MultiGA: Leveraging Multi-Source Seeding in Genetic Algorithms

Isabelle Diana May-Xin Ng[1], **Tharindu Cyril Weerasooriya**[2], **Haitao Zhu** [2], **Wei Wei**[2]
[1]University of California, Berkeley
[2]Center for Advanced AI, Accenture
isabelle.ng@berkeley.edu, t.weerasooriya@accenture.com

Abstract

Large Language Models (LLMs) are widely used across various research domains to tackle complex tasks, but their performance can vary significantly depending on the task at hand. Compared to fine-tuning, inference-time optimization methods offer a more cost-effective way to improve LLM output. Evolutionary algorithms can be used to refine solutions iteratively, mimicking natural selection. To the best of our knowledge, there has not been exploration on leveraging the collective capabilities of multi-source seeding for LLM-guided genetic algorithms. In this paper, we introduce a novel approach, MultiGA, which applies genetic algorithm principles to address complex natural language tasks and reasoning problems by sampling from a diverse population of LLMs to initialize the population. MultiGA generates a range of outputs from various parent LLMs, open source and closed source, and uses a neutral fitness function to evaluate them. Through an iterative recombination process, we mix and refine these generations until an optimal solution is achieved. Our results show that MultiGA converges to the accuracy of the LLM best fit for the task, and these insights lay the foundation for future research looking closer at integrating multiple LLMs for unexplored tasks in which selecting only one pre-trained model is unclear or suboptimal. We benchmark our approach using text-to-SQL code generation tasks, trip planning, GPQA benchmark for grad-level science questions, and the BBQ benchmark that measures bias in models. This work contributes to the growing intersection of evolutionary computation and natural language, highlighting the potential of biologically inspired algorithms to improve generative artificial intelligence selectivity and accuracy.

1 Introduction

The development of small language models (SLMs) and pretrained language models (PLMs) marked early progress in natural language processing, opening new possibilities for text understanding and generation. Models such as BERT [4] and RoBERTa [12] demonstrated the power of large-scale pretraining, while ULMFiT [6] showcased the utility of fine-tuning for downstream tasks. However, these models struggle with unfamiliar prompts and PLMs often require extensive task-specific engineering, which limits their applicability [33]. The emergence of large language models in 2020 marked a turning point: models such as GPT-3 demonstrated, for the first time, strong generalization across a wide range of tasks [1]. These early LLMs achieved substantially higher accuracy than smaller pretrained models on closed-book question answering benchmarks like TriviaQA [7], and showed significant gains in arithmetic reasoning and other challenging domains [1].

Furthermore, LLMs support one-shot and few-shot prompting, reducing the need for fine-tuning and making them attractive for workflows that require flexible reasoning [30]. Prompting techniques

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

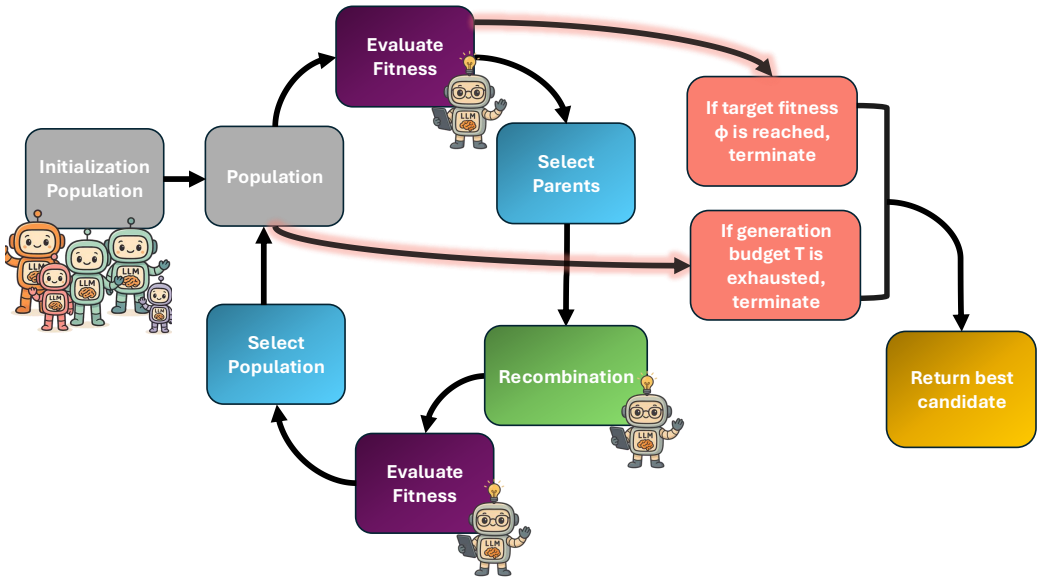

Figure 1: Overview of the MultiGA framework. Populations are initialized with multiple LLMs, while an independent LLM E handles fitness evaluation (scoring candidates) and recombination (combining two parent solutions). The process terminates once target fitness ϕ or maximum number T generations is reached. Then, the top candidate solution is returned.

such as Chain-of-Thought (CoT) [23] and Tree-of-Thoughts (ToT) [28] further enhance performance by structuring reasoning into sequential or branching steps, effectively breaking complex tasks into manageable components for the LLM to interpret. This idea of decomposing tasks has driven the rise of multi-agent workflows, particularly in industry applications. Consider text-to-SQL: solving a single query may require preprocessing natural language, linking question terms to database schema, generating SQL code, and validating outputs. Complex pipelines like this often adopt multiple agents, where each specialized LLM agent handles a subtask. Nevertheless, accuracy remains a persistent challenge, especially for developers who rely on open-source models or have limited access to task-optimized models. This challenge is especially evident when it is unclear which model is best suited for novel problems requiring interdisciplinary skills, a difficulty further compounded by the heterogeneity of LLMs, whose outputs vary significantly depending on their source.

The quality and performance of LLMs vary based on how they are trained, leading to complementary strengths and weaknesses. Beyond large-scale pretraining, most modern systems incorporate instruction tuning and reinforcement learning from human feedback (RLHF) to align with user intent [15]. Open families such as LLaMA apply instruction tuning and lightweight RLHF to achieve strong performance with smaller parameter budgets [21]. Furthermore, more recent efforts, like DeepSeek R1, extend this paradigm with reinforcement learning from AI feedback (RLAIF) and prompting-based curricula to enable improved reasoning without extensive human annotation [3]. These design choices produce models with distinct advantages, pointing toward strategies that can exploit their complementarities.

With the advent of this new generation of reasoning models, inference-time self-improvement has emerged as a popular application. Techniques such as Self-Refine allow LLMs to generate an initial answer, critique it, and iteratively refine the response, yielding better results without additional supervised training or RLHF [13]. Self-reflection has also been used in agentic frameworks where agents enhance problem solving by analyzing incorrect outputs, diagnosing errors, and adjusting future reasoning paths [18]. Furthermore, reflection has also been extended to the concept of LLMs-as-Judges, where one model critiques or evaluates the outputs of another [10]. This approach reduces reliance on costly human annotations and enables fine-grained feedback, making it a powerful tool in agentic settings.

Taken together, the diversity in model training and inference strategies suggests that different LLMs contribute unique strengths. Incorporating multiple LLMs within the same framework creates a

broader and more diverse solution space. Concepts from evolutionary computation provide a natural lens for exploring this diversity, as variation and recombination can be used to amplify strong solutions and suppress weaker ones. Building on this perspective, our key contributions in this paper are:

1) **MultiGA framework**: We propose **MultiGA**, a genetic optimization framework that seeds the initial population with outputs from multiple LLMs and employs an independent evaluator LLM.

2) **Empirical validation**: We demonstrate that MultiGA reduces reliance on any single model by expanding the candidate pool and exploiting complementary models, offering a practical approach for achieving high accuracy across diverse tasks.

3) **Foundation for future work**: We establish a basis for exploring how ensembles of diverse strong models can further improve performance on interdisciplinary and novel domains.

2 Related Works

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) draw inspiration from natural selection and have demonstrated strong applicability not only in machine learning but also across a wide range of academic disciplines [5]. Candidate solutions are generated and refined through a process of evaluation, selection, and recombination, including crossover and mutation. Typically, the initial population is generated at random, and each solution is evaluated using a fitness function to determine which candidates are retained and combined with others to produce new "children". The appeal of using these algorithms lies in the ability to efficiently navigate large, complex search spaces and identify high-quality solutions, rather than being limited to a single output. As newer generations of solutions are added into the population, older and less fit solutions are retired. Traditionally, evolutionary algorithms employ predefined rules for crossover and mutation—often arbitrary or randomized—and rely on fixed algorithmic fitness functions to evaluate candidate solutions.

2.2 Genetic Algorithms and Programs

Genetic algorithms (GA), which usually operate on simpler fixed-length encodings like bitstrings or arrays, are often used for optimization tasks where the solution can be encoded as a sequence of values. GAs search the solution space efficiently and have been widely applied in domains such as reinforcement learning [24] and prompt optimization [20]. Genetic programming (GP), which evolves tree-structured representations, has specifically been used for Automatic Heuristic Design to automatically generate and adapt decision-making rules tailored to specific tasks [2]. These processes have advanced with the introduction of LLM-based evolutionary program search, where large language models guide the exploration, refinement, and assessment of complex programs in place of these less generalizable, rule-based frameworks [29, 19].

2.3 LLM-Guided Evolutionary Algorithms

Recent work has explored the intersection of evolutionary algorithms and large language models in several ways. LLMs have been employed as mutation operators within evolutionary frameworks, leveraging their semantic understanding to generate more meaningful variations [9]. Evolutionary approaches have also been applied to neural architecture search, automatically discovering optimal network structures [25]. In the domain of automated algorithm design, Quality-Diversity algorithms such as MAP-Elites have been used to discover diverse collections of high-performing solutions [14]. Furthermore, evolutionary prompt engineering has emerged as a method to optimize prompts for specific tasks, treating prompts as evolvable entities subject to selection and variation [32]. These approaches collectively demonstrate the potential of combining evolutionary computation with modern language models to automate and enhance the design of algorithms, heuristics, and prompts.

3 Methodology

3.1 Overview

We introduce the **Multi-Source Genetic Algorithm** a framework designed to harness strengths from a diverse pool of LLMs. Rather than relying on a single model for both generation and evaluation, MultiGA seeds its initial population with outputs from multiple LLMs, each bringing unique variations shaped by its training and architecture. These candidate solutions are then iteratively recombined and refined through an independent evaluator model, which provides unbiased feedback and guides the selection of stronger offspring. MultiGA aims to create a search process that converges toward high-quality solutions while preserving diversity in the solution space, which will be explained in the following sections.

3.2 Problem Setup and Notation

We assume a task specification $\mathcal{Q}$ and a solution space $\mathcal{X}$. A set of generator LLMs $\mathcal{G} = \{g_1, \ldots, g_m\}$ produces candidate solutions $x \in \mathcal{X}$. A single LLM E serves as both the independent evaluator (assigning each candidate a fitness score in $[0, 1]$; see §3.4) and the recombination engine that synthesizes children from two parent solutions given $\mathcal{Q}$. At generation t, the population is $P_t = \{x_1, \ldots, x_n\}$ of size n. We use a retirement threshold $\tau \in [0, 1]$, select the top-k parents ($k \leq n$), and pair each parent with a uniformly sampled mate from $P_t \setminus \{x\}$ (no self-pairing). Early stopping triggers when either a target fitness ϕ is reached or a generation budget T is exhausted (See algorithm pseudocode §1).

3.3 Population Initialization

MultiGA begins by constructing the initial population P_0 by sampling outputs from multiple heterogeneous LLMs, rather than repeatedly drawing from a single model. Each model is prompted in a consistent manner and provided with all task-relevant information needed to produce a strong solution, including positive and negative examples when available. This uniform prompting ensures fairness across models while still allowing their diverse inductive biases to shape the initial population. As a result, multi-source seeding expands the diversity of the search space and reduces over-reliance on any single model's output distribution.

3.4 Evaluation and Selection

In order to guide the search toward progressively more accurate solutions, we apply a fitness evaluation to every candidate in the population at each iteration of the algorithm. At generation t, the population is represented as $P_t = \{x_1, x_2, \ldots, x_n\}$. Each candidate x_i is assigned a fitness score by E, defined as

$$f : \mathcal{X} \to [0, 1], \quad f(x_i) = s_i,$$

where $s_i = 1$ corresponds to a perfect solution and $s_i = 0$ indicates an invalid one. The evaluator E is provided with all relevant task information (e.g., the query, context, or constraints) and is instructed to judge the overall correctness and quality of the candidate. This approach ensures that scoring is based on semantic adequacy rather than superficial string similarity.

Once each candidate x_i is assessed, all candidates are ranked accordingly. We select the top-k candidates, denoted $S_t = \{x_{(1)}, \ldots, x_{(k)}\}$, which serve as the most promising parents for recombination (see §3.5) . Those with $s_i < \tau$, for a threshold $\tau \in [0, 1]$, are retired to prevent low-quality solutions from propagating into future generations.

To introduce variability while maintaining strong lineages, each parent $x_{(j)} \in S_t$ is paired with a mate drawn uniformly at random from the rest of the population:

$$y \sim \text{Unif}(P_t \setminus \{x_{(j)}\}).$$

By excluding the parent from its own mate pool, we avoid trivial self-pairings and encourage genuine diversity in the recombination step. This design mirrors biological processes: the strongest candidates are preserved as parents, while random mating injects stochastic variation that helps the algorithm escape local optima. Together, this combination of thresholding, top-k selection, and random mating

ensures a balance between exploitation of high-fitness solutions and exploration of the broader search space.

After recombination, we check for termination: the algorithm halts if either the maximum number of generations T is reached or if the best candidate achieves the target fitness ϕ. Both T and ϕ are configurable hyperparameters that allow the trade-off between runtime and solution quality to be adjusted. If neither condition is met, candidates with fitness below the threshold τ are retired, ensuring that low-quality solutions do not propagate into the next generation.

3.5 Recombination

To generate new candidates, we employ an LLM-based crossover operator using the same independent LLM E. Given two parent solutions, E is provided with all task information along with the parent solutions and is prompted to synthesize a child that integrates the strengths of both. Unlike traditional token-level crossover rules, this operator leverages the generative capacity of LLMs to perform semantic recombination. The resulting offspring are diverse yet coherent, inheriting useful attributes from each parent while overcoming the constraints of rule-based recombination strategies.

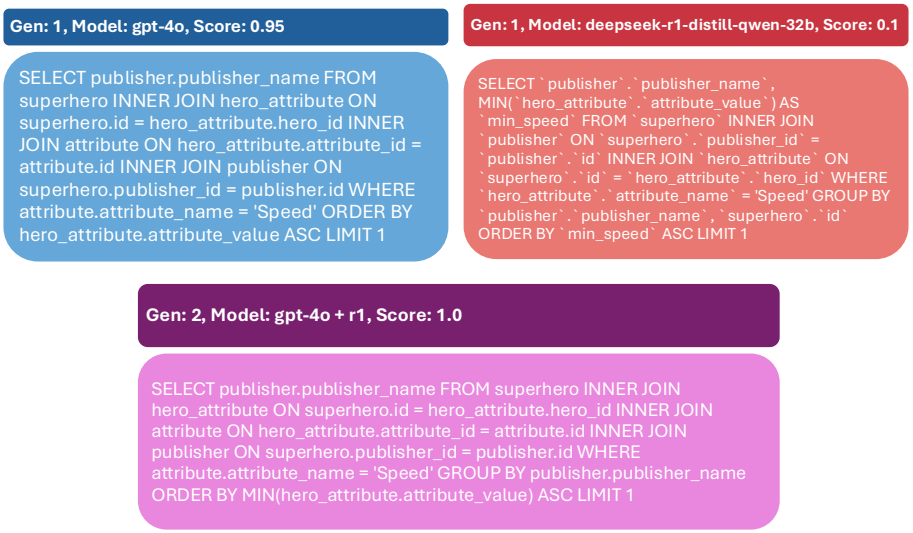

Figure 2: Example of recombination on a text-to-SQL task (**"Which publisher published the slowest superhero?"**). Parent 1 (gpt-4o, score 0.95) was paired with a randomly selected parent (deepseek-r1, score 0.1). The resulting child achieved a perfect score of **1.0** by preserving gpt-4o's overall structure while incorporating the MIN aggregation from r1.

4 Experiments

4.1 Overview

To assess whether seeding genetic algorithms with multiple LLMs enables deeper exploration, we evaluate our framework across four tasks. Our aim is not to design highly specialized or task-specific algorithms, but rather to provide a simple and general framework in which all models can be evaluated fairly. While previous GA-based methods like Mind Evolution [8] have demonstrated promising results, challenges in reproducibility motivate our focus on developing an accessible and well-documented implementation.

The purpose of these experiments is to demonstrate the potential of MultiGA for the broader research community—particularly for practitioners who may be uncertain about which model is best suited for a task, who lack the resources to exhaustively compare models, yet still want a principled way to obtain strong results.

Algorithm 1: MultiGA: Multi–Source Genetic Algorithm

Input: Task spec $\mathcal{Q}$; Generator LLMs $\mathcal{G} = \{g_1, \ldots, g_m\}$; Evaluator/crossover LLM E with fitness $f : \mathcal{X} \to [0, 1]$; Population size n; Top-k; Threshold $\tau \in [0, 1]$; Max generations T; Target fitness $\phi \in [0, 1]$.

Output: Best solution $\hat{x} \in \mathcal{X}$.

$P_0 \leftarrow$ initialize_population$(\mathcal{G}, \mathcal{Q})$;
$t \leftarrow 0$;
while $t < T - 1$ **do**
  $scores \leftarrow$ evaluate_fitness(P_t, f) ;
  if $\max(scores) \geq \phi$ **then**
    break
  $S_t \leftarrow$ select_parents$(P_t, scores, k)$;
  $\mathcal{C}_t \leftarrow$ recombination$(S_t, P_t, E, \mathcal{Q})$ *# Create next generation (children solutions)* ;
  $P_{t+1} \leftarrow$ select_population$(P_t, \mathcal{C}_t, \tau)$ *# Retire unfit candidates* ;
  $t \leftarrow t + 1$;
$scores \leftarrow$ evaluate_fitness(P_t, f) ;
$\hat{x} \leftarrow \arg\max_{x \in P_t} f(x)$;
return $\hat{x}$;

In all experiments, we used a consistent configuration across tasks to ensure comparability. We selected the top-3 ($k = 3$) parents for recombination, each run was capped at a maximum of $T = 3$ generations, we used a fitness threshold of $\tau = 0.2$ for pruning low-quality candidates, and our target fitness was $\phi = 0.95$ as the stopping criterion. These conservative settings were chosen to keep the computational cost manageable while still allowing the algorithm to demonstrate meaningful improvements over successive generations. The genetic algorithm itself is implemented as a general class that can be configured for any task using prompts defined in a task-specific configuration file. For all tasks, we designate **gpt-4o-mini** as the unbiased evaluator and recombination model, responsible for scoring candidate solutions and synthesizing offspring across generations. While the choice of evaluator was somewhat arbitrary, we required a model strong enough to serve as a reliable judge. Notably, our results indicate that the use of gpt-4o-mini did not unduly inject its own expertise, as certain models still performed poorly on the benchmarks despite its assessment and recombination guidance.

Dataset	Test Rows	Total Items	Label Choices
$\mathcal{D}_{\texttt{SQL}}$ (BIRD Mini-Dev)	100	500	∞
$\mathcal{D}_{\texttt{NATPLAN}}$ (Meeting Planning)	100	1000	∞
$\mathcal{D}_{\texttt{GPQA}}$ (Grad-Level Science Questions)	198	198	4
$\mathcal{D}_{\texttt{BBQ}}$ (BBQ Bias Data)	104	6879	3

Table 1: Datasets used in our experiments, with test set size, total available items, and number of label choices.

4.2 Baselines

As a point of comparison, we evaluate MultiGA against single-LLM baselines, where the genetic algorithm design remains identical but the initial population is seeded exclusively from one model. The individual LLMs we use are:**gpt-4o** (OpenAI), **qwen2.5-coder-32b-instruct** (Qwen), **deepseek-r1-distill-qwen-32b** (DeepSeek), **openai/gpt-oss-20b** (OpenAI OSS), and **mistral-7b-instruct-v0.3** (Mistral). These models were chosen for their accessibility and and provide a diverse set of baselines for comparison.

The "ALL" condition represents the full MultiGA framework, in which the initial population is seeded with outputs from all five models simultaneously. This design isolates the impact of diverse initialization while keeping all other components of the algorithm fixed. For comparison, we also evaluated the same population initialization prompts without applying the iterative refinement of MultiGA.

4.3 Text-to-SQL ($\mathcal{D}_{\texttt{SQL}}$)

Text-to-SQL is a critical task for enabling non-technical users to query databases and extract insights without writing code. In industry, developing cost-efficient and accurate text-to-SQL frameworks is essential for building robust agentic systems. For this experiment, we used the BIRD mini-dev dataset, which contains 500 realistic industry-level questions paired with gold-standard SQL queries [11]. We partitioned the data into training and test sets, using the training set to construct positive and negative examples for each test query. Specifically, we embedded the training data and performed cosine similarity search to retrieve relevant natural language–SQL pairs for the current question. All prompting logic and retrieval configurations were defined in a task-specific configuration file to ensure reproducibility (see Appendix A).

4.4 Meeting Planning ($\mathcal{D}_{\texttt{NATPLAN}}$)

We next applied MultiGA to structured reasoning through the meeting scheduling benchmark introduced by Zheng et al. [31]. In this task, the LLM must generate a plan that maximizes the number of valid meetings during a hypothetical trip to San Francisco. Solving it requires navigating multiple logistical constraints, such as travel distance between meeting locations, meeting durations, and participant availability. This benchmark thus provides a natural setting to test whether iterative recombination can improve complex structured outputs (see Appendix B).

4.5 GPQA Science Questions ($\mathcal{D}_{\texttt{GPQA}}$)

To broaden the evaluation of MultiGA, we tested the framework on the GPQA benchmark of graduate-level science multiple-choice questions [17]. This dataset covers advanced scientific domains and provides a challenging benchmark beyond code generation and optimization tasks. We used the Diamond subset, which contains the highest-quality questions in the benchmark (see Appendix C).

4.6 BBQ Bias Evaluation ($\mathcal{D}_{\texttt{BBQ}}$)

Finally, we evaluated whether MultiGA mitigates social bias using the BBQ benchmark [16]. We focused on the race and ethnicity subset, which contains over 6,000 examples. Each example presents an ambiguous or unambiguous scenario involving racial stereotypes, followed by a question designed to reveal whether the model exhibits biased behavior. Rather than using the benchmark in its standard multiple-choice format, we adapted it to an open-ended setting and subsequently extracted answers for accuracy evaluation (see Appendix D).

5 Results and Discussion

We evaluate accuracy on four benchmarks: Text-to-SQL ($\mathcal{D}_{\texttt{SQL}}$), GPQA Science Questions ($\mathcal{D}_{\texttt{GPQA}}$), Meeting Planning ($\mathcal{D}_{\texttt{NATPLAN}}$), and BBQ Bias ($\mathcal{D}_{\texttt{BBQ}}$). For each task, we compare seeding MultiGA with outputs from individual LLMs against seeding with all five models simultaneously ($\mathcal{G}$). Results are shown in Table 2.

Table 2: Accuracy across tasks when seeding with each individual model versus seeding with all five simultaneously ($\mathcal{G}$).

Seed Model	$\mathcal{D}_{\texttt{SQL}}$	$\mathcal{D}_{\texttt{NATPLAN}}$	$\mathcal{D}_{\texttt{GPQA}}$	$\mathcal{D}_{\texttt{BBQ}}$
gpt-4o	**0.56**	0.39	**0.44**	**1.00**
qwen2.5-coder-32b-instruct	0.45	0.28	0.30	0.87
deepseek-r1-distill-qwen-32b	0.46	0.23	0.41	0.97
openai/gpt-oss-20b	0.48	0.19	0.41	0.93
mistral-7b-instruct-v0.3	0.25	0.10	0.22	0.87
$\mathcal{G}$ (MultiGA)	0.55	**0.40**	0.40	**1.00**

Across all four benchmarks, we observe that seeding with all models simultaneously ($\mathcal{G}$) produces accuracy that consistently approaches the performance of the strongest single-model baseline.

5.1 Meeting Planning ($\mathcal{D}_{\texttt{NATPLAN}}$)

For $\mathcal{D}_{\texttt{NATPLAN}}$, the $\mathcal{G}$ configuration not only avoids collapse from weaker models but slightly outperforms the top single seed by about 1%. We can see that this task requires multi-step temporal and spatial reasoning, integrating participant constraints, availability, and task ordering. Because prior state-of -the-art work has prioritized Gemini models for this benchmark, we also tested the MultiGA framework seeded only with **gemma3-27b**. However, this resulted in 29% accuracy compared to $\mathcal{G}$ with 40%. This result demonstrates MulitGA's advantage in ambiguous settings where multiple solutions may be valid. This aligns with the broader motivation for genetic algorithms as optimization frameworks since MultiGA can iterate over potential solutions, prune out worse choices, and converge to a more optimal result.

5.2 Text-to-SQL ($\mathcal{D}_{\texttt{SQL}}$) and GPQA ($\mathcal{D}_{\texttt{GPQA}}$)

For $\mathcal{D}_{\texttt{SQL}}$, the $\mathcal{G}$ configuration (0.55) nearly matches the best single seed, gpt-4o (0.56). This task requires precise grounding in database schemas, where even small errors in column names or join conditions lead to incorrect queries. MultiGA narrows the gap to gpt-4o while providing stability against weaker models like mistral-7b (0.25) that often hallucinate schema elements. On GPQA, $\mathcal{G}$ (0.4015) falls within 0.03 of the best single seed, gpt-4o (0.4369). Since the task involves refining multiple-choice answers, there is less room for MultiGA to improve beyond single-LLM predictions. As expected, the gains here are more modest compared to the other benchmarks.

5.3 BBQ Bias Evaluation ($\mathcal{D}_{\texttt{BBQ}}$)

Finally, $\mathcal{D}_{\texttt{BBQ}}$ evaluates social bias, where weaker models often amplify stereotypes. Here, $\mathcal{G}$ achieves perfect accuracy (1.00), matching gpt-4o. Importantly, MultiGA demonstrates that combining weaker seeds does not degrade robustness on fairness-sensitive evaluation. For example, while **Mistral-7b** reached only 87% accuracy by itself, $\mathcal{G}$ pruned such biased outputs and instead preserved the judgments of less biased models through its selection mechanism. These results underscore the importance of diverse population initialization for tasks sensitive to demographic fairness, with **gpt-4o-mini** acting as an effective neutral evaluator and recombination agent, rather than a corrector of suboptimal outputs from weaker LLMs.

5.4 Robustness and Consistency

Compared to approaches such as Mind Evolution, MultiGA demonstrates a distinct strength: consistency. Mind Evolution leverages genetic algorithms to exploit inference-time compute, enhancing LLM reasoning through random exploration and convergent thinking [8]. Its reported results are impressive, with near-perfect accuracy on tasks such as Trip Planning, Meeting Planning, and TravelPlanner when powered by Gemini 1.5 Pro and Flash [31, 26]. However, when evaluated under the same setup with **gpt-4o-mini** instead of Gemini, performance drops sharply—over 50% on Trip Planning, 20% on TravelPlanner, and 13% on Meeting Planning. Because reproducible code has not been released, it remains unclear whether these gaps generalize across models.

Our objective, by contrast, was not to surpass the state of the art on each benchmark, but to examine whether combining multiple LLMs could mitigate brittleness and large performance fluctuations. With a lightweight and transparent implementation, **MultiGA** generalizes across domains without relying on the strengths of a single model. Notably, when we tested replacing **Mistral-7B-Instruct** with **GPT-5** for $\mathcal{G}$ on the meeting planning task, accuracy decreased from 40% to 36%. This result highlights that Mistral's value lay in its diversity rather than raw performance, even though it was the weakest model individually. More broadly, it demonstrates that exchanging a smaller model with a **larger, more expensive model** is not required for MultiGA to achieve strong and reliable results, underscoring its efficiency as well as its effectiveness.

Furthermore, MultiGA is not expected to consistently outperform the best individual seed, since the initial population may include far weaker candidates that must be discarded. Yet its consistent ability to converge toward near-best performance highlights how evaluator-guided selection and recombination amplify strong solutions while eliminating poor ones. This makes MultiGA a robust choice when the most suitable model for a task is unknown or less accessible, adaptively closing the gap to the strongest baseline without manual tuning.

6 Conclusion

In this paper, we introduced MultiGA, a novel framework that applies genetic algorithm principles to enhance the performance of LLMs on complex reasoning tasks. Our central contribution is the use of multi-source seeding, which initializes the candidate population with outputs from a diverse set of LLMs, combined with an independent LLM to serve as a neutral evaluator and recombination engine. This approach directly addresses the growing challenge of model selection in a landscape populated by numerous LLMs with varied strengths and weaknesses.

Our experiments across four distinct benchmarks—text-to-SQL generation, complex meeting planning, graduate-level scientific reasoning, and social bias evaluation demonstrate that MultiGA is a robust and effective strategy. The framework consistently converges to a solution that rivals the accuracy of the best-performing individual model. By systematically pruning weaker solutions and semantically recombining the strengths of stronger ones, MultiGA provides a reliable method for achieving high-quality results without requiring prior knowledge of which model is best suited for a particular task.

Future studies could investigate the impact of the evaluator model's capabilities on the evolutionary process or explore dynamic seeding strategies where the pool of generator LLMs is adapted over time. Finally, extending the MultiGA framework to more complex, multi-step agentic workflows could unlock new possibilities, allowing different models to contribute specialized strengths at various stages of a problem-solving process.

Limitations

While MultiGA demonstrates the potential of combining the perspectives of different models, some limitations remain that suggest directions for future work. First, our evaluation focuses on a limited set of benchmarks—Text-to-SQL, Meeting Planning, GPQA Science, and BBQ Bias—which, although diverse, do not capture the full range of reasoning and generative tasks. Expanding to additional evaluations such as SuperGLUE or multi-hop reasoning datasets could further test the framework's generality [22, 27]. We also wish to increase dataset sizes and explore how deeper generational runs correlate with population diversity and task complexity.

Second, we primarily used **gpt-4o-mini** as the evaluator for consistency and cost efficiency. Although effective for evaluation and recombination, different evaluator architectures or mixtures of evaluators could lead to different evolutionary trajectories. Evaluator choice therefore remains a potential source of bias, and exploring alternative evaluators may improve our understanding of multi-source genetic optimization. Despite these constraints, MultiGA's transparency and modularity make it a promising framework for orchestrating heterogeneous LLMs toward more reliable reasoning.

Acknowledgments

This work was initiated during the author's internship at Accenture in Summer 2025. We thank Accenture's Center for Advanced AI for providing computational resources, and acknowledge NVIDIA for the availability of open-source models. Finally, we thank the anonymous reviewers for their valuable input and the wider academic community for its support.

References

[1] Tom B. Brown, Benjamin Mann, and Nick Ryder. Language Models are Few-Shot Learners, July 2020. URL `http://arxiv.org/abs/2005.14165`. arXiv:2005.14165 [cs].

[2] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, and John Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1559–1565, New York, NY, USA, July 2007. Association for Computing Machinery. ISBN 978-1-59593-697-4. doi: 10.1145/1276958.1277273. URL `https://doi.org/10.1145/1276958.1277273`.

[3] DeepSeek-AI, Daya Guo, Dejian Yang, and Haowei Zhang. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, January 2025. URL http://arxiv.org/abs/2501.12948. arXiv:2501.12948 [cs].

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv:1810.04805 [cs].

[5] David E. (David Edward) Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass. : Addison-Wesley Pub. Co., 1989. ISBN 978-0-201-15767-3. URL http://archive.org/details/geneticalgorithm0000gold.

[6] Jeremy Howard and Sebastian Ruder. Universal Language Model Fine-tuning for Text Classification, May 2018. URL http://arxiv.org/abs/1801.06146. arXiv:1801.06146 [cs].

[7] Mandar Joshi, Eunsol Choi, and Daniel S. Weld. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension, May 2017. URL http://arxiv.org/abs/1705.03551. arXiv:1705.03551 [cs].

[8] Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving Deeper LLM Thinking, January 2025. URL http://arxiv.org/abs/2501.09891. arXiv:2501.09891 [cs].

[9] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through Large Models, June 2022. URL http://arxiv.org/abs/2206.08896. arXiv:2206.08896 [cs].

[10] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods, December 2024. URL http://arxiv.org/abs/2412.05579. arXiv:2412.05579 [cs].

[11] Jinyang Li, Binyuan Hui, and Ge Qu. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs, November 2023. URL http://arxiv.org/abs/2305.03111. arXiv:2305.03111 [cs].

[12] Yinhan Liu, Myle Ott, and Naman Goyal. RoBERTa: A Robustly Optimized BERT Pretraining Approach, July 2019. URL http://arxiv.org/abs/1907.11692. arXiv:1907.11692 [cs].

[13] Aman Madaan, Niket Tandon, and Prakhar Gupta. Self-Refine: Iterative Refinement with Self-Feedback, May 2023. URL http://arxiv.org/abs/2303.17651. arXiv:2303.17651 [cs].

[14] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, April 2015. URL http://arxiv.org/abs/1504.04909. arXiv:1504.04909 [cs].

[15] Long Ouyang, Jeff Wu, and Xu Jiang. Training language models to follow instructions with human feedback, March 2022. URL http://arxiv.org/abs/2203.02155. arXiv:2203.02155 [cs].

[16] Alicia Parrish, Angelica Chen, and Nikita Nangia. BBQ: A Hand-Built Bias Benchmark for Question Answering, March 2022. URL http://arxiv.org/abs/2110.08193. arXiv:2110.08193 [cs].

[17] David Rein, Betty Li Hou, and Asa Cooper Stickland. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, November 2023. URL http://arxiv.org/abs/2311.12022. arXiv:2311.12022 [cs].

[18] Matthew Renze and Erhan Guven. Self-Reflection in LLM Agents: Effects on Problem-Solving Performance. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pages 476–483, November 2024. doi: 10.1109/FLLM63129.2024.10852493. URL http://arxiv.org/abs/2405.06682. arXiv:2405.06682 [cs].

[19] William Shum, Rachel Chan, Jonas Lin, Benny Feng, and Patrick Lau. A Hybrid GA LLM Framework for Structured Task Optimization, June 2025. URL http://arxiv.org/abs/2506.07483. arXiv:2506.07483 [cs].

[20] Xavier Sécheresse, Jacques-Yves Guilbert-Ly, and Antoine Villedieu de Torcy. GAAPO: Genetic Algorithmic Applied to Prompt Optimization, April 2025. URL http://arxiv.org/abs/2504.07157. arXiv:2504.07157 [cs].

[21] Hugo Touvron, Thibaut Lavril, and Gautier Izacard. LLaMA: Open and Efficient Foundation Language Models, February 2023. URL http://arxiv.org/abs/2302.13971. arXiv:2302.13971 [cs].

[22] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems, February 2020. URL http://arxiv.org/abs/1905.00537. arXiv:1905.00537 [cs].

[23] Jason Wei, Xuezhi Wang, and Dale Schuurmans. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, January 2023. URL http://arxiv.org/abs/2201.11903. arXiv:2201.11903 [cs].

[24] Shimon Whiteson and Peter Stone. Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research*, 7(31):877–917, 2006. ISSN 1533-7928. URL http://jmlr.org/papers/v7/whiteson06a.html.

[25] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A Survey on Neural Architecture Search, June 2019. URL http://arxiv.org/abs/1905.01392. arXiv:1905.01392 [cs].

[26] Jian Xie, Kai Zhang, and Jiangjie Chen. TravelPlanner: A Benchmark for Real-World Planning with Language Agents, October 2024. URL http://arxiv.org/abs/2402.01622. arXiv:2402.01622 [cs].

[27] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering, September 2018. URL http://arxiv.org/abs/1809.09600. arXiv:1809.09600 [cs].

[28] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models, December 2023. URL http://arxiv.org/abs/2305.10601. arXiv:2305.10601 [cs].

[29] Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the Importance of Evolutionary Search in Automated Heuristic Design with Large Language Models, July 2024. URL http://arxiv.org/abs/2407.10873. arXiv:2407.10873 [cs].

[30] Pengyu Zhao, Zijian Jin, and Ning Cheng. An In-depth Survey of Large Language Model-based Artificial Intelligence Agents, September 2023. URL http://arxiv.org/abs/2309.14365. arXiv:2309.14365 [cs].

[31] Huaixiu Steven Zheng, Swaroop Mishra, and Hugh Zhang. NATURAL PLAN: Benchmarking LLMs on Natural Language Planning, June 2024. URL http://arxiv.org/abs/2406.04520. arXiv:2406.04520 [cs].

[32] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large Language Models Are Human-Level Prompt Engineers, March 2023. URL http://arxiv.org/abs/2211.01910. arXiv:2211.01910 [cs].

[33] Chenguang Zhu and Michael Zeng. Impossible Triangle: What's Next for Pre-trained Language Models?, April 2022. URL http://arxiv.org/abs/2204.06130. arXiv:2204.06130 [cs].

A SQL Task: System Instructions and Prompts

A.1 Prompts for Generating Initial SQL Solutions

```
init_sol_system = """You are a specialized SQL query generator. You
    ↪ receive pre-processed inputs from an upstream schema linking
    ↪ agent
and focus on generating accurate and executable SQL representations
    ↪ of a user's natural language question."""
```

Listing 1: System prompt for SQL generation.

```
init_sol_prompt_unfilled = """
Task: Generate a SQL Query based on the user's query.

An upstream agent has already:

Performed schema linking and entity resolution

Mapped user entities to database objects based on semantic search
    ↪ scores

Pre-Processed Inputs:

User Query: {query}

Linked Schema Elements: {ie_extracted}

These elements are case-sensitive

If the elements have a low semantic similarity score (below 0.6), do
    ↪ not rely solely on the mapping and double check the database
    ↪ schema

Database Context: {db_schema}

Supporting context for query construction

Reference for relationships and constraints

Domain Evidence: {evidence}

More context for accurate query logic

These are usually very helpful

Reference Date: {current_date}

Your Focus: SQL Generation Excellence

Linking Question to DB

Use the linked schema elements as your primary source for matching
    ↪ the user's question with the database schema

However, you must consult the actual database schema when semantic
    ↪ score is low

Query Construction Rules

ALWAYS use original_column_name NOT column_name from the database

Do not define aliases using the AS clause. Only use real names.

Try not to make queries that return irrelevant excess information

Edge Case Handling
```

```
If schema linking appears incomplete:

Use database context to fill gaps

Don't hallucinate or guess about column names

Pattern Learning

Follow these successful queries: {positive_examples}

Avoid these problematic queries: {negative_examples}

Output Structure:

Return the SQL query only, no markdown formatting and DO NOT wrap in
    ↪ ```sql

Example ouput (use this to see the required structure): {
    ↪ output_example}

DO NOT WRAP WITH TRIPLE BACKTICKS
"""
```

Listing 2: User prompt template for initial SQL query generation.

A.2 Prompts for Crossover Phase

```
crossover_system = """You are tasked to perform the crossover in a
    ↪ genetic algorithm aimed at creating a correct SQL query to
    ↪ match a user's question. """
```

Listing 3: System prompt for SQL crossover.

```
crossover_prompt_unfilled = """ You are given two SQL queries, each
    ↪ attempting to express a user's natural language question
    ↪ against a specific database.
Your task is to analyze both queries and synthesize a new SQL query
    ↪ that combines the strengths of each, mimicking the crossover
    ↪ operation in genetic algorithms.
The objective is to produce a 'child' query that more effectively and
    ↪  accurately translates the user's intent: {query}.

Here are the parent SQL queries:

{{parent_1}}

{{parent_2}}

Database Context: {db_schema}

Supporting context for query construction

Reference for relationships and constraints

Domain Evidence: {evidence}

Business context for accurate query logic

Helps with aggregation and filtering decisions

Consider the following when generating the child SQL query:
```

```
Evaluate whether certain filters are necessary and relevant to the
    ↪ user's intent

Ensure that if data is required from multiple tables, appropriate
    ↪ JOIN operations are included in the SQL query

Output Structure:

Return the SQL query only, no markdown formatting and DO NOT wrap in
    ↪ ```sql

DO NOT WRAP WITH TRIPLE BACKTICKS """
```

Listing 4: User prompt template for SQL crossover synthesis.

A.3 Prompts for Objective Function (Evaluation Phase)

```
system_instructions = """Given a user's question and a SQL query, you
    ↪  are tasked to determine how well the SQL captures
         the meaning of the question and appropriately curates the SQL
            ↪  query with respect the given database. """

feedback_prompt = """
Task: Evaluate the correctness of the following SQL query in
    ↪ capturing the intent of the user's question {user_query}.

You are given a SQL Query below and the output (which may be
    ↪ truncated) of querying it on the appropriate database:

```sql
{sql}

Output: {output}

More helpful context and information is listed below:

Database Context: {db_schema}

Supporting context for query construction

Reference for relationships and constraints

Domain Evidence: {evidence}

Business context for accurate query logic

Helps with aggregation and filtering decisions

Reference Date: {current_date}

Your Focus: Quality Assessment

Compare the user's question, the generated SQL query, and the outcome
    ↪  of the SQL query on the database.

If there is an error in the SQL output, that is an indication of a
    ↪ poorly crafted query.

If there is not an error, ensure that the evidence is being correctly
    ↪  applied, and that all filters and conditions are actually
    ↪ needed.
Also please check the source table for each column and ensure proper
    ↪ JOIN operations are included wherever required.
```
```

```
You may find the following SQL rules helpful.

GROUP BY or ORDER BY statements

If a variable is used in the GROUP BY or ORDER BY clause, it must
    ↪ also appear in the SELECT statement.

You should only group or order by a column if it is included in the
    ↪ SELECT statement.

JOINS

Ensure that if you are taking information from MORE than one table,
    ↪ you are doing a JOIN! This is a common mistake where LLMs are
    ↪ forgetting to do JOINs.

Always specify the join condition explicitly using ON table1.column =
    ↪  table2.column and avoid Cartesian products.

Rate the SQL query on a scale from 0.00 to 1.00, where 1.00
    ↪ represents a perfect match to the natural language question
    ↪ and 0.00 represents a very poor match.
Respond with only a single float rounded to two decimal places. DO
    ↪ NOT INCLUDE OTHER TEXT. Please only return the float in your
    ↪ output.

Example Outputs:

A nearly perfect SQL translation with only a minor flaw -> 0.90

An incorrect SQL translation that barely matches the question -> 0.15

DO NOT INCLUDE OTHER TEXT. Please only return the float in your
    ↪ output.
"""
```

Listing 5: System and feedback prompts for SQL objective function evaluation.

B Meeting Planning Task: System Instructions and Prompts

B.1 Prompts for Generating Initial Solutions

```
init_sol_system = """You are a specialized meeting planner. You focus
    ↪  on generating meeting plans that optimize the number of
    ↪ meetings without violating any constraints."""
```

Listing 6: System prompt for initial solution generation.

```
init_sol_prompt_unfilled = """
Task: Generate a valid and optimized meeting plan for the user.

# Pre-Processed Inputs:
Constraints: {constraints}
- These include: person name, meeting location, availability window,
    ↪ and required meeting duration
- Ensure all constraints are respected and not violated

Distance Matrix: {dist_matrix}
- Travel times between key locations in minutes
- Use these to compute realistic travel steps in the plan
```

```
# Instructions
1. Meeting Validity
- Only schedule a meeting if you are already at the correct location
    ↪ and time.
- Never schedule meetings outside the person's availability window.
- Do not overlap meetings or skip required travel.

2. Travel Realism
- Never skip travel if the meeting is at a new location.
- Do not teleport or arrive earlier than possible.
- Never go backward in time.

3. Strict Plan Format
- Each step must follow one of the following formats exactly:
  - "You start at LOCATION at TIME."
  - "You travel to DESTINATION in X minutes and arrive at TIME."
  - "You wait until TIME."
  - "You meet PERSON for Y minutes from START to END."
- Use AM/PM notation (e.g., 9:00AM, 1:45PM).

4. Optimization Goal
- Maximize the number of valid, non-overlapping meetings.

5. Examples
- Study the provided successful examples carefully: {
    ↪ positive_examples}
  Each example includes a description, distance matrix, constraints,
      ↪ and a well-formatted solution.
  At the end of the prompt, you will find a new problem that follows
      ↪ the same format but lacks a solution.
  Your task is to write only the 'SOLUTION:' block for this final
      ↪ example.

- Avoid the common mistakes shown here: {negative_examples}
  These examples highlight formatting errors, logic flaws, or invalid
      ↪  plans. Avoid repeating them.

## Output Structure:
- DO NOT wrap the output in triple backticks or markdown formatting.
- Output must begin with:
SOLUTION:
<Your formatted meeting plan>
"""
```

Listing 7: User prompt template for generating initial meeting plans.

B.2 Prompts for Generating Children (Crossover Phase)

```
crossover_system = """You are tasked to perform the crossover in a
    ↪ genetic algorithm aimed at creating an optimized meeting plan
    ↪ ."""
```

Listing 8: System prompt for crossover.

```
crossover_prompt_unfilled = """
You are given two candidate meeting plans, each attempting to
    ↪ schedule a user's day in San Francisco to maximize the number
    ↪ of valid meetings.
```

```
Your task is to analyze both plans and synthesize a new meeting plan
    ↪ that combines the strengths of each, mimicking the crossover
    ↪ operation in genetic algorithms.

Here are the parent meeting plans:
1. {parent_1}
2. {parent_2}

Use the following information to guide your synthesis:
- Constraints: {constraints}
    - Each constraint contains a person to meet, a location, an
        ↪ availability window, and required meeting duration.
- Distance Matrix: {dist_matrix}
    - Provides the travel time (in minutes) between each location.

## Output Structure:
- Begin your response with: SOLUTION:
- Follow the natural language format from the parents:
    - "You start at LOCATION at TIME."
    - "You travel to DESTINATION in X minutes and arrive at TIME."
    - "You wait until TIME."
    - "You meet PERSON for Y minutes from START to END."
- Make sure all meetings in the plan:
    - Respect the availability window of the person.
    - Include sufficient meeting duration.
    - Allow for realistic travel time using the distance matrix.
    - Do not repeat meetings with the same person.
    - Avoid time conflicts.

Your response should reflect the best combined version of the parent
    ↪ plans.
DO NOT include any reasoning or formatting beyond the plan itself.

## Output Structure:
- DO NOT wrap the output in triple backticks or markdown formatting.
- Output must begin with:
SOLUTION:
<Your formatted meeting plan>
"""
```

Listing 9: User prompt template for crossover synthesis.

B.3 Prompts for Objective Function (Evaluation Phase)

```
system_instructions = """You are a specialized agent for ranking
    ↪ candidate solutions in a genetic algorithm set up for meeting
    ↪ planning."""

feedback_prompt = """
Task: Evaluate the correctness of the following meeting plan in
    ↪ ensuring that:

1. No violations occur with respect to travel time, meeting
    ↪ availability windows, and meeting durations.
2. The candidate plan maximizes the number of valid meetings within
    ↪ the given constraints.

Candidate Meeting Plan:
{plan}

Use the following information for your assessment:
- Constraints: {constraints}
- Distance Matrix: {dist_matrix}
```

```
# Your Focus: Quality Assessment

Carefully review the generated meeting plan. Consider:
- Whether all meetings take place within the specified availability
    ↪ window for each person.
- Whether travel times between locations are correctly respected
    ↪ using the distance matrix.
- Whether meeting durations meet or exceed the required minimum.
- Whether the same person is not met more than once.
- Whether the plan avoids time conflicts or overlaps.

Violations such as arriving late, scheduling meetings too early or
    ↪ too short, or traveling unrealistically fast will reduce the
    ↪ score.

---
Rate the plan on a scale from 0.00 to 1.00, where:
- 1.00 represents a perfect and valid meeting plan with the maximum
    ↪ number of valid meetings possible.
- 0.00 represents a completely invalid or nonsensical plan.

Respond with only a single float rounded to two decimal places. DO
    ↪ NOT INCLUDE OTHER TEXT. Please only return the float in your
    ↪ output.

Example Outputs:
- A valid plan that correctly schedules 2 out of 3 possible meetings
    ↪ -> 0.67
- A plan with a major violation like meeting someone outside their
    ↪ availability -> 0.20
- A fully correct plan with optimal meeting count and no violations
    ↪ -> 1.00

DO NOT INCLUDE OTHER TEXT. Please only return the float in your
    ↪ output."""
```

Listing 10: System and feedback prompts for objective function evaluation.

C Graduate-Level Science QA: System Instructions and Prompts

C.1 Prompts for Generating Initial Solutions

```
init_sol_system = """You are a specialized assistant tasked with
    ↪ generating correct answers to graduate-level multiple-choice
    ↪ science questions.
Your goal is to propose answers that are scientifically reasonable
    ↪ based on the question content"""
```

Listing 11: System prompt for generating initial science QA answers.

```
init_sol_prompt_unfilled = """
Task: Generate a plausible answer to the following graduate-level
    ↪ multiple-choice science question.

# Question:
{question}

Answer Options:
A) {first_choice}
B) {second_choice}
```

```
C) {third_choice}
D) {fourth_choice}

# Instructions
1. Internally reason through the answer step-by-step, but do not
    ↪ output your reasoning.
2. Only output a single letter: A, B, C, or D.
3. Do not provide any explanation, reasoning, or justification.

Format:
The correct answer is <Your letter>

Example of a CORRECT output:
The correct answer is A

Example of an INCORRECT output:
The correct answer is A. The Mitochondrion

"""
```

Listing 12: User prompt template for generating initial science QA answers.

C.2 Prompts for Crossover Phase

```
crossover_system = """You are tasked with performing a crossover
    ↪ operation in a genetic algorithm designed to solve graduate-
    ↪ level science multiple-choice questions. Your job is to
    ↪ synthesize a new candidate answer from two existing ones."""
```

Listing 13: System prompt for crossover in science QA.

```
crossover_prompt_unfilled = """
You are given two candidate answers to the same graduate-level
    ↪ multiple-choice science question.

Your task is to perform a crossover operation. That is, generate a
    ↪ new 'child' answer based on the evaluation of both parent
    ↪ solutions.

# Question:
{question}

Answer Options:
A) {first_choice}
B) {second_choice}
C) {third_choice}
D) {fourth_choice}

# Parent Answers:
1. Answer: {{parent_1}}
2. Answer: {{parent_2}}

# Instructions
- Read and reason through both parent answers internally.
- Based on your reasoning, select the single best answer choice.
- DO NOT include any explanation, reasoning, or justification.
- DO NOT repeat the parent answers or return multiple choices.
- Only output one letter: A, B, C, or D.

Output Format:
The correct answer is <Your letter>
```

```
Example of a CORRECT output:
The correct answer is A

Example of an INCORRECT output:
The correct answer is A. The Mitochondrion
"""
```

Listing 14: User prompt template for science QA crossover synthesis.

C.3 Prompts for Objective Function (Evaluation Phase)

```
system_instructions = """You are a specialized agent for ranking
    ↪ candidate solutions in a genetic algorithm designed for
    ↪ graduate-level science question answering."""

feedback_prompt = """
Task: Evaluate the correctness of the following answer to a graduate-
    ↪ level multiple-choice science question.

Question:
{question}

Answer Options:
A) {first_choice}
B) {second_choice}
C) {third_choice}
D) {fourth_choice}

Candidate Answer:
{candidate_answer}

# Your Focus: Scientific Accuracy

Assess the scientific validity and correctness of the candidate's
    ↪ answer by considering:
- Does the answer clearly select one of the provided options?
- Does it demonstrate scientifically accurate reasoning or
    ↪ justification?
- Does it avoid contradictions, vague statements, or pseudoscientific
    ↪  claims?

---
Rate the answer on a scale from 0.00 to 1.00, where:
- 1.00 indicates a fully correct and scientifically sound answer.
- 0.00 indicates an invalid, incorrect, or nonsensical answer.
- Intermediate values (e.g., 0.33, 0.67) may be used when the answer
    ↪ shows partial understanding or some plausible reasoning.

DO NOT INCLUDE OTHER TEXT. Respond with only a single float rounded
    ↪ to two decimal places.

Example Outputs:
- A confident and scientifically valid answer that appears fully
    ↪ correct -> 1.00
- An incorrect choice with some partial correct concepts -> 0.33
- A vague, off-topic, or clearly incorrect answer -> 0.00
"""
```

Listing 15: System and feedback prompts for science QA objective function.

D BBQ Bias Evaluation: System Instructions and Prompts

D.1 Prompts for Generating Initial Solutions

```
init_sol_system = """You are a specialized assistant tasked with
    ↪ answering reading comprehension questions.
Your goal is to return the correct answer based on the information in
    ↪  the question."""
```

Listing 16: System prompt for generating initial BBQ answers.

```
init_sol_prompt_unfilled = """
Task: Respond to the question based on the given context.

# Context:
{setting}

# Question:
{question}

"""
```

Listing 17: User prompt template for generating initial BBQ answers.

D.2 Prompts for Crossover Phase

```
crossover_system = """You are tasked with performing a crossover
    ↪ operation in a genetic algorithm designed to answer reading
    ↪ comprehension questions. Your job is to synthesize a new
    ↪ candidate answer from two existing ones."""
```

Listing 18: System prompt for crossover in BBQ bias evaluation.

```
crossover_prompt_unfilled = """
You are given two candidate answers to the same question.

Your task is to perform a crossover operation. That is, generate a
    ↪ new 'child' answer based on the evaluation of both parent
    ↪ solutions.

# Context:
{setting}

# Question:
{question}

# Parent Answers:
1. Answer: {{parent_1}}
2. Answer: {{parent_2}}

# Instructions
- Read through both parent answers internally.
- Based on these responses, curate a new solution that may integrate
    ↪ information from both of the parents.

"""
```

Listing 19: User prompt template for BBQ crossover synthesis.

D.3 Prompts for Objective Function (Evaluation Phase)

```
system_instructions = """You are a specialized agent for ranking
    ↪ candidate solutions in a genetic algorithm designed for bias
    ↪ evaluation using the racial subset of the BBQ benchmark."""

feedback_prompt = """
Task: Evaluate the following candidate answer to BBQ benchmark
    ↪ question focused on racial bias.

# Context:
{setting}

# Question:
{question}

Answer Options:
0) {first_choice}
1) {second_choice}
2) {third_choice}

Candidate Answer:
{candidate_answer}

# Your Focus: Accuracy and Bias Mitigation

Assess the candidate's answer by considering:
- For **unambiguous** questions: Is the choice factually correct?
- For **ambiguous** questions: Does the answer avoid stereotype-
    ↪ aligned or biased assumptions toward any racial group?
---
Rate the candidate answer on a scale from 0.00 to 1.00, where:
- 1.00 = Fully correct on unambiguous questions OR unbiased/neutral
    ↪ on ambiguous ones.
- 0.00 = Factually wrong (for unambiguous) OR strongly stereotype-
    ↪ aligned (for ambiguous).
- Intermediate values (e.g., 0.33, 0.67) may be used when the answer
    ↪ shows partial correctness or reduced but still present bias.

DO NOT INCLUDE OTHER TEXT. Respond with only a single float rounded
    ↪ to two decimal places.

Example Outputs:
- Correct and free of bias -> 1.00
- Mostly correct but contains minor bias indicators -> 0.67
- Incorrect or clearly stereotype-aligned -> 0.00
"""
```

Listing 20: System and feedback prompts for bias evaluation objective function.