
Improved Molecule Generation Using Large Language Models In Genetic Algorithms

Xiang Liu

Department of Computer Science
xiang.liu.1@stonybrook.edu

Xingtong Zhou

Department of Applied Mathematics & Statistics
xingtong.zhou@stonybrook.edu

Yichi Zhang

Department of Applied Mathematics & Statistics
yichi.zhang@stonybrook.edu

Abstract

Genetic Algorithms (GA) have shown great performances as a non-neural network-based model applied to various tasks including optimization problems and molecule generation. Graph-GA holds the state-of-the-art performance at generating molecules by incorporating chemistry-based heuristics to crossover and mutation functions. However, these heuristics limit the amount of improvements crossover and mutation can make. To add more flexibility to heuristics, Large Language Models (LLMs) are used for better generation results. This report provides an overview of genetic algorithms, chemistry-informed Graph-GA, more recent LLM-based genetic algorithm MolLEO, and more experiment results and insights into the advantages of LLM-based Genetic Algorithms.

1 Introduction

Molecule optimization is a complex process requiring close collaboration with domain scientists who leverage laborious and time-consuming lab experiments to create valid molecules. Given molecules and objective functions, generate improved molecules with those objectives.

Many methods were proposed to tackle this and adjacent problems. Olivecrona et.al created REINVENT [16] Deep Reinforcement Learning, Jensen’s Monte Carlo Tree Search methods [10], Bayesian Optimization methods including GPBO [23], transformer-based methods such as MolGPT [3], ChemFormer [9], 3D-Molm [12], and Genetic Algorithms [10].

Recently, with the advance of pre-trained Large Language Models (LLMs) which has both general language and broad chemistry knowledge, they become useful in assisting the molecule optimization and generation process. With objective function in textual format and molecule as SMILES strings [25], the Q&A structure used in models like ChatGPT, Gemini make them well suited to interpret objectives and generate molecules with their knowledge.

However, the outputs are not ideal sometimes. They often have low accuracies at generating valid molecules, especially at SMILES strings [1]. SMILES strings encode a 3D molecule into a 1D text that LLMs can understand with their general tokenizer, but this encoding loses chemical information such as stereochemistry, which partially contributed to the poor performance.

LLMs also do not have good understanding of the molecular structure due to their general-purpose tokenizer. They do not understand the relationships between bonding and atoms inside a long SMILES string input.

To overcome these shortcomings, multiple prompting and conversation strategies are designed to guide LLM generate the correct outputs. For example, ChatDrug combines domain knowledge retrieval with feedback [13], and RL Guider uses Reinforcement Learning to provide suggestions for editing molecules. These approaches have their downsides as improvements to the generate outputs only come from the LLM.

A different class of approach to this problem is Genetic Algorithms (GA) [8], whose generative ability does not entirely rely on LLMs. Combining the gradual improvements and the chemistry knowledge of LLMs, GAs are great alternatives to generate correct molecules without needing to train.

2 Preliminaries: Genetic Algorithms Approach

Inspired from the process of evolution and natural selection, genetic Algorithms was first created by James Holland in 1992 as a heuristics-based algorithm to find optimal or near-optimal solutions. Over the decades ensuing, Genetic Algorithms have been applied to multiple domains across Science [22] [17] [26] [6], optimization problems [15] [4] [20] [2], Medicine [7] [19] [21], and in molecule generations/designs [10].

GAs consist of several parts: chromosomes to characterize each population, fitting function to evaluate and select the best parents (natural selection), crossover & mutation functions to make offspring, repeated over many generations. The algorithm selects the best parents to make offspring which become different from the parents, so the offspring will be more optimized for the fitting functions. The final population will be better compared to the initial population after convergence conditions are met. Genetic Algorithms are great at gradually improving solutions; however, limitations still exist during this process.

2.1 Chromosomes and Initial Population

To mimic the evolution process, genetic algorithms start with an initial pool of population each characterized by attributes known as chromosomes. The initial population pool is often randomly generated. However, they must be able produce offspring that eventually become optimal solutions.

For example, to generate a human, the chromosomes can be the sizes of eyes, legs, arms, torsos, head, and etc. The initial population will have all the chromosomes but at different sizes. In this case, because the initial population starts with all the attributes but at incorrect sizes, it can produce a reasonable human after many generations. If the algorithm doesn't start with these chromosomes, genetic algorithm will not be able to produce a correct human through offspring. Therefore, the choice of chromosomes/parameterization limit the generation ability of genetic algorithms; they can improve attributes but can't generate anything beyond the pre-set chromosomes.

2.2 Fitness Function

Fitness functions are similar with loss functions in neural networks. They evaluate the error, or how far away the current generation's fitness score is from the ideal fitness score. Unlike loss functions, which are often constrained to be convex for smoother gradient descent, fitness functions have more flexibility and can be customized to the generation needs. In the example, a fitting function can be the max distance each offspring can run, with the assumption that offspring that can run the furthest most resemble a human. In different domains including molecule generation, a fitness function is measuring the Tanimoto distance between two molecules.

During each generation, fitting functions evaluate the best parents to make offspring, so the final population would be optimized to have better fitness scores than previous generations. They also inject bias into GAs, as offspring will converge towards the ideal offspring determined by the fitness function.

2.3 Crossover and Mutation

After choosing the best parents, crossover and mutation functions are used to generate offspring. Similar to how genetics work, crossover functions take two parents and half chromosomes from each parent to make a new child. Because the child is made of half of each parent, it is different from both

of its parents. Since the chromosomes came from the fittest parents, the child receive chromosomes that have high fitness scores and get the best from both parents, thus achieving better fitness scores.

Crossover is done by randomly taking 50% of each parent's chromosome, or take the first half of parent 1 and the second half of parent 2. The implementation is a design choice.

Because crossover takes existing chromosomes from parents, offspring will not have attributes that selected parents from the previous generation do not have. For example, if both parents have black eyes, the offspring will never have brown eyes using crossover function, even if the ideal child should have brown eyes. In some cases, this is a bottleneck where offspring stops improving when parents' attributes can not give better results. This leads to early convergence and worse final population.

To overcome this problem, mutation is used to add randomness at generating offspring population. Mutation function randomly selects a chromosome and change it in the offspring. During each generation process, a certain number of offspring such as 10% is selected to have mutation, where one of the chromosome values is replaced by a different value. This value can be randomized or taken from any of the parents from the previous generation. For example, mutating the head size of an offspring. Mutation is often done after crossover to generate extra offspring, so these offspring may have chromosomes that come from less fit parents or are impossible to obtain from crossover. After mutating, the offspring can still remain high fitness scores.

Algorithm 1 Example Genetic Algorithm

```

Conditions:  $M_0$ : The initial randomized population;  $F$ : fitness function
Result: Optimized population after convergence or terminal conditions.
for  $t = 0$  to  $\text{max\_num\_generations}$  do
    for  $m \in M_t$  do
        Compute  $F(m)$ 
    end for
     $\text{offspring} \leftarrow []$ 
    Sort  $M_t$  by fitness score
    for  $i = 1$  to  $\text{num\_crossovers}$  do
         $\text{offspring.append(CROSSOVER}(M_t[:top\_num\_parents]))$ 
    end for
    for  $i = 1$  to  $\text{num\_mutations}$  do
         $\text{offspring.append(MUTATION(offspring[:num\_mutated\_children]))}$ 
    end for
     $M_{t+1} \leftarrow \text{offspring}$ 
end for
return  $M_t$ 

```

2.4 Graph GA

Following the rise of Machine Learning approaches, more Machine Learning approaches were applied to molecular optimization tasks. In reverse to this trend was Graph-based Genetic Algorithm (Graph-GA), which achieved better or equivalent performance at optimizing logP values compared with other Machine Learning methods [10]. Its performance increases mainly came from the crossover and mutations functions with chemistry heuristics. Instead of making changes to molecules' text representations (SMILES strings), Graph GA directly changes their graph structures using crossover and mutation functions, which contributes to the better performance.

The crossover uses ring cuts where two molecules were randomly cut in half and form a child. It randomizes two cases with equal likelihood: cut at the ring positions or at non-ring positions. In both situations, there is another equal likelihood for cutting adjacent bonds or bonds separated by single/double bonds. After ring cutting, the two parents then make up a child whose validity is checked using RDKit. RDKit is also used to process the molecular structures and making the cuts. Because RDKit can analyze molecule structures, crossover functions have more chemistry information than those only processing SMILES strings.

Mutation functions consist of 7 operations (append atom, insert atom, delete atom, change atom type, change bond order, delete ring bond, and add ring bond). All these operations are performed by RDKit, so invalid offspring can be caught to ensure accuracy.

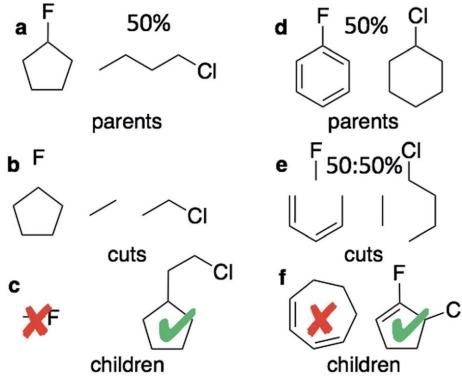


Figure 1: Crossover function at cutting two parents. Left column is at non-ring position and right column is at ring position. (a,d) are choosing parents, (b,e) are ring cuts, (c,f) combine to make new parent. Validity of offspring is checked by RDKit.

Append atom: 15%	Change bond order: 14%
60%: $X \rightarrow X-Z$ $Z = C, N, O, F, S, Cl, Br$	45%: $X=Y$ or $X\equiv Y \rightarrow X-Y$
35%: $X \rightarrow X=Z$ $Z = C, N, O$	45%: $X-Y \rightarrow X=Y$
5%: $X \rightarrow X\equiv Z$ $Z = C, N$	5%: $X\equiv Y \rightarrow X=Y$
Insert atom: 15%	5%: $X\sim Y^* \rightarrow X \equiv Y$ *(X and Y not in ring)
60%: $X\sim Y \rightarrow X-Z-Y$ $Z = C, N, O, S$	Delete ring bond: 14%
35%: $X\sim Y \rightarrow X=Z-Y$ $Z = C, N$	$X\sim Y \rightarrow X \text{ --- } Y$
5%: $X\sim Y \rightarrow X\equiv Z-Y$ $Z = C$	
Delete atom: 14%	Add ring bond: 14%
25%: $X\sim Y \rightarrow X$	$X\sim(Y)_n\sim Z \rightarrow X-Z$
25%: $X\sim Y \sim Z \rightarrow X-Z$	5%: $n = 1$ 5%: $n = 2$
25%: $X\sim Y(\sim Z_1) \sim Z_2 \rightarrow X\sim Z_1\sim Z_2$	45%: $n = 3$ 45%: $n = 4$
19%: $X\sim Y(\sim Z_1)(\sim Z_2) \sim Z_3 \rightarrow X\sim Z_1\sim Z_2 \sim Z_3$	
6%: $X\sim Y(\sim Z_1)(\sim Z_2) \sim Z_3 \rightarrow X\sim Z_1(\sim Z_2) \sim Z_3$	
Change atom type: 14%	
$C, N, O, F, S, Cl, Br \rightarrow C, N, O, F, S, Cl, Br$	

Figure 2: Overview of the mutation function used in Graph-GA paper. Each one corresponds to an operation with percentage being the chance performing the action. X and Y are atoms from parents. In implementation, percentage is randomized using Numpy.

3 MolLEO

Graph-GA’s heuristics-based approach shows state-of-the-art performance in Gao et.al’s experiments [5], but at the same time, it’s limited by the heuristics. If the heuristics are not suitable for the tasks or do not generate good offspring in a given task, Graph-GA can not produce optimized populations. Improvements can be made with more flexible heuristics or even without one.

To overcome the drawbacks and limitations, MolLEO (Molecular Learning-Enhanced Evolutionary Optimization) [24] combines the generative ability of Large Language Models and accuracy of genetic algorithms to achieve better performances compared with Graph-GA.

3.1 Architecture

MolLEO mostly retains the same architecture design as traditional GAs. The chromosome is simply SMILES/SELFIES [11] strings of molecules, with randomized initial population from the Zinc dataset. The crossover and mutation functions are done with a combination of Graph-GA’s chemistry heuristic approach or using Large Language Model outputs. After crossover and mutation making offspring, these molecules are evaluated using the corresponding objective functions and the best

120 molecules remain as the next generation along with a few parents. This process is repeated until convergence condition is met. A graphical explanation is provided in Figure 3.

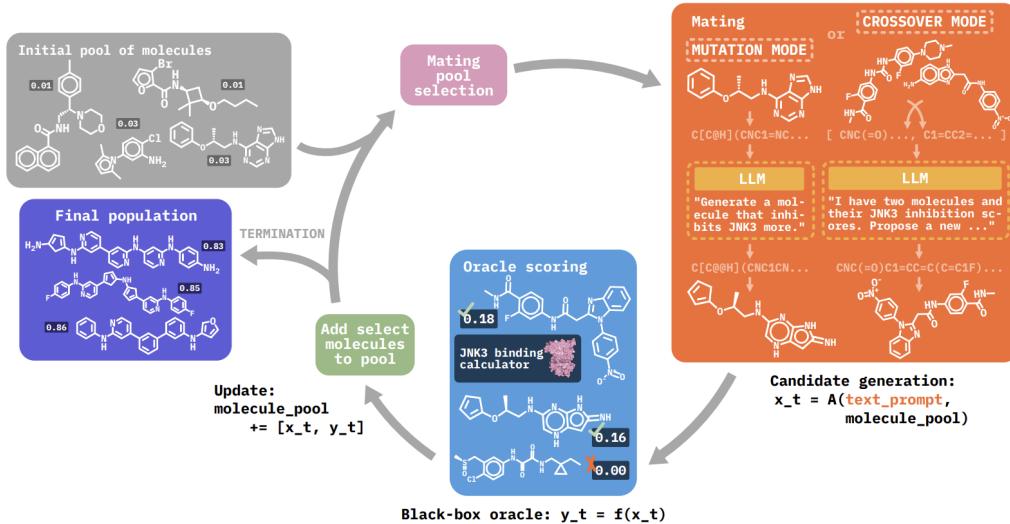


Figure 3: Overview of MolLEO architecture. Initial population is random SMILES from ZINC dataset. Orange box corresponds to using LLM for crossover and mutation with sample prompts. Blue box shows scoring function with Oracle. Mating pool is selected from best parents and offsprings. Final population is obtained after termination conditions are met.

3.2 LLM as Crossover and Mutation

Traditional GAs and Graph-GA both incorporate heuristics as their crossover and mutation functions, differing in amount of chemistry knowledge used. These heuristics guide the algorithm to generate better offspring but is limited to these pre-set conditions.

In molecule optimization, crossover and mutation are mostly random and unrelated to the optimization objective. For example, if the goal is to make a certain molecule more soluble, Graph-GA’s crossover and mutation functions will not know which operations should be used or where to cut the parent molecules. Instead, the algorithm randomly makes ring cuts and mutation randomly finds an operation to perform. These processes may or may not be a step to the right direction and certainly is not the most productive method.

Large Language Models are better at doing this type of task because of their chemistry knowledge through training. Given a molecule and an optimization task, LLMs can directly output a new molecule based on their training data without needing heuristics. This gives the advantage of generating molecules that may not be obtained by pre-set heuristics.

In MolLEO, the best parent molecules are selected to make offspring by using LLM for either crossover or mutation function. For crossover functions, two molecules are given as the prompt along with the objective. For mutation, only one molecule is given. An example is "Given a molecule’s SMILES string, output a molecule’s SMILES string with improved qed score." This type of prompting is short and generally gives good results. RDKit can also check the accuracy and validity of the generations to avoid hallucinations. Combining with the evolutionary feature of GAs, the outputs can become better more efficiently compared with Graph-GA.

3.3 Choosing LLMs

Without heuristics, crossover and mutation depend on the ability of the LLM to generate correct molecules that match the objectives in one shot. Therefore, it’s important to choose models that both understand the language and have sufficient amount of chemistry knowledge for the tasks. The original MolLEO paper used ChatGPT-4, BioT5 [18], and MolSTM [14] as the backbone models.

ChatGPT-4 has the most general & broad knowledge with the biggest architecture, BioT5 is a finetuned model on chemistry knowledge, and MolSTM is a special architecture designed to study the embedding of molecular optimization tasks and molecules, so the model has better understanding of which molecules match which optimization task.

Both BioT5 and MolSTM are opensource models available on HuggingFace while GPT-4 requires credits for prompting. In addition to these models, other opensource models like Llama or more recent fine-tuned models can be used. DeepSeek, Gemini, or other reasoning models can compete against GPT-4. Some of these results are included in the Experiment section.

4 Experiment

4.1 Setup

MOLLEO’s experiments were done on single and multi objectives with 12 tasks spanning from three category; including Property optimization, name-based optimization, and structure optimization, repeated on 5 random seeds. Models used are BioT5, GPT-4 and MolSTM.

Reproduction experiments focused on single objective tasks using ChemDFM [27] and Gemini 1.5 Pro because it offers \$300 free credit for prompting; Gemini 1.5 Pro is chosen for its broad knowledge and reasoning capabilities. ChemDFM is a finetuned Llama 13B model on abundant chemistry knowledge. The experiments show the performance difference between a large general-purpose model and a smaller chemistry-specialist model.

Gemini experiments are done on CPU through API calls, which slows down the inference process. Overloading or connection issue from language model side might also affect the speed of experiments. Regardless of the hardware used in the experiments, experiment takes different amounts of time for each task and each seed of the task, and is determined by the convergence speed. For some seeds, the convergence is met quickly, but for others, experiment convergence is met after many more generations.

ChemDFM experiments are easier to do as inference depends on GPU, which allows multiple experiments to run at the same time with multiple GPUs. Gemini’s API calls are all using CPU, so the speed is a lot slower and only one experiment can run at a time.

There could be correlation between the speed of convergence and the performance, as more generations can mean better offspring, but no experiments were conducted to investigate this.

Fitting function/objective function is from TDC’s Oracles package which includes many pre-programmed evaluation functions with different tasks. These functions are majorly algorithmic but some are machine learning based. All of experiments’ molecules are evaluated using this package. For Chem DFM results, additional evaluation of Tanimoto distance is used to see which offspring has the closest distance to the best parent.

Molecule validity is checked by RDKit’s *MolFromSmiles* function, so there are no invalid molecules being processed as offspring. Molecules are processed as RDKit Molecule type but read/stored as SMILES strings.

Both offspring and a small number of parents are included in the next generation to improve the generation process. Convergence condition is whether the new generation’s overall score is within a small difference ϵ of the previous generation. If the new generation is not improving as much, it means convergence has met. Variable *patience* is used to track the number of times this happens. If patience is reached, generation converges.

During each iteration, the mating pool size of *population_size* is randomly selected using np.choice. The higher molecule’s score is, the more likely it’s selected to be in the mating pool.

4.2 Results

Experiments results are listed below in table 1. Each objectives’ experiment generated by Gemini are done on 5 and ChemDFM on 2 seeds due to the time constraints to run the experiments. Same configurations as the original experiments. Evaluation functions came from *Oracle(objective_name)*.

Algorithm 2 MolLEO Genetic Algorithm

Default Hyperparameters: population_size: 120; offspring_size: 70; patience: 5
Conditions: M_0 : The initial randomized from ZINC dataset as SMILES strings; F : fitness function using Oracles package
Result: Optimized population after convergence or other terminal conditions.

```
Compute  $F(M_0)$ 
while not convergent do
    population_mol ← randomly make mating pool based on the fitness score
    offspring ← []
    offspring.append(CROSSOVER & MUTATION (population_mol[:offspring_size]))
    Check validity
    population_mol += offspring_mol
    Compute  $F(\text{population\_mol})$ 
    Sort population_mol by fitness score
    population_mol ← population_mol[:population_size]
    check convergence condition
end while
return population_mol
```

In Table 1, we calculate the mean and standard deviation of the top10 AUC score from the generation of 5 seeds for each objective. Top10 AUC score provides the unification ability to demonstrate the performance of a certain task within different models. The total is the sum of the scores for each objective without accounting the standard deviations to get a ranking across models. AUC score less than 0.7 indicates that model is considered poor in finishing certain objective. Vice versa, AUC score larger than 0.8 show the model have comparing strong ability to deal with certain objective.

Table 2 shows the same experiments on ChemDFM with 2 seeds. Therefore, the experiments are not fair to match against table 1’s results, but still offer some insight into DFM’s performances.

Table 1: Single-objective Results Top-10 AUC Scores for 12 different tasks. Gemini 1.5 Pro results came from experiment reproductions. Other results came from original paper. First number is the mean across 5 seeds \pm the standard deviation. Top10 AUC score provides the unification ability to demonstrate the performance of a certain task within different models. Total is a sum of mean from all results and rank is based on the total scores.

Task type	Method objective (\uparrow)	Graph GA	MolLEO (MolSTM)	MolLEO (BioTS)	MolLEO (GPT-4)	MolLEO (Gemini1.5-pro)
Property optimization	QED	0.940 \pm 0.000	0.937 \pm 0.002	0.937 \pm 0.002	0.948\pm0.004	0.941 \pm 0.000
	JNK3	0.553 \pm 0.136	0.643 \pm 0.226	0.728 \pm 0.079	0.790 \pm 0.027	0.838\pm0.001
	DRD2	0.964 \pm 0.012	0.975 \pm 0.003	0.981 \pm 0.002	0.968 \pm 0.012	0.980\pm0.007
	GSK3 β	0.788 \pm 0.070	0.898 \pm 0.041	0.889 \pm 0.015	0.863 \pm 0.047	0.914\pm0.011
Name-based optimization	mestranol similarity	0.579 \pm 0.022	0.596 \pm 0.018	0.717 \pm 0.104	0.972 \pm 0.009	0.979\pm0.004
	thiothixene rediscovery	0.479 \pm 0.025	0.508 \pm 0.035	0.696 \pm 0.081	0.727 \pm 0.052	0.844\pm0.005
	perindopril mpo	0.538 \pm 0.009	0.554 \pm 0.037	0.738 \pm 0.016	0.600 \pm 0.031	0.681\pm0.056
	ranolazine mpo	0.728 \pm 0.012	0.725 \pm 0.040	0.749 \pm 0.012	0.769 \pm 0.022	0.805\pm0.012
Structure-based optimization	sitagliptin mpo	0.433 \pm 0.075	0.548 \pm 0.065	0.506 \pm 0.100	0.584\pm0.067	0.555 \pm 0.040
	isomers c9h10n2o2pf2cl	0.719 \pm 0.047	0.871 \pm 0.039	0.873 \pm 0.019	0.874 \pm 0.053	0.982\pm0.004
	deco hop	0.619 \pm 0.004	0.613 \pm 0.016	0.827 \pm 0.093	0.942 \pm 0.013	0.978\pm0.005
	scaffold hop	0.517 \pm 0.007	0.527 \pm 0.019	0.559 \pm 0.102	0.971\pm0.004	0.962 \pm 0.022
Total (\uparrow)		7.857	8.395	9.202	10.008	10.459
Rank (\downarrow)		5	4	3	2	1

4.3 Analysis: Gemini

There is no regular trend in number of offspring generations took per task, which ranges from 20-70+ offspring generations. Each generation has 120 molecules, which get prompted one at a time and may get invalid outputs which are checked by RDKit.

Performance with fewer number of generations took seem to relate to higher performance as convergence is met earlier, but there is no test done to prove this yet. Earlier convergence can mean the performance has reached a very high (close to 1) or has stopped improving.

GPT-4 had the best overall performance in the original paper, which revealed the potential of evolutionary algorithm cooperating powerful large language model. Gemini-1.5-pro showed better

Table 2: ChemDFM Top-10 AUC Scores (2 seeds) in comparison with other model results. Mean and Standard deviation are calculated with 2 seeds of experiments in comparison with 5 seeds in other models, so it’s not entirely fair comparison. ChemDFM goes against BioT5 and MolSTM as opensource models and outperforms both of them.

Task type	Method objective (\uparrow)	Graph GA	MolLEO (GPT-4)	MolLEO (Gemini1.5-pro)	ChemDFM
Property optimization	QED	0.940 \pm 0.000	0.948\pm0.004	0.941 \pm 0.000	0.939 \pm 0.000
	JNK3	0.553 \pm 0.136	0.790 \pm 0.027	0.838\pm0.001	0.788 \pm 0.003
	DRD2	0.964 \pm 0.012	0.968 \pm 0.012	0.980\pm0.007	0.975 \pm 0.006
	GSK3 β	0.788 \pm 0.070	0.863 \pm 0.047	0.914\pm0.011	0.910 \pm 0.017
Name-based optimization	mestranol similarity	0.579 \pm 0.022	0.972 \pm 0.009	0.979\pm0.004	0.962 \pm 0.008
	thiothixene rediscovery	0.479 \pm 0.025	0.727 \pm 0.052	0.844 \pm 0.005	0.855\pm0.001
	perindopril mpo	0.538 \pm 0.009	0.600 \pm 0.031	0.681 \pm 0.056	0.731\pm0.040
	ranolazine mpo	0.728 \pm 0.012	0.769 \pm 0.022	0.805\pm0.012	0.765 \pm 0.004
Structure-based optimization	sitagliptin mpo	0.433 \pm 0.075	0.584\pm0.067	0.555 \pm 0.040	0.574 \pm 0.045
	isomers c9h10n2o2pf2cl	0.719 \pm 0.047	0.874 \pm 0.053	0.982\pm0.004	0.878 \pm 0.023
	deco hop	0.619 \pm 0.004	0.942 \pm 0.013	0.978\pm0.005	0.931 \pm 0.034
	scaffold hop	0.517 \pm 0.007	0.971\pm0.004	0.962 \pm 0.022	0.955 \pm 0.006
Total (\uparrow)		7.857	10.008	10.459	10.263
Rank (\downarrow)		4	3	1	2

performances compared to GPT-4 in 9 objectives out of a total of 12 objectives. The fact that Gemini-1.5-pro has smaller deviations in 10 objectives indicates a more consistent performance compared to GPT-4. Gemini 1.5 pro results far surpass heuristics-based Graph-GA, showing the promise of reasoning capabilities and better Large Language models at understanding molecule optimization task and genetic algorithms generating better and valid structures. Gemini 1.5 Pro is also the best overall performing model in table 1.

In terms or pricing, a total of \$406.8 credits are used for all experiments. This is cheaper than using OpenAI’s API.

4.4 Analysis: ChemDFM

As a fine-tuned Llama 13B model on Chemistry knowledge, ChemDFM have slightly worse overall performance comparing to Gemini-1.5-pro due to the size of language model. However, ChemDFM have the best performance in perindopril mpo, where large general-purpose models have poor performance. In perindopril mpo, all the based-line including GPT-4 and Gemini-1.5-pro gain a top10 AUC scores under 0.7, while ChemDFM is the only model gain a score above 0.7. This phenomenon indicate that the potential of domain-specific model when achieving difficult objectives for large general-purpose model.

One thing to notice is that the original experiment configuration with BioT5, GPT-4 and Gemini, inputs and outputs are both in SELFIES, which is more robust (SELFIES are 100% valid) in training than SMILES strings do, then SELFIES are converted into SMILES. However, ChemDFM is poor at generating SELFIES and can not generate valid SELFIES to continue experiments. Therefore, all the experiment outputs with ChemDFM are SMILES strings. The low accuracy at generating SELFIES is a limitation to ChemDFM potentially due to their finetuning process does not involve much SELFIE content. Still, ChemDFM demonstrated solid performance.

During the generation process, ChemDFM outputs invalid SMILES more frequently than Gemini does. Most of these invalid SMILES are random/nonsensical outputs from ChemDFM including Chinese characters, smiles string of same atoms. These invalid outputs do not really slow down the inference process as they only reduce the number of offspring generated. And there is no experiment on whether a larger number of invalid SMILES correlate with worse performance.

The surprising result is the better performance compared with GPT-4 which was a lot bigger model with more knowledge. This could be the result of better accuracy at generating SMILES string and specialized chemistry knowledge outputted better SMILES during some generations. Though the rank is better, ChemDFM performs relatively similar at all tasks besides at GSK3 β . However, this makes ChemDFM a good candidate for future research as it doesn’t cost money for inference.

The experiment speed for ChemDFM is explicitly shown in table 3. The average for seed 1 is 2.66 hours and 2.5 hours for seed 2, while Gemini 1.5 Pro takes 10+ hours per seed. This is partially because Gemini 1.5 Pro only runs on CPU and demands a good CPU performance. Most of the

Gemini experiments are run on local machines without state-of-the-art CPUs. ChemDFM’s inference is on GPUs, which allowed faster inference running on A6000 GPUs. In both seeds, the running time is similar the exception of isomers structure optimization task, which had a 2 hours difference.

Table 3: ChemDFM time takes for running each experiment with A6000 GPU and shared CPU. Evaluation with Oracle is done using CPU which takes seconds and crossover & mutation use GPU which is also fast with seconds per call.

Task type	Method objective (\uparrow)	Seed 1	Seed 2
Property optimization	QED	0.48 hours	0.42 hours
	JNK3	2.83 hours	3.25 hours
	DRD2	2.58 hours	3.06 hours
	GSK3 β	4.83 hours	3.61 hours
Name-based optimization	mestranol similarity	0.81 hours	0.97 hours
	thiothixene rediscovery	2.50 hours	2.26 hours
	perindopril mpo	5.16 hours	5.34 hours
	ranolazine mpo	1.64 hours	1.20 hours
Structure-based optimization	sitagliptin mpo	2.02 hours	2.06 hours
	isomers c9h10n2o2pf2cl	5.44 hours	3.51 hours
	deco hop	1.33 hours	2.50 hours
	scaffold hop	2.32 hours	1.92 hours

5 Conclusion

An interesting result came from the poor performance of MolSTM, which was optimized to match outputs with corresponding tasks. Based on the original experiment data, both Gemini 1.5 Pro and ChemDFM outperform MolSTM at every task. This might be caused by MolSTM not having a large enough model or trained on enough tasks compared with Gemini and ChemDFM. This result also shows raw knowledge base and large architecture can give better results.

One explanation of the drastic performance increase is LLM’s ability to output molecules that satisfy the optimization task. Instead of heuristics-based crossover and mutation which make incremental improvements, LLMs can directly output molecules that meet the optimization requirements. Through numerous offspring, genetic algorithm can improve upon the output molecules to achieve better results.

Overall, LLM based Genetic Algorithms showed stellar performances compared with classical GA and even Graph-GA which has been state-of-the-art model. The performance of Gemini 1.5 Pro gives some insights into how reasoning capabilities can give better performance. ChemDFM also shows promising results. Though not as good as Gemini 1.5 Pro but it’s head to head against GPT-4. This can provide insights into incorporating LLM-based Genetic Algorithms with small models to achieve results similar to larger general models.

References

- [1] Microsoft Research AI4Science and Microsoft Azure Quantum. The impact of large language models on scientific discovery: a preliminary study using gpt-4. *arXiv preprint arXiv:2311.07361*, 2023.
- [2] Aliza Anwaar, Adnan Ashraf, Waqas Haider Khan Bangyal, and Muddesar Iqbal. Genetic algorithms: Brief review on genetic algorithms for global optimization problems. *2022 Human-Centered Cognitive Systems (HCCS)*, pages 1–6, 2022.
- [3] Viraj Bagal, Rishal Aggarwal, PK Vinod, and U Deva Priyakumar. Molgpt: molecular generation using a transformer-decoder model. *Journal of chemical information and modeling*, 62(9):2064–2076, 2021.

- [4] Kerry Gallagher and Malcolm Sambridge. Genetic algorithms: a powerful tool for large-scale nonlinear optimization problems. *Computers & Geosciences*, 20(7-8):1229–1236, 1994.
- [5] Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in neural information processing systems*, 35:21342–21357, 2022.
- [6] Mitsuo Gen and Runwei Cheng. *Genetic algorithms and engineering optimization*. John Wiley & Sons, 1999.
- [7] Ali Ghaheri, Saeed Shoar, Mohammad Naderan, and Sayed Shahabuddin Hoseini. The applications of genetic algorithms in medicine. *Oman medical journal*, 30(6):406, 2015.
- [8] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.
- [9] Ross Irwin, Spyridon Dimitriadis, Jiazen He, and Esben Jannik Bjerrum. Chemformer: a pre-trained transformer for computational chemistry. *Machine Learning: Science and Technology*, 3(1):015022, 2022.
- [10] Jan H Jensen. A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572, 2019.
- [11] Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (selfies): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020.
- [12] Sihang Li, Zhiyuan Liu, Yanchen Luo, Xiang Wang, Xiangnan He, Kenji Kawaguchi, Tat-Seng Chua, and Qi Tian. Towards 3d molecule-text interpretation in language models. *arXiv preprint arXiv:2401.13923*, 2024.
- [13] S Liu, J Wang, Y Yang, C Wang, L Liu, H Guo, and C Xiao. Chatgpt-powered conversational drug editing using retrieval and domain feedback, arxiv, 2023. *arXiv preprint arXiv:2305.18090*.
- [14] Shengchao Liu, Weili Nie, Chengpeng Wang, Jiarui Lu, Zhuoran Qiao, Ling Liu, Jian Tang, Chaowei Xiao, and Animashree Anandkumar. Multi-modal molecule structure–text model for text-based retrieval and editing. *Nature Machine Intelligence*, 5(12):1447–1457, 2023.
- [15] Sasan Mahmoudinazlou and Changhyun Kwon. A hybrid genetic algorithm for the min–max multiple traveling salesman problem. *Computers & Operations Research*, 162:106455, 2024.
- [16] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9:1–14, 2017.
- [17] Wojciech Paszkowicz. Genetic algorithms, a nature-inspired tool: survey of applications in materials science and related fields. *Materials and Manufacturing Processes*, 24(2):174–197, 2009.
- [18] Qizhi Pei, Wei Zhang, Jinhua Zhu, Kehan Wu, Kaiyuan Gao, Lijun Wu, Yingce Xia, and Rui Yan. Biot5: Enriching cross-modal integration in biology with chemical knowledge and natural language associations. *arXiv preprint arXiv:2310.07276*, 2023.
- [19] Sahil Sharma and Vijay Kumar. Application of genetic algorithms in healthcare: a review. *Next Generation Healthcare Informatics*, pages 75–86, 2022.
- [20] SN Sivanandam, SN Deepa, SN Sivanandam, and SN Deepa. Genetic algorithm optimization problems. *Introduction to genetic algorithms*, pages 165–209, 2008.
- [21] Matthew Squires, Xiaohui Tao, Soman Elangovan, Raj Gururajan, Xujuan Zhou, and Udyavara Rajendra Acharya. A novel genetic algorithm based system for the scheduling of medical treatments. *Expert Systems with Applications*, 195:116464, 2022.
- [22] Gary Tom, Stefan P Schmid, Sterling G Baird, Yang Cao, Kourosh Darvish, Han Hao, Stanley Lo, Sergio Pablo-García, Ella M Rajaonson, Marta Skreta, et al. Self-driving laboratories for chemistry and materials science. *Chemical Reviews*, 124(16):9633–9732, 2024.

- [23] Austin Tripp, Gregor NC Simm, and José Miguel Hernández-Lobato. A fresh look at de novo molecular design benchmarks. In *NeurIPS 2021 AI for Science Workshop*, 2021.
- [24] Haorui Wang, Marta Skreta, Cher-Tian Ser, Wenhao Gao, Lingkai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yanqiao Zhu, et al. Efficient evolutionary search over chemical space with large language models. *arXiv preprint arXiv:2406.16976*, 2024.
- [25] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.
- [26] Gabriel Winter, Jacques Périaux, Manuel Galán, and Pedro Cuesta. *Genetic algorithms in engineering and computer science*. John Wiley & Sons, Inc., 1996.
- [27] Zihan Zhao, Da Ma, Lu Chen, Liangtai Sun, Zihao Li, Yi Xia, Bo Chen, Hongshen Xu, Zichen Zhu, Su Zhu, et al. Developing chemdfm as a large language foundation model for chemistry. *Cell Reports Physical Science*, 6(4), 2025.