LLM and Simulation as Bilevel Optimizers:
A New Paradigm to Advance Physical Scientific Discovery

Pingchuan Ma[1] **Tsun-Hsuan Wang**[1] **Minghao Guo**[1] **Zhiqing Sun**[2]
Joshua B. Tenenbaum[1][3][4] **Daniela Rus**[1] **Chuang Gan**[5][6] **Wojciech Matusik**[1]

Abstract

Large Language Models have recently gained significant attention in scientific discovery for their extensive knowledge and advanced reasoning capabilities. However, they encounter challenges in effectively simulating observational feedback and grounding it with language to propel advancements in physical scientific discovery. Conversely, human scientists undertake scientific discovery by formulating hypotheses, conducting experiments, and revising theories through observational analysis. Inspired by this, we propose to enhance the knowledge-driven, abstract reasoning abilities of LLMs with the computational strength of simulations. We introduce Scientific Generative Agent (SGA), a bilevel optimization framework: LLMs act as knowledgeable and versatile thinkers, proposing scientific hypotheses and reason about discrete components, such as physics equations or molecule structures; meanwhile, simulations function as experimental platforms, providing observational feedback and optimizing via differentiability for continuous parts, such as physical parameters. We conduct extensive experiments to demonstrate our framework's efficacy in constitutive law discovery and molecular design, unveiling novel solutions that differ from conventional human expectations yet remain coherent upon analysis.

1. Introduction

In physical science, spanning physics, chemistry, pharmacology, etc., various research streams aim to automate and speed up scientific discovery (Wang et al., 2023). Each

[1]MIT CSAIL [2]CMU LTI [3]MIT BCS [4]Center for Brains, Minds and Machines [5]UMass Amherst [6]MIT-IBM Watson AI Lab. Correspondence to: Pingchuan Ma <pcma@csail.mit.edu>.

stream innovates within its field, creating methods tailored to its specific challenges and nuances. However, this approach often misses a universally applicable philosophy (Popper, 2005; Fortunato et al., 2018), which can be pivotal to democratizing access to advanced research tools, standardizing scientific practices, and enhancing efficiency across disciplines. Our goal aims to transcend specific domains, offering a unified approach to physical science.

As an inspiration, we observe how human scientists conduct scientific discovery experiments and conclude a few key experiences: (i) iteratively propose a hypothesis and make observations from experimentation to correct theories (Popper, 2005), (ii) divide the solutions into discrete components, such as physics equations or molecule structures, and continuous components, such as parameters for physics and molecule properties (Wang et al., 2023), (iii) exploit the existing knowledge while occasionally explore novel ideas aggressively in pursuits of breakthrough (Wuestman et al., 2020), (iv) follow a generic, universal principle for all types of physical scientific discovery yet with specific nuance of each discipline (Rosenberg & McIntyre, 2019).

Standing out as generalist tools with an extensive repository of knowledge (AI4Science & Quantum, 2023), large language models (LLMs) have recently risen to prominence in scientific discovery for their expansive knowledge bases, advanced reasoning capabilities, and human-friendly natural language interface. One line of research focuses on fine-tuning LLMs with domain-specific data to align natural language with scientific information, such as chemical (Chithrananda et al., 2020) or drug (Liu et al., 2021) structures; however, these methods are domain-bound and demand extensive data for broader application. Another research direction seeks to leverage the innate capabilities of pre-trained LLMs, augmented by external resources like the internet, programming, or documentation. LLMs serve as optimizers or agents (Huang et al., 2023) for mathematical problem-solving (Romera-Paredes et al., 2023), conducting chemical experiments (Boiko et al., 2023), and advancing molecular (Li et al., 2023) and drug discovery (Sharma & Thakur, 2023). Nevertheless, these approaches are confined to the computational capability of LLMs, a crucial factor
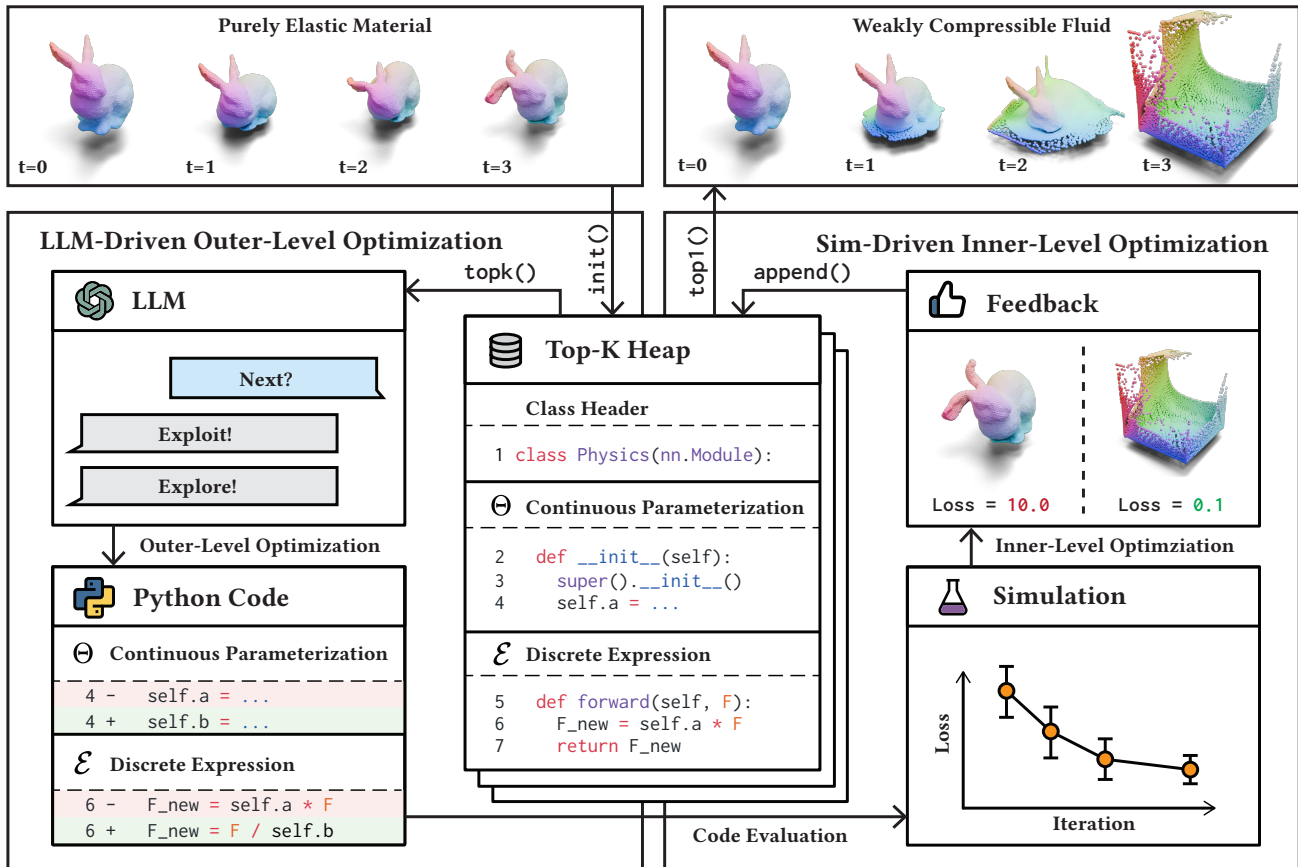
Figure 1. **The overall pipeline of Scientific Generative Agent (SGA).** Taking the constitutive law searching problem as an example, the input is an initial guess (a purely elastic material), and the output is another constitutive law optimized towards the ground-truth (weakly compressible fluid). The initial guess first initialize a top-K heap for storing the solutions. In the outer-level optimization, an LLM takes in top-K previously proposed solutions and generates a better one upon them with modified continuous parameterization Θ and discrete expression $\mathcal{E}$. In the inner-level optimization, a gradient-based optimization solves for optimal Θ via simulation and appends these optimized solutions in the heap. After a few iterations of bilevel optimization, the heap returns the top-1 solutions as the final solution.

in physical science for tasks like calculating numerical results based on physics law hypotheses to predict natural phenomena. To address this limitation, we propose to augment LLMs with physical simulation, hereby merging the knowledge-driven, abstract reasoning abilities of LLMs with the computational structures and accuracy of simulations.

To this end, inspired by the overarching philosophy of human scientists, we introduce **Scientific Generative Agent (SGA)**, a bilevel optimization approach wherein the outer-level engages LLMs as knowledgeable and versatile thinkers for generating and revising scientific hypothesis, while the inner-level involves simulations as experimental platforms for providing observational feedback. First, we employ LLMs to generate hypotheses, which then guide the execution of simulations. These simulations, in turn, yield observational feedback that helps refine and improve the proposed hypotheses. Secondly, we introduce a bilevel optimization framework: one level performs search-based optimization on discrete, symbolic variables like physics laws

or molecule structures via LLMs; the other level performs gradient-based optimization via differentiable simulation for continuous parameters like material stiffness or molecule coordinates. Thirdly, we devise an exploit-and-explore strategy for the hypothesis proposal by adjusting LLM's generation temperature. Lastly, we demonstrate our pipeline is generally applicable across scientific disciplines, with only minimal modification such as altering the prompts.

For the empirical study, we focus on (i) *molecular design* that aims to discover molecular structure and atoms' coordinates based on its conformation and quantum mechanical properties and (ii) *constitutive law discovery* that aims to discover material constitutive equations and its corresponding mechanical properties directly from a recorded motion trajectory. To provide a concrete example, let's assume that we initially have simply the code for a purely linear material. We then task our model to uncover a more complex representation by optimizing its code to fit a highly nonlinear trajectory. In this task, our method capitalizes on the

strengths of bilevel optimization: the outer-level utilizes LLMs to identify the correct symbolic material constitutive equations and formulates a proposition for potentially beneficial continuous parameterization (e.g., Young's modulus and Poisson's ratio); and the inner-level refines the proposed material parameters and provides informative feedback using differentiable simulation. Generally, our method can discover the desired molecules and constitutive laws, outperforming other LLM-based baselines; more interestingly, it can propose well-performing solutions that are beyond human expectation yet sensible under analysis by domain experts. Overall, our contributions are concluded as:

- We present a generic framework for physical scientific discovery that combines LLMs with physical simulations.
- We propose a bilevel optimization with LLMs for discrete-space search-based optimization and differentiable simulations for continuous-space gradient-based optimization.
- We conduct extensive experiments to demonstrate the effectiveness and generality of the proposed framework in physics law discovery and molecular design; moreover, we showcase novel molecules or constitutive laws, while unexpected from a conventional perspective, are deemed reasonable upon examination by domain experts.

2. Scientific Generative Agent

SGA is a bilevel optimization framework where the upper level features LLMs as proposers of scientific solutions, and the lower level utilizes simulations as experimental platforms for validation. In Sec. 2.1, we describe a formal definition of the bilevel optimization, followed by Sec. 2.2 for outer optimization and Sec. 2.3 for inner optimization.

2.1. Bilevel Optimization Pipeline

We formally describe the pipeline of our method, including the input/output of the system and the underlying submodules, and the overall optimization formulation. Suppose we are given a metric to evaluate a physical phenomenon y (e.g., a configuration of deformation) for a scientific problem $\mathcal{L}(y)$ (e.g., reconstruction of a mechanistic behavior). First, we describe the simulation (as an experimental platform) as,

$$y, z = \Phi(\theta; \mathcal{E}), \tag{1}$$

where Φ is a simulator that takes in scientific expression $\mathcal{E}$ (e.g., constitutive equations) and continuous components θ (e.g., material parameters) as inputs and gives simulated physical phenomenon y and additional observational feedback z (e.g., particles' trajectories) as outputs. Next, the LLM is prompted to act as a thinker to propose expressions $\mathcal{E}$ based on past experimental results from simulation,

$$\mathcal{E}, \Theta = \text{LLM}\left(\{\mathcal{L}(y_k), z_k, o_k, \mathcal{E}_k, \Theta_k\}_{k \in [K]}; \mathcal{P}\right), \tag{2}$$

Algorithm 1 Scientific Generative Agent

Input: Discrete expression and continuous param $(\mathcal{E}, \theta \in \Theta)$,
    Num of exploiting M_l, Num of exploring M_h,
    Exploiting temperature T_l, Exploring temperature T_h

```
 1: # Store ranked (solution,param) by heap
 2: H ← heap()
 3: # Continuous optimization
 4: θ̂ ← optim(E,θ;Φ)
 5: H.append((E,θ̂))
 6: for i = 1,...,N do
 7:    # Generate Ml solutions from LLM
 8:    (E,Θ)[:Ml] ← LLM(H.topk(K),Tl)
 9:    # Generate Mh solutions from LLM
10:    (E,Θ)[Ml:Ml+Mh] ← LLM(H.topk(K),Th)
11:    for m = 1,...,Ml + Mh do
12:       # Continuous optimization
13:       θ̂ ← optim(E,θ ∈ Θ;Φ)
14:       H.append((E,θ̂))
15:    end for
16: end for
```
Output: H.topk(1) # Return the best

---

where the set $[K]$ summarizes the pointers to the past simulation results containing an evaluation of the scientific problem $\mathcal{L}(y_k)$, other physical feedback (z_k, o_k), and past proposals $(\mathcal{E}_k, \Theta_k)$; o_k summarizes the intermediate results of the inner optimization (later detailed in Sec. 2.3); Θ determines the continuous parameterization for the decision variables of the inner optimization (e.g., which variables to be optimized within a proposed equation); $\mathcal{P}$ is prompt. With these, we define the bilevel optimization problem as,

$$\min_{\mathcal{E}, \Theta} \mathcal{L}\left(y\left(\mathcal{E}, \Theta, \hat{\theta}; \Phi\right)\right) \tag{3a}$$

$$\text{s.t. } G(\mathcal{E}, \Theta; \Phi) \leq 0 \tag{3b}$$

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} \mathcal{L}(y(\theta; \Phi, \mathcal{E})), \tag{3c}$$

where $G(\cdot) \leq 0$ refers to the validity of the simulation (i.e., whether an expression $\mathcal{E}$ is simulatable). The outer optimization searches for (i) an expression $\mathcal{E}$ that defines what experiments to be conducted $\Phi(\cdot; \mathcal{E})$ and (ii) continuous parametrization Θ that defines the search space of the inner continuous optimization $\min_{\theta \in \Theta}$. With the dependencies on the outer-level variables $(\mathcal{E}, \Theta)$, the inner optimization searches for the optimal continuous parameters $\hat{\theta}$ given the proposed expression via differentiable simulation.

2.2. LLM-Driven Outer-Level Search

We dive deeper into how we use LLMs (Eq. 2) and their interaction with the simulation for outer-level search (Eq. 3).

LLM-driven Optimization   LLMs have shown to be effective sequential decision makers for generic optimization, providing proper guidance via prompting and sufficiently informative contexts (Yang et al., 2024; Romera-Paredes

et al., 2023). We craft prompts to direct LLMs in a structured manner, enabling them to (i) perform *analysis* on past experimental results from the simulation, e.g., the deviatoric parts of stress tensor are likely correct based on the loss curve; (ii) devise a high-level *plan* on how to formulate a hypothesis or improve upon previous experiments, e.g., ensure numerical stability with the usage of the determinant of deformation gradient; (iii) suggest a *solution* that can be executed as experiments via simulation for hypothesis testing; e.g., a code snippet describing a constitutive equation. For Eq. 3a, inspired by (Ma et al., 2024), we adopt an evolutionary search that generates multiple offspring $\{\mathcal{E}_m, \Theta_m\}_{m \in [M]}$ (M is offspring size) in each iteration and retain the best selection. Distinctively, our approach (Alg. 1) involves selecting several high-performing candidates rather than the best only, which (i) enhances the feasibility of hypotheses in simulation (Eq. 3b) and (ii) facilitates evolutionary crossover, with LLMs generating new hypotheses from various past experiments ("breeds") for better exploration, akin to the findings in (Romera-Paredes et al., 2023).

Interfacing with Simulation   The primary challenge in integrating LLMs with simulation lies in devising a protocol that enables efficient, structured, yet adaptable communication between the two modules. We observe that physical scientific solutions are often represented as mathematical expressions or structured entities. Hereby, from LLMs to simulation, we consider two settings: equation searching and entity searching, both unified as the abstraction $(\mathcal{E}, \Theta)$ in Eq. 2. In *equation searching*, LLMs are allowed to propose equations $\mathcal{E}$ along with the search space of the inner-level continuous optimization Θ; for practitioners, an example using PyTorch can be Θ as `__init__` that defines continuous parameters via `nn.Parameter` and $\mathcal{E}$ as `forward` that defines computation of equations (see Fig. 1). In *entity searching*, LLMs propose descriptions of structures $\mathcal{E}$ (e.g., how atoms are connected to form a molecule) with Θ simply reduced to constant (e.g., every atom has its 3D coordinates to be optimized) and omitted from the optimization Eq. 3a as decision variables. On the other hand, from simulation to LLMs, we leverage domain experts' knowledge to craft functions for extracting compact, relevant information z as observational feedback; this process is akin to an experienced scientist offering guidance to a junior colleague on how to document experimental findings effectively. For instance, human experts often monitor the movements of specific body regions to derive constitutive laws. Therefore, to aid in this process, we include a function in the simulation that records the particle trajectories. Lastly, the subsequent section Sec. 2.3 will provide an in-depth explanation of the inner optimization results denoted as o. These results serve as feedback from the simulation to the LLMs.

Exploitation and Exploration   Inspired by human scientists achieving breakthroughs by skillfully balancing careful progression with bold exploration, we devise an exploit-and-explore strategy by tuning the LLMs' decoding temperature (Yang et al., 2024). When generating offspring $\{\mathcal{E}_m, \Theta_m\}_{m \in [M]}$ in Eq. 3a, we divide them into two groups: one ($m \in \mathcal{M}_{\text{exploit}}$) consists of cautious followers that keep the "gradient" and conservatively trails previous solutions, while the other ($m \in \mathcal{M}_{\text{explore}}$) comprises daring adventurers that take risks and suggest unique solutions. Empirically, we observed that (i) $\mathcal{M}_{\text{exploit}}$ often contains repetitive solutions from previous iterations, and (ii) $\mathcal{M}_{\text{explore}}$ tends to yield solutions too random to be informative for guiding optimization, or invalid (i.e., violating Eq. 3b), thus providing little feedback signal. As a rule of thumb, we have found that a 1:3 ratio between $\mathcal{M}_{\text{exploit}}$ and $\mathcal{M}_{\text{explore}}$ is effective.

2.3. Differentiable Inner-Level Optimization

Under the search space Θ and expression for simulation $\mathcal{E}$ from the outer level, inner optimization (Eq. 3c) involves a gradient-based optimization that solves for optimal continuous parameters $\hat{\theta} \in \Theta$ via differentiable simulation (Eq. 1). Essentially, the domain-specific knowledge is distilled via gradients $\nabla_\theta \Phi(\theta; \mathcal{E})$ from the simulation to the intermediate optimization results o (like loss curve). The $(\hat{y}, o)$ are then fed back to LLMs for revising solutions. Note that o may involve the loss curve toward the target metric $\mathcal{L}$ and other auxiliary recordings throughout optimization, carrying information of how to improve solutions in various aspects; for example, with $\mathcal{L}$ as displacement of position, o may include velocities across the inner optimization iterations.

3. Experiments

3.1. Problem Definitions

Constitutive Law Discovery   Identifying the constitutive law from motion observations stands as one of the most difficult challenges in fields such as physics, material science, and mechanical engineering. Here we follow the recent advances in physical simulation and formulate the constitutive law discovery task as an optimization problem (Ma et al., 2023) using differentiable Material Point Method (MPM) simulators (Sulsky et al., 1995; Jiang et al., 2016). Note that our method is not specifically tailored to MPM simulators and applies to any physical simulation. The objective of this task is to identify both the discrete expression and continuous parameters in a constitutive law, specifically the symbolic material models $\varphi(\cdot)$ and their corresponding material parameters θ, from a ground-truth trajectory of particle positions $\hat{X}_{t \in [1,...,T]}$ where T denotes the number of steps. In this problem, we consider two types of constitutive laws, $\varphi_E(\cdot; \theta_E)$ and $\varphi_P(\cdot; \theta_P)$, for modeling elastic and plastic

Table 1. **Benchmark**. We compare our method against 4 baselines and 2 variations of our method, while also noting the difference in architecture or hyper-parameters. We use column **#Iter.** as the number of iterations, **#Hist.** as the K value for the top-k retrieval in the historical optimization steps, $\frac{\text{\#Exploit}}{\text{\#Explore}}$ as the number of offspring for exploitation versus exploration, **Bilevel** as if bilevel optimization is enabled. Our experiments encompass 8 different tasks, which are divided into constitutive law search (**a-d**) and molecule design (**e-h**). A lower loss value is preferable across all tasks. The best method with the lowest loss is highlighted in **bold** text.

Method	#Iter.	#Hist.	$\frac{\text{\#Exploit}}{\text{\#Explore}}$	Bilevel	Constitutive Law Search				Molecule Design			
					(a)$\downarrow$	(b)$\downarrow$	(c)$\downarrow$	(d)$\downarrow$	(e)$\downarrow$	(f)$\downarrow$	(g)$\downarrow$	(h)$\downarrow$
CoT	1	5	N/A	✗	298.5	1462.3	150.0	384.1	3.0	32.1	18.6	6.0
FunSearch	20	2	0 / 4	✗	210.3	872.2	82.8	139.5	1.1	7.1	8.3	1.1
Eureka	5	1	0 / 16	✗	128.0	531.0	101.7	150.1	4.3	9.8	3.3	9.7e-1
OPRO	5	5	0 / 16	✗	136.2	508.3	99.2	128.8	2.4	9.4	3.1	1.3
Ours (no bilevel)	5	5	4 / 12	✗	90.2	517.0	83.6	68.4	8.6e-1	9.1	1.8	1.4
Ours (no exploit)	5	5	0 / 16	✓	3.0e-3	3.9e-1	6.6e-2	**1.4e-12**	4.0e-4	1.5e-1	6.1e-1	**2.8e-5**
Ours	5	5	4 / 12	✓	**5.2e-5**	**2.1e-1**	**6.0e-2**	**1.4e-12**	**1.3e-4**	**1.1e-1**	5.4e-1	3.6e-5

materials respectively, and they are formally defined as:

$$\varphi_E\left(\mathbf{F};\theta_E\right) \mapsto \boldsymbol{\tau} \tag{4a}$$

$$\varphi_P\left(\mathbf{F};\theta_P\right) \mapsto \mathbf{F}^{\text{corrected}}, \tag{4b}$$

where $\mathbf{F} \in \mathbb{R}^{3\times3}$ is the deformation gradient, $\boldsymbol{\tau} \in \mathbb{R}^{3\times3}$ is the Kirchhoff stress tensor, $\mathbf{F}^{\text{corrected}} \in \mathbb{R}^{3\times3}$ is the deformation gradient after plastic return-mapping correction, and θ_E and θ_P are the continuous material parameters for elastic and plastic constitutive laws respectively. Given a specific constitutive law, we input it to the differentiable simulation and yields a particle position trajectory:

$$X_{t\in[1,...,T]} = \text{sim}\left(\varphi\left(\cdot;\theta\right)\right), \tag{5}$$

and we optimize the constitutive law by fitting the output trajectory to the ground truth $\hat{X}_{t\in[1,...,T]}$.

Molecule Design In this study, we focus on a prevalent task in molecule design: discovering molecules with specific quantum mechanical properties. Our objective is to determine the optimal molecular structure and its 3D conformation to match a predefined target quantum mechanical property. The design process involves both the discrete expression – the molecular structure represented by SMILES strings (Weininger, 1988), and the continuous parameters – the 3D coordinates of each atom in the molecule. The methodology comprises two loops: In the outer loop, the LLM generates the initial molecular structure as a SMILES string, along with a preliminary guess for the 3D atom coordinates. The inner loop involves simultaneous optimization of both the molecule's 3D conformation and quantum mechanical properties, both determined by 3D atom positions.

For the generation of 3D conformations, we utilize the ETKGD algorithm (Riniker & Landrum, 2015) followed by optimization using the Merck Molecular Force Field (MMFF) (Halgren, 1996), both implemented within the RD-Kit (Landrum et al., 2013). To get the quantum mechanical
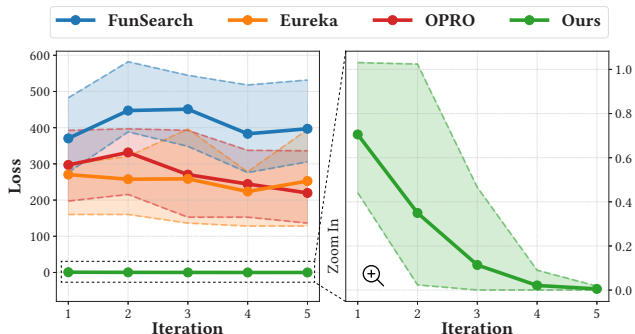

Figure 2. **Loss trends comparison.** Loss of the best solution averaged across seeds at different iterations of LLM-driven optimization, where the shading shows the min/max value.

property values, we employ UniMol (Zhou et al., 2023), a pre-trained transformer-based large model, which has been fine-tuned on the QM9 dataset (Ramakrishnan et al., 2014).

3.2. Experiment Setup

Task Design We design a diverse set of challenging tasks for evaluation. For constitutive law discovery, we propose 4 tasks including: **(a)** fitting the non-linear elastic material starting from a linear elastic material, **(b)** fitting the von Mises plastic material starting from a purely elastic material, **(c)** fitting the granular material starting from a purely elastic material, and **(d)** fitting the weakly compressible fluid starting from a purely elastic material. For molecular design task, we consider 4 popular tasks, centering on 3 commonly evaluated quantum mechanical properties (Fang et al., 2022; Zhou et al., 2023), each set to different target values: **(e)** HOMO (Highest Occupied Molecular Orbital) set to 0, **(f)** LUMO (Lowest Unoccupied Molecular Orbital) set to 0, **(g)** the HOMO-LUMO energy gap set to 0, and **(h)** the HOMO-LUMO energy gap set to -2. All these values are normalized on all data in QM9 dataset.

Table 2. **Comparison with symbolic regression**. We compare our method against 5 most performant methods in SRBench (La Cava et al., 2021) and 3 pre-trained symbolic regression methods. **Sym.** denotes whether the result is symbolic or not.

Method	$R2 \uparrow$	$MSE \downarrow$	$MAE \downarrow$	Sym.
FFX	0.9824	4.5e+5	3.7e+2	✓
MLP	0.9876	3.2e+5	3.4e+2	✗
FEAT	0.9964	9.2e+4	1.7e+2	✓
DSO	0.9968	8.2e+4	9.2e+1	✓
Operon	0.9988	2.8e+4	9.8e+1	✓
SymbolicGPT	0.5233	6.9e+6	1.7e+3	✓
NeSymReS	N/A to >3 variables			✓
T-JSL	N/A to >2 variables			✓
Ours	**0.9990**	**1.7e+4**	**8.6e+1**	✓

Implementation Details We run all our experiments 5 times with different random seeds following previous practices (Ma et al., 2024). Due to the complexity of the task, we provide a simple bootstrapping example of a valid design to ensure the success rate. We use warp (Macklin, 2022) for the differentiable MPM simulation, and we develop our inner-level optimization upon PyTorch (Paszke et al., 2019). In all our experiments, we use mean square error as the criteria and Adam optimizer (Kingma & Ba, 2015). We choose `gpt-4-turbo-preview` as the backbone model for LLM and tentatively set the exploiting temperature $T_l = 0.5$ and exploring temperature $T_h = 1.0$.

3.3. Physical Scientific Discovery

We consider 6 strong baselines for evaluation: (i) **Chain-of-Thoughts (CoT)** prompting (Wei et al., 2022) solves the problem by looking at step-by-step solutions from examples. We provide 5 examples with an explanation to CoT as the initial solution. (ii) **FunSearch** (Romera-Paredes et al., 2023) utilizes evolutionary strategy to avoid local optimum. We adopt the given hyperparameters from the original implementation with 2 optimization histories and 4 explorers. We set the number of iterations to 20, yielding the same number of solutions evaluated, for a fair comparison to other methods. (iii) **Eureka** (Ma et al., 2024) generates multiple solutions in each iteration to improve the success rate of the generated code. We keep the hyperparameters from the original implementation. (iv) **Optimization by PROmpting (OPRO)** (Yang et al., 2024) highlights the advantages of involving a sorted optimization trajectory. We set the hyperparameters to be equal to **Eureka** except for the number of historical optimization steps. In all these works (i-iv), we notice the temperatures for LLM inference are all 1.0, which is equal to the exploring temperature in our method, so we denote them with 0 exploiter. We also consider 2 variants of our method: (v) **Ours (no bilevel)**

Table 3. **Comparison with population-based molecule design**. We compare our method against a traditional population-based molecule design method GhemGE (Yoshikawa et al., 2018) and report the results of molecule design tasks **(e-h)**.

Method	(e) $\downarrow$	(f) $\downarrow$	(g) $\downarrow$	(h) $\downarrow$
GhemGE	4.8e-3	1.8	1.5	9.8e-5
Ours	**1.3e-4**	**1.1e-1**	**5.4e-1**	**3.6e-5**

Table 4. **Experiment in imaginary constitutive law**. We construct an imaginary constitutive law to keep LLM from cheating by memorization and report the results of our method and baselines.

Method	FunSearch	Eureka	OPRO	Ours
Loss	105.0	89.1	98.0	**1.3e-3**

removes the bilevel optimization by only searching with LLM. (vi) **Ours (no exploit)** removes the exploitation by setting the temperature to 1.0 all the time.

We present our experiments against the 8 designed tasks and show the results in Table 1. Compared to baselines (i-iv), our method is significantly better by a number of magnitudes. When the bilevel optimization is removed from our method, the performance drops dramatically, but still statistically better than baselines (i-iv), indicating the choice of hyperparameters and the integration of exploitation is helpful for the task. When we remove the exploitation but restore the bilevel optimization, we notice the performance grows back. It has comparable performance compared to our method in **(d)** or even better results in **(h)**. However, in some tasks, especially hard ones (e.g., **(b)** and **(f)**) that we care more in reality, the performance gap is over 50%, indicating the effectiveness of our exploit-and-explore strategy. We also present the loss trend in task **(a)** in Figure 2, our method outstands with a much lower loss and a converging trend.

We also compare our method with traditional methods in each specific area to demonstrate the generalizability of our method. First, we reformulate our constitutive law search task **(a)** into a symbolic regression task by (i) capture the ground-truth output (the stress tensors) as the supervision, and (ii) separate the 9 output dimension into 9 independent problems and ensemble them for evaluation. Note that these modifications dramatically simplified the original task: we removed back-propagation through time (BPTT) and directly discover the constitutive law without surrogate loss. We evaluate 14 traditional baselines in SRBench (La Cava et al., 2021) and 3 data-driven pre-trained baselines. We select the top few baselines in Table 2 and show the rest in the Appendix C.1. As shown the table, our method topped on this task even with a much more challenging setting. Also, since our method depends on the in-context learning ability of LLMs, it has little constraint in the num-
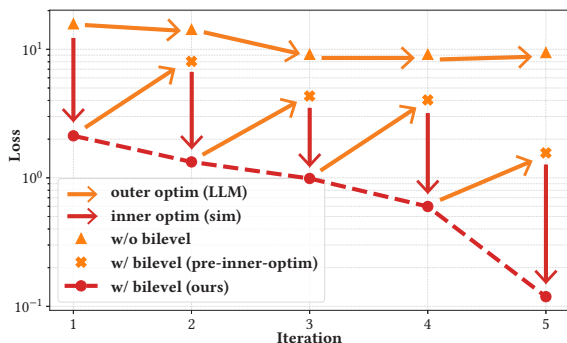
Figure 3. **Ablation on bilevel optimization.** We denote the optimization trajectory with and without out bilevel optimization with red dot and orange triangle respectively. We visualize the intermediate step of our method before the inner-level optimization using orange cross. We also highlight the outer LLM optimization and inner simulation optimization using orange and red arrows.

Figure 4. **Ablation on the backbone LLM.** We compare the performances of 4 selected backbone LLMs and report the rank of them. A outer curve indicates a better performance.

ber of variables than the data-driven pre-trained baselines. For moledule design tasks, we also compare our method with GhemGE (Yoshikawa et al., 2018), which employs a population-based molecule design algorithm. As shown in Table 3, our method presents a much lower loss, demonstrating the general effectiveness of our method.

3.4. Ablation Study

Generalization or Memorization   In order to figure out if the improvement introduced by our method is merely because the LLM saw the solutions in its training phase, we design an experiment ablating it by making it invent an imaginary constitutive law that does not exist on the earth. We mix the constitutive law of von Mises plasticity, granular material, and weakly compressible fluid by 50%, 30%, and 20%, so that the new constitutive law represents an imaginary material whose behavior is extremely complex. We repeat our experiment setup as in Figure 1. We compare our method against the baselines and report the performances in Table 4. As shown in the table, our method can still discover the constitutive law with a low quantitative loss. From our observation, there is *very little visual difference* between the ground-truth material and the optimized constitutive law. We show the discovered constitutive law in Appendix D.9.

Bilevel Optimization is the Key   Here we evaluate the importance of bilevel optimization in Figure 3 using the task **(h)**. Comparing the blue triangle curve and the red dot curve, which represent the LLM-driven outer-level optimization and the simulation-driven inner-level optimization, it is easy to conclude that the loss performance with bilevel optimization is better. Nevertheless, we are also interested in how bilevel optimization works inside each optimization step and how much LLMs and simulations help respectively. As shown as a zigzag curve, we found that LLMs and simulations help each other over all optimization steps: the
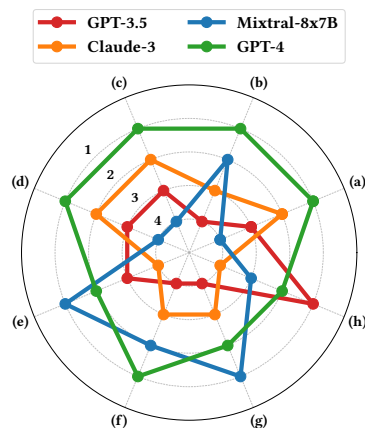
next proposal from LLMs will be better with simulation-optimized results, and vice versa. We argue that LLMs and simulations have different expertise: LLMs are generalist scientists who have cross-discipline knowledge, while simulations are domain experts who have specialized knowledge.

LLM Backbone   In addition to GPT-4 (OpenAI, 2023), we repeat the experiments in Table 1 using 3 additional LLM backbones: (i) GPT-3.5 (Ouyang et al., 2022), (ii) Claude-3-Sonnet (Anthropic, 2024), and (iii) Mixtral-8x7B (Jiang et al., 2024), and report the rank of them in Figure 4. Indicated by the largest area, GPT-4, as our choice, statistically outperforms the other methods. Interestingly, we found Claude-3-Sonnet is the second top method on most of constitutive law search task, while Mixtral-8x7B even tops on 2 molecule design tasks. As a result, our workflow also works for other LLMs, however, our suggestion for practitioners is to try GPT-4 as the first choice but also consider open-source model (e.g., Mixtral-8x7B) for budget or customizability.

Exploitation v.s. Exploration   We visualize the statistics of the simulation execution status in Figure 5 (a) using the task **(b)**, which is one of the most challenging tasks in our experiments. When the exploitation is removed, the error rate dramatically increases, as shown by a decrease in green bars. It leads to a degeneration in the performance of the methods with exploitation as shown in Figure 5 (b). However, even though the success rate remains high, when exploration is removed, the optimization result is still worse than keeping them both. We argue that exploration is significant when the optimization problem is challenging, especially in our case, where the search space is highly non-linear and unstructured and resulting in numerous local optimum.
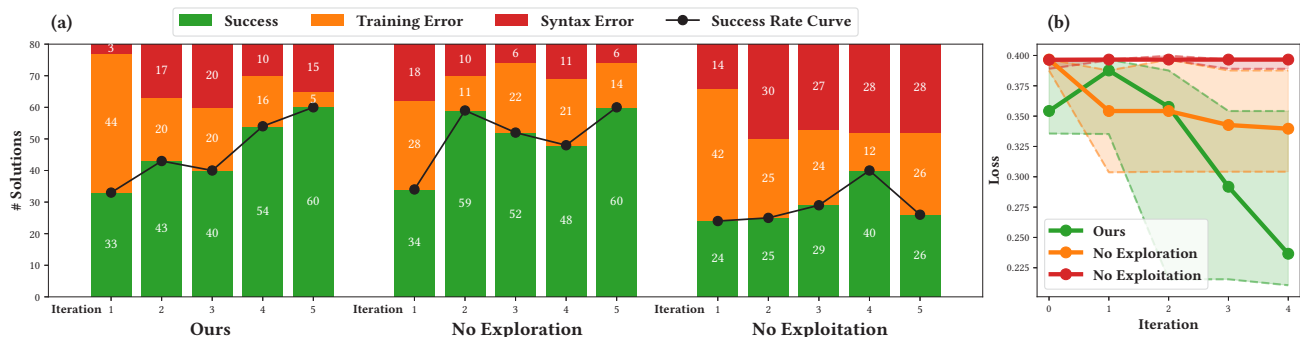
Figure 5. **Ablation on exploration-exploitation.** (a) Histogram of solutions that are valid for simulation (Eq. 3b) across iterations. (b) Loss ($\mathcal{L}$ in Sec. 2.1) of the best solution averaged across seeds at different iterations, where the shading indicates the min/max values.

3.5. Case Study

Constitutive Law Search We provide a trimmed snippet of our searched constitutive law in Figure 6 (a) for task (a) where a highly non-linear material is provided as the trajectory to fit. We reformat the code slightly to fit into the text, where the complete example can be found in the Appendix. Starting from a linear material, our method is able to automatically generate the constitutive law with a quadratic deviatoric term. Note that our method also provides a concrete implementation of __init__ function that defines the continuous parameters in the computational graph for later inner-level optimization.

Molecule Design When comparing the two molecules with respect to their HOMO-LUMO energy gap based on optimized results from the LLM as shown in Figure 6 (b), we observe distinct characteristics in each: (i) **Molecule A** (gap-0) includes sulfur and chlorine atoms attached to a ring, coupled with a trifluoromethyl group, introducing electron-withdrawing effects, and (ii) **Molecule B** (gap-2) includes oxygen (notably in ethers) and sulfur within the ring structures introducing localized non-bonding electron pairs. Furthermore, the overall structure of Molecule B is more complex than that of Molecule A, containing multiple rings. An intriguing aspect of Molecule B, which might initially defy expectations, is the presence of a single fluorine atom. The high electronegativity of fluorine typically leads to electron density withdrawal, influencing the gap value. However, due to the complexity of Molecule B's structure, the impact of the fluorine atom is somewhat localized, thereby not significantly altering the gap value.

4. Related Work

4.1. Automated Scientific Discovery

Automated scientific discovery, enhanced by machine learning methods, serves as a powerful accelerator for research, enabling scientists to generate hypotheses, design experiments, interpret vast datasets, and unearth insights that may


```python
class Physics(nn.Module):
    def __init__(self, youngs_modulus_log: float = 13.03,
                       poissons_ratio_sigmoid: float = -1.99):
        super().__init__()
        self.youngs_modulus_log = nn.Parameter(
            torch.tensor(youngs_modulus_log))  # Log of Young's modulus
        self.poissons_ratio_sigmoid = nn.Parameter(
            torch.tensor(poissons_ratio_sigmoid))  # Sigmoid of Poisson's ratio
    def forward(self, F: torch.Tensor) -> torch.Tensor:
        youngs_modulus = self.youngs_modulus_log.exp()
        poissons_ratio = torch.sigmoid(self.poissons_ratio_sigmoid) * 0.49
        mu = youngs_modulus / (2 * (1 + poissons_ratio))  # Shear modulus
        lam = youngs_modulus * poissons_ratio / (
            (1 + poissons_ratio) * (1 - 2 * poissons_ratio))
        # Deformation gradient determinant J
        J = F.det().view(-1, 1, 1)
        F_invT = F.inverse().transpose(1, 2)
        # Volumetric part
        P_vol = lam * (J - 1) * F_invT
        # Deviatoric part
        P_dev = mu * (F - (1 / J) * F_invT)
        # Compute Kirchhoff stress tensor
        kirchhoff_stress = P_vol + P_dev @ F.transpose(1, 2)
        return kirchhoff_stress
```
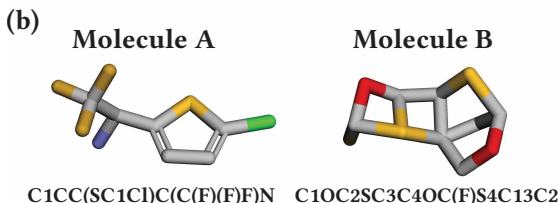

Figure 6. **Case Study.** (a) We give a concrete example of the searched constitutive law. (b) We provide 2 novel molecules optimized for different objectives with their SMILES stings.

elude traditional scientific methodologies (AI4Science & Quantum, 2023; Kramer et al., 2023; Wang et al., 2023). This multifaceted process unfolds through two synergistically linked stages: hypothesis formation and the collection and analysis of experimental data. The integration of automated systems not only augments the scientific inquiry process but also streamlines the discovery pipeline, from conceptualization to empirical validation. This paper places a particular emphasis on, but is not limited to, constitutive law discovery and molecular design. These areas exemplify the profound impact of automation in unraveling complex material behaviors and in the innovative design of molecules with tailored properties. Automatic identification of constitutive material models has been a long-standing problem and recent works utilizes differentiable simulation (Du et al., 2021; Ma et al., 2023; 2021) to address it as a system identification problem. Leveraging machine learning and artificial intelligence, researchers are able to predict molecular be-

havior, optimize chemical structures for specific functions, and thus, rapidly accelerate the development of new drugs, materials, and chemicals (Jin et al., 2018; Zhou et al., 2019; Schneider, 2018).

4.2. Large Language Models and Agents

The advancement of Large Language Models (LLMs) such as ChatGPT and GPT-4 has sparked considerable interest in their potential as autonomous agents (Brown et al., 2020; OpenAI, 2022; 2023). Recent developments have shown that LLMs can be enhanced to solve complex problems by creating and utilizing their own tools, as demonstrated in the LATM framework (Sumers et al., 2024), and by acting as optimizers in the absence of gradients, as seen in the OPRO methodology (Yang et al., 2024). These approaches signify a shift towards more independent and versatile LLM-based agents capable of generating solutions through self-crafted tools and optimization techniques (Cai et al., 2024; Yao et al., 2023b;a), showcasing their evolving problem-solving capabilities. In the realm of scientific discovery, LLMs have begun to make significant contributions, particularly in mathematics and computational problems. The FunSearch method (Romera-Paredes et al., 2023) pairs LLMs with evaluators to exceed known results in extremal combinatorics and online bin packing, illustrating LLMs' ability to discover new solutions to established problems. Similarly, AlphaGeometry's success (Trinh et al., 2024) in solving olympiad-level geometry problems without human demonstrations highlights the potential of LLMs in automating complex reasoning tasks. These examples underline the transformative impact of LLMs in pushing the boundaries of scientific inquiry and automated reasoning.

4.3. Bilevel Optimization

Bilevel optimization involves a hierarchical structure with two levels of optimization problems, where the solution to the upper-level problem is contingent upon the outcome of the lower-level problem (Colson et al., 2007). Bilevel optimization problems are inherently more complex than their single-level counterparts due to the nested nature of the optimization tasks and the intricate interdependencies between them. Recent advancements have focused on developing efficient algorithms, including evolutionary algorithms (Sinha et al., 2017b), gradient-based approaches (Liu et al., 2022), and approximation techniques (Sinha et al., 2017a), to tackle the computational challenges presented by the non-convex and non-differentiable characteristics of many bilevel problems. Among a wide span of application domains of bilevel optimization, neural architecture search (NAS) (Liu et al., 2019; Bender et al., 2018; Cai et al., 2019; Xue et al., 2021) is prominent and close to the problem setting in this paper: the upper level optimizes the discrete neural network architecture while the lower level optimizes the continu-

ous weights of the neural network. However, typical NAS methods require a predefined search space, constraining the exploration of discrete network architectures to manually specified boundaries. Our framework distinguishes itself by employing LLM encoded with general knowledge and gets rid of the limitations imposed by manual design constraints.

5. Conclusion

We consider a few limitations and future directions. (i) Although we prompt the LLM to generate pseudo-code plans and comments, it is generally hard to ensure the interpretability of LLM-generated solutions. (ii) Since the LLM-generated codes are executed directly without any filtering in our application, there exists potential AI safety risk that hazards the operating system. (iii) Our method only utilizes the internal knowledge of LLMs as the prior, where in reality people design manual constraints and rule to regularize and improve the optimization (Udrescu et al., 2020). We leave these domain-specific applications and human feedback-based regularization methods as our future work. (iv) The performance our method highly depends on the differentiablity of the generated code. However, Zero-order optimizers (Hansen, 2006) should also shine since the number of continuous parameters is relatively limited. (v) LLM inference requires large computational resources and thus increases expense. For example, it spends around $10 for our method to complete one task using GPT-4, which will be increasingly inacceptable when the number of iteration grows. (vi) Due to the reuse of previously generated solutions in our proposed top-k heap, the KV cache in LLM will be highly similar between neighbor iterations. It opens a gate for recent KV cache optimization methods (Zheng et al., 2023) to speedup our method by KV cache reusing.

In conclusion, we present Scientific Generative Agent, a bilevel optimization framework: LLMs serve as knowledgeable and adaptable thinkers, formulating scientific solutions like physics equations or molecule structures; concurrently, simulations operate as platforms for experimentation, offering observational feedback and optimizing continuous components like physical parameters. We focused on two scientific problems: constitutive law search and molecular design. Our approach outperforms other LLM-based benchmark methods, delivering consistent, robust, and nearly monotonic improvement. Furthermore, it shows exceptional ability in identifying unknown, true constitutive laws and molecular structures. Remarkably, our system generates innovative solutions that, despite being unconventional, are deemed reasonable after being thoroughly analyzed by experts in their respective domains. We view our process as a trailblazer, establishing a new paradigm for utilizing LLMs and simulations as bilevel optimization to further advancements in physical scientific discoveries.

Acknowledgements

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

AI4Science, M. R. and Quantum, M. A. The impact of large language models on scientific discovery: a preliminary study using gpt-4. *arXiv preprint arXiv:2311.07361*, 2023.

Anthropic. Introducing the next generation of claude, 2024. URL https://www.anthropic.com/news/claude-3-family.

Arnaldo, I., Krawiec, K., and O'Reilly, U.-M. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 879–886, 2014.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pp. 550–559. PMLR, 2018.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pp. 936–945. Pmlr, 2021.

Boiko, D. A., MacKnight, R., Kline, B., and Gomes, G. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Cai, T., Wang, X., Ma, T., Chen, X., and Zhou, D. Large language models as tool makers. In *International Conference on Learning Representations*, 2024.

Cava, W. L., Singh, T. R., Taggart, J., Suri, S., and Moore, J. Learning concise representations for regression by evolving networks of trees. In *International Conference on Learning Representations*, 2019.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Chithrananda, S., Grand, G., and Ramsundar, B. Chemberta: large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885*, 2020.

Colson, B., Marcotte, P., and Savard, G. An overview of bilevel optimization. *Annals of operations research*, 153: 235–256, 2007.

Du, T., Wu, K., Ma, P., Wah, S., Spielberg, A., Rus, D., and Matusik, W. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (TOG)*, 41(2):1–21, 2021.

Fang, X., Liu, L., Lei, J., He, D., Zhang, S., Zhou, J., Wang, F., Wu, H., and Wang, H. Geometry-enhanced molecular representation learning for property prediction. *Nature Machine Intelligence*, 4(2):127–134, 2022.

Fortunato, S., Bergstrom, C. T., Börner, K., Evans, J. A., Helbing, D., Milojević, S., Petersen, A. M., Radicchi, F., Sinatra, R., Uzzi, B., et al. Science of science. *Science*, 359(6379):eaao0185, 2018.

Halgren, T. A. Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94. *Journal of computational chemistry*, 17(5-6):490–519, 1996.

Hansen, N. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pp. 75–102, 2006.

Huang, Q., Vora, J., Liang, P., and Leskovec, J. Benchmarking large language models as ai research agents. *arXiv preprint arXiv:2310.03302*, 2023.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., and Selle, A. The material point method for simulating continuum materials. In *Acm siggraph 2016 courses*, pp. 1–52. 2016.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pp. 2323–2332. PMLR, 2018.

Jin, Y., Fu, W., Kang, J., Guo, J., and Guo, J. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, San Diega, CA, USA, 2015.

Kommenda, M., Burlacu, B., Kronberger, G., and Affenzeller, M. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21(3):471–501, 2020.

Kramer, S., Cerrato, M., Džeroski, S., and King, R. Automated scientific discovery: From equation discovery to autonomous discovery systems. *arXiv preprint arXiv:2305.02251*, 2023.

La Cava, W., Helmuth, T., Spector, L., and Moore, J. H. A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation*, 27(3):377–402, 2019.

La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. Contemporary symbolic regression methods and their relative performance. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Landrum, G. et al. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling. *Greg Landrum*, 8:31, 2013.

Li, J., Liu, Y., Fan, W., Wei, X.-Y., Liu, H., Tang, J., and Li, Q. Empowering molecule discovery for molecule-caption translation with large language models: A chatgpt perspective. *arXiv preprint arXiv:2306.06615*, 2023.

Li, W., Li, W., Sun, L., Wu, M., Yu, L., Liu, J., Li, Y., and Tian, S. Transformer-based model for symbolic regression via joint supervised learning. In *The Eleventh International Conference on Learning Representations*, 2022.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Liu, R., Mu, P., Yuan, X., Zeng, S., and Zhang, J. A general descent aggregation framework for gradient-based bi-level optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):38–57, 2022.

Liu, Z., Roberts, R. A., Lal-Nag, M., Chen, X., Huang, R., and Tong, W. Ai-based language models powering drug discovery and development. *Drug Discovery Today*, 26 (11):2593–2607, 2021.

Ma, P., Du, T., Tenenbaum, J. B., Matusik, W., and Gan, C. Risp: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation. In *International Conference on Learning Representations*, 2021.

Ma, P., Chen, P. Y., Deng, B., Tenenbaum, J. B., Du, T., Gan, C., and Matusik, W. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In *International Conference on Machine Learning*. PMLR, 2023.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

Macklin, M. Warp: A high-performance python framework for gpu simulation and graphics, March 2022. NVIDIA GPU Technology Conference.

McConaghy, T. Ffx: Fast, scalable, deterministic symbolic regression technology. *Genetic Programming Theory and Practice IX*, pp. 235–260, 2011.

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., and Petersen, B. K. Symbolic regression via neural-guided genetic programming population seeding. In *Advances in Neural Information Processing Systems*, 2021.

OpenAI. OpenAI: Introducing ChatGPT, 2022. URL https://openai.com/blog/chatgpt.

OpenAI. OpenAI: GPT-4, 2023. URL https://openai.com/research/gpt-4.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2020.

Popper, K. *The logic of scientific discovery*. Routledge, 2005.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Riniker, S. and Landrum, G. A. Better informed distance geometry: using what we know to improve conformation generation. *Journal of chemical information and modeling*, 55(12):2562–2574, 2015.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, pp. 1–3, 2023.

Rosenberg, A. and McIntyre, L. *Philosophy of science: A contemporary introduction*. Routledge, 2019.

Schapire, R. E. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, pp. 149–171, 2003.

Schneider, G. Automating drug discovery. *Nature reviews drug discovery*, 17(2):97–113, 2018.

Sharma, G. and Thakur, A. Chatgpt in drug discovery. 2023.

Sinha, A., Malo, P., and Deb, K. Evolutionary algorithm for bilevel optimization using approximations of the lower level optimal solution mapping. *European Journal of Operational Research*, 257(2):395–411, 2017a.

Sinha, A., Malo, P., and Deb, K. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017b.

Sulsky, D., Zhou, S.-J., and Schreyer, H. L. Application of a particle-in-cell method to solid mechanics. *Computer physics communications*, 87(1-2):236–252, 1995.

Sumers, T., Yao, S., Narasimhan, K., and Griffiths, T. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. Survey Certification.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *Advances in Neural Information Processing Systems*, 33:4860–4871, 2020.

Valipour, M., You, B., Panju, M., and Ghodsi, A. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

Virgolin, M., Alderliesten, T., and Bosman, P. A. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pp. 1084–1092, 2019.

Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P. A. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation*, 29(2):211–237, 2021.

Wang, H., Fu, T., Du, Y., Gao, W., Huang, K., Liu, Z., Chandak, P., Liu, S., Van Katwyk, P., Deac, A., et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Weininger, D. 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.

Wuestman, M., Hoekman, J., and Frenken, K. A typology of scientific breakthroughs. *Quantitative Science Studies*, 1(3):1203–1222, 2020.

Xue, C., Wang, X., Yan, J., Hu, Y., Yang, X., and Sun, K. Rethinking bi-level optimization in neural architecture search: A gibbs sampling perspective. In *AAAI Conference on Artificial Intelligence*, volume 35, pp. 10551–10559, 2021.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. R. Tree of thoughts: Deliberate problem solving with large language models. In *Conference on Neural Information Processing Systems*, 2023a.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Yoshikawa, N., Terayama, K., Sumita, M., Homma, T., Oono, K., and Tsuda, K. Population-based de novo molecule generation, using grammatical evolution. *Chemistry Letters*, 47(11):1431–1434, 2018.

Zheng, L., Yin, L., Xie, Z., Huang, J., Sun, C., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104*, 2023.

Zhou, G., Gao, Z., Ding, Q., Zheng, H., Xu, H., Wei, Z., Zhang, L., and Ke, G. Uni-mol: A universal 3d molecular representation learning framework. In *International Conference on Learning Representations*, 2023.

Zhou, Z., Kearnes, S., Li, L., Zare, R. N., and Riley, P. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):10752, 2019.

A. Full Prompts

System prompt for constitutive law discovery:

```
You are an intelligent AI assistant for coding, physical simulation, and scientific discovery.
Follow the user's requirements carefully and make sure you understand them.
Your expertise is strictly limited to physical simulation, material science, mathematics, and coding.
Keep your answers short and to the point.
Do not provide any information that is not requested.
Always document your code as comments to explain the reason behind them.
Use Markdown to format your solution.
You are very familiar with Python and PyTorch.
Do not use any external libraries other than the libraries used in the examples.
```

System prompt for molecule design:

```
You are an intelligent AI assistant for coding, molecule design, and scientific discovery.
Follow the user's requirements carefully and make sure you understand them.
Your expertise is strictly limited to physical simulation, material science, chemistry, molecule design, mathematics,
and coding.
Keep your answers short and to the point.
Do not provide any information that is not requested.
Always document your code as comments to explain the reason behind them.
Use Markdown to format your solution.
You are very familiar with PyTorch.
Your are very familiar with the SMILES notation (Simplified Molecular-Input Line-Entry System).
Do not use any external libraries other than the libraries used in the examples.
```

Coding format prompt for elastic constitutive law discovery:

```
## Format Requirements

### PyTorch Tips
1. When element-wise multiplying two matrix, make sure their number of dimensions match before the operation. For
example, when multiplying 'J' (B,) and 'I' (B, 3, 3), you should do 'J.view(-1, 1, 1)' before the operation. Similarly,
'(J - 1)' should also be reshaped to '(J - 1).view(-1, 1, 1)'. If you are not sure, write down every component in the
expression one by one and annotate its dimension in the comment for verification.
2. When computing the trace of a tensor A (B, 3, 3), use 'A.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)'. Avoid
using 'torch.trace' or 'Tensor.trace' since they only support 2D matrix.

### Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For example, '# (B, 3, 3)'.
3. The only library allowed is PyTorch. Follow the examples provided by the user and check the PyTorch documentation to
learn how to use PyTorch.
4. Separate the code into continuous physical parameters that can be tuned with differentiable optimization and the
symbolic constitutive law represented by PyTorch code. Define them respectively in the '__init__' function and the
'forward' function.
5. The first output of the 'forward' function is the updated deformation gradient. Always remember the second output of
the 'forward' function is Kirchhoff stress tensor, which is defined by the matrix multiplication between the first
Piola-Kirchhoff stress tensor and the transpose of the deformation gradient tensor. Formally, 'tau = P @ F^T', where tau
is the Kirchhoff stress tensor, P is the first Piola-Kirchhoff stress tensor, and F is the deformation gradient tensor.
Do not directly return any other type of stress tensor other than Kirchhoff stress tensor. Compute Kirchhoff stress
tensor using the equation: 'tau = P @ F^T'.
6. The proposed code should strictly follow the structure and function signatures below:

```python
import torch
import torch.nn as nn


class Physics(nn.Module):

    def __init__(self, param: float = DEFAULT_VALUE):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        Tentatively initialize the parameters with the default values in args.

        Args:
            param (float): the physical meaning of the parameter.
        """
        super().__init__()
        self.param = nn.Parameter(torch.tensor(param))

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute Kirchhoff stress tensor from deformation gradient tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            kirchhoff_stress (torch.Tensor): Kirchhoff stress tensor (B, 3, 3).
        """
        return kirchhoff_stress
```

Solution Requirements
```

```
1. Analyze step-by-step what the potential problem is in the previous iterations based on the feedback. Think about why
the results from previous constitutive laws mismatched with the ground truth. Do not give advice about how to optimize.
Focus on the formulation of the constitutive law. Start this section with "### Analysis". Analyze all iterations
individually, and start the subsection for each iteration with "#### Iteration N", where N stands for the index.
Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about how to separate your algorithm into a
continuous physical parameter part and a symbolic constitutive law part. Describe your plan in pseudo-code, written out
in great detail. Remember to update the default values of the trainable physical parameters based on previous
optimizations. Start this section with "### Step-by-Step Plan".
3. Output the code in a single code block """python ... """ with detailed comments in the code block. Do not add any
trailing comments before or after the code block. Start this section with "### Code".
```

Coding format prompt for plastic constitutive law discovery:

```
## Format Requirements

### PyTorch Tips
1. When element-wise multiplying two matrix, make sure their number of dimensions match before the operation. For
example, when multiplying 'J' (B,) and 'I' (B, 3, 3), you should do 'J.view(-1, 1, 1)' before the operation. Similarly,
'(J - 1)' should also be reshaped to '(J - 1).view(-1, 1, 1)'. If you are not sure, write down every component in the
expression one by one and annotate its dimension in the comment for verification.
2. When computing the trace of a tensor A (B, 3, 3), use 'A.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)'. Avoid
using 'torch.trace' or 'Tensor.trace' since they only support 2D matrix.

### Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For example, '# (B, 3, 3)'.
3. The only library allowed is PyTorch. Follow the examples provided by the user and check the PyTorch documentation to
learn how to use PyTorch.
4. Separate the code into continuous physical parameters that can be tuned with differentiable optimization and the
symbolic deformation gradient correction model represented by PyTorch code. Define them respectively in the '__init__'
function and the 'forward' function.
5. The proposed code should strictly follow the structure and function signatures below:

'''python
import torch
import torch.nn as nn


class Physics(nn.Module):

    def __init__(self, param: float = DEFAULT_VALUE):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        Tentatively initialize the parameters with the default values in args.

        Args:
            param (float): the physical meaning of the parameter.
        """
        super().__init__()
        self.param = nn.Parameter(torch.tensor(param))

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute corrected deformation gradient from deformation gradient tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            F_corrected (torch.Tensor): corrected deformation gradient tensor (B, 3, 3).
        """
        return F_corrected
'''

### Solution Requirements

1. Analyze step-by-step what the potential problem is in the previous iterations based on the feedback. Think about why
the results from previous constitutive laws mismatched with the ground truth. Do not give advice about how to optimize.
Focus on the formulation of the constitutive law. Start this section with "### Analysis". Analyze all iterations
individually, and start the subsection for each iteration with "#### Iteration N", where N stands for the index.
Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about if the plasticity is needed to improve
performance. Remember that plasticity is not necessary. If your analysis supports plasticity, think about how to update
deformation gradient using plasticity. Think about how to separate your algorithm into a continuous physical parameter
part and a symbolic deformation gradient correction model part. Describe your plan in pseudo-code, written out in great
detail. Remember to update the default values of the trainable physical parameters based on previous optimizations.
Start this section with "### Step-by-Step Plan".
3. Output the code in a single code block """python ... """ with detailed comments in the code block. Do not add any
trailing comments before or after the code block. Start this section with "### Code".
```

Coding format prompt for molecule design:

```
## Format Requirements

### Code Requirements

1. The programming language is always python.
```

```
2. Annotate the size of the tensor as comment after each tensor operation. For example, '# (B, 3, 3)'.
3. Separate the code into: (1) python string 'SMILES': the SMILES string describing the molecular topology structure and
atomic types, and (2) matrix 'coordinates' the 3D coordinates of all atoms. These representations should not include
hydrogens.
4. The SMILES string should be valid. Use your knowledge about Simplified Molecular-Input Line-Entry System to help you
design a valid one.
5. The number of atoms in the SMILES string should be no less than 8, which means the number of atoms should be >= 8.
Try to generate molecule with diverse atoms.
6. The 3D coordinates of the atoms should not be overlapping with each other. In another word, every row in the matrix
'coordinates' should be distinct from each other.
7. The 'coordinates' matrix is of shape '(N, 3)' where 'N' stands for the number of atoms in the molecule. It should be
identical to the number of atoms that the proposed SMILES string represents. State out the shape of any matrix defined
in the comment as shown in the following example. State out the number of atoms that the SMILES string represents in the
comment as shown in the following example.
8. The discrete SMILES string is critical in this problem since it defines the structure and cannot be tuned using
differentiable optimization. Please propose different SMILES string from all examples or iterations above to discover
and evaluate more structure. This is very important.
9. The proposed code should strictly follow the structure and function signatures below:

'''python
SMILES: str # N atoms

coordinates: list[list[float]] # (N, 3)
'''

### Solution Requirements

1. Analyze step-by-step what the potential problem is in the previous iterations based on the feedback. Think about why
the results from previous molecule structure mismatched with the ground truth. Do not give advice about how to optimize.
Focus on the formulation of the SMILES string. Start this section with "### Analysis". Analyze all iterations
individually, and start the subsection for each iteration with "#### Iteration N", where N stands for the index.
Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about how to separate your algorithm into a
continuous 3D coordinate system part and a discrete SMILES string part. Remember the SMILES string proposed should
always be different from previous iterations. After propose the new SMILES string, compute and count step-by-step how
many atoms it contains. The continuous parameter should follow the number of atoms in the SMILES string. Describe your
plan in pseudo-code, written out in great detail. Start this section with "### Step-by-Step Plan".
3. Output the code in a single code block "'''python ... '''" with detailed comments in the code block. After the SMILES
string, compute the number of atoms in it by counting. Remember that the number of atoms in the SMILES string should be
no less than 8, which means the number of atoms should be >= 8. Try to generate molecule with diverse atoms. Do not add
any trailing comments before or after the code block. Start this section with "### Code".
```

B. More Explanations

B.1. Data Workflow

The full input to LLM has 3 main parts: (i) system prompt, (ii) iteration information, and (iii) format prompt. For the system prompt, we insert it into the LLM at the beginning or input it as a special instruction depending on the type of LLM. For the iteration information, we first concatenate the code and its feedback and then simply stack the top K solutions. Finally, we append the format prompt at the end of the prompt to regularize the expected output. From our experiments, it is important to keep the order of prompts to ensure the performance and the successful parsing. More precisely, we show this process in the following python-like code:

```python
1 prompts = []
2 prompts.append(system_prompt)
3 for solution in reversed(solutions.topk()):
4     iteration_prompt = solution.code + '\n' + solution.feedback
5     prompts.append(iteration_prompt)
6 prompts.append(format_prompt)
7 full_prompt = '\n'.join(prompts)
```

B.2. Differences to Symbolic Regression Task

- Our problem focuses on loss-guided general scientific discovery, which is a super-set of regular regression problems. In the constitutive law search tasks, we do not directly feed the input/output pair to our method. Instead, we consider a much more challenging task: apply the generated constitutive law recursively and use the overall loss as the performance metric. Concretely, a classic SR methods solve $\arg\min_f \|f(X) - y\|$ given $< X, y >$ pairs, whereas our method solves $\arg\min_f \|g(f(X))\|$ given $< X, g(f(X)) >$ pairs and g is a complex function like physical simulation. It is easy to construct g to cover the former case using the later formulation, proving the generality of our problem setup. We formulate our problem as such to reflect a more realistic scenario in scientific discovery, where direct supervision is extremely sparse.

- Our method supports arbitrary number of input variables and output features, where most of SR methods (Valipour et al., 2021) have limitation on the number of input and output. The input limitation strongly caps the complexity

of tasks they can solve, and the output limitation forces them ignore the structural correlation between each output dimension. In a comparison, our method supports arbitrary problem settings thanks to the code-based representation, which enables multi-dimensional arrays and tensor operations.

- Our model adapts to multi-discipline application easily, while traditional SR methods typically incorporate with domain-experts' priors via hard-coded constraints and heuristic (Udrescu et al., 2020), which is limited, domain-specific, and difficult to customize. Our method is built upon LLMs pre-trained on internet-level data that contains multi-discipline natural languages, mathematical expressions, and codes. As a result, it is easy for users to customize it and adapt to their own diverse applications via natural language guidance.

C. More Experiments

C.1. Symbolic Regression

We present the full results of the comparison to symbolic regression methods in Table 5

Table 5. **Symbolic Regression**

Method	R2 ↑	MSE ↓	MAE ↓	Symbolic
AIFeynman (Udrescu et al., 2020)	0.05105	22814675.8	2520.0	✓
DSR (Petersen et al., 2020)	0.57527	10966411.0	2045.0	✓
BSR (Jin et al., 2019)	0.66526	8642965.0	1938.6	✓
AdaBoost (Schapire, 2003)	0.75058	6439962.9	1777.7	✗
GP-GOMEA (Virgolin et al., 2021)	0.77734	5749076.4	1580.1	✓
SBP-GP (Virgolin et al., 2019)	0.81773	4706077.0	1367.5	✓
LightGBM (Ke et al., 2017)	0.83368	4294433.7	1129.9	✗
XGBoost (Chen & Guestrin, 2016)	0.87775	3156500.5	1109.2	✗
MRGP (Arnaldo et al., 2014)	0.91074	2304682.5	950.5	✓
EPLEX (La Cava et al., 2019)	0.91851	2104070.1	122.2	✓
FFX (McConaghy, 2011)	0.93124	1775263.7	801.7	✓
MLP	0.98240	454461.5	366.3	✗
FEAT (Cava et al., 2019)	0.98761	319800.6	336.1	✓
DSO (Mundhenk et al., 2021)	0.99642	92374.9	168.6	✓
Operon (Kommenda et al., 2020)	0.99684	81577.9	92.4	✓
SymbolicGPT (Valipour et al., 2021)	0.52333	6862154.7	1680.7	✓
NeSymReS (Biggio et al., 2021)	N/A to >3 variables			✓
T-JSL (Li et al., 2022)	N/A to >2 variables			✓
Ours	**0.99901**	**17424.6**	**86.4**	✓

C.2. Longer Iteration

In order to further investigate the potential of our method and ablate the hyper-parameters for practitioners, we add a new study in terms of the number of iterations (question-answering cycles). We repeat our experiment in Table 1 with a prolonged number of iterations to 20 and report the performance in Table 6.

Table 6. **Longer Iteration**

#Iterations	(a) ↓	(b) ↓	(c) ↓	(d) ↓	(e) ↓	(f) ↓	(g) ↓	(h) ↓
5	5.2e-5	2.1e-1	6.0e-2	**1.4e-12**	**1.3e-4**	1.1e-1	5.4e-1	3.6e-5
20	**4.2e-6**	**4.0e-4**	**2.5e-3**	**1.4e-12**	**1.3e-4**	**6.5e-2**	**1.2e-1**	**5.6e-6**
Improvement	+1138.1%	+52400.0%	+2300.0%	0.0%	0.0%	+69.2%	+350.0%	+542.9%

As shown in the table, the number of iterations turns out to be a determining hyper-parameter with significant impart on the performance. While it has little affect on relatively easier tasks, it dramatically improves the performance of the most

challenging tasks including **(b)** and **(c)**. For practitioners, the number of iteration should be first considered as the most important hyper-parameter when adapting our method to their own tasks.

D. More Results

D.1. Constitutive Law Discovery (a)

The best solution on task **(a)** optimized by our method:

```python
import torch
import torch.nn as nn

class Physics(nn.Module):
    # Best values from the training curves
    DEFAULT_YOUNGS_MODULUS_LOG = 13.03
    DEFAULT_POISSONS_RATIO_SIGMOID = -1.99

    def __init__(self, youngs_modulus_log: float = DEFAULT_YOUNGS_MODULUS_LOG, poissons_ratio_sigmoid: float =
      DEFAULT_POISSONS_RATIO_SIGMOID):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        Initialize the parameters with the best values from previous feedback.
        """
        super().__init__()
        # Initialize the parameters as trainable parameters
        self.youngs_modulus_log = nn.Parameter(torch.tensor(youngs_modulus_log))  # Log of Young's modulus
        self.poissons_ratio_sigmoid = nn.Parameter(torch.tensor(poissons_ratio_sigmoid))  # Sigmoid of Poisson's ratio

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute Kirchhoff stress tensor from deformation gradient tensor.

        Args:
            F (torch.Tensor): Deformation gradient tensor (B, 3, 3).

        Returns:
            kirchhoff_stress (torch.Tensor): Kirchhoff stress tensor (B, 3, 3).
        """
        # Convert the parameters to their actual values
        youngs_modulus = self.youngs_modulus_log.exp()  # (1,)
        poissons_ratio = torch.sigmoid(self.poissons_ratio_sigmoid) * 0.49  # (1,)

        # Lame parameters
        mu = youngs_modulus / (2 * (1 + poissons_ratio))  # Shear modulus (1,)
        lam = youngs_modulus * poissons_ratio / ((1 + poissons_ratio) * (1 - 2 * poissons_ratio))  # First Lame parameter
      (1,)

        # Deformation gradient determinant J and its reshape for operations (B,)
        J = F.det().view(-1, 1, 1)

        # Inverse transpose of F for stress computation (B, 3, 3)
        F_invT = F.inverse().transpose(1, 2)

        # Compute first Piola-Kirchhoff stress tensor P (B, 3, 3)
        # Volumetric part
        P_vol = lam * (J - 1) * F_invT

        # Deviatoric part combining neo-Hookean behavior
        # This accounts for the near incompressible nature of the material
        P_dev = mu * (F - (1 / J) * F_invT)

        # Compute Kirchhoff stress tensor tau by multiplying the first Piola-Kirchhoff with the transpose of F (B, 3, 3)
        kirchhoff_stress = P_vol + P_dev @ F.transpose(1, 2)

        return kirchhoff_stress
```

D.2. Constitutive Law Discovery (b)

The best solution on task **(b)** optimized by our method:

```python
import torch
import torch.nn as nn

class Physics(nn.Module):

    def __init__(self, gamma: float = -0.07):  # Based on best value from iteration 5
        """
        Initialize gamma as a trainable parameter which will be used for scaling the soft
        deformation correction.

        Args:
            gamma (float): scaling factor for the deformation correction.
        """
        super().__init__()
        self.gamma = nn.Parameter(torch.tensor(gamma))  # Initialize gamma, (1,)
```

```
16
17    def forward(self, F: torch.Tensor) -> torch.Tensor:
18        """
19        Compute corrected deformation gradient tensor F, by applying a soft correction
20        proportional to the deviation of its determinant from 1, effectively guiding the
21        gradient towards physically realistic states.
22
23        Args:
24            F (torch.Tensor): deformation gradient tensor (B, 3, 3).
25
26        Returns:
27            F_corrected (torch.Tensor): corrected deformation gradient tensor (B, 3, 3).
28        """
29        # Compute determinant of F and create a condition based on its value, (B,)
30        J = torch.det(F)  # (B,)
31
32        # Apply a smooth step function as a deviation condition, (B,)
33        J_deviation_condition = torch.tanh(J - 1)  # (B,)
34
35        # Prepare for correction, taking into account the batch dimension (B,)
36        gamma_correction = self.gamma * J_deviation_condition.view(-1, 1, 1)  # (B, 1, 1)
37
38        # Identity matrix, expanded for batch size (B, 3, 3)
39        I = torch.eye(3, device=F.device).repeat(F.size(0), 1, 1)  # (B, 3, 3)
40
41        # Correct F by pulling towards identity matrix when determinant deviates from 1, (B, 3, 3)
42        F_corrected = F - gamma_correction * (F - I)  # (B, 3, 3)
43
44        return F_corrected
```

D.3. Constitutive Law Discovery (c)

The best solution on task **(c)** optimized by our method:

```
1  import torch
2  import torch.nn as nn
3
4  # The default value for elastic_limit is set to the best from the last iteration, and
5  # we initialize a new parameter for capturing the hardening effect
6  DEFAULT_ELASTIC_LIMIT = 0.92
7  DEFAULT_HARDENING_FACTOR = 0.1
8
9  class Physics(nn.Module):
10
11     def __init__(self, elastic_limit: float = DEFAULT_ELASTIC_LIMIT,
12                  hardening_factor: float = DEFAULT_HARDENING_FACTOR):
13         """
14         Define trainable continuous physical parameters for differentiable optimization.
15
16         Args:
17             elastic_limit (float): the parameter determining the initial yield strength.
18             hardening_factor (float): the parameter controlling the rate of hardening.
19         """
20         super().__init__()
21         self.elastic_limit = nn.Parameter(torch.tensor(elastic_limit))  # ()
22         self.hardening_factor = nn.Parameter(torch.tensor(hardening_factor))  # ()
23
24     def forward(self, F: torch.Tensor) -> torch.Tensor:
25         """
26         Compute corrected deformation gradient from deformation gradient tensor.
27
28         Args:
29             F (torch.Tensor): deformation gradient tensor (B, 3, 3).
30
31         Returns:
32             F_corrected (torch.Tensor): corrected deformation gradient tensor (B, 3, 3).
33         """
34         # Obtain the polar decomposed rotation (R) and stretch (S)
35         U, S, V = torch.svd(F)  # U: (B, 3, 3), S: (B, 3), V: (B, 3, 3)
36         R = U @ V.transpose(-2, -1)  # R: (B, 3, 3)
37
38         # Correct the S tensor with hardening
39         # Assuming hardening affects the elastic limit linearly with accumulated plastic strain
40         plastic_strain = torch.relu(S - self.elastic_limit)  # Presumed plastic strain
41         hardening_adjustment = 1.0 + (self.hardening_factor * plastic_strain)
42
43         S_clamped = torch.min(S, self.elastic_limit * hardening_adjustment)  # Clamp S with hardening
44         S_corrected = torch.diag_embed(S_clamped)  # S_corrected: (B, 3, 3)
45
46         F_corrected = R @ S_corrected  # (B, 3, 3) Corrected deformation gradient tensor
47
48         # Ensure volume preservation
49         J = torch.det(F).view(-1, 1, 1)  # (B, 1, 1) Determinant of the input F for volume
50         J_corrected = torch.det(F_corrected).view(-1, 1, 1)  # (B, 1, 1) Determinant of the corrected F
51         volume_ratio = (J / J_corrected) ** (1/3)
52         F_corrected = F_corrected * volume_ratio  # (B, 3, 3) Volume-preserved F_corrected
53
54         return F_corrected
```

D.4. Constitutive Law Discovery (d)

The best solution on task (d) optimized by our method:

```python
import torch
import torch.nn as nn

class Physics(nn.Module):

    DEFAULT_VALUE = 0.0  # Best guess based on previous behavior

    def __init__(self, param: float = DEFAULT_VALUE):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        The parameter modulates corrections towards nearly isochoric behavior.
        """
        super().__init__()
        self.param = nn.Parameter(torch.tensor([param]))  # Scalar modulation parameter

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Symbolic deformation gradient correction model.
        """
        # Compute the determinant of the deformation gradient (volumetric change)
        J = torch.det(F)  # (B,)

        # Compute the volumetric part of the deformation gradient: J^(1/3)*I
        I = torch.eye(3).to(F.device)  # (3, 3)
        # Expand the identity matrix to the entire batch
        I = I.view(1, 3, 3).expand(F.size(0), -1, -1)  # (B, 3, 3)
        # Calculate volumetric part
        vol_deform = torch.pow(J, 1.0 / 3.0).view(-1, 1, 1) * I  # (B, 3, 3)

        # Calculate the deviatoric part of F: divide F by J^(1/3)
        dev_deform = F / torch.pow(J, 1.0 / 3.0).view(-1, 1, 1)  # (B, 3, 3)

        # Modulate correction by self.param and construct the correction term
        correction = self.param * (I - dev_deform)  # (B, 3, 3)

        # Combine the volumetric part with the deviatoric correction
        F_corrected = vol_deform + correction  # (B, 3, 3)

        return F_corrected  # (B, 3, 3)
```

D.5. Molecule Design (e)

The Top-20 solution on task (e) optimized by our method:

1. C1=CC=C(Br)C=C1C2=CN=CC=C2

2. C1=CC=C(I)C=C1C2=CC=NC=C2

3. C1=CN(C=C1)C2=CC(=CC(=C2)F)O

4. C1C2CC3CC(C1)C(C2)(C3)N

5. C1CC2OC1COC2=O

6. C1=CC=C(C=C1)C2=NC(Cl)=NC=C2

7. C1=CC=C(I)C=C1C2=NC(C(F)(F)F)=CN=C2

8. C1N2C3C4OC(C5)C13C24C5

9. C1=CC=C2N=CC=C(Br)C2=C1

10. C=CC1C(=O)NC(=S)N1C

11. C1=CC=CS1C2=NC=CC(Cl)=C2

12. C1OC2C(O)C3C(N)C1C23

13. O=C(NC1=CC=CC=C1)C2CCOCC2Cl

14. C1=CC=C2C(=C1)N=CN2C3=CC=CC=C3Br

15. `C1OC2C3N4C5C6C7C8C1C2C3C4C5C6C7C8`

16. `C1=CC=C(C=C1)C2=CN=CC=C2C=CCl`

17. `C1=CC=C(S1)C2=CC=C(O2)C(F)F`

18. `C1=CC=C2C(=C1)C(=NN2)C3=CC=CC=C3Br`

19. `C1=CC=C2N=C(C(=O)NC2=C1)C3=CC=CS3`

20. `C1=CC=C(C=C1)C2=NN=C(S2)Cl`

D.6. Molecule Design (f)

The Top-20 solution on task **(f)** optimized by our method:

1. `SC(F)(F)CC(Cl)(Cl)N`

2. `C1=CSC(=C1)NNC(=O)CF`

3. `C1C2C3CSC1N2OC3F`

4. `C1CSC(Cl)C(N)C1Cl`

5. `CC(C(=O)O)NC(F)(F)F`

6. `O=C(O)C(F)=C(Cl)C(Br)C=O`

7. `C1CSC(C(=O)O)N=C1F`

8. `C1=CC(=CS1)C2=CC=C(Br)C(F)=C2`

9. `O=S(=O)(N)C1=CC=C(Br)C=C1`

10. `CC(C(=O)O)C(Cl)C(Br)C(N)C(I)`

11. `C1SC(Cl)C(Cl)C(=O)O`

12. `FC1=CC=C(C=C1)CS`

13. `C1CSC2C3CC(Cl)C1C23`

14. `C1=CC(=O)N(C2=CS1)C2=O`

15. `C1=CC=C(N)C(Cl)=C1Cl`

16. `C1SCC2C1C1=C(C=O)C=CC1C2Cl`

17. `C1=CC2=C(N=C(I)C=C2)C=C1`

18. `NC(CS)C(C(=O)O)Cl`

19. `C1=CC=NC2=C1C(=O)SC2=CCBr`

20. `C1=CC=C2C(=C1)C(=CS2)Cl`

D.7. Molecule Design (g)

The Top-20 solution on task **(g)** optimized by our method:

1. `c1cc(sc1Cl)C(C(F)(F)F)N`
2. `C(C(Cl)Cl)C1=CSC(N)=N1`
3. `C1C2CC3NC1C3C2O`
4. `C1OC2C3NOC1C3C2`
5. `C1C(Cl)C2CNC1C2O`
6. `C1(=CC(=C(N1Cl)O)C#N)S`
7. `C1COC2C3CC4(NC2C34)O1`
8. `c1cc(c(c(c1)Cl)O)C(F)(F)F`
9. `O1CCN2CC(F)C2C1`
10. `N1CC2OC(F)C2C1`
11. `C1(O)C2C(NC2C1C)C`
12. `C1=CC2=C(S1)C(=CC(=C2)P)I`
13. `C1CC2NCC(C1)C2O`
14. `C1CC2SCC(C1)N2`
15. `C1=NC2=CS(=O)(=O)N=C12`
16. `C1OC2C3CC(S)C1C23`
17. `C1C2C(NC=O)C(Cl)C1OC2`
18. `O=S(=O)(c1ccccc1)N`
19. `C1C=CC(O1)(F)N2C=CC(Br)=C2Cl`
20. `O=N(=O)C=C(C)C=C(C)N=O`

D.8. Molecule Design (h)

The Top-20 solution on task **(h)** optimized by our method:

1. `CC(NC(=O)C(Cl)C(=O)O)CSC`
2. `C1=CC(=CC=C1)C2=NSN=C2`
3. `FC(F)Oc1ccccc1N`
4. `O=C1NC(=O)SC2=C1C=CC=C2`
5. `C1NOC2C1SC1C2N1Cl`
6. `C1CSCC(N)C1N=C(O)C2=CC=CS2`
7. `C1C2CC(NC1=O)C(Cl)C2I`
8. `C1OC2SC3C4OC(F)S4C13C2`

9. `SC(Cl)(Cl)C1=CC=CC=C1O`

10. `C1=NOC(=C1)C2C(C(=O)NC2=O)Cl`

11. `C1OC2C3NCC4S1C23C4`

12. `C1C2C(N(C1Cl)C2=O)F`

13. `OC1C2C(O1)N=CS(=O)C2`

14. `C1OC2C3N4C1SC23N4`

15. `C1CSC(C2=NC=CS2)N1`

16. `C1CSCC(N1)C2=CC=CS2`

17. `O=P1(OP(=O)(OC1=O)C2=CN=CC=C2)I`

18. `C1CC2(CNC2)C(O1)C3=CN=CN3`

19. `O=C1SCCN(C1)C2=CN=CO2`

20. `C1=CC=NC2=C1C(=O)N=C(S2)Cl`

D.9. Imaginary Constitutive Law

The best solution on the imaginary constitutive law invention task optimized by our method:

```python
import torch
import torch.nn as nn

# Default values for the physical parameters based on previous iterations
DEFAULT_KAPPA = 0.08
DEFAULT_MU = 0.28

class Physics(nn.Module):
    def __init__(self, kappa: float = DEFAULT_KAPPA, mu: float = DEFAULT_MU):
        """
        Initialize the continuous physical parameters kappa and mu for differentiable optimization.
        """
        super().__init__()
        self.kappa = nn.Parameter(torch.tensor(kappa))  # Bulk modulus correction factor (scalar)
        self.mu = nn.Parameter(torch.tensor(mu))        # Shear modulus correction factor (scalar)

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute the corrected deformation gradient from the deformation gradient tensor.
        """
        B = F.size(0)  # Batch size (scalar)
        I = torch.eye(3, device=F.device).unsqueeze(0).expand(B, -1, -1)  # Identity matrix (B, 3, 3)
        J = torch.det(F).view(-1, 1, 1)  # Jacobian determinant (B, 1, 1)

        # Volume correction factor (B, 1, 1)
        vol_correction_factor = torch.clamp(self.kappa * (J - 1), min=0.0, max=1.0)
        vol_correction = vol_correction_factor * I  # Volume correction term (B, 3, 3)

        # Compute trace of F for shape correction (B, 1, 1)
        trace_F = F.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)
        dev_F = F - (trace_F / 3) * I  # Deviatoric part of F (B, 3, 3)

        # Shape correction factor (scalar)
        shape_correction_factor = torch.clamp(self.mu, min=0.0, max=1.0)
        shape_correction = shape_correction_factor * dev_F  # Shape correction term (B, 3, 3)

        F_corrected = F - vol_correction - shape_correction  # Corrected deformation gradient (B, 3, 3)
        return F_corrected
```