly specify the word limit of 50 in the task but we discover several successful strategies that indirectly achieve the task instead of directly specifying a word limit. These include "Explicit single-unit constraint", "Structural constraints", "Implicit tone modification".

For T2I generation, GPT-Image shows degrading T2I alignment by removing relational and action-focused elements like "Removing a Critical Relational Element" and "Removing the Object of a Verb". This points to a highly structured semantic understanding of the prompt where the relationships and actions are treated as distinct, critical components. On the other hand, Imagen shows decreased alignment for strategies like "Adding Unrelated and Prominent Elements" and "Introducing Conflicting Setting or Environment Details" which may indicate the model's on the holistic scene's context.

Figure D.4 compares the constitutions composed of "attack" strategies to make math equations more difficult for each model. It shows introducing distractor variables may cause GPT-5 to fail at solving simple math problems, but does not tend to impact the correctness of the answers from Gemini 2.5 Flash, Gemini 2.5 Pro and o4-mini. Moreover, shared strategies like introducing exponents demonstrate that LLMs in general still cannot master complex algebraic problems.

The above examples demonstrate that the insights from the constitutions are informative and non-trivial. Moreover, when we use constitution-guided ACE for that task and model, we observe that success is achieved in fewer steps.

5.3 Quantitative Results: Steering Models

Constitution Impact and Generality We measure the accuracy of surrogate classifier in predicting whether a proposed ACE to a prompt satisfies the given task according to the model and autorater, for the initial constitution (at epoch 0 in the evolution) and the final optimized constitution (final epoch when the evolution ends). Our results in Table 1 confirm that the optimization is successful observing improved performance and generalization (on the test split) over training iterations. Depending on the simplicity of the task, like some models assessed on word count, we also observe that the constitution may not evolve significantly over iterations, maintaining the surrogate accuracy. In the Appendix Section B, we compare our surrogate classifier to TextGrad Yuksekgonul et al. (2024) and observe that we outperform TextGrad on all tasks.

Effectiveness and Success Taking a set of unseen initial prompts, we use the ACE generation (with and without constitution-guidance) to randomly generate sequences of ACEs that result in successfully satisfying the task. For every initial prompt, we randomly select one sequence and we measure an aggregate score of the percentage of sequences that were successful, i.e., the ACE success rate. This score is averaged over 10 seeds and reported in Table 1. We observe that the success rate of ACE is higher with constitution-guidance. In Figure 5, we measure the cumulative probability distribution of ACE sequence lengths for satisfying the given task. Specifically, this measures if a successful mutation is achieved for the set of initial prompts within N steps of sampling ACEs (x-axis in Figure 5. Using the constitution (blue curves) results in higher likelihood of success in all models and tasks. This further confirms the effectiveness of using an optimized constitution with ACE since it not only achieves success quickly (as shown in Figure 1), but also shows a higher likelihood of success over multiple sequences, as shown in Figure 5.

We measure the diversity of initial prompts and the diversity of final prompts to ensure our approach is not simply converging on a trivial solution of similar, less-diverse prompts. The Self-BLEU scores in Table 1 confirms this by showing that the diversity does not significantly decrease after ACEs are applied.

6 CONCLUSIONS

We introduced a novel framework, Constitution-guided ACE, for understanding and steering black-box large generative models. The core objective was to systematically decode the "black box" behavior of these models by establishing a link between concept-level prompt modifications and shifts in model behavior. Our approach efficiently learns a human-interpretable constitution, which acts as a rulebook for tasks like adversarial steering.

A key direction for future work is extending this framework to other input modalities and applications such as image, video or agentic systems where, concepts and ACEs may be complex and extracting natural language explanations can be challenging. An added hurdle in such advanced systems can be the unavailability of robust autoraters. We also study model behavior in conjunction with the autorater by assuming the reliability of the autorater which may not be the case for all models and tasks in practice. More research is needed to be able to use ACE to disentangle vulnerabilities in model behavior from those in autorater behavior. We demonstrate applying ACE in sequences (multiple steps, or trajectories) however, the guiding constitution is optimized on single ACEs in turn promoting a greedy approach. Further research is needed to understand if agents are capable of optimally traversing trajectories of ACEs to achieve even higher likelihood of success.

ACKNOWLEDGMENTS

We would like to thank William Isaac and Helen King for valuable discussions and support. We would also like to thank Uri Shalit, Utsav Prabhu, Ken Burke, Caroline Pantofaru, Oyvind Tafjord, John Hewitt, Robert Geirhos for insightful comments and help, Dirichi Ike-Njoku for code review, and anonymous reviewers for feedback on the paper.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples, 2018. URL `https://arxiv.org/abs/1804.07998`.

Dana Angluin. Queries and concept learning. *Machine learning*, 2:319–342, 1988.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Jerome Bruner. *A study of thinking*. Routledge, 2017.

Brent Bryan, Robert C Nichol, Christopher R Genovese, Jeff Schneider, Christopher J Miller, and Larry Wasserman. Active learning for identifying function threshold boundaries. *Advances in Neural Information Processing Systems (NeurIPS)*, 2005.

Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. *arXiv preprint arXiv:1807.08024*, 2018.

David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal or Artificial Intelligence Research (JAIR)*, 4:129–145, 1996.

Google DeepMind. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL `https://arxiv.org/abs/2507.06261`.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Gemini Team Google. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024. URL https://arxiv.org/abs/2403.05530.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

Yash Goyal, Amir Feder, Uri Shalit, and Been Kim. Explaining classifiers with causal concept effect (cace). *arXiv preprint arXiv:1907.07165*, 2019.

Meera Hahn, Wenjun Zeng, Nithish Kannen, Rich Galt, Kartikeya Badola, Been Kim, and Zi Wang. Proactive agents for multi-turn text-to-image generation under uncertainty. In *International Conference on Machine Learning (ICML)*, 2025.

Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. Part II: Explanations. *The British journal for the philosophy of science*, 2005.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1215. URL https://aclanthology.org/D17-1215/.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active testing: Sample-efficient model evaluation. In *International Conference on Machine Learning (ICML)*, 2021.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Thomas Rainforth. Active surrogate estimators: An active learning approach to label-efficient model evaluation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Harold J Kushner. A versatile stochastic model of a function of unknown and time varying form. *Journal of Mathematical Analysis and Applications*, 5(1): 150–167, 1962.

Belinda Z Li, Been Kim, and Zi Wang. Questbench: Can llms ask the right question to acquire information in reasoning tasks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yang Li, Jie Ma, Miguel Ballesteros, Yassine Benajiba, and Graham Horwood. Active evaluation acquisition for efficient LLM benchmarking. *arXiv preprint arXiv:2410.05952*, 2024.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL https://arxiv.org/abs/1405.0312.

Tania Lombrozo. The structure and function of explanations. *Trends in cognitive sciences*, 10(10):464–470, 2006.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Vivek Miglani, Aobo Yang, Aram H Markosyan, Diego Garcia-Olano, and Narine Kokhlikyan. Using Captum to explain generative language models. *arXiv preprint arXiv:2312.05491*, 2023.

Ali Modarressi, Mohsen Fayyaz, Ehsan Aghazadeh, Yadollah Yaghoobzadeh, and Mohammad Taher Pilehvar. Decompx: Explaining transformers decisions by propagating token decomposition. *arXiv preprint arXiv:2306.02873*, 2023.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Associa-*

tion for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*, 2022.

Savvas Petridis, Benjamin D Wedin, James Wexler, Mahima Pushkarna, Aaron Donsbach, Nitesh Goyal, Carrie J Cai, and Michael Terry. Constitutionmaker: Interactively critiquing large language models by converting feedback into principles. In *Proceedings of the 29th International Conference on Intelligent User Interfaces*, pages 853–868, 2024.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021. URL https://arxiv.org/abs/2102.12092.

Javier Rando, Jie Zhang, Nicholas Carlini, and Florian Tramèr. Adversarial ml problems are getting harder to solve and to evaluate, 2025. URL https://arxiv.org/abs/2502.02260.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022. URL https://arxiv.org/abs/2205.11487.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning (ICML)*, pages 3145–3153. PMlR, 2017.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning (ICML)*, pages 3319–3328. PMLR, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014.

Joshua Tenenbaum. Bayesian modeling of human concept learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1998.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations (ICLR)*, 2024.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

James Woodward. *Making things happen: A theory of causal explanation*. Oxford university press, 2005.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2023.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text, 2024. URL https://arxiv.org/abs/2406.07496.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:55006–55021, 2023a.

Yongchao Zhou, Andrei Ioan Muresanu, Zixuan Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR)*, 2023b.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023a.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023b. URL https://arxiv.org/abs/2307.15043.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Not Applicable]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Interpreting and Controlling Model Behavior via Constitutions for Atomic Concept Edits: Supplementary Materials

A Additional Experiment Details

We summarize our choices for data, autorater and constitution hyper-parameters in Table A.1. For ACE generation, we use a fixed number of initial prompts and sample 2 ACEs per prompt, repeating this for each mutated prompt up to a max depth (sequence length). We stop sampling once we reach a prompt that successfully satisfies the task, hit the max depth or encounter an error during concept extraction, mutation, autorater or target model. We therefore report an approximate number of valid ACEs we generated that were used for optimizing the constitution and reporting metrics as this differs slightly among the different target models we studied.

We also experiment with dynamic constitution update strategies. Particularly, per epoch, we control the percentage of strategies to be updated via a hyperparameter. Moreover, we also control the number of strategies to be added or removed in the constitution via another hyperparameter. Both these hyperparameters lead to a dynamic change in both these parameters across epochs. Empirically, for complex tasks like math and T2I, we observed that gradually increasing the number of strategies across epochs, combined with decreasing the allowed percentage change gives the best results as per the performance and stability of training. The accuracy plots over iterations of evolving the constitution are displayed in Figure B.1.

	Parameter	**Word Count**	**Math**	**T2I Align**
Task definition	Task definition	Force the model to adhere to a word count constraint	Modify the math problem to make it more difficult to solve correctly	Update the prompt to decrease its alignment with its generated image
Autorater Details	Autorater	Response length	SymPy	Gemini 2.5 Flash
	Autorater success	Response length < 50	Answer $\neq$ Ground truth	VQA Score = 0
	Autorater failure	Response length ≥ 50	Answer = Ground truth	VQA Score = 1
ACE Generation	Initial prompt dataset	LIMA	GSME (QuestBench)	COCO Captions
	# initial prompts	100	100	150
	# ACEs sampled for each prompt	2	2	2
	ACEs sequence length	3	3	4
	# valid ACEs generated	$\sim$2500	$\sim$400	$\sim$1200
Constitution Optimization	Epochs	5	10	20
	Batch Size	100	200	50
	Constitution Size (Epoch 0)	10	5	10
	Constitution Size (Epoch E)	10	10	10
	Change % (Epoch 0)	10	100	100
	Change % (Epoch E)	10	5	5

Table A.1: **Experiment Details:** We summarize all the parameters used for each task for ACE generation and constitution optimization including autorater details, datasets, sizes and hyper-parameters.

B Using TextGrad as a Surrogate Classifier

We include results from using TextGrad Yuksekgonul et al. (2024) (a SOTA or near-SOTA approach for obtaining textual gradients) as a baseline for surrogate classification i.e., to predict whether a given update to a prompt (ACE) results in satisfying the given task using the autorater scores as ground truth. We use the same train, validation, test data and system instruction for both TextGrad and our surrogate model and we observe that our surrogate model outperforms TextGrad across all the tasks.

More importantly, the optimized prompt given by TextGrad lists strategies to help the LLM solve the surrogate task i.e., to determine if the proposed ACE satisfies the task or not. For example, the TextGrad optimized

Task	Model	TextGrad	Surrogate Model (ours)
Decrease T2I Alignment	Imagen 4	76.67	**92.08**
Adhere to Word Count	GPT-4o	89.97	**95.81**
Make Math Incorrect	GPT-5	66.66	**76.20**

Table B.2: **Using TextGrad as a Surrogate Model:** We measure the accuracy on the test split while using TextGrad for surrogate classification on each task. We observe that our model out-performs TextGrad and results in a more interpretable constitution to understand model behavior.

prompt for the word count task includes a strategy as follows - "Identify the proposed update, analyze its impact on word count, and conclude with a clear statement and numerical answer. Use a template for consistency". Such instructions do not uncover patterns in the ACEs and prompts to understand the model. On the other hand, our surrogate model is guided by the constitution which is explicitly optimized to summarize model behavior on the given task using the ACE data. Our method is therefore more performant and also inherently interpretable.

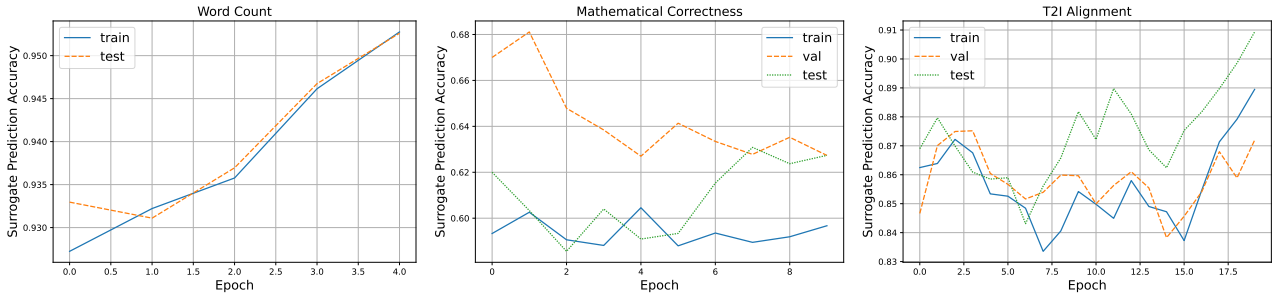

Figure B.1: **Accuracy of Surrogate Model against Training Iterations**: We plot the classification accuracy of the train, test and validation split over iterations of constitution evolution. We observe that the test accuracy improves over iterations. Simpler tasks require fewer iterations while more complex tasks can benefit from longer training.

C Constitutions for all Tasks

We provided the best constitutions for each tasks in Figure D.4 (Math), Figure D.5, D.6, D.7 (Word Count) and Figure D.3 (T2I Tasks).

D Prompts for ACE Generation

We provide the prompt used for extracting concepts and the relevant bag of ACEs in Figure D.9. We further provide the exact `preamble` used in Figure D.9 in Figure D.11. For 'Math' task, we observed that using this approach would lead to a significant number of ACEs being wasted as the modified problem could not be parsed by the deterministic Sympy solver. To counter this, we used an expanded pipeline for concept extraction in 'Math' as described in Figure D.8. For every bag of concept and ACEs from a `Propose` LLM call, we attempt a recycling of unparseable modified problems through an additional LLM call `Correct`. The parseable set of problems are made to pass though a `Expand` step to enforce diversity in the proposed ACEs. This was motivated after we observed that not enough required ACEs were being proposed by the original pipeline. We further provide the prompts for `Correct` and `Expand` in Figure D.10. The `Propose` prompt is the same as the main concept extractor prompt used for other tasks, as descriped in Figure D.9. We made minor adjustments to the `preamble` for different tasks especially in the the task-specific parts.

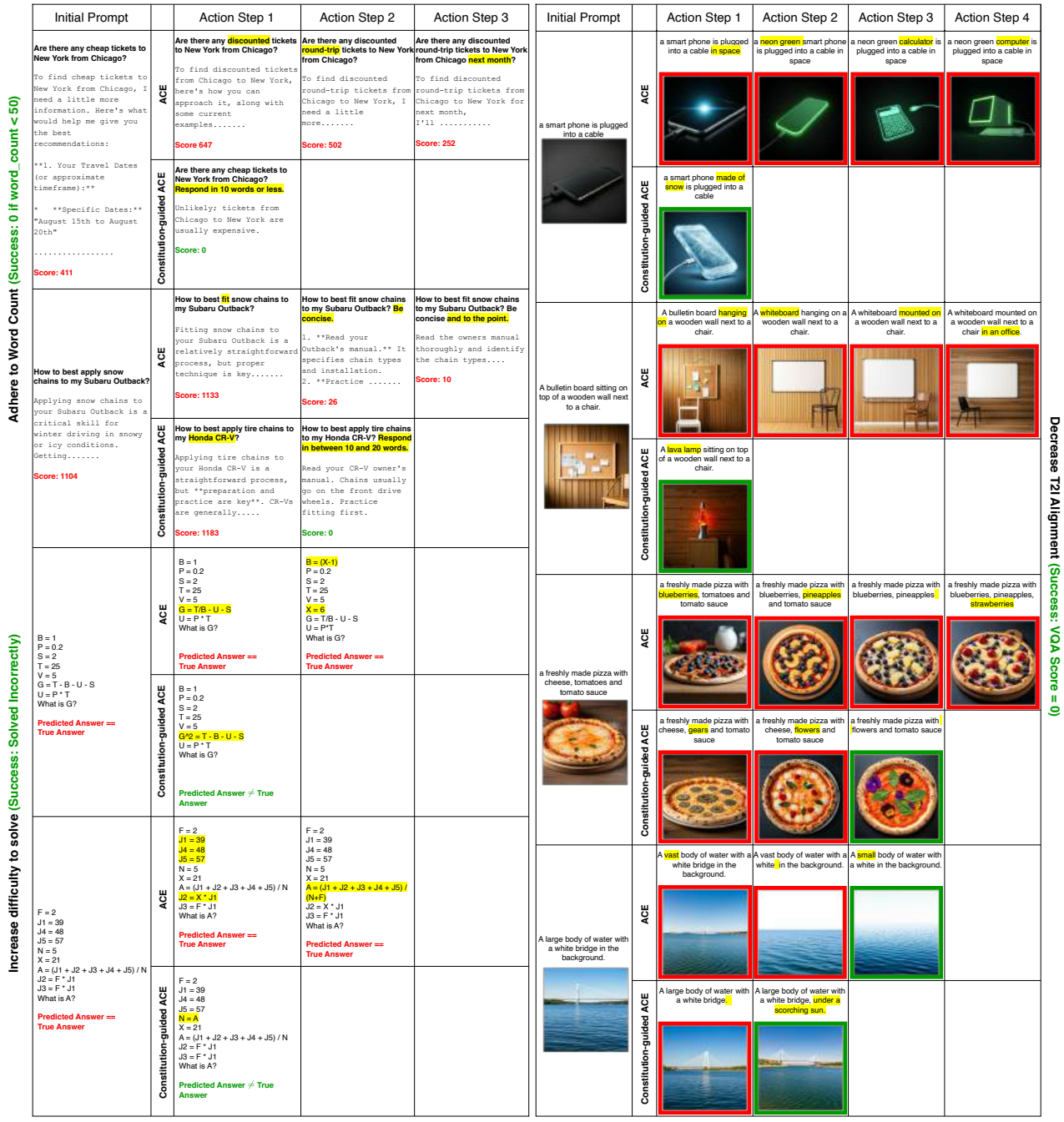

Figure B.2: Qualitative Examples of ACE: We illustrate how ACE may create mutations that eventually satisfy a given task.

Imagen	**Good Strategies** • **Adding Contrasting Actions or States:** Introduce actions or states for the main subject that are actively in conflict with its nature or typical behavior. This creates a strong dissonance that the AI struggles to represent coherently alongside the original subject, reducing overall alignment. The conflict resides in what the subject is doing or its current condition. • **Replacing Central Concepts with Contradictory Objects:** Substitute a key noun describing the main subject with a term representing a physically incompatible or conceptually opposed object. This creates a fundamental visual conflict that is challenging for the AI to integrate smoothly, leading to a significant reduction in alignment. This alters the core focus by changing the primary object. • **Adding Unrelated and Prominent Elements:** Include seemingly random objects or concepts that demand significant visual space or attention without a clear connection to the original prompt. If these added elements are visually dominant or inherently incompatible with the existing scene's context, they can disrupt the overall coherence and reduce alignment. The impact here depends on the visual weight of the added element. • **Introducing Conflicting Setting or Environment Details:** Add descriptors for the location or surroundings that are visually or contextually incompatible with the main subject or action described in the original prompt. The clash between the central elements and their environment creates a challenge for the AI in generating a cohesive image, thus decreasing alignment. **Bad Strategies** • **Direct Replacement with Visually Similar Concepts:** Substituting one object or characteristic with another that shares a very similar visual appearance or function within the scene has minimal impact on alignment. The AI can easily substitute the new element without significantly altering the overall composition or meaning of the generated image. • **Removing Redundant or Implicit Information:** Deleting concepts or objects that are strongly implied or inherently associated with other elements already present in the prompt usually does not decrease alignment. The AI can often infer and generate the removed element based on the remaining context, maintaining the original intent. • **Adding or Modifying Subtle Spatial Descriptors:** Altering phrases that describe the precise position or arrangement of objects in a way that results in only a minor or easily resolvable visual change has limited impact on alignment. The AI can typically accommodate small positional shifts without fundamentally changing the image's structure or message. • **Replacing a Descriptor with a More Aligned Alternative:** Substituting a descriptive word with another that provides a clearer or more accurate visual characteristic will likely increase alignment. This action clarifies the user's intent and makes it easier for the AI to generate an image that matches the specified details. • **Adding Inherently Aligned or Expected Descriptors:** Including a descriptive word or concept that is a common or expected attribute of another element in the prompt does not introduce conflict and will likely be incorporated by the AI without reducing alignment. It simply reinforces existing visual assumptions. • **Replacing an Action with a Related or Plausible Alternative:** Substituting a verb describing an action with a different action that is still a plausible or related activity for the subject has minimal impact on alignment. The AI can easily adapt to the new action without creating significant visual dissonance relative to the subject.
DALLE-3	**Good Strategies** • **Introduce Ambiguity with Unrelated Elements:** Add descriptive terms or concepts that are not closely related to the core subject of the prompt. The disconnect between the core subject and the added terms introduces ambiguity that the model might struggle to resolve correctly. • **Modify Core Concept with Dissimilar Property:** Directly alter a central descriptive element of the initial prompt to exhibit a property significantly different from the original description. This forces the model to interpret and integrate a new characteristic into the scene, potentially increasing the likelihood of a less aligned output. • **Add Trivial and Conflicting Constraints:** Introduce conditions or constraints into the new prompt that are logically incompatible with the existing elements of the scene. The model attempting to reconcile these conflicting constraints can lead to a decreased alignment with the original scene description. • **Combine Unlikely Concepts:** Merge two or more concepts or subjects that are not typically found together in reality or commonly associated in training data. The juxtaposition of unrelated elements challenges the model to create a coherent image that correctly reflects all instructions, leading to reduced alignment. **Bad Strategies** • **Change a Characteristic Property:** Alter a descriptive element of an object or scene in a way that doesn't significantly alter the fundamental nature of that object or scene. The change might be incorporated without disrupting the overall representation, resulting in no meaningful decrease in alignment. • **Introduce Synonyms or Paraphrased Descriptions:** Replace a noun or phrase with a synonym or paraphrase that carries the same core meaning and context. The underlying concepts remain the same, so the alignment is likely preserved. • **Introduce Redundant Details:** Add detailed descriptions of elements already present in the prompt or easily inferred terms. Because these terms don't introduce new information or alter existing concepts, the model will likely produce a result that still aligns with the original prompt. • **Specify the Color of an Object:** Make a minor change like altering a color without introducing any other conflicting elements or concepts. This single change is unlikely to drastically challenge the model's interpretation of the scene. • **Introduce a Common Action:** Add a common action that is consistent with the existing elements of the prompt and doesn't introduce any ambiguity or conflict. The model will likely integrate this action seamlessly, leading to the prediction reflecting this addition.
GPT-Image	**Good Strategies** • **Removing a Key Subject:** This strategy involves deleting a main noun or entity from the prompt that is central to the original description. By removing a primary subject, the generated image is likely to depict a scene significantly different from the original intent, thereby decreasing alignment. • **Replacing a Key Subject with a Dissimilar One:** Replacing a central noun or entity with something completely different and often semantically unrelated. This drastically alters the core content of the image, leading to a clear decrease in alignment with the original prompt's intention. • **Removing a Critical Relational Element:** This approach focuses on removing the words or phrases that describe the relationship or interaction between two or more subjects in the equation. Without this explicit connection, the image generation process may interpret the subjects as being depicted separately or in a different manner, reducing adherence to the intended scene or action. • **Removing the Object of a Verb:** Deleting the noun phrase that the verb in the prompt acts upon. This removes the focus of the action described, leading to ambiguity or a different interpretation of the scene in the generated image, thus decreasing alignment. **Bad Strategies** • **Merely Clarifying or Rephrasing an Existing Element without a Substantive Change:** Restating existing descriptors, like clarifying the color of an object that was already present, without fundamentally altering the core elements of the prompt. This type of change is unlikely to impact the image generation in a meaningful way. • **Adding Minor Descriptive Details that Don't Impact the Core Scene:** Incorporating minor adjectives or adverbs that don't alter the main subjects, actions, or relationships in a significant way. For example, adding "slightly" or "a little" to a modifier that doesn't dramatically change the core meaning. • **Adding a Highly Specific Element to an Underspecified Prompt:** Introducing a concrete detail (e.g., a specific object, color, or action) into a prompt that was previously very general or abstract. This often increases alignment as the image generation process now has a clearer instruction to follow. • **Removing a Non-Essential Modifier:** Deleting an adjective, adverb, or phrase that provides additional detail but is not crucial to the core meaning or structure of the prompt. The absence of this modifier is unlikely to significantly alter the main subjects, actions, or relationships depicted in the generated image. • **Setting an Attribute Already Present:** Explicitly stating an attribute (like color, style, or lighting) that was already clearly implied or mentioned in the original prompt. This action doesn't introduce a new conflicting element, merely reinforces an existing one, which will likely maintain or increase alignment. • **Removing a Subject Not Explicitly Mentioned:** Deleting a noun or entity in the action that was not a core or even minor part of the original prompt's description. Since the subject was not present in the original prompt, its removal from the action will not affect the alignment of the generated image with the original prompt.

Figure D.3: Optimized constitutions for Text-to-Image Alignment Task

o4-mini	**Good Strategies** • **Introduce a System of Equations:** Modify an existing equation into a system of two or more linear equations. This increases the complexity by forcing the user to solve for multiple variables simultaneously. The new system should be solvable and produce a valid result within the context of the overall problem. • **Increase Order of Equation:** Alter an existing linear equation to a higher-order polynomial, such as a quadratic or cubic equation. This requires more complex algebraic manipulation to solve. Ensure the new equation has a real solution that fits logically within the problem's structure. **Bad Strategies** • **Change an Operator:** Replace an arithmetic operator (e.g., addition, subtraction, multiplication, division) within an equation with a different operator. This often fundamentally changes the logic of the problem, potentially invalidating it or making it trivially easy or difficult in a way that is not algebraically challenging. Such a change can lead to nonsensical results or alter the problem type entirely. • **Introduce an Unused Variable:** Add a new variable to the problem that is not used in any of the subsequent calculations. This adds an extraneous piece of information that doesn't affect the solution path or the final answer. While it might slightly increase cognitive load, it doesn't make the algebraic solving process more difficult. • **Add a Superfluous Step:** Append a new, simple calculation at the end of the problem that uses the final result of the original problem. This simply adds a trivial step without increasing the complexity of the core algebraic problem. The added step usually involves a basic arithmetic operation.
GPT-5	**Good Strategies** • **Introduce an Exponent or Root:** Modify an existing equation to include an exponent or a root. This increases the complexity of the calculation beyond simple arithmetic, often requiring more steps and a deeper understanding of mathematical order of operations. This strategy is distinct because it introduces a non-linear operation. • **Add a Distractor Variable:** Introduce a new variable and an associated equation that is not necessary to solve the main problem. This tests the user's ability to identify relevant information and ignore irrelevant data, increasing the cognitive load required to solve the problem. • **Cross-Equation Variable Substitution:** Modify an equation to use a variable defined in another, seemingly unrelated equation. This forces the solver to track dependencies between different parts of the problem, increasing the number of steps and the potential for error. This is different from simply adding a variable, as it creates a more intricate web of relationships. **Bad Strategies** • **Simple Value Change:** Change the numerical value of a single variable without altering the underlying equations or relationships. While this changes the final answer, it doesn't fundamentally increase the difficulty of the required algebraic steps. • **Operator Swap:** Replace a basic arithmetic operator (e.g., addition, subtraction, multiplication, division) with another basic operator within an equation. This alters the calculation but doesn't necessarily make the problem more difficult to solve from an algebraic standpoint.
Gemini-2.5-Flash	**Good Strategies** • **Introduce a Non-Linear Relationship:** This strategy involves modifying a linear equation to a non-linear one, such as introducing exponents, square roots, or trigonometric functions. This increases the complexity of the calculation by altering the fundamental relationship between variables. • **Introduce Inter-Variable Dependency:** This strategy involves replacing a constant with a variable, or making one variable dependent on another. This increases the mental load required to solve the problem by introducing a new relationship between variables. • **Compound Operations:** This strategy involves combining multiple arithmetic operations into a single step, rather than a single operation. This forces the user to apply the order of operations and track multiple calculations at once. **Bad Strategies** • **Redundant Modification:** This strategy involves adding a new variable or equation that does not affect the calculation of the final answer. This fails to increase the difficulty of the problem. • **Value Swapping:** This strategy involves changing the numerical value of a constant without altering the underlying equations. This typically makes the problem easier to solve, as it only requires a simple substitution.
Gemini-2.5-Pro	**Good Strategies** • **Alter the Order of Operations:** This strategy involves changing the order in which mathematical operations are performed by adding, removing, or modifying parentheses. This can significantly impact the final result by forcing a different order of operations than what would be followed by default. This makes the problem more difficult to solve correctly, as it requires careful attention to the new structure of the equation. • **Operator Substitution:** This strategy involves replacing one mathematical operator with another, such as changing addition to subtraction, multiplication to division, or vice versa. This can drastically alter the problem's outcome and requires careful adjustment of the solving process. The change in operator makes the problem more difficult by introducing a new relationship between the variables. • **Introduce an Exponent:** This strategy involves introducing an exponent into one of the equations, either by raising a variable to a power or by using an exponential function. This significantly increases the complexity of the problem, as it requires understanding and applying the rules of exponents. The introduction of an exponent can lead to a much larger or smaller result, making it more challenging to solve correctly. **Bad Strategies** • **Add a New Variable or Constant:** This strategy involves adding a new variable or numerical constant to an existing equation. While this changes the final answer, it doesn't fundamentally increase the problem's conceptual difficulty. The steps to solve the problem remain the same, just with an additional simple arithmetic operation, making this an ineffective way to increase difficulty. • **Change a Numerical Value:** This strategy involves changing a numerical value in the problem to a different number, such as an integer to a fraction or a positive to a negative. While this may alter the final answer, it does not fundamentally change the problem's structure or the steps required to solve it. This is considered an ineffective strategy because it only makes the arithmetic more tedious rather than increasing the problem's conceptual difficulty. • **Replace a Variable with an Expression:** This strategy involves replacing a variable with a more complex mathematical expression. This can require additional steps to solve but does not necessarily make the overall problem more difficult. The added complexity is often a distraction rather than a fundamental increase in the problem's difficulty, making this an ineffective strategy.

Figure D.4: Optimized constitutions for 'Math'

o4-mini	**Good Strategies** • **Direct Numerical Length Constraint (Significantly Deviating from Natural Output):** Explicitly state the desired number of words, characters, sentences, or paragraphs when this length is significantly different from the model's natural output for the given prompt. This direct numerical instruction guides the model to produce responses within the specified limit, ensuring precise adherence when the constraint actively forces a change in length by either significantly shortening or lengthening it. This strategy is effective when the numerical target is clearly outside the model's default response length. • **Word Count Range with Significant Bounds:** Define a minimum and maximum word count, where the range itself is a meaningful constraint beyond the model's natural output. This encourages the model to expand or contract its response within a clearly defined and impactful range. The bounds must be sufficiently far from the typical output length to compel a change in behavior. • **Conciseness Modifiers with Clear Reduction:** Use terms that clearly indicate a requirement for extreme brevity (e.g., "one-word answer," "single sentence," "extremely brief"). These modifiers provide an unambiguous instruction for a drastically shorter output that would not typically be generated by the model by default. This strategy is effective when a significant and undeniable reduction in length is desired. • **Targeted Prompt Refinement for Brevity:** Rephrasing or removing superfluous elements from the original prompt, specifically to make the core query more direct, can sometimes lead to shorter, more focused responses from the LLM. This indirect approach can aid in conciseness, especially when elements are removed or replaced with a more concise alternative, altering the model's response length without explicitly stating a count. This strategy is about simplifying the prompt to elicit a naturally shorter response. • **Minimum Word Count (Significantly High for Elaboration):** Specify a minimum word count that is substantially higher than a typically brief answer. This strategy forces the model to elaborate and expand its response to meet the imposed lower bound, effective when you want to force more detail than it naturally would provide. The minimum count must be clearly impactful to compel an expanded response. **Bad Strategies** • **Numerical Word Count (Near Natural Output Tendency):** Setting a numerical word count that aligns closely with, or is slightly below or above, the model's inherent tendency for a given prompt often fails to compel a specific length. This includes numerical constraints that are generally met or exceeded by the model's default response length, acting as a loose ceiling rather than an active constraint without forcing a behavioral change. Numerical constraints are ineffective if they don't significantly alter the model's natural output length in either direction. • **Ambiguous Length by Detail Level:** Using subjective terms like "brief," or "high-level" without a numerical or structural element is too vague. The model's interpretation of these terms may not match the user's intent, leading to inconsistent length. These terms do not provide a clear enough instruction for the model to alter its natural output length. • **Indirect Content Modification without Length Influence:** Changing the prompt's focus (e.g., synonyms, audience, context) without any direct or indirect instruction related to length. These modifications alter the content but generally do not influence the response's word count unless they naturally simplify the query significantly. This strategy alters context but not necessarily length. • **Altering Response Format (Lacking Explicit Length):** Requesting a different output format such as bullet points or paragraphs, without explicitly adding a length constraint, does not guarantee adherence to word count. The model can still produce verbose outputs within the new format, as the format itself does not imply a length. A format change alone is insufficient for length control. • **Minimum Word Count (Insufficiently High):** Specifying a minimum word count that is not significantly greater than what the model would naturally produce for a brief answer. This weak minimum doesn't effectively compel the model to expand its response, failing to push the model beyond its default behavior. The minimum word count must be aggressively high to be effective.
GPT-5	**Good Strategies** • **Direct Numeric Word Count (Low & Strict Constraint):** Explicitly stating a numeric word count for the entire response, such as "Set word count as X words" or "Set the maximum word count to X words" or "Set the word count to X", where X is a relatively low number (e.g., 50 words or less). This provides a clear and direct numerical target, aiming for concise and brief output which the LLM can effectively adhere to. • **Constraint on Output Type:** Apply length constraints directly to a specific *type or category of output* mentioned in the prompt, such as "SET word count of informational answer as X words" or "Set the length of the summary to X words." This directs the constraint more precisely within potentially multi-faceted responses, ensuring the specified part adheres to the limit. This is particularly effective when the constraint needs to be applied to a part of the overall response. • **Explicit Single Unit Constraint:** Limiting the response to a single, defined textual unit, such as "limited to a single paragraph." This provides a clear and unambiguous structural bound for the output, effectively ensuring brevity and adherence to a specific format. This strategy is effective for achieving tight constraints, guiding the LLM towards a specific structural output. • **Output Length Specification:** Use phrases that specifically refer to the *output* or *response* length, such as "SET output length of response as X words" or "SET length of response as X words." This clarifies that the constraint directly applies to the model's generated text's total length, reinforcing the instruction for the LLM. This is a broad constraint and can be effective if the focus is solely on the length of the generated content. • **Qualitative Length Indicator with Structural Focus:** Using qualitative terms like "short" or "brief" in conjunction with a direct structural constraint for the output, such as "SET response format as 'a short paragraph'" or "Set length constraint type as paragraph count." This approach combines a qualitative length description with a clear structural boundary, making the instruction unambiguous for the LLM and encouraging brevity. **Bad Strategies** • **High or Unrealistic Word Counts:** Setting a word count constraint that is significantly higher than what might be considered a truly "constrained" output (e.g., "Set the word count to 200 words," "Set the maximum word count as 100 words"). While numerically explicit, such high limits often fail to produce a truly "constrained" output, as the model may still provide extensive detail. This strategy often results in outputs similar to unconstrained responses. • **Minimum Word Count:** Setting a *minimum* word count (e.g., "Set minimum words to 40"). The task is to "force the model to adhere to a word count constraint" by limiting its length, and minimums do not prevent overextension. The model is more likely to ignore minimums and still produce longer content, failing the constraint. • **Ambiguous Qualitative Length Indicators:** Using qualitative terms like "brief" or "concise" without explicitly linking them to a specific structural format or numerical target, or when they are applied to the prompt itself rather than the response. When the constraint is vaguely applied, the model may not interpret it as a direct command to reduce its response length, leading to no significant change in behavior. • **Non-Word-Count Numerical Constraints:** Specifying a numeric length constraint using units other than words, such as character count or sentence count (e.g., "Set the maximum length as 100 characters", "Set response to 4 sentences"). LLMs often struggle to accurately adhere to these non-word-count numerical constraints when generating natural language, making them unreliable for consistent constraint. • **Non-Output Constraint Application:** Applying a word count constraint to an aspect of the prompt that is not the LLM's generated output, such as "SET word count of query as 50 words" or "Set word count of prompt as fifty words." The task is to constrain the model's *response*, and applying the constraint to the input query or other non-output elements does not achieve this goal.
Gemini-2.5-Flash	**Good Strategies** • **Direct Word Count Specification:** Explicitly state the desired word count using numerical values. This provides the LLM with a clear and unambiguous target for response length, especially when seeking a very short or precise output. • **Maximum Length Constraint:** Use phrasing like "maximum length" or "under X words" to set an upper boundary for the response. This prevents the LLM from exceeding a certain word threshold. • **Word Count Ranges:** Provide a numerical range for the word count, giving the LLM some flexibility while still imposing a clear boundary. This allows for slight variance while keeping the response contained. • **Clear Brevity Instruction:** Employ terms that directly instruct the LLM for brevity and specificity, such as "short and to the point" or "very brief and factual." This guides the model towards economical language when combined with a clear implicit or explicit length target. • **Structural Constraints (Sentences/Paragraphs) for Enforced Brevity:** Define the response length by specifying a very low number of sentences or paragraphs (e.g., 1 sentence, 1 paragraph). This offers a structural constraint that naturally limits word count and effectively enforces brevity when the target is genuinely short. **Bad Strategies** • **Implicit Tone Modification for Length:** Attempting to control word count by only setting a "tone" (e.g., "simple," "direct") does not directly address length. The LLM may adopt the tone but not the desired brevity, unless combined with an explicit length constraint. • **Broad Contextual or Topical Changes:** Modifying the core topic or adding unrelated context (e.g., replacing keywords, specifying audience) distracts from the length constraint. The LLM will focus on the new information rather than brevity. • **Detailed Specification of Output Format Alone:** While mentioning formats like "bullet points" can contribute to brevity, it doesn't guarantee a specific word count. A detailed bulleted list can still be very long if no explicit length is given. • **High Word Count for Simple Queries:** Setting a word count constraint that is disproportionately high for the complexity of the query (e.g., 100+ words for a simple factual question) often leads to the model ignoring the constraint, as its natural answer would be shorter. This can also happen if a short natural answer is constrained to a much higher word count than is needed. • **Vague Conciseness Keywords Without Numerical Context:** Using general terms like "concise," "brief," or "short" by themselves, without any accompanying numerical or structural constraint, often fails to reliably enforce a specific word count. The LLM may interpret these terms broadly, leading to inconsistent results regarding length.

Figure D.5: Optimized constitutions for 'Word Count'

<table>
<tr><td>Gemini-2.5-Pro</td><td>

Good Strategies

- **Direct Numerical Word Count:** Explicitly state the desired response length using a specific number of words, clearly indicating it applies to the model's output. This offers an unambiguous and precise target for the LLM.
  - *Description:* This strategy involves directly specifying the exact number of words the model's *response* should contain. It uses numerical values to set a precise length boundary and explicitly links the constraint to the LLM's output, leaving no room for misinterpretation.
- **Relative Length Constraint:** Use terms like "single sentence" or "briefly" combined with a specific structural output to imply a very short word count. This indicates a clear, concise response without specifying a number.
  - *Description:* This strategy sets a general brevity requirement by instructing the model to provide a response within a very limited structural unit, such as a single sentence or a small, fixed number of sentences. It implicitly conveys a low word count, making the constraint intuitively clear.
- **Character Count Limit:** Define the maximum number of characters allowed in the response. This offers an alternative and highly granular way to control length.
  - *Description:* This strategy controls response length by setting a maximum character limit. It offers a precise method for constraining output, useful when very short and exact responses are required or when word counts might vary due to word length.
- **Explicit Numerical Constraint for Output:** Clearly state a numerical word count constraint using natural language, ensuring it is unambiguously directed at the *model's output*. This makes the intention clear for the LLM.
  - *Description:* This strategy clearly specifies a numerical word count that applies to the model's generated text. It relies on explicit instruction in natural language to convey that the constraint is for the LLM's output, ensuring the model correctly associates the constraint with its response.
- **Implied Numerical Constraint for Output:** Set a numerical word count constraint that is clearly intended for the model's output, even if not explicitly stating "response" or "output," as long as the context makes it unambiguous.
  - *Description:* This strategy involves setting a numerical word count using natural language, where the constraint is clearly about the generated text despite not using explicit terms like "response" or "output." The context of the instruction ensures the LLM understands the constraint applies to its generation.

Bad Strategies

- **Vague Brevity Indicators:** Using non-quantifiable terms like "brief," "short summary," or "concise" without a numerical or structural constraint. These are too ambiguous for an LLM to confidently adhere to.
  - *Description:* This strategy uses subjective and qualitative terms to indicate a preference for a shorter response. Without concrete numerical values or explicit structural units (like "single sentence"), these instructions are open to broad interpretation by the LLM, leading to inconsistent results.
- **Unrelated Constraint Units:** Specifying length in terms of paragraphs or a large number of sentences without also providing a numerical word count constraint. This still leaves too much room for variation within each unit.
  - *Description:* This strategy attempts to control length by specifying the number of structural units, such as paragraphs or sentences, but fails to define a maximum word or character count *within* those units. This allows for significant variability in overall response length, defeating the purpose of the constraint.
- **Altering Content/Tone While Constraining Length:** Combining a word count request with other significant modifications like changing the tone, audience, or content. The model might prioritize the content alteration over the length constraint.
  - *Description:* This strategy dilutes the word count instruction by simultaneously requesting other significant modifications to the prompt, such as changes in tone, audience, or core content. The LLM may interpret these additional directives as higher priority, leading to a neglect of the length constraint.
- **Minimum Word Counts:** Specifying a *minimum* word count. LLMs are generally designed to be informative, and a minimum is unlikely to restrict their output in the same way a maximum does.
  - *Description:* This strategy attempts to influence response length by setting a lower boundary for the word count. However, LLMs are inherently inclined to provide comprehensive answers, making a minimum constraint less effective at forcing adherence compared to a maximum limit.
- **Word Count on the Prompt Itself:** Applying a word count constraint to the input prompt rather than the model's generated response. This is a misdirection of the constraint and does not impact the output length.
  - *Description:* This strategy incorrectly applies the word count constraint to the prompt provided by the user, rather than the desired length of the model's output. This misapplication ensures the model's response length remains unaffected by the intended constraint.

</td></tr>
<tr><td>GPT-5-mini</td><td>

Good Strategies

- **Direct Word Count Specification:** Explicitly state the desired word count and the unit "words." This directly communicates the constraint to the model.
- **Specific Numeric Word Count (Low-to-Medium):** Use exact, relatively low numbers (e.g., 10, 15, 20, 25, 50) for word count specifications. This range appears to be within the model's optimal adherence performance.
- **Targeting Response Word Count:** Clearly indicate that the word count applies to the model's output. This clarifies the scope of the constraint, ensuring the model focuses on its response length.
- **Word Count Constraint via Action Term:** Use action-oriented phrasing like "SET word count as X" or "Set the word count constraint to X". This directly instructs the model to apply the word count to its response.
- **Character Count Specification:** Explicitly specifying a desired character count for the model's response, along with the unit "characters," effectively constrains the output length. This provides a clear and measurable limit for the model to adhere to.

Bad Strategies

- **High Numeric Word Count Request:** Requesting a significantly larger word count (e.g., 100 words, 200 words, 250 words) often fails. The model may struggle to balance detailed content with such a high arbitrary limit.
- **Vague Brevity Terms:** Using subjective terms like "concise," "brief," "summary," or "short" without a numeric value is often ineffective. These terms lack the specificity the model needs to apply a constraint.
- **Paragraph or Sentence Range Constraints:** Setting constraints based on a number of paragraphs or a range of words (e.g., "Set the word count range as 50-75 words") tends to be ignored. The model prefers single, exact numeric targets.
- **Altering Subject Matter:** Changing the core subject of the prompt (e.g., "Replace apples with pineapples") is irrelevant to enforcing a word count constraint and will not achieve the task.
- **Applying Constraint to Input Prompt:** Imposing a word count or length constraint explicitly on the user's input prompt itself, rather than the model's generated output, is ineffective for controlling the response. The model will not interpret this as a constraint on its own response.

</td></tr>
<tr><td>GPT-5-nano</td><td>

Good Strategies

- **Direct Numeric Word Count (Low & Strict Constraint):** Explicitly stating a numeric word count for the entire response, such as "Set word count as X words" or "Set the maximum word count to X words" or "Set the word count to X", where X is a relatively low number (e.g., 50 words or less). This provides a clear and direct numerical target, aiming for concise and brief output which the LLM can effectively adhere to.
- **Constraint on Output Type:** Apply length constraints directly to a specific *type or category of output* mentioned in the prompt, such as "SET word count of informational answer as X words" or "Set the length of the summary to X words." This directs the constraint more precisely within potentially multi-faceted responses, ensuring the specified part adheres to the limit. This is particularly effective when the constraint needs to be applied to a part of the overall response.
- **Explicit Single Unit Constraint:** Limiting the response to a single, defined textual unit, such as "limited to a single paragraph." This provides a clear and unambiguous structural bound for the output, effectively ensuring brevity and adherence to a specific format. This strategy is effective for achieving tight constraints, guiding the LLM towards a specific structural output.
- **Output Length Specification:** Use phrases that specifically refer to the *output* or *response* length, such as "SET output length of response as X words" or "SET length of response as X words." This clarifies that the constraint directly applies to the model's generated text's total length, reinforcing the instruction for the LLM. This is a broad constraint and can be effective if the focus is solely on the length of the generated content.
- **Qualitative Length Indicator with Structural Focus:** Using qualitative terms like "short" or "brief" in conjunction with a direct structural constraint for the output, such as "SET response format as 'a short paragraph'" or "Set length constraint type as paragraph count." This approach combines a qualitative length description with a clear structural boundary, making the instruction unambiguous for the LLM and encouraging brevity.

Bad Strategies

- **High or Unrealistic Word Counts:** Setting a word count constraint that is significantly higher than what might be considered a truly "constrained" output (e.g., "Set the word count to 200 words," "Set the maximum word count as 100 words"). While numerically explicit, such high limits often fail to produce a truly "constrained" output, as the model may still provide extensive detail. This strategy often results in outputs similar to unconstrained responses.
- **Minimum Word Count:** Setting a *minimum* word count (e.g., "Set minimum words to 40"). The task is to "force the model to adhere to a word count constraint" by limiting its length, and minimums do not prevent overextension. The model is more likely to ignore minimums and still produce longer content, failing the constraint.
- **Ambiguous Qualitative Length Indicators:** Using qualitative terms like "brief" or "concise" without explicitly linking them to a specific structural format or numerical target, or when they are applied to the prompt itself rather than the response. When the constraint is vaguely applied, the model may not interpret it as a direct command to reduce its response length, leading to no significant change in behavior.
- **Non-Word-Count Numerical Constraints:** Specifying a numeric length constraint using units other than words, such as character count or sentence count (e.g., "Set the maximum length as 100 characters", "Set response to 4 sentences"). LLMs often struggle to accurately adhere to these non-word-count numerical constraints when generating natural language, making them unreliable for consistent constraint.
- **Non-Output Constraint Application:** Applying a word count constraint to an aspect of the prompt that is not the LLM's generated output, such as "SET word count of query as 50 words" or "Set word count of prompt as fifty words." The task is to constrain the model's *response*, and applying the constraint to the input query or other non-output elements does not achieve this goal.

</td></tr>
</table>

Figure D.6: Optimized constitutions for ‘Word Count’

GPT-4o	**Good Strategies** • **Direct Word Count Specification**: Explicitly state the desired word count using numerical values (e.g., "100 words"). This clearly communicates the exact length expectation to the model, aiming for a specific output size. This is more effective for setting an upper limit than a minimum. • **Character/Sentence Limits**: Specify the maximum number of characters or sentences allowed for the response. This provides alternative objective metrics for controlling output length, especially when a precise word count might be less practical or when focusing on brevity of individual components. This strategy is generally more reliable for upper limits. • **Word Count Range**: Define an acceptable upper and lower bound for the word count. This allows for slight flexibility within a desired brevity, providing the model with a target window for response length, though lower bounds can be less reliably enforced. • **Targeted Brevity Commands**: Use direct commands that specify brevity for specific elements of the response, often implying a single unit of output (e.g., "brief explanation"). This ensures that a particular part of the output adheres to a short length. • **Removal of Irrelevant Context or Superfluous Information**: Remove parts of the prompt that dilute the core request or add extraneous details. By making the prompt more focused, it can lead to a more concise and controlled output, indirectly aiding in word count adherence if the removed context previously expanded the response. **Bad Strategies** • **Irrelevant Tone/Audience Adjustments**: Modifying the prompt's tone, target audience, or level of detail *without explicitly linking it to length*. These changes focus on style or context, not word count, and will not reliably influence output length. • **Generic Prompt Rewording**: Simply rephrasing the prompt using synonyms or slightly different phrasing. These actions change the wording but do not introduce any new constraints related to length. • **Changing Specifics/Core Request**: Altering factual elements or the specific context of the prompt in a way that changes the fundamental request. These modifications change the core request rather than its length, making them irrelevant to word count control. • **Format Changes (without length implication)**: Requesting specific output formats without any accompanying explicit instruction about the length of those formatted elements. The format itself does not guarantee a word count. • **Vague Qualitative Length Descriptors**: Using qualitative terms that imply a constrained length but are often too ambiguous or non-specific for reliable word count adherence. These terms like "concisely" or "briefly" do not provide a sufficiently clear instruction for consistent word count restriction, often failing to yield a specific length. The model's ability to adhere to these is inconsistent and often does not result in a precise word count.

Figure D.7: Optimized constitutions for 'Word Count'

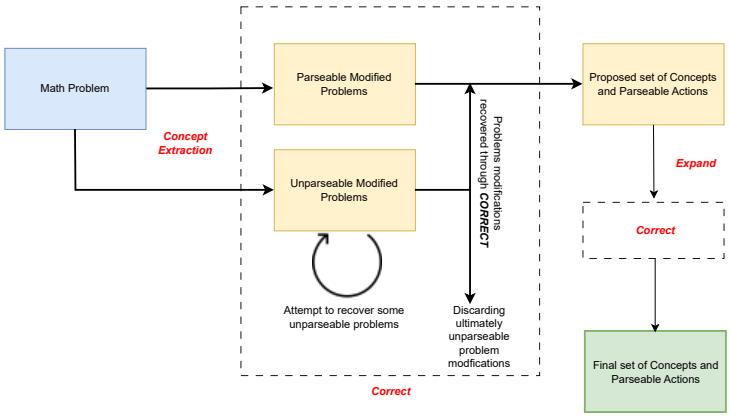

Figure D.8: Expanded view of the pipeline used for ACE generation in 'Math'. We propose an initial set of concepts and corresponding ACEs through the concept extraction prompt in Figure D.9. We filter out unparseable modifications to the problems through using signals from the SymPy Solver. We attempt to recover some unparseable problems through the user of `Correct` prompt (Figure D.10). Finally, to enforce diversity in concepts and their respective ACEs, we use the `Expand`(Figure D.10) prompt to propose newer concepts and ACEs.

Concept Extractor Prompt	`{preamble}` `Prompt: {prompt}` `Generate the concept bag given the input below. Your response should be in JSON format WITHOUT the JSON tags.` `Respond only in the ConceptBag dataclass format above without the ```json tags. Do not output anything else.` `Input: {{` ` "prompt": "{prompt}"` ` "task": "{task}",` ` "task_description": "{task_description}"` `}}` `Output:`

Figure D.9: Concept extractor prompt used for obtaining concepts and relevant ACEs related to that concept.

Expand	You are an expert at refining and expanding upon a list of ideas. A previous step has generated an initial ConceptBag. Your task is to make it better by adding more concepts and actions that are highly creative and laser-focused on the TASK. TASK: {task} TASK DESCRIPTION: {task_description} ***CREATIVITY MANDATE*** - **Think outside the box!** Propose creative and unconventional modifications. - **Don't be afraid to be "crazy."** Sometimes the most unexpected ideas are the most effective at misleading a language model. The goal is to find the model's blind spots. Push the boundaries of the initial ideas. ***YOUR INSTRUCTIONS*** 1. **Analyze the existing ConceptBag**: Review the concepts and actions below. 2. **Add New Concepts**: Add at least 3 NEW, distinct concepts that were missed in the first pass. 3. **Expand Existing Concepts**: For EACH of the existing concepts, add at least 2 NEW, creative actions that are different from the ones already there. 4. **Maintain Rules**: Every new action you create must follow the original rules (solvable prompt, correct verbalization, etc.). The primary goal is always the TASK. Additionally, you must adhere to these **Structural Modification Rules**: - All `updated_prompt` values MUST remain valid, solvable math problems in the same structured format as the original. - Modifications should be structural changes to the math problem itself (e.g., changing variable names, altering equations, adding distractors) rather than adding conversational English text. - The core structure of equations and the final question ("What is ...?") must be preserved. 5. **Return a Complete, Updated ConceptBag**: Your final output should be a single ConceptBag object that includes both the original and the new concepts and actions. Do not repeat any actions. ***EXISTING_CONCEPT_BAG*** {existing_concept_bag}
Correct	"""You are an expert at crafting deceptively simple-looking but tricky math problems. The following JSON objects represent actions that were intended to make a math problem more misleading for an AI, but an error made them unsolvable by a symbolic deterministic solver. Your task is to **fix the `updated_prompt`** in each JSON object to make it a valid, solvable math problem, while **preserving or even enhancing the original misleading intent**. Analyze the error and the original intent. Then, rewrite the `updated_prompt` to be both mathematically sound and highly deceptive. Here are the actions that need fixing: {json.dumps([info for _, info in actions_to_correct], indent=2)} Return a JSON list of the corrected actions. Only return the actions that you were able to successfully make both solvable and misleading. If you cannot fix an action in a way that preserves its deceptive spirit, do not include it in your response. Each corrected action should be a complete JSON object with the same structure as the input. """

Figure D.10: `Expand` and `Correct` prompts used for enforcing diversity and recovering unparseable ACEs, for concept generation in 'Math'

preamble	"""You are an expert at extracting a ConceptBag from a given prompt, task and task description. A ConceptBag represents 1. Collection of concepts that exist in the given prompt or in your belief state around the prompt 2. A number of actions proposed on each concept that can be taken to modify the prompt. ***BEGIN_CONCEPT_DEFINITION*** Some descriptions of what a concept is: 1. Concepts are independent ideas or abstractions that may exist in the prompt or generated image. 2. A concept can take several forms as long as it corresponds to a single idea or notion. 3. A concept cannot be a mixture of multiple independent ideas or notions. Types of concepts: ** Explicit concepts: A concept that is explicitly present in the prompt. It can also be a metadata concept like style, mood, image style etc. ** Relationship concepts: A type of concept that represents a relationship between two concepts. ** Explicit Attribute concepts: A type of concept that represents a property or an attribute of another concept to further describe it. For example, in a prompt 'a white rabbit eating a carrot', the concept 'white' can be an attribute of 'rabbit' and the concept 'eating' can be a relationship between 'rabbit' and 'carrot'. ** Implicit attribute concepts: An attribute concept need not be explicitly present in the prompt. An implicit attribute concept cannot exist if the concept it is describing is not present in the prompt. This could also be a metadata concept that covers the full prompt. It MUST be associated with an explicit concept already in the prompt or some metadata to better describe the context. For example, in a text-to-image prompt like "a rabbit in a barn", an unmentioned concept could be the background, mood, image style etc. ***END_CONCEPT_DEFINITION*** ***BEGIN_ACTION_RULES*** The actions to the concept are updates to the prompt or generated image that correspond to the concept. The actions can only be ADD, REMOVE or REPLACE type actions. The actions MUST BE ATOMIC. This means that they can ADD, REMOVE or REPLACE only ONE concept at a time. 1. ADD action: This can involve adding a concept that is not present in the prompt. This may be adding an implicit concept, setting an attribute of a concept or setting a relationship of two concepts. Add action is only permitted if the concept is not already present in the prompt. HIGH LEVEL STRUCTURES FOR VERBALIZATION: - SET <implicit_attribute_concept_name> of <explicit_concept> as <value> - SET <relationship_concept_name> as <value> between <concept_1> and <concept_2> 2. REPLACE action: This can involve replacing a concept name with an alternative name. This can only be done if the concept already exists in the prompt. You can replace any kind of concept including explicit concepts, attributes, relationships as long as it is already present in the prompt. HIGH LEVEL STRUCTURES FOR VERBALIZATION: - REPLACE <concept> with <alternative_value> - REPLACE <explicit_attribute_concept_name> of <explicit_concept> from <current_value> to <alternative_value> - REPLACE <relationship_concept_name> between <concept_1> and <concept_2> from <current_value> to <alternative_value> 3. REMOVE action: This can involve removing a concept from the prompt. This can only be done if the concept already exists in the prompt. You can remove any kind of concept including explicit concepts, attributes, relationships as long as it is already present in the prompt. HIGH LEVEL STRUCTURES FOR VERBALIZATION: - REMOVE <concept> - REMOVE <explicit_attribute_concept_name> of <explicit_concept> - REMOVE <relationship_concept_name> between <concept_1> and <concept_2> ***END_ACTION_RULES*** ***BEGIN_GENERAL_RULES*** * Extract as many concepts that make sense as possible from the prompt with respect to the task and task description * Extract as many actions that make sense as possible for each concept with respect to the task and task description * MAKE SURE THE ACTIONS ARE DIVERSE. * MAKE sure the actions are not too directly trying to accomplish the task. They should be exploratory in nature. * Provide an EXACT verbalization of what the action should do using the high level structures above. Mention the exact concept name and value to update in the prompt. * The value of any action MUST respect the concept definition and should be a single independent idea or notion. It must not be a mixture of multiple concepts. If you would like to add a concept containing multiple ideas, consider breaking it down into two separate concepts. * Do not propose implicit attribute concepts for other implicit concepts. They MUST be associated with an explicit concept in the prompt. ***END_GENERAL_RULES*** ***BEGIN_CONCEPT_BAG_DEFINITION*** Your job is to use the given task and task description to prepare a ConceptBag that creates concepts and actions that satisfy the given task. The dataclass ConceptBag contains the following fields: "concepts": a set of concepts as defined in the Concept class. "prompt": the given prompt that you are extracting concepts and possible actions from. The dataclass Concept contains the following fields: "name": name of the concept. "description": description of the concept. "actions": a list of ADD, REMOVE or REPLACE actions that can be taken to modify the prompt with respect to the concept. The dataclass ConceptAction contains the following fields: "action_type": the type of action, an ActionType enum. Can be either ActionType.ADD_ACTION, ActionType.REMOVE_ACTION or ActionType.REPLACE_ACTION. "verbalization": the verbalization of the action. This should follow the action rules above. For example, given a prompt, "A rabbit eating a carrot", some action verbalizations can be "Set the color of the rabbit as white", "Remove rabbit", "Set the mood to scary" etc. Remember, the action can only be applied to ONE concept at a time. "updated_prompt": the updated prompt after taking the action. Make sure to use the verbalization to update the prompt. Change only what is proposed in the verbalization. Do not change anything else in the prompt. Remember, the prompt is only updated to reflect the changed concept, no other changes. ***END_CONCEPT_BAG_DEFINITION*** Extract as many concepts and actions as possible with respect to the task and task description. Make sure to respond in the dataclass format provided. Do not respond with any other text.

Figure D.11: Preamble used in the Concept Extractor Prompt (Figure D.9).