

# LLM-SRBench: A New Benchmark for Scientific Equation Discovery with Large Language Models



\*Parshin Shojaee



\*Ngoc-Hieu Nguyen



Kazem Meidani



Amir Barati Farimani



Khoa D Doan



Chandan K Reddy



VIRGINIA  
TECH.



VINUNIVERSITY

Carnegie  
Mellon  
University



ICML  
International Conference  
On Machine Learning

# Outline

**Background**

Motivation

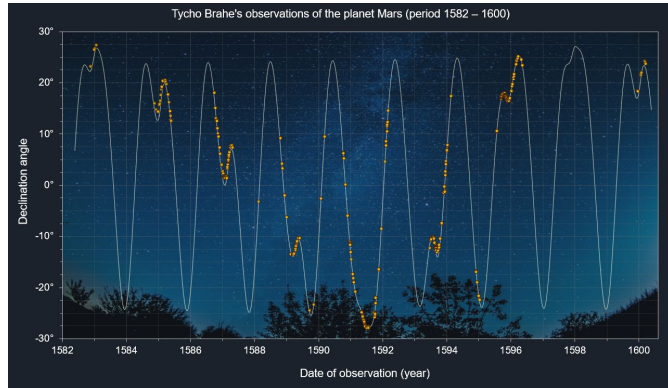
The Proposed Approach

Results

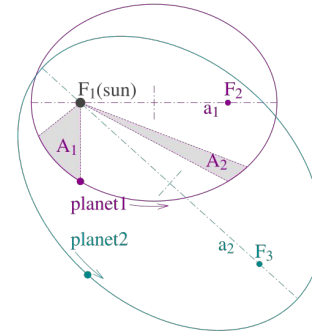
# Scientific Equation Discovery (SED)

Find a concise and interpretable mathematical expression that closely models empirical observations.

Tycho's observatory data



Kepler's laws of planetary motion

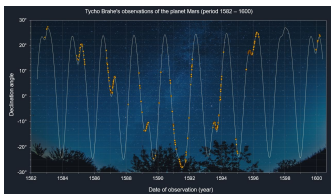


$$\frac{dA}{dt} = \frac{1}{m} \frac{1}{2} (\vec{r} \times \vec{p}).$$

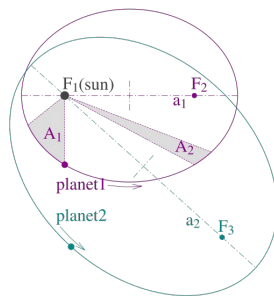
# Scientific Equation Discovery (SED)

Find a concise and **interpretable** mathematical expression that closely models empirical observations.

Tycho's observatory data

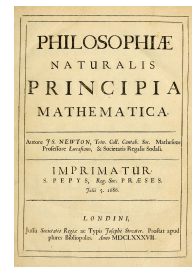


Kepler's laws



$$\frac{dA}{dt} = \frac{1}{m} \frac{1}{2} (\vec{r} \times \vec{p}).$$

Newton's laws



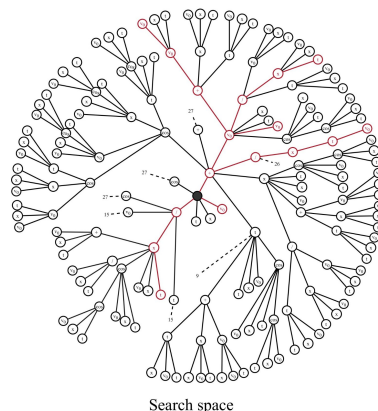
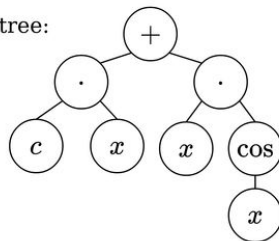
- Mathematical equations vs Predictive model: provide insights, extrapolation, knowledge transfer

# Approaches

- Evolutionary search algorithms
- DL methods using Transformers
- Recent works have shown that using **LLMs for SED is a promising approach**

$$c \cdot x + x \cdot \cos(x)$$

Expression tree:



# Motivation of using LLMs

- LLMs have the ability to
  - process vast amounts of scientific literature
  - extract relevant information
  - generate coherent hypotheses
- LLMs have shown the potential to enhance automatic reasoning and problem-solving capabilities [1, 2]

[1] Large language models for scientific synthesis, inference and explanation. Zheng et al (2023)

[2] Mathematical discoveries from program search with large language models. Romera-Paredes et al (2024)

# Scientific Equation Discovery & LLMs



## Goal / Instruction

- Discover the mathematical equation/law that describes **[output variable]** based on given **[input features]**.
- Use domain-specific knowledge of **[the scientific field]** and provided data samples to find an equation that is scientifically valid and fits the data well.



## Scientific Context

- Problem description
- Variable names and descriptions
- Example:

Find an equation in the field of classical mechanics that describes the mass ( $m$ ) needed to store energy in an oscillating system, given physical input variables: mean stored energy ( $E_n$ ), driving frequency ( $\omega$ ), natural frequency ( $\omega_0$ ), and amplitude ( $x$ ).



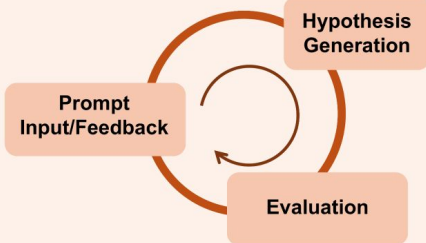
## Data

$E_n$	$\omega$	$\omega_0$	$x$	$m$
4.7	1.2	2.3	1.5	1.2
3.4	2.7	2.7	3.1	0.1
		⋮		
2.8	1.5	3.6	1.4	0.4



## Typical Workflow

- LLM internal scientific knowledge
- Reasoning and planning
- Programming



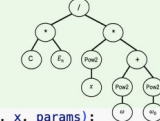
- Parameter Optimization
- Simulation Experiments
- Statistical Fit to Data



## Hypothesis

- Discovered mathematical equation represented by expressions, trees, programs, etc.
- Supporting explanations / reasoning

$$m = 4 * E_n / (x ** 2 * (\omega ** 2 + \omega_0 ** 2))$$



```

def equation(E_n, omega, omega_0, x, params):
    # Energy-mass ratio normalized by parameter
    numerator = params[0] * E_n
    # Combined frequency and amplitude scaling effects
    denominator = omega**2 * x**2 + omega_0**2 * x**2
    m = numerator / denominator
    return m
  
```



## Evaluation

- **Data Fidelity:**
  - In-Domain accuracy
  - Out-of-Domain generalization
- **Symbolic Accuracy:**
  - Human expert / LLM evaluator
  - Scientific plausibility
  - Interpretability
- **Computational Efficiency**

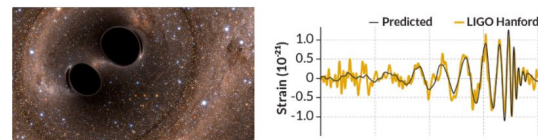
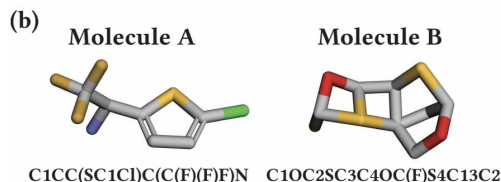
# Scientific Equation Discovery & LLMs

- **LLM-SR: Scientific equation discovery via programming with large language models.** Shojaee et al 2024
- **In-context symbolic regression: Leveraging large language models for function discovery.** Merler et al 2024.
- **LaSR: Symbolic Regression with a Learned Concept Library.** Arya et al (NeurIPS 2024).
- **LLM and Simulation as Bilevel Optimizers: A New Paradigm to Advance Physical Scientific Discovery.** Pingchuan et al (ICML 2024).

(a)

```

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)
        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
    
```



Concepts (by Physicist or LLM)

“Wave strain diminishes as distance increases”  
 “Wave strain has extraordinarily small magnitude”

Guide the search for

$$h = \frac{2G}{c^4} \frac{1}{r} \frac{\partial^2 Q}{\partial t^2}$$

# Outline

Background

**Motivation**

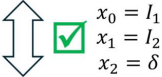
The Proposed Approach

Results

# The need of a new SED benchmark

## 1. Data contamination

A better score might not reflect an improvement but indicating that a model memorised the benchmark data.

Input Prompt	LLM Response
<pre>Can you give me the complete version of last function? Consider physical meaning and relationships of inputs while completing the function.  Find the mathematical function structure that represents the resultant intensity due to the interference of two coherent light sources.  import numpy as np from scipy.optimize import minimize  def loss_function(params, x0, x1, x2, y_true):     y_pred = equation(x0, x1, x2, params)     return np.mean((y_pred - y_true) ** 2)  def evaluate(data: dict) -&gt; float:     """Evaluate equation on input and output observations."""     # Load first data observations     inputs, outputs = data['inputs'], data['outputs']     # Optimize equation skeleton parameters     loss_partial = lambda params: loss_function(params, *inputs.T[:3], outputs)     params_initial_guess = [1, 0, 0, pi]     result = minimize(loss_partial, params_initial_guess, method='BFGS')     optimized_params = result.x     # Return evaluation score     score = loss_function(optimized_params, *inputs.T[:3])     return -score if not np.isnan(score) and not np.isinf(score) else None  def equation_v0(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray) -&gt; np.ndarray:     """Find mathematical function for the resultant intensity due to interference of     two waves given their individual intensities and the phase difference     between them.     Args:         x0: A numpy array representing the intensity of the first wave.         x1: A numpy array representing the intensity of the second wave.         x2: A numpy array representing the phase difference between the two waves.         params: Array of numeric parameters to be optimized     Returns:         A numpy array with the calculated resultant intensity.     """     return params[0]*x0 + params[1]*x1 + params[2]*x2 + params[3]  def equation_v1(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray) -&gt; np.ndarray:     """Improved version of equation_v0"""</pre>	<pre>def equation(x0: np.ndarray, x1: np.ndarray, x2: np.ndarray, params: np.ndarray) -&gt; np.ndarray:     """     Find mathematical function for the resultant intensity due to interference of     two waves given their individual intensities and the phase difference     between them.     Args:         x0: A numpy array representing the intensity of the first wave.         x1: A numpy array representing the intensity of the second wave.         x2: A numpy array representing the phase difference between the two waves.         params: Array of model numeric constants or parameters to be optimized.     Returns:         A numpy array with the calculated resultant intensity.     """     I = params[0]*np.sqrt(x0*x1)+np.cos(x2) + params[1]*x0 + params[2]*x1 +     params[3]     return I</pre>
	$\begin{aligned} x_0 &= I_1 \\ x_1 &= I_2 \\ x_2 &= \delta \end{aligned}$
$I_* = I_1 + I_2 + 2\sqrt{I_1 I_2} \cos(\delta)$	Feynman I.37.4 Problem

# Current SR benchmarks

- AI Feynman dataset (Udrescu & Tegmark, 2020)
  - 120 equations derived from the Feynman Lectures on Physics
- Strogatz dataset (Strogatz, 2018)
  - 14 equations modeling nonlinear and chaotic dynamical processes
- SRBench (La Cava et al., 2021)
  - combine AI Feynman dataset with Strogatz dataset, different noise levels
  - 14 SR methods
- SRSD (Matsubara et al., 2022)
  - address some issues in the AI Feynman dataset

# Outline

Background

Motivation

**LLM-SRBench**

Results

# Creating a new benchmark with transformed equations

$$E_n = \frac{1}{4} m (\omega^2 + \omega_0^2) x^2$$



$$\omega = 4 \frac{\sqrt{\frac{E_n}{m} - \omega_0^2 x^2}}{x}$$

```

1 def equation(m: np.ndarray, omega: np.ndarray, omega_0: np.ndarray, x:
2   np.ndarray, params: np.ndarray) -> np.ndarray:
3     """Mathematical function for Energy
4     This function represents the energy of a vibrating string as a function of
5     mass, frequency, length, and natural frequency.
6     Args:
7       m: A numpy array representing observations of Mass.
8       omega: A numpy array representing observations of Frequency.
9       omega_0: A numpy array representing observations of Natural Frequency.
10      x: A numpy array representing observations of Length.
11      params: Array of numeric constants or parameters to be optimized
12      (currently unused)
13     Returns:
14       A numpy array representing Energy as the result of applying the
15       mathematical function to the inputs.
16     """
17     # calculate terms
18     squared_omega = omega**2
19     squared_omega_0 = omega_0**2
20     squared_x = x**2
21     # calculate energy
22     energy = 0.25 * m * (squared_omega + squared_omega_0) * squared_x
23     # return the energy
24     return energy
25

```

```

1 def equation(E_n: np.ndarray, m: np.ndarray, omega_0: np.ndarray, x:
2   np.ndarray, params: np.ndarray) -> np.ndarray:
3     """Improved version of 'equation_v1'.
4     Args:
5       E_n: A numpy array representing observations of Energy.
6       m: A numpy array representing observations of Mass.
7       omega_0: A numpy array representing observations of Frequency (initial
8       frequency or reference frequency).
9       x: A numpy array representing observations of Length.
10      params: Array of numeric constants or parameters to be optimized.
11     Returns:
12       A numpy array representing Frequency as the result of applying the
13       mathematical function to the inputs.
14     """
15     # calculate the expected frequency based on energy and mass
16     expected_frequency = np.sqrt(E_n / m)
17     # Introduce a scaling factor based on length, with a decay factor in the
18     # exponential term
19     output = expected_frequency * params[0] * np.exp(-params[1] * x / (x ** 2 +
20     params[2])) * (params[3] + params[4] * np.sin(np.sqrt(x)) + params[5] *
21     np.cos(np.sqrt(x)))
22     # Add a reference frequency term with a decay factor
23     output += omega_0 * params[6] * np.exp(-params[7] * x)
24     return output
25

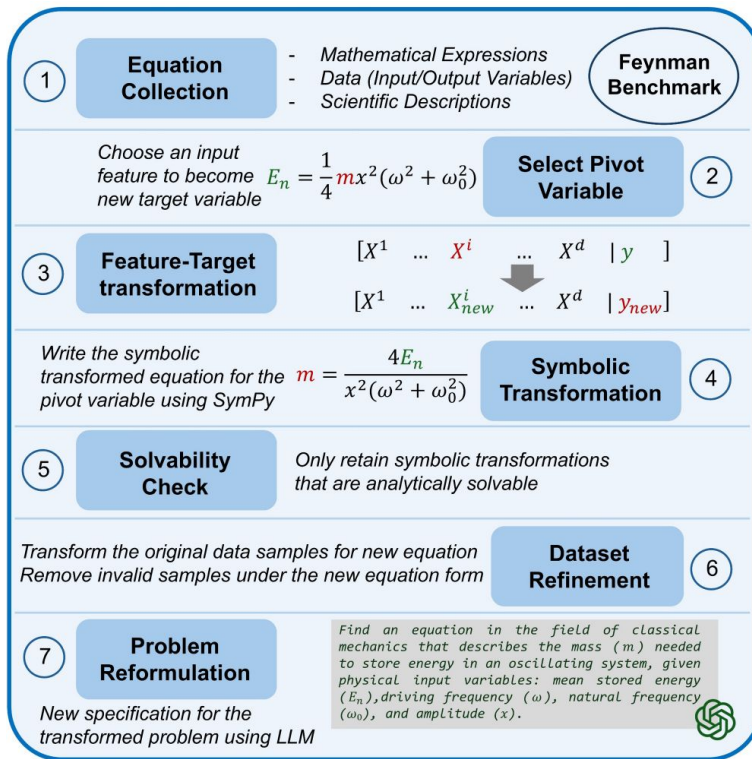
```

$$E = 0.25 \cdot m \cdot (\omega^2 + \omega_0^2) \cdot x^2$$

$$\left[ \sqrt{\frac{E_n}{m}} \cdot p_0 \cdot e^{-\frac{p_1 x}{x^2 + p_2}} \cdot (p_3 + p_4 \sin(\sqrt{x}) + p_5 \cos(\sqrt{x})) \right] + [\omega_0 \cdot p_6 \cdot e^{-p_7 x}]$$

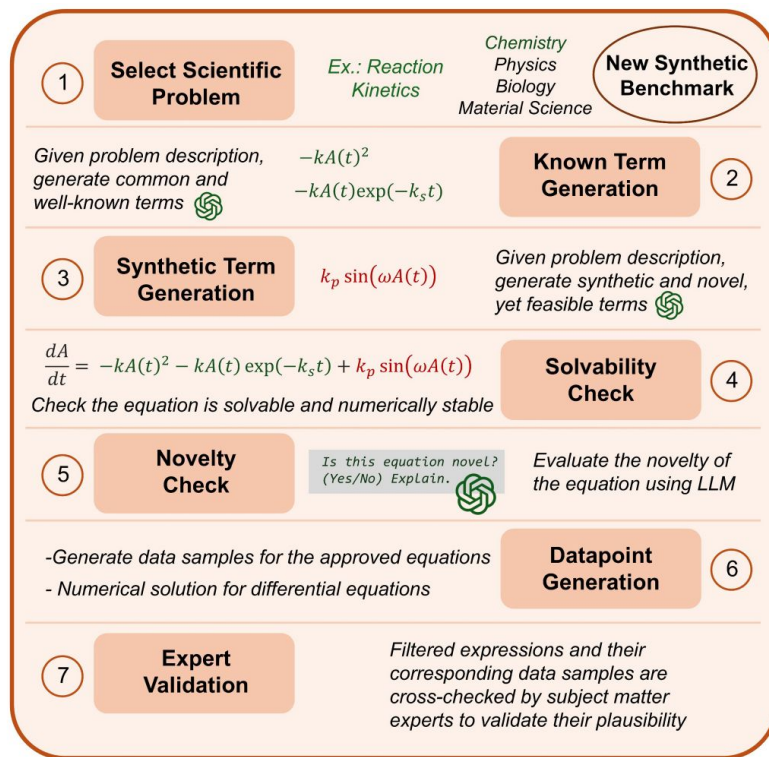
LLM-SR found the correct expression of the original but failed on the transformed equation

# Creating a new benchmark with transformed equations



(a) LSR-Transform

# Creating a new benchmark with synthetic equations



(b) LSR-Synth

# LLM-SRBech | Metrics



## Data Fidelity + Symbolic Accuracy

**Data Fidelity:** How well discovered equations fit data

- **Acc<sub>0.1</sub>** (accuracy to tolerance) + **NMSE** (normalized error)
- **Out-of-Domain testing:** True scientific equations must generalize

**Symbolic Accuracy:** LLM (GPT-4o) as Judge



**Handles any hypothesis representations:** Programs, expressions, strings



**Mathematical equivalence:** Beyond exact matching



**94.6% human agreement + Strong OOD correlation**

# Outline

Background

Motivation

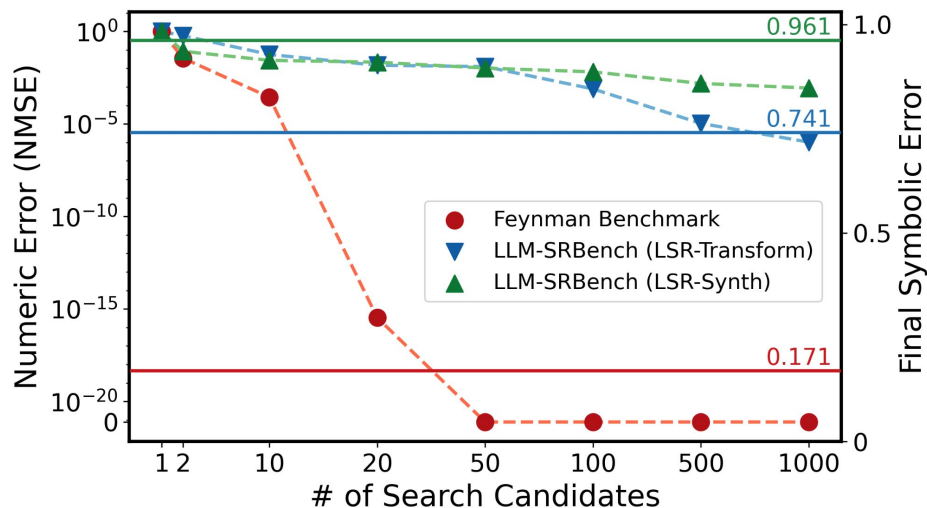
The Proposed Approach

**Results**


# The need of a new SED benchmark

## 1. Data contamination

A better score might not reflect an improvement but indicating that a model memorised the benchmark data.



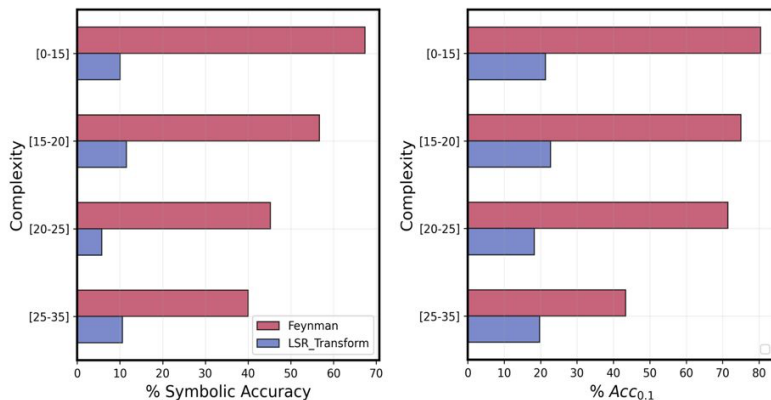
# LLM-SRBench | Results

 Key finding: Even state-of-the-art discovery methods using different LLM backbones only achieve ~31% symbolic accuracy on our benchmark, highlighting the challenging nature of scientific equation discovery!

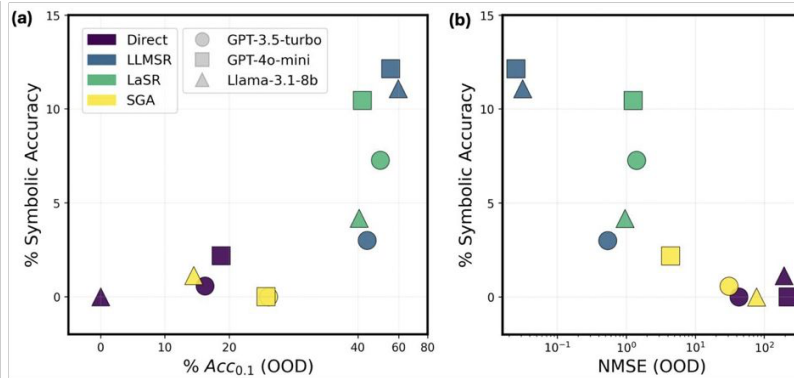
Models	LSR-Transform			LSR-Synth											
				Chemistry			Biology			Physics			Material Science		
	SA (%) $\uparrow$	Acc <sub>0,1</sub> (%) $\uparrow$	NMSE $\downarrow$	SA (%) $\uparrow$	Acc <sub>0,1</sub> (%) $\uparrow$	NMSE $\downarrow$	SA (%) $\uparrow$	Acc <sub>0,1</sub> (%) $\uparrow$	NMSE $\downarrow$	SA (%) $\uparrow$	Acc <sub>0,1</sub> (%) $\uparrow$	NMSE $\downarrow$	SA (%) $\uparrow$	Acc <sub>0,1</sub> (%) $\uparrow$	NMSE $\downarrow$
<i>Direct Prompting (DataBlind)</i>															
Llama-3.1-8B-Instruct	3.61	1.801	0.3697	0.0	0.0	0.0644	0.0	0.0	0.5481	0.0	0.0	0.0459	0.0	0.0	0.0826
GPT-3.5-turbo	2.10	1.801	0.3553	0.0	8.33	<b>0.0023</b>	0.0	4.16	0.5990	0.0	2.27	<b>0.0274</b>	0.0	0.0	<b>0.0277</b>
GPT-4o-mini	<b>7.21</b>	<b>6.306</b>	<b>0.2631</b>	0.0	<b>13.88</b>	0.0221	0.0	<b>4.16</b>	<b>0.4648</b>	4.54	<b>9.09</b>	0.0647	0.0	0.0	0.0484
<i>SGA (Ma et al., 2024)</i>															
Llama-3.1-8B-Instruct	2.70	0.909	0.3519	0.0	8.33	0.0458	0.0	0.0	0.2416	0.0	2.27	0.1549	0.0	12.12	0.0435
GPT-3.5-turbo	0.0	0.909	0.3465	0.0	8.33	0.0071	0.0	8.33	0.1279	2.27	4.54	<b>0.0249</b>	0.0	28.10	0.0019
GPT-4o-mini	<b>9.91</b>	<b>8.11</b>	<b>0.2321</b>	0.0	<b>16.66</b>	<b>5.46e-4</b>	<b>4.16</b>	<b>12.51</b>	<b>0.0128</b>	<b>4.54</b>	<b>9.09</b>	0.0511	0.0	<b>36.11</b>	<b>6.02e-4</b>
<i>LaSR (Grayeli et al., 2024)</i>															
Llama-3.1-8B-Instruct	5.41	45.94	0.0021	0.0	27.77	2.77e-4	4.16	16.66	2.73e-4	4.54	25.02	0.0018	8.21	64.22	7.44e-5
GPT-3.5-turbo	<b>12.61</b>	47.74	0.0015	0.0	38.89	1.51e-4	0.0	16.66	2.31e-4	6.81	22.71	0.0011	20.66	64.09	3.77e-5
GPT-4o-mini	6.31	<b>50.45</b>	<b>0.0011</b>	<b>2.77</b>	<b>38.92</b>	<b>9.11e-5</b>	<b>8.33</b>	<b>20.83</b>	<b>1.53e-4</b>	<b>9.91</b>	<b>31.81</b>	<b>9.94e-4</b>	<b>28.12</b>	<b>72.04</b>	<b>9.23e-6</b>
<i>LLM-SR (Shojaee et al., 2024b)</i>															
Llama-3.1-8B-Instruct	30.63	38.55	0.0101	8.33	<b>66.66</b>	8.01e-6	<b>25.30</b>	<b>58.33</b>	<b>1.04e-6</b>	6.97	34.09	1.23e-4	4.10	88.12	1.15e-7
GPT-3.5-turbo	10.81	10.81	0.1449	0.0	50.22	2.87e-5	0.0	25.03	2.33e-5	0.0	25.12	8.84e-4	12.42	82.14	2.75e-8
GPT-4o-mini	<b>31.53</b>	<b>39.64</b>	<b>0.0091</b>	<b>11.11</b>	52.77	<b>4.12e-6</b>	16.66	29.16	3.06e-6	<b>9.91</b>	<b>36.36</b>	<b>7.62e-5</b>	<b>20.24</b>	<b>88.28</b>	<b>3.21e-9</b>

# LLM-SRBechn | Results

⚠ Same complexity  
↓  
Considerably different  
performance



📈 Strong correlation between  
Symbolic Accuracy & OOD  
generalization



# Conclusion

- Introduced **LLM-SRBench**: A new benchmark addressing data contamination in SED
- Proposed Symbolic Accuracy evaluated by an LLM-as-a-Judge
- Highlighted SED Challenge: SOTA methods achieve ~31% symbolic accuracy on our benchmark

Thank you