Combinatorial Promptimization: A Genetic Algorithm for Prompt Evolution with Chromosome Representation

Joshua Nielsen

June 2025

Abstract

Large language models (LLMs) are highly sensitive to the prompts that guide their behavior, yet most prompt engineering today remains manual, brittle, and unoptimized. In this paper, we introduce Combinatorial Promptimization, a novel framework that reimagines prompt design as a combinatorial optimization problem over a structured search space. Drawing inspiration from genetic algorithms, our approach encodes prompts as interpretable chromosomes composed of modular components, enabling systematic exploration, recombination, and mutation of prompt structures. Unlike prior methods such as PromptBreeder, which operate in unstructured and sample-inefficient spaces, our method constrains the search to a tractable, meaningful subset while retaining expressiveness and adaptability. We incorporate dynamic hyper-mutation to expand the search space over time, guided by performance-based feedback. Experiments on the GSM8K benchmark using the Mixtral-8x7B model demonstrate that our framework consistently improves prompt performance, outperforming strong baselines and matching or exceeding the results of more computationally intensive approaches. Beyond benchmark accuracy, we highlight the framework's interpretability, extensibility, and potential for industrial applications in LLM alignment and prompt deployment. This work offers a new paradigm for prompt design grounded in optimization theory, and invites further exploration at the intersection of language models and combinatorial search.

1 Introduction

1.1 From Prompt Engineering to Promptimization

Prompt Engineering is the discipline of crafting prompts that instruct large language models (LLMs) in such a way that they exhibit the intended behavior and *only* the intended behavior [13]. Prompt Engineering is a vital practice because all AI systems are fundamentally driven by prompts—whether explicitly input by

the user or embedded as a system prompt behind-the-scenes. From agentic AI workflows to AI assistants like GitHub Copilot[1] or Cursor[2], system prompts define core behaviors and constraints. These prompts are typically hand-crafted through trial-and-error, yet they are not guaranteed to be empirically optimized, and are often brittle when transferred across domains. Since different use cases demand different capabilities, effective deployment requires tailoring prompts to specific contexts, reinforcing the need for systematic and scalable prompt design methods.

Promptimization is a direct appendage of elaborate problem-solving efforts, where automated approaches iteratively improve a baseline prompt (or set of prompts) until it reaches near-optimal performance for a given task. While popularized by the release of ChatGPT in November 2022, researchers had already been exploring methods for automated prompt generation. One such effort used reinforcement learning to train a policy network that generated prompts by searching over discrete token sequences for optimal task performance [4]. A key finding from this work is that the highest-performing prompts were often unintelligible to humans—for example, one optimal prompt for a sentiment transfer task was "(Parameters comparison)=(Compare either)." This suggests that large language models may process prompts in ways not easily anticipated by human intuition, and that manual prompt crafting may not achieve optimal performance.

However, the models used in that study—BERT, GPT-2, and RoBERTa—were base models, not explicitly trained to follow the conversational natural language instructions. Many modern systems, such as GPT-4, are instruction-tuned—that is, fine-tuned on datasets containing human instructions and responses, enabling them to follow prompts in a more conversational and intuitive manner [10]. While this makes interaction with the model more accessible for end-users, it does not eliminate the need for automated prompt optimization when task performance is paramount. Empirical methods are still necessary to identify truly optimal prompts, regardless of how intuitively they align with human expectations.

Since the emergence of ChatGPT, most promptimization efforts can be traced back to the Automatic Prompt Engineering (APE) algorithm introduced by Zhou et al. [16], and many additional algorithms are emerging that seek to improve it or address gaps. These include hint generation [14], actor-critic editing [5], "gradient descent"[3] and beam search [11], and some interesting evolutionary algorithmic approaches [6, 3, 8]. Among the most powerful of these is PromptBreeder by Google DeepMind [6], which serves as the inspiration for our proposed methodology.

[1] https://github.com/features/copilot, accessed April 2, 2025

[2] https://www.cursor.com/en, accessed April 2, 2025

[3] The quotes are used by the original authors because it is not literal; no gradient descent was used in the proper mathematical sense, rather the process was emulated in principle.

1.2 Promptimization as A Constrained Optimization Problem

Promptimization is the automated process of crafting prompts that maximize the performance of a LLM $\mathcal{M}$ on a target task T. Formally, given an LLM $\mathcal{M}$, a dataset $\mathcal{D} = \{(Q, A)\}$ of input-output pairs, and an evaluation metric f, the goal is to identify an optimal prompt p^* that maximizes the expected performance of $\mathcal{M}$ on $\mathcal{D}$. As written by Zhou et al. [16], this is commonly framed as:

$$p^* = \arg\max_p f(p) = \arg\max_p E_{(Q,A)\sim\mathcal{D}}[f(p, Q, A)],$$

where $f(p, Q, A)$ measures the performance of $\mathcal{M}$ prompted with p on input Q and output A, e.g., classification accuracy.

Existing methods seek to optimize directly in the global feasible region $\mathcal{P}$, defined as the set of all possible prompts:

$$\mathcal{P} = \{p \mid p \in \mathcal{V}^k, k \leq K\},$$

where $\mathcal{V}$ is the LLM's vocabulary, p is a sequence of tokens and K is the context limit. However, the vastness of $\mathcal{P}$ renders exhaustive exploration computationally infeasible, i.e., it is not reasonable to explore every possible permutation of tokens. The rules of meaningful language act as a natural filter, excluding nonsensical prompts that would obviously fail (e.g., "flibberty gibberty") in favor of semantically plausible instructions (e.g., "Solve the following math problem"). As a result, the effective search space is implicitly constrained to a more promising subregion of $\mathcal{P}$. Nonetheless, the number of ways to express even a single task remains astronomically large, especially in domains requiring specialized knowledge which may require more verbose and specific instructions [15]. This combinatorial explosion is characteristic of problems best approached with meta-heuristics, such as genetic algorithms, and directly motivated the design of PromptBreeder.

1.3 Combinatorial Optimization and Meta-Heuristics

Combinatorial optimization is a subfield of mathematical optimization that consists of finding an optimal object from a finite set of objects, where the set of feasible solutions is discrete or can be reduced to a discrete set. The prototypical example of a combinatorial optimization problem is the Traveling Salesman Problem, wherein one seeks to find the shortest path one can take to visit each city exactly once and return to the city of origin. In many such problems, exhaustively searching all possible solutions is intractable, so high-level strategies known as meta-heuristics are designed to find near-optimal solutions. Unlike problem-specific heuristics, meta-heuristics provide a general framework that can be adapted to a wide range of problems. Prominent examples include genetic algorithms (GAs), simulated annealing, particle swarm optimization,

and ant colony optimization [2].

An essential component of meta-heuristic approaches is the structured encoding or representation of solutions. This can include representing solutions as a molecule, as in the case of Chemical Reaction Optimization [9], or an object which has a position, an inertial mass, active gravitational mass, and passive gravitational mass, as in the case of Gravitational Search Algorithms [12]. In our chosen case of GAs, solutions are typically encoded as chromosomes, or sequences of alpha-numeric elements. This is an imitation of how DNA is composed of the four bases A (Adenine), C (Cytosine), T (Thymine) and G (Guanine), which can be strung together in various patterns like A-T-T-G-C, etc. This encoding enables operators like crossover to recombine solutions in a way that preserves and potentially enhances their fitness. Without a well-defined encoding, the algorithm loses its power to exploit patterns in the solution space effectively.

These methods often balance two key principles: exploration and exploitation. Exploration refers to the ability to search broadly across the solution space to avoid getting trapped in local optima, or a neighborhood of good - but not best - solutions. Exploitation focuses on refining promising solutions to converge toward an optimal or near-optimal solution. GAs, in particular, are a canonical example of meta-heuristics that leverage evolutionary principles to optimize solutions. They maintain a population of candidate solutions (chromosomes), which evolve over time through selection, crossover (recombination), and mutation. Fitness functions guide the selection of individuals to ensure that better-performing solutions have a higher probability of propagating their features to subsequent generations.

In the case of PromptBreeder [6], while the algorithm is inspired by evolutionary principles, it diverges from a traditional genetic algorithm by lacking a chromosome-like representation for prompts. Instead, each prompt is treated as an independent entity without any structured decomposition into meaningful subcomponents that can be recombined or mutated in a controlled manner. This lack of structured representation limits the algorithm's ability to leverage the full potential of genetic operators. Though some of the aforementioned promptimization methods are inspired by meta-heuristic methods, none have used strong representation methods that facilitate their proper usage. In short, none are attempting to solve a combinatorial optimization problem because there is no discrete set of solutions to explore.

1.4 Combinatorial Promptimization and Structured Chromosome Representation

To address these challenges, we introduce Combinatorial Promptimization and re-frame the promptimization problem by introducing a structured search space. This framework establishes a foundation for systematic

optimization of prompts, bridging the gap between heuristic-driven methods and rigorous combinatorial approaches. The best way to create this structured representation is an open - and very exciting - question. We propose the first approach for creating a structured chromosome space, $\mathcal{C}$, where each configuration $C \in \mathcal{C}$ represents a chromosome composed of modular components:

$$C = [c_1, c_2, \ldots, c_n], \quad c_i \in \mathcal{S}_i,$$

with $\mathcal{S}_i$ representing the set of valid settings for the i-th component. Each chromosome C maps to a prompt $p \in \mathcal{P}$ through an LLM-based mapping function:

$$g : \mathcal{C} \to \mathcal{P}, \quad p = g(C).$$

The optimization problem is then reformulated in terms of $\mathcal{C}$:

$$C^* = \arg\max_{C \in \mathcal{C}} E_{(Q,A)}[f(g(C), Q, A)].$$

However, this function assumes that we are changing only a single prompt. And while a single best prompt is still sought, we seek to find it by evolving a population of prompts. We therefore write

$$\text{Maximize } E_{C_i \in \mathcal{P}_t}\big[f(g(C_i), t)\big],$$

where $C_i \in \mathcal{P}_t$ is a chromosome in the population $\mathcal{P}_t$ at time t; $g(C_i)$ maps the chromosome C_i to a prompt $p_i \in \mathcal{P}$; $f(g(C_i), t)$ is the fitness of the prompt p_i at time t, based on the number of correct responses generated by the LLM.

This approach constrains the search to a smaller, more tractable space while maintaining access to the larger solution space through g. By leveraging systematic exploration techniques within $\mathcal{C}$, we can achieve better performance - paired with interpretability - compared to unstructured methods. The following section details our methodology for implementing this framework. The results are then given, followed by discussion, future work, and conclusions.

2 Methodology

2.1 Chromosomes as Lists of Setting Indices

Our approach defines a chromosome as a list of integers, where each value serves as an index pointing to a specific setting within a predefined list of prompt components. The key challenge lies in identifying which components should be included. As noted earlier, there are countless ways to construct a prompt, and the design of an effective representation remains an open problem. While some prompt features—such as assigning an expert persona or using chain-of-thought reasoning—are known to improve performance, we hypothesize that many other useful features remain unexplored or undefined, especially for each unique domain.

To address this, we generate candidate settings for each component following the method in the appendix of [6]. Specifically, the LLM is prompted with: "[Make a] List of 10 diverse ideas helpful in solving tasks like this one: INSTRUCTION: + problem description". This pattern is used both to initialize the list of components and to generate up to 10 possible settings for each one, focused on instruction strategies and task completion.

For our GSM8K test case, we select four components to seed the system: (1) instruction and reasoning approach, (2) creativity in solving the problem, (3) use of examples (e.g., few-shot learning), and (4) the persona assigned to the LLM. A chromosome is represented as a list $[x_1, x_2, x_3, x_4]$ where each x_i is an integer index corresponding to a setting within its respective component 1.

Index	**Instruction and Reasoning Approach**	**Creativity**	**Examples Used**	**Persona**
1	N/A	N/A	N/A	N/A
2	Step-by-step guidance	High creativity	Single example	Grade school math teacher
3	Simulate dialogue between two people	Low creativity	Two examples	High-achieving student
4	Outline common pitfalls	Lateral thinking	Three examples	Determined learner

Table 1. Example of chromosome settings

It is important that each component has an N/A setting, allowing the algorithm to learn to ignore a component if it adds no value to the prompt. Suppose a chromosome was $[3, 2, 2, 4]$, and the LLM is given the corresponding settings and these instructions:

> You are an expert in writing and designing instructions for language models. We say that instructions/prompts are composed of components, each of which have different settings. Generate written instructions for the following task using the given settings. Task: Solve a given math word problem. Settings:
> - Instruction and reasoning approach: Simulate dialogue between two people.
> - Creativity: High creativity
> - Examples used: One example
> - Persona: Determined learner

This may result in a prompt such as: "Show a discussion between two determined learners as they work creatively through an example math problem, then solve the given problem." Naturally, the exact output will vary depending on the LLM used and the phrasing of the meta-instruction, but this example illustrates the intended behavior.

2.2 The Promptimizer

2.2.1 Genetic Algorithm Methods

With a functional chromosome representation in place, we can now implement a genetic algorithm. For selection, we use a binary tournament, as was done in PromptBreeder. We retain the two highest-performing individuals as elites in every generation. The remaining 18 individuals participate in binary tournaments to form a parent pool. The nine individuals who lose their tournaments are replaced with children produced by randomly sampling two parents (with replacement) from this pool. Children are generated by one of three randomly selected crossover methods: one-point, two-point, or uniform.

2.2.2 Hyper Mutation

One of the innovations proposed in PromptBreeder is that of hyper mutation. That is, throughout the algorithm, it is not just the prompts themselves that improve, but also the process of mutating existing prompts. We follow their example, but instead of mutating the mutation instructions, we allow the chromosomes to grow dynamically by adding new components and settings over time. This allows us to evolve the *chromosome representation itself*, adapting to performance at various points.

Recall our example in Table 1; suppose that after the first generation, all prompts that use the "low creativity" setting perform very poorly, suggesting that it is not useful. We do not remove this setting, as

all settings remain part of the search space throughout (e.g., the setting may later pair extremely well with some other new setting), but we introduce new settings that are hypothesized to be stronger. Similarly, we introduce new components to the chromosome that are selected to improve performance.

Hyper-mutation occurs probabilistically based on a fixed probability P_h, provided that the chromosome representation has not reached its predefined maximum size (an upper limit k of components and l of settings). If a chromosome is selected for hyper-mutation, we apply one of three types of structural changes: dding a new setting to an existing component, adding a new component to the chromosome, or feedback-based modification of an existing prompt.

To guide hyper-mutation, we track the relative contribution of components and settings to fitness over time. Given a population of size N_t at time step t, we define the *setting score* for setting s within component c as:

$$S_s(c, s, t) = \sum_{i=1}^{N_t} f(C_i, t) \cdot \mathbb{1}(C_i[c] = s) \tag{1}$$

where $f(C_i, t)$ is the fitness of individual C_i at time t; $\mathbb{1}(C_i[c] = s)$ is an indicator function that equals 1 if individual C_i uses setting s in component c. Similarly, the *component score* for component c at time t is computed as:

$$S_c(c, t) = \sum_{s \in S_c} S_s(c, s, t) \tag{2}$$

To ensure fair comparisons, scores are normalized to be between zero and 1:

$$\tilde{S}_s(c, s, t) = \frac{S_s(c, s, t)}{\max_{s \in S_c} S_s(c, s, t)} \tag{3}$$

$$\tilde{S}_c(c, t) = \frac{S_c(c, t)}{\max_c S_c(c, t)} \tag{4}$$

When hyper-mutation selects the *Add Setting* operation, we first identify:

$$s^* = \arg\min_{s \in S_c} \tilde{S}_s(c, s, t) \tag{5}$$

$$s^+ = \arg\max_{s \in S_c} \tilde{S}_s(c, s, t) \tag{6}$$

where s^* is the lowest-performing setting and s^+ is the highest-performing setting within component c. The LLM is instructed to generate a new setting that is an improvement over all existing options, using s^*

to indicate what does not work and s^+ to indicate what has been successful. This promotes generation of new settings in useful directions, rather than randomly.

If hyper-mutation selects the *Add Component* operation, we identify:

$$c^+ = \arg\max_c \tilde{S}_c(c, t) \tag{7}$$

where c^+ is the best-performing component. Unlike setting generation, the LLM is only provided the best-performing component to generate a completely new component. The newly generated component is then added to the chromosome, with an initial set of settings, selected by the LLM. The first of these settings is always N/A so that the component can be ignored when necessary.

In the *Feedback* operation, a prompt is randomly selected from the population and evaluated using an LLM-based function, which examines both correct and incorrect classifications from its history. The LLM then generates a modified version of the prompt. The settings are extracted from this new prompt and convert them into a new chromosome representation. The LLM is instructed to use the settings as currently available to perform this conversion when possible, but if the new prompt uses a setting or component which cannot be mapped to the original, the overall representation schema is expanded to include it.

Each time a new setting or component is introduced, all chromosomes must expand accordingly. If a component is added at time $t+1$, each existing chromosome C_t is extended with a default value for the new component, yielding:

$$C_{t+1} = [C_t, 0] \tag{8}$$

Where 0 corresponds to the N/A setting that is present in all components. This ensures backward compatibility with previously generated chromosomes while allowing the search space to grow dynamically.

Hyper-mutation enables the search space itself to evolve over time while preserving structure. Rather than relying solely on random mutations, our approach uses performance-driven expansion to introduce new components and settings. This balances exploration—by adding new search dimensions—and exploitation—by guiding expansion based on fitness scores.

However, expanding the chromosome representation introduces a critical challenge: as the search space grows, convergence becomes harder to achieve. The set of possible chromosomes increases exponentially with each new component or setting. For example, starting with four components, each with four settings (see Table 1), there are $4^4 = 256$ possible chromosomes. To keep the problem tractable, we impose upper bounds on both the number of components and their available settings. In principle, a genetic algorithm with enough generations could explore this space effectively. In practice, the main constraint is compute—specifically, the

cost of inference required to evaluate each prompt. As models become smaller and inference more efficient, we expect this bottleneck to diminish. Nonetheless, as we will show, even within these practical limits, the Promptimizer achieves strong performance. The pseudocode for the whole process is given in the appendix.

2.3 Experiment Settings

Promptimizer was run using the open-source Mixtral-8x7B model on an A30 GPU with 24GB of VRAM hosted by the Logistics and Distribution Institute at the University of Louisville. We used a population of 20 individuals with an evaluation size of 50, to a maximum of 20 generations. The primary benchmark we used was the GSM-8K benchmark for grade-school math word problems. Other benchmarks will also be used for the final publication.

We conducted a 2^3 full factorial design of experiments to evaluate the effects of three factors with two levels: hyper-mutation rate, max component size, and max setting size. That is, at what rate do we attempt to trigger hyper-mutations which expand the chromosome representation, and what is the max size of the final representation? Hyper-mutation rates were either 5% and 10% and the max component size was either 6 or 8. When the max component size was 6, the max setting size could be either 8 or 10, and when the max component size was 8, the max setting size could be 10 or 12. For analysis, we examined the best and mean accuracies across generations. We expect that both measures will increase over time generally, though we anticipate that there will be additional variation across the generations as the search space expands.

3 Results

The 8 experiments took approximately one month to complete. The range of accuracies (% out of 50 samples) at the beginning and end of the experiments are shown in table 2.

Experiment	**Hyper mutation rate**	**Max components**	**Max settings**	**Initial range**	**Final range**
1	10%	8	10	(28, 82)	(26, 84)
2	10%	8	12	(26, 82)	(56, 84)
3	10%	6	8	(10, 72)	(50, 88)
4	10%	6	10	(28, 78)	(56, 86)
5	5%	8	10	(28, 80)	(70, 86)
6	5%	8	12	(24, 76)	(66, 88)
7	5%	6	8	(14, 82)	(62, 86)
8	5%	6	10	(30, 86)	(58, 90)

Table 2. Worst and best accuracies at start and end of 20 generations.

As expected, accuracy improves across all populations over the course of each experiment. Initial best scores are already relatively high, ranging from 72% (Experiment 3) to 86% (Experiment 8). Initial worst

scores are more variable, spanning 10% to 30% across the same experiments. By the final generation, nearly all worst-case accuracies have improved, with the exception of Experiment 1, where the lowest score decreases slightly. In all other cases, the final worst scores approach or exceed the initial best scores, indicating consistent population-wide improvement.

Experiments 5–8, which used a lower hyper-mutation rate, result in a narrower final accuracy range, with higher worst-case scores than Experiments 1–4. Nevertheless, the best final scores across all runs converge in the low-to-mid 80% range, with Experiment 8 achieving the highest at 90%.

To analyze performance trends, we group the results by hyper-mutation rate. Figures 1 and 2 show best accuracy over time for high and low mutation rate experiments, respectively. As anticipated, best accuracy generally increases across generations, with more significant gains early on, followed by marginal improvements later. Notably, Experiments 3 and 6 start from lower baselines but reach 88% by the final generation. Overall, we observe no substantial difference in final best accuracy attributable to hyper-mutation rate.

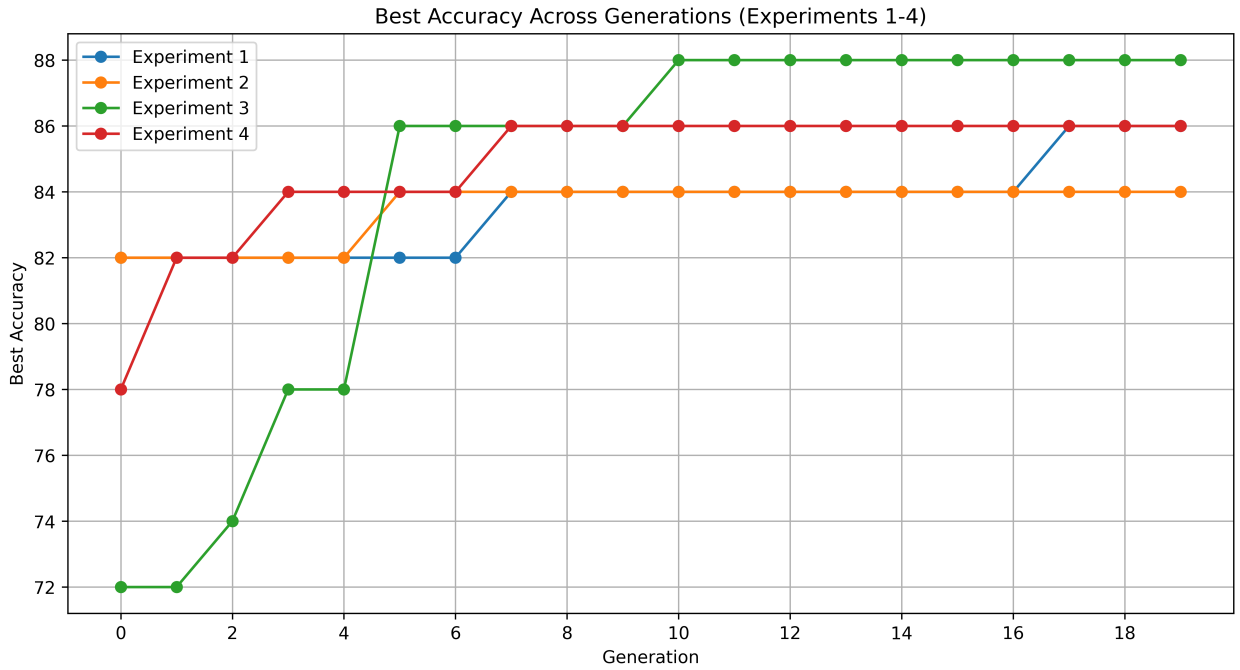

Figure 1. Best accuracies across generations for experiments 1-4 (high hyper-mutation rate)

Figures 3 and 4 show the progression of mean accuracy across generations for the high and low hyper-mutation rate experiments, respectively. The impact of the mutation rate is more clearly observed here: in Experiments 1–4 (high mutation), mean accuracy varies substantially in later generations. Notably, Experiment 1 exhibits a sharp decline after generation 16, eventually dropping below its initial mean at generation zero. In contrast, Experiments 5–8 (low mutation) progress more steadily, with all runs converging

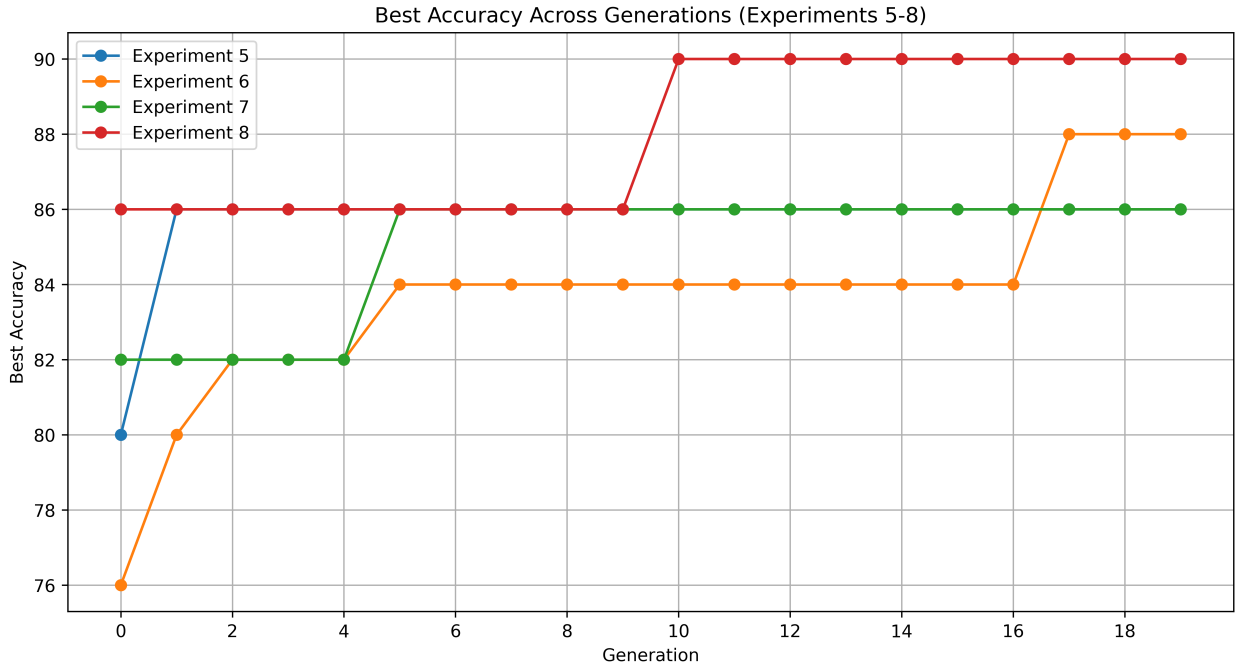

Figure 2. Best accuracies across generations for experiments 5-8 (low hyper-mutation rate)

toward similar mean accuracies after just a few generations. This behavior aligns with expectations, as a lower mutation rate emphasizes exploitation earlier in the evolutionary process.

Figure 5 further illustrates the distribution of accuracies across generations for Experiment 1. The blue line denotes the mean, and the black line the median, for each generation. While accuracy generally improves over time, a sharp decline begins around generation 17, ending with a lower mean and modal accuracy than the initial population. Though suboptimal, this is not surprising. As discussed earlier, expanding the search space introduces increased exploration, often at the expense of exploitation. This trade-off explains both the decline and the heightened variance observed from generation 7 onward. The long-term behavior of the distribution remains unknown due to computational limits, but because elite individuals are preserved in each generation, high-performing prompts are not lost even when average performance declines.

Due to space constraints, we do not include a table of all final component settings here. However, four new components and their various settings were introduced during the run:

- **Difficulty Level**: (N/A, Easy, Medium, Hard, Very Hard, Expert, Challenging)
- **Interaction Style**: (N/A, Interactive Quiz, Fill-in-the-Blanks, Multiple Choice, Short Answer, Step-by-Step Guided Solution),
- **Assessment and Feedback**: (N/A, Automated Scoring, Peer Review, Self-Assessment, Expert Feedback, Personalized Feedback, Expert-guided Adaptive Feedback)

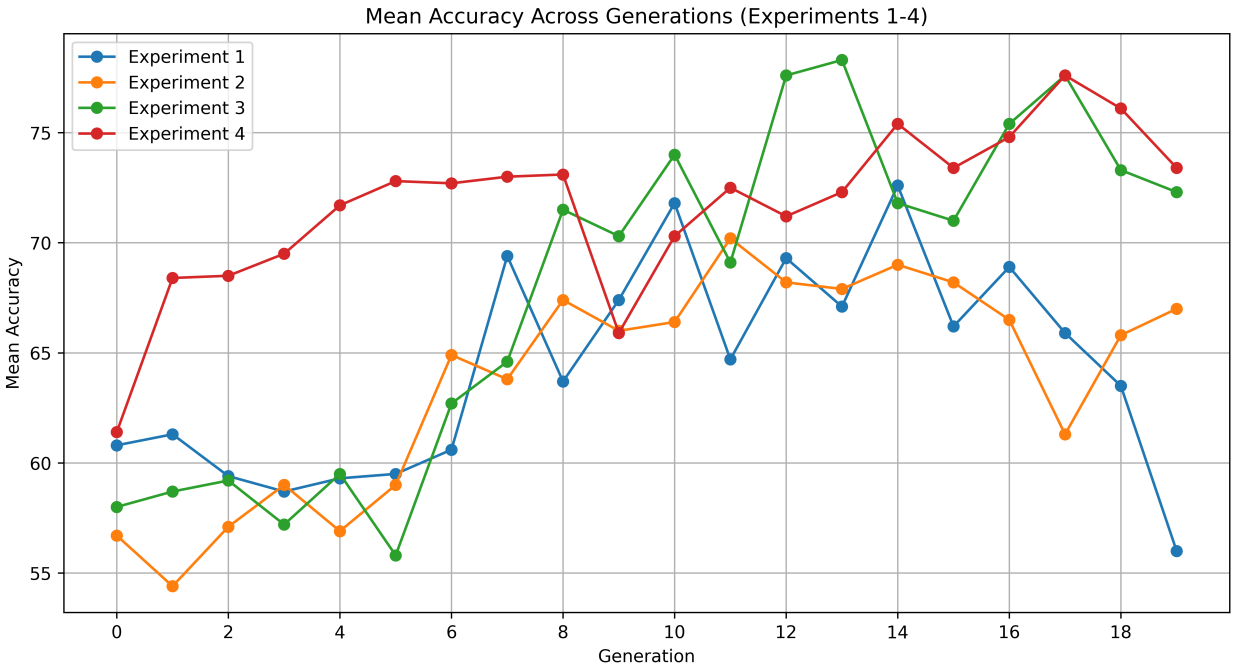

Figure 3. Mean accuracies across generations for experiments 1-4 (high hyper-mutation rate)

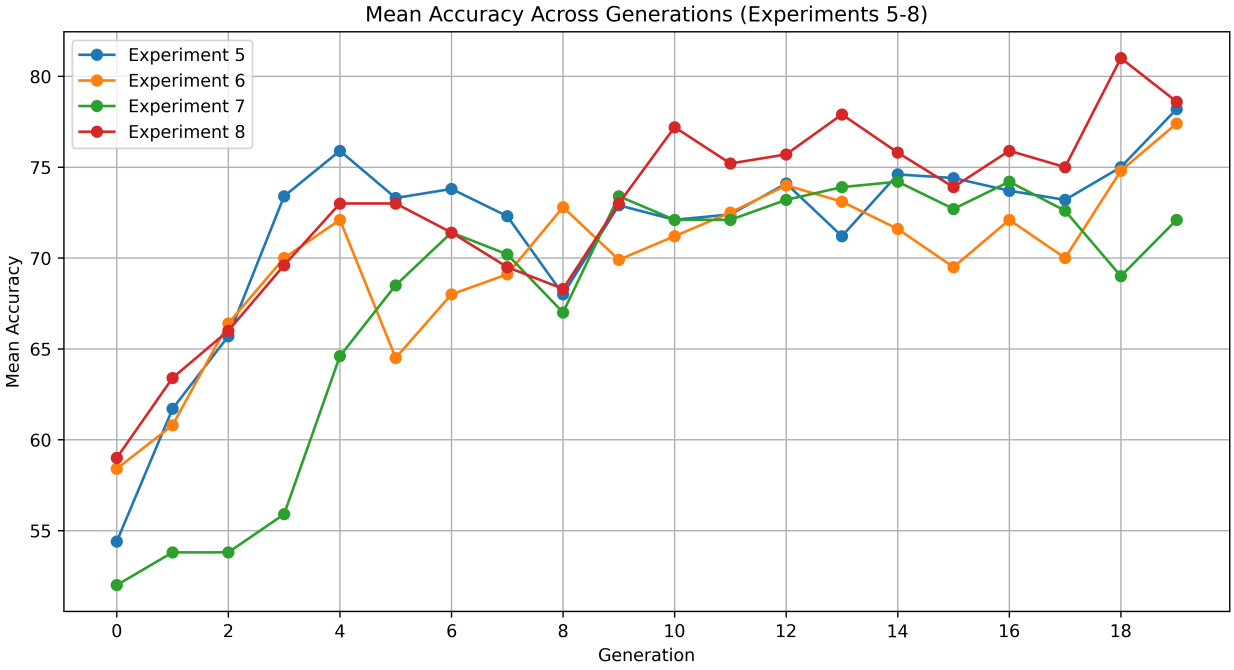

Figure 4. Mean accuracies across generations for experiments 5-8 (low hyper-mutation rate)

- **Use of Visual Aids** (N/A, Use of Diagrams, Use of Graphs, Use of Animated Illustrations, Use of Real-world Images).

Additionally, the following settings were appended to the original components:

- **Instruction and Reasoning Approach**: Provide Analogical Reasoning
- **Creativity**: Innovative Thinking
- **Examples Used**: Adaptive Examples, Curated and Progressive Examples
- **Persona**: Experienced Mathematician

The best chromosome for experiment 1 is [2, 2, 1, 3, 0, 0, 0, 0], notably containing none of the added settings, and its prompt which scores 86% is:

> Imagine you are a determined learner engaged in a low creativity simulation of a dialogue between two people to solve a math word problem. Consider using a single example as reference, if needed. Here's the problem [PROBLEM]. Now, let's solve it.

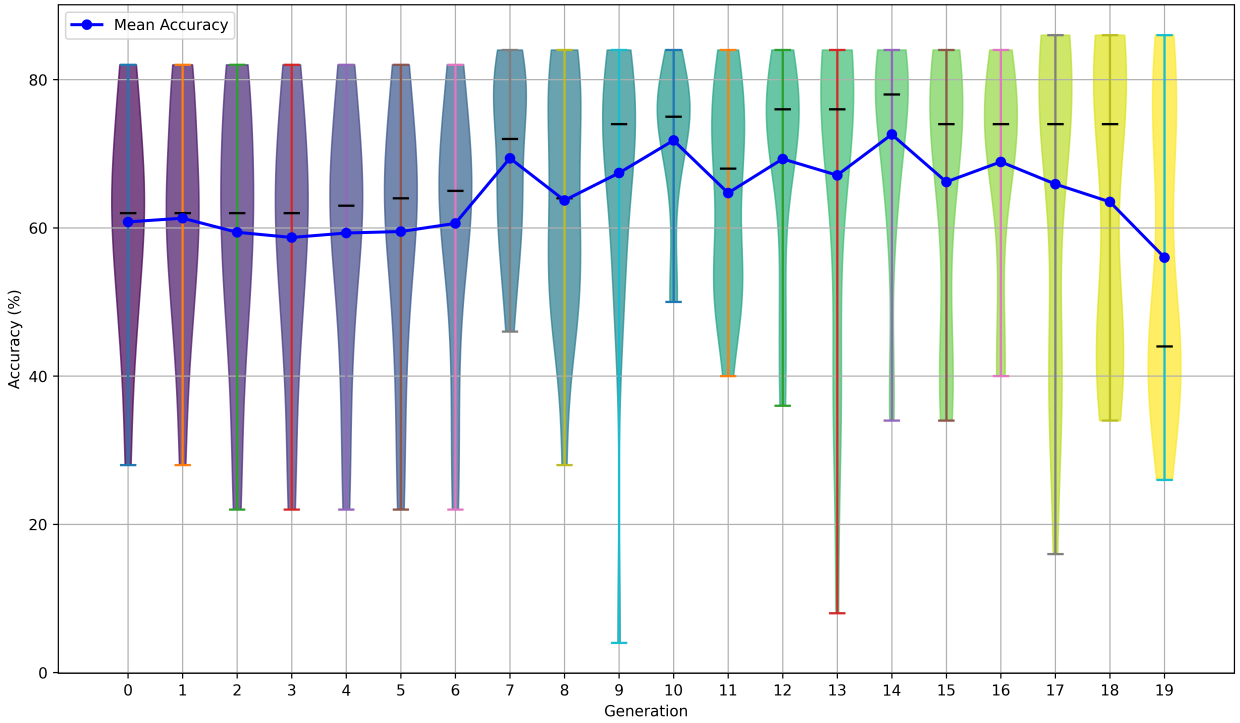

Figure 5. Population accuracy across generations for experiment 1

In contrast, Figure 6 shows the change in accuracy distributions for experiment 8, which uses a much smaller hyper mutation rate and produces the highest final prompt which scores 90%, and even generates a score of 86% in the initial population! While variation in median and accuracy persists through the generations, we observe relative stability and by generation 20, the majority of the population converges to relatively high performance. However, as with experiment 1, we do not yet know how the distributions may change if it were allowed to continue beyond the 20 generations. In contrast to experiment 1, the variation within distributions reduces over time. Interestingly, with the lower hyper-mutation rate, the overall settings do not even reach the maximum size by the end of the 20 generations (see Table 3). Two additional components are added, and one setting to Persona is added.

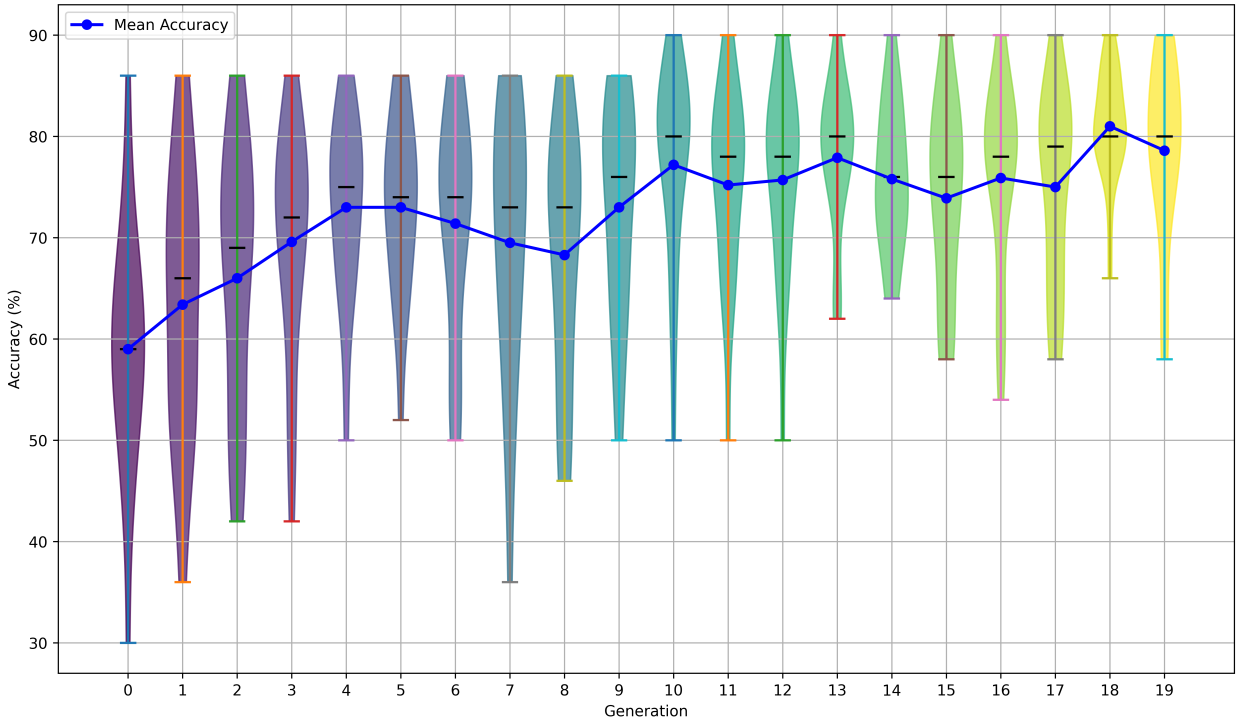

Figure 6. Population accuracy across generations for experiment 8

The best performing chromosome from experiment 8 was [2, 0, 0, 4, 2, 3] and its corresponding prompt is:

Index	Instruction and Reasoning Approach	Creativity	Examples Used	Persona	Assumptions Made	Difficulty Level
1	N/A	N/A	N/A	N/A	N/A	N/A
2	Step-by-step guidance	High creativity	Single example	Grade school math teacher	No assumptions made	Easy
3	Simulate dialogue between two people	Low creativity	Two examples	High-achieving student	Assumes given information is accurate	Intermediate
4	Outline common pitfalls	Lateral thinking	Three examples	Determined learner	Assumes given information is complete	Hard
5				Experienced Math Tutor	Makes reasonable assumptions	Very Hard

Table 3. Settings after 20 generations for Experiment 8

> "As an experienced math tutor, imagine you are explaining the solution to a hard level math word problem to a student, discussing each step and reasoning as if you were having a conversation. While explaining, assume that the given information is accurate."

Notably, the best-performing prompt in Experiment 8 omits the use of examples, despite the well-documented effectiveness of few-shot prompting. This suggests that while few-shot strategies often enhance performance, they do not necessarily yield the highest performance in all cases. It also reinforces the importance of including N/A settings, allowing the model to disregard components that may be unnecessary—an outcome that supports our initial hypothesis.

To better understand which components and settings contribute most to high performance, we analyzed the full chromosome history from all generations of Experiment 8. We identified 143 chromosomes that achieved over 80% accuracy, among which there were 30 unique configurations. Using the open-source software Gephi, we visualized the co-occurrence of component settings (Figure 7). In the graph, each node represents a specific setting, with node and edge sizes proportional to their frequency of appearance. The six node colors correspond to the six components listed in Table 3.

Among the high-performing prompts, the most frequent setting by far was "Simulate dialogue between two people", appearing in 133 of 143 high-performing chromosomes and in 24 of the 30 unique ones. This serves as a strong validation of our DUCC approach introduced in Chapter 4, particularly in light of its apparent under-performance in Chapter 5. These findings suggest that simulated dialogue may warrant its own dedicated component in future iterations, potentially with more granular sub-settings. It also appears to be a flexible and synergistic strategy, exhibiting high co-occurrence with multiple settings across all other

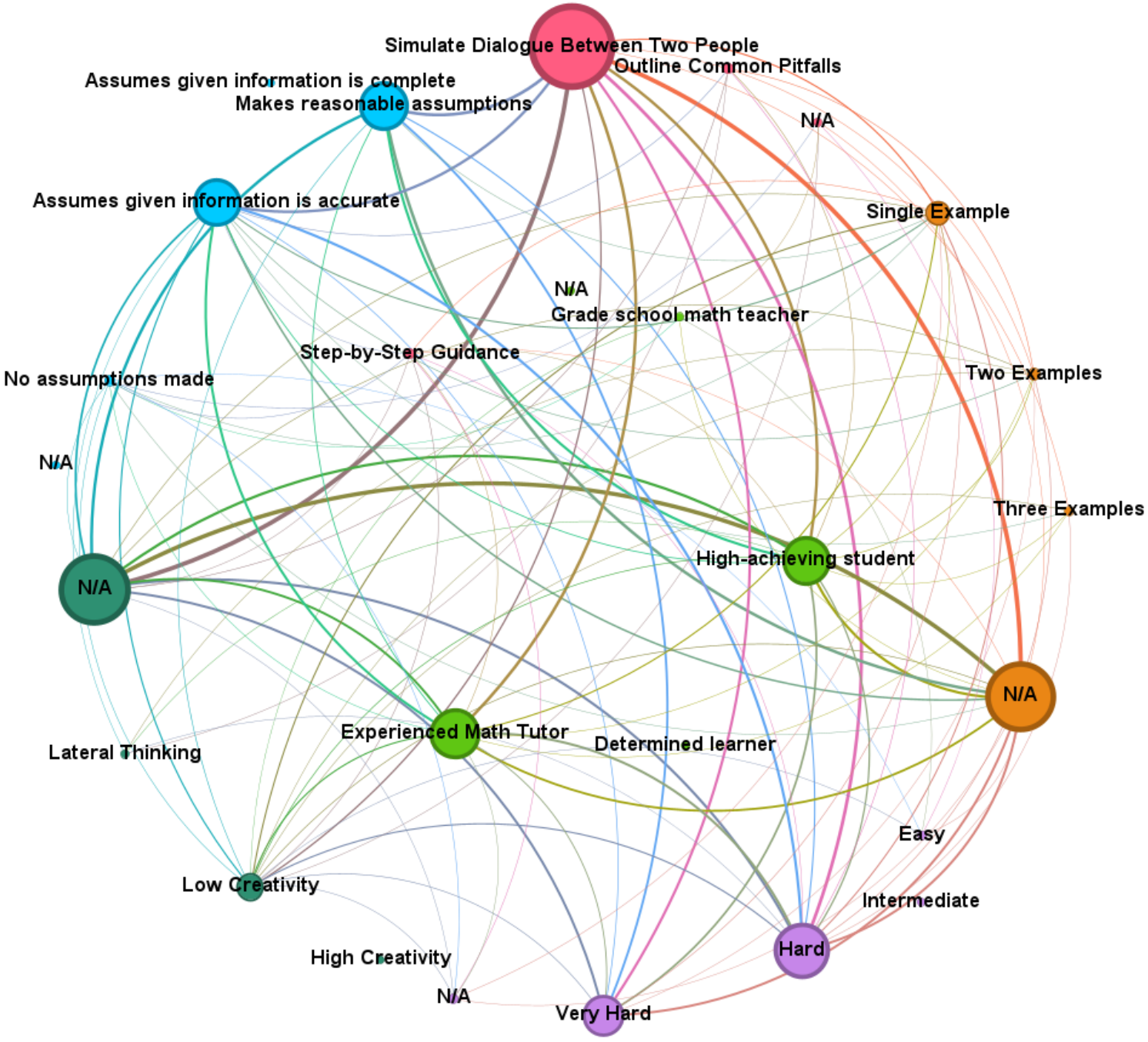

Figure 7. Network illustration of chromosome settings which scored $\geq 80\%$

components.

We also observe that the N/A settings for Examples (orange) and Creativity (dark green) are dominant among top-performing prompts, suggesting these components may be less critical or even unnecessary in certain tasks. For the Persona component, no substantial performance difference is observed between the Experienced Math Tutor and High-Achieving Student settings, indicating either may suffice.

4 Discussion

Our results indicate that a lower hyper-mutation rate may be more effective in experiments constrained to a limited number of generations, as it facilitates convergence without excessively expanding the search space. However, one of the more striking findings is that strong performance can often be achieved from the initial population alone, provided that seed prompts are thoughtfully designed. In several experiments, the best-performing prompts appeared in the initial generation and remained competitive throughout (see Table 2). This highlights the potential for future work to focus on the design of robust initialization strategies, which could reduce or even eliminate the need for extensive evolutionary cycles.

Moreover, while population-level improvements tend to require multiple generations, improvements at the individual prompt level can occur rapidly, as shown in Figures 1 and 2. This reinforces the idea that even minimal evolutionary pressure, when paired with strong initial diversity, can yield meaningful gains.

4.1 PromptBreeder Comparison

A significant contribution of this work is to show the improvements made over its source of inspiration, PromptBreeder. When the authors PromptBreeder sought peer review on OpenReview.net [7], it was rejected on the grounds that genetic algorithms (GAs) are inherently sample inefficient. This critique is well-founded because traditional GAs rely on stochastic mutations, often producing a large number of low-quality candidates before converging on an optimal solution. This problem is exacerbated when applied to prompt engineering, where every fitness evaluation requires executing an inference on an LLM, making sample efficiency a significant concern. PromptBreeder, in particular, was affected by this inefficiency due to its reliance on an unstructured search space and a mutation strategy that lacked clear constraints or guidance. With a virtually infinite number of possible prompts, the algorithm had no mechanism for systematically refining high-quality solutions beyond selective retention.

Our approach fundamentally reframes promptimization as a combinatorial search problem, introducing a structured chromosome representation to define a finite but expressive search space. By treating promptimization as a well-defined combinatorial optimization problem rather than an open-ended evolutionary

search, we mitigate some of the inefficiencies that plagued PromptBreeder. Rather than generating arbitrary prompt variations, our method decomposes prompts into structured components with predefined sets of values, dramatically reducing the number of possible candidates. This ensures that every generated prompt exists within a constrained, meaningful space, increasing the likelihood that a mutation results in an improved candidate. Furthermore, the chromosome representation is highly interpretable, allowing human evaluators to clearly identify which components and settings contribute to success. Additionally, rather than relying solely on stochastic mutations, we employ contribution scores to guide the search toward components and settings that have historically correlated with higher fitness. By incorporating hyper-mutation strategies that adapt the representation itself over time, we allow the search space to expand dynamically while maintaining control over its growth, preventing premature convergence without requiring an unbounded number of evaluations.

Despite these improvements, it would be inaccurate to claim that our approach completely eliminates sample inefficiency. Fitness evaluations remain computationally expensive since each candidate prompt must be assessed through an LLM inference, and while our structured search space reduces the number of meaningless candidates, some settings may still contribute only marginally to performance. Additionally, our approach, like any GA, relies on maintaining a diverse population to prevent premature convergence, necessitating a large number of evaluations per generation. While PromptBreeder's reliance on an ill-defined search space led to severe inefficiencies, our approach does not eliminate the underlying constraints of evolutionary algorithms, though it does mitigate them.

Compared to gradient-based methods, which directly adjust model parameters based on differentiable loss functions, our approach remains indirect in its optimization. Since we operate in a black-box setting where LLMs cannot be fine-tuned during the prompt optimization process, we lack the ability to systematically adjust prompts with the same efficiency that reinforcement learning or Bayesian optimization might afford. However, these alternatives require direct access to model internals, making them infeasible for practical use in most large-scale LLM applications.

Future work should explore additional ways to improve efficiency without sacrificing the flexibility and interpretability that GAs offer. One possibility is the introduction of early stopping mechanisms that terminate low-performing candidates before a full evaluation is conducted, reducing unnecessary computations. Another is an adaptive mutation strategy that dynamically adjusts mutation probabilities based on performance trends, ensuring that search efforts focus on the most promising areas of the space. Hybrid methods that integrate Bayesian optimization with GAs could also provide a more refined balance between exploration and exploitation, further mitigating sample inefficiency while maintaining the ability to explore novel prompt structures.

Ultimately, while our structured search method significantly improves upon the inefficiencies of PromptBreeder, it does not fully escape the fundamental limitations of sample-based optimization. However, by imposing structure on the search space, leveraging contribution scores for guided mutations, and introducing a dynamic expansion mechanism for the chromosome representation, we create a more efficient and scalable framework for promptimization—one that provides a meaningful step toward overcoming the longstanding efficiency challenges of genetic algorithms in this domain.

To emphasize Promptimizer's success, consider that the performance of this same GSM8K benchmark using this same Mixtral LLM with 5-shot learning (showing 5 successful examples in their prompt) was only 58.4% [1]! Even in our worst performing experiment (experiment 3), the best initial prompt had a score of 72%, so the best prompt from the initial population outperforms their standalone benchmark. This shows that even the simple act of creating a strong population of seed prompts can result in great success. Additionally, Promptbreeder's best performance on the same benchmark is 83.5%, using a *much* larger model, PaLM-2L.

4.2 Future Work

To thoroughly evaluate Promptimizer's capabilities, future work will involve longer experiments with more generations and expanded evaluation sets. In particular, we aim to assess performance on additional benchmarks and increase the evaluation set size beyond the current 50 samples, allowing for more robust and generalizable comparisons. This may reveal whether our results were limited by evaluation scope rather than underlying model performance.

There is also potential for machine learning methods to enhance this framework. The chromosome representation bears resemblance to the latent spaces used in models such as variational autoencoders (VAEs). In the appendix, we outline an alternative method inspired by this observation, which we were unable to test due to computational constraints. Reinforcement learning could also play a role, particularly in addressing sample inefficiency—a known limitation in evolutionary approaches.

Beyond performance optimization, we encourage readers to consider the broader utility of population-level prompting, even without explicit evolution. In contexts such as outreach to potential living kidney donors, different messages resonate with different demographic groups. This same chromosome representation can be used to systematically generate message variants at scale. For example, the LLM might be instructed: "Given the following settings, create a short advertising message about living kidney donation, and an idea for an accompanying image." Relevant components might include **Audience Age, Audience Sex, Audience Location, Action Invitation** (e.g. read more, visit a website, call a center), and **Emotional**

Appeal (e.g., heroism, sacrifice, generosity, family, etc.). Another application could focus on addressing known barriers to donation, such as the financial concerns identified in the addendum to Chapter 4. The chromosome representation allows for rapid generation of ideas grounded in domain expertise—especially valuable when experts can identify relevant attributes but cannot manually create targeted content for every audience.

If a reliable validation method can be developed that does not rely on real-world user testing—for instance, by having an LLM score messages based on specified criteria—then the same evolutionary process described in this chapter can be applied. One could even employ multiple LLMs, each simulating different personas or demographic profiles. While this has high potential, executing it well is nontrivial, as illustrated by a recent debacle at Meta.[4]

5 Conclusions

While our experiments were limited in scope due to computational constraints, Promptimizer consistently outperformed both the baseline Mixtral model and Google's PromptBreeder. Like PromptBreeder, our method demonstrates that evolving a population of prompts—rather than refining a single one—can yield superior performance. However, we advance this approach by formalizing it within a genetic algorithm framework, enabling systematic exploration and reproducibility.

To our knowledge, all prior work in prompt engineering, including informal promptimization strategies, has focused on optimizing performance directly within the vast and unstructured space of all possible prompts. This work is the first to treat prompt engineering as a combinatorial optimization problem, enabling the use of meta-heuristic search over a dramatically reduced and structured search space. Our results show that meaningful prompt representations can be constructed and optimized using established techniques from combinatorial optimization.

We propose that combinatorial promptimization represents a novel and promising research direction—one that warrants rigorous exploration. We invite experts in optimization, combinatorics, and related fields to contribute their tools and perspectives to this emerging domain. With further development, this approach has the potential to significantly advance both the theory and practice of prompt engineering.

[4]High potential, but implementing this approach responsibly is challenging, especially when simulating demographic characteristics—see this criticism of Meta's chatbot project: https://www.npr.org/2025/01/12/nx-s1-5253945/why-critics-say-metas-chatbot-is-digital-blackface.

6 Appendix

For sake of brevity, the supporting functions shown in the algorithm are not all contained in detail here, but we briefly describe each function's purpose. *InitializePopulation* creates the starting population using the settings shown in Table 1. The *EvaluatePopulation* function uses the prompt on each of the 50 math problems from our evaluation set and scores the number of correct answers. *Elitism* and *StochasticTournamentSelection* follow the pattern described at the end of the subsection headed Selection above. The *RandomCrossover* function is described in the Crossover subsection.

Algorithm 1 Genetic Algorithm with Hyper-Mutation and Stochastic Tournament Selection

```
Parameters:
    pop_size                                                          ▷ Population size
    max_generations                                                   ▷ Maximum number of generations
    num_parents                                                       ▷ Number of parents selected per generation
    mutation_rate                                                     ▷ Probability of mutation per chromosome
    hyper_mutation_rate                                               ▷ Probability of expanding chromosome representation
    elite_size                                                        ▷ Number of elite individuals preserved each generation
    tournament_size                                                   ▷ Size of tournament for selection
    max_components, max_settings                                      ▷ Limits on chromosome expansion
procedure GeneticAlgorithm(pop_size, settings, max_generations, num_parents, mutation_rate, hyper_mutation_rate, elite_size, tournament_size, max_components, max_settings)
    population ← InitializePopulation(pop_size, settings)
    fitnesses ← EvaluatePopulation(population)
    generation ← 0
    while generation < max_generations do
        elites ← Elitism(population, elite_size)
        parents ← StochasticTournamentSelection(population, fitnesses, tournament_size) + elites
        new_population ← parents
        for individual in population do
            if individual not in parents or elites then
                parent1, parent2 ← RandomSample(parents, 2)
                child ← RandomCrossover(parent1, parent2)
                child ← Mutate(child, mutation_rate, settings)
                new_population ← new_population + child
            end if
        end for
        fitnesses ← EvaluatePopulation(new_population)
        population ← ReplacePopulation(population, fitnesses, new_population)
                                                  ▷ Perform Hyper-Mutation (Structural Expansion)
        if NotAtMaxCapacity(population, max_components, max_settings) then
            for individual in population do
                if individual not in elites and Random() < hyper_mutation_rate then
                    individual.chromosome, individual.prompt, mutation_type ← HyperMutate(individual, settings, max_components, max_settings)
                end if
            end for
        end if
        generation ← generation + 1
    end while
    best_chromosome ← GetBestChromosome(population, fitnesses)
    return best_chromosome, max(fitnesses)
end procedure
```

References

[1] M. AI. Mixtral of experts. https://mistral.ai/news/mixtral-of-experts/. Accessed: Jul. 09, 2024.

[2] Z. Beheshti and S. M. H. Shamsuddin. A review of population-based meta-heuristic algorithms. *Int. j. adv. soft comput. appl*, 5(1):1–35, 2013.

[3] A. Chen, D. M. Dohan, and D. R. So. EvoPrompting: Language Models for Code-Level Neural Architecture Search, Oct. 2023.

[4] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. P. Xing, and Z. Hu. RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning, Oct. 2022.

[5] Y. Dong, K. Luo, X. Jiang, Z. Jin, and G. Li. PACE: Improving Prompt with Actor-Critic Editing for Large Language Model, Aug. 2023.

[6] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel. Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution, Sept. 2023.

[7] C. Fernando, D. S. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2024.

[8] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang. Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers, Sept. 2023.

[9] A. Y. Lam and V. O. Li. Chemical-reaction-inspired metaheuristic for optimization. *IEEE transactions on evolutionary computation*, 14(3):381–399, 2009.

[10] OpenAI. GPT-4 Technical Report, Mar. 2023.

[11] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng. Automatic Prompt Optimization with "Gradient Descent" and Beam Search, May 2023.

[12] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi. Gsa: a gravitational search algorithm. *Information sciences*, 179(13):2232–2248, 2009.

[13] L. Reynolds and K. McDonell. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. (arXiv:2102.07350), Feb. 2021.

[14] H. Sun, X. Li, Y. Xu, Y. Homma, Q. Cao, M. Wu, J. Jiao, and D. Charles. AutoHint: Automatic Prompt Optimization with Hint Generation. (arXiv:2307.07415), Aug. 2023.

[15] X. Wang, C. Li, Z. Wang, F. Bai, H. Luo, J. Zhang, N. Jojic, E. P. Xing, and Z. Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization, 2023.

[16] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large Language Models Are Human-Level Prompt Engineers, Mar. 2023.