Does LLM dream of differential equation discovery?

Elizaveta Ivanchik
AI Institute
ITMO University
Saint-Petersburg, Russia
eaivanchik@itmo.ru

Timur Bavshin
AI Institute
ITMO University
Saint-Petersburg, Russia
trbavshin@itmo.ru

Alexander Hvatov
AI Institute
ITMO University
Saint-Petersburg, Russia
alex_hvatov@itmo.ru

Abstract

Large Language Models (LLMs) have shown promise in symbolic regression tasks, yet their application to partial differential equation (PDE) discovery faces fundamental challenges. Unlike traditional symbolic regression, where models directly generate data enabling fast feedback, PDE discovery requires solving implicit equations and extracting derivative information from physical field data, capacities that current LLMs lack out of the box. We address these challenges through three key contributions: (1) reformulating PDE discovery as a code generation task that leverages LLMs' programming capabilities, (2) developing an optimal data representation format that preserves physical field properties while fitting within context limitations and enabling derivative extraction, and (3) integrating LLMs into a meta-learning framework with the EPDE (Evolutionary Partial Differential Equation) algorithm, where LLMs serve as informed oracles suggesting physically plausible equation forms. Our approach bridges the gap between LLMs' theoretical knowledge of differential equations and the practical requirements of scientific discovery from data. We demonstrate that properly formatted physical field data combined with code generation prompts enables general-purpose LLMs to participate meaningfully in the equation discovery process, despite not being specifically trained for this task. This work establishes a foundation for leveraging pre-trained LLMs in automated scientific discovery while acknowledging current limitations and the need for hybrid human-AI validation approaches.

1 Introduction

The field of symbolic regression for partial differential equations (PDEs), starting from PDE-FIND [1], has experienced remarkable innovation [2], driven by the convergence of Large Language Models (LLMs) [3], advanced evolutionary algorithms [4, 5], and physics-informed neural approaches [6]. The most significant trend is the emergence of hybrid methodologies that combine LLM scientific knowledge with evolutionary robustness, fundamentally changing how we approach automated equation discovery from complex data.

In symbolic regression, we observe a paradigm shift from combinatorial search to knowledge-guided generation. While evolutionary algorithms, such as PySR [7], navigate vast hypothesis spaces, LLM-based approaches, like LLM-SR [8], leverage pre-trained scientific knowledge. The main differences in restoring an expression from a differential equation are that the model generates data by itself, i.e., we have direct (and fast) feedback from the model for the evolutionary algorithm or for LLM. For LLM, we note that significant advances have been made by shifting the problem toward the code-generation domain [9].

In differential equation discovery, we try to find an implicit expression in the form of a differential equation. In contrast, (a) we cannot extract the data field directly from the differential equation (we basically have to "solve" the equation somehow, which is generally a problem by itself) and (b) we cannot easily unpack differential symbols into the data domain. The latter means that if the equation solver is not used, differentiation should be performed within the differential equation discovery algorithm using numerical methods to assess the discovered equation indirectly.

Recent algorithms solve the problem basically in two directions. First, we discuss the numerical difference and how to mitigate the errors associated with it. We use neural networks to filter the data and also differentiate them [10] as weak forms to make the error weight-averaged [11]. For example, classically, the error at the boundaries is larger than that within the bounded domain. The second direction is to develop a differentiation "agnostic" algorithm that can solve equations to form feedback. The solution to the differential equation, in general, makes the process unarguably slower but more robust.

The ambiguity of equation solutions creates a fundamental validation crisis - without automated methods to assess physical plausibility, discovered equations require extensive human expertise to verify, thereby limiting their practical deployment. For example, [12] shows the 31.5 % of quality at the top in the physical data for all models. However, recent advances in neural operators show that differentiation (and sometimes the whole differential equation solution process) could be learned [13]. The recent paper demonstrates that we can obtain a solution without relying on solvers [14]. However, most of the success is obtained when we use many preliminary assumptions and rely on them.

The LLM, as a portmanteau for any problem nowadays, theoretically contains a wealth of knowledge about differential equations. It can effectively cite and apply the knowledge from the textbook [15]. However, differential equation discovery operates on physical field data, creating significant challenges: any LLM or VLM does not often meet the physical data "pictures" (equation solutions) in the training dataset. It is not able to handle the differentiation of such data out of the box. There is initial research on how LLM could be adjusted to the equation discovery problem [16].

In this paper, we **aim** to test these abilities of general LLM. We formulate differential equation discovery as a code generation problem, and we develop an optimal format of data such that LLM can handle the differentiation as well. The data must not be too compressed to retain physics, and on the other hand, must be compressed to fit the context. Ultimately, we utilize LLM as an oracle to infer the initial possible forms of the equation, which are then passed to the algorithm in a meta-learning loop.

Contribution: We formulate PDE discovery as a code-generation task for LLMs, introduce a compact physics-preserving textual representation for field+derivatives, and integrate LLMs as oracles inside an EPDE meta-learning loop.

Limitation: - We consider the EPDE single equation discovery framework EPDE. However, it could be replaced if necessary.
- The models with large context, fine-tuned models, etc., may perform better. It is actually a separate task to find a physics-aware pre-training. We use only the publicity available pre-trains.

Code and data are avaliable in the GitHub repository `https://anonymous.4open.science/r/EPDE_LLM-2028`

2 Differential equation discovery background

In all cases for the equation discovery problem, it is assumed that the data are placed on a discrete grid $X = \{x^{(i)} = (x_1^{(i)}, ...x_{\text{dim}}^{(i)})\}_{i=1}^{i=N}$, where N is the number of observations and dim is the dimensionality of the problem. We mention a particular case of time series, for which dim $= 1$ and $X = \{t_j\}_{i=1}^{i=N}$.

It is also assumed that for each point on the grid, there is an associated set of observations $U = \{u^{(i)} = (u_1^{(i)}, ..., u_L^{(i)})\}_{i=1}^{N}$ to define a grid map $u : X \subset \mathbb{R}^{\text{dim}} \to U \subset \mathbb{R}^L$.

There are two further ways. First is when we formally determine symbols in form:

$$J^r = (x_1, ..., x_{\text{dim}}; u; D_1 u; D_2 u; ...; D_r u) \quad (1)$$

,where $D_r = \bigcup\limits_{|\alpha|=r} \{\frac{\partial^r u}{\partial x_1^{\alpha_1} ... \partial x_r^{\alpha_r}}\}$ is the set of all partial differentials of order r and $\alpha = \{\alpha_1, ...\alpha_{\text{dim}}\}, |\alpha| = \sum\limits_{i=1}^{i=\text{dim}} \alpha_i$ is just a differential multi-index. *Simply speaking*, (1) is a set of symbols that represent differentials up to a given order r. Since we usually have a single observation set u we omit it from the notation $J^r(u)$

From these symbols, we get a formal symbolic expression using a possible set of actions $\mathcal{T}$ (monomials, products, powers) acting on J^r. Then $S \subset \mathcal{T}$ represents selected terms (equation structure), and P is the set of admissible coefficients. Coefficients by themselves could be a function of independent coordinates or just constants. Then the equation has the following form:

$$M(S, P) = \sum_{s \in S} p_s \cdot s(J^r) = 0 \tag{2}$$

The described process has two differences from symbolic regression: we have an implicit dependency in the form of the equation $M(S, P) = 0$, and also, this equation is differential. To assess any quality measure, we must use a solver to extract a solution from (2) and then compare it with the data U on a grid X. Using a solver is a computationally intensive approach, even for non-differential expressions; however, for differential equations, it also requires expert solver tuning.

Second way is to use numerical differentiation D_h of data $\bar{J}^r = \{(x^{(i)}, u^{(i)}, D_h u^{(i)}, ...(D_h)^r u^{(i)})\}_{i=1}^{i=N}$. In this case, we can replace symbols with their numerical counterparts, which are essentially tensors of the same dimensionality as the input data. Therefore, numerical differentiation is used to form a resulting tensor that can be used to assess the equation indirectly, for example, by using the mean error, which in the case of the equation is referred to as discrepancy.

For the SINDy case, we manually determine the longest sentence Σ_{long} possible and fix it. The optimization is performed only by P, which is essentially a vector of the numerical coefficients near each word of Σ_{long}. We need to make P as sparse as possible, which is done with classical LASSO regression. In SINDy, we compute the loss function by using the discrepancy over the discrete grid.

$$P^* = \underset{P \in \Pi}{\text{argmin}} ||M(\Sigma_{\text{long}}, P)||_2 + \alpha ||P||_1 \tag{3}$$

In (3) we denote by $|| \cdot ||_2$ the mean discrepancy in the computation grid X and by $|| \cdot ||_1$ is the l_1 norm. Since SINDy usually works with constant coefficients, we could use the l_1 norm to determine the sparsity of the set of parameters P. In some sense, it is a measure of the complexity of the surface in terms of the number of symbols needed to describe it.

Evolutionary approaches and reinforcement learning have their own rules to construct S for a model. Every equation S_i appearing within the optimization process is evaluated using the SINDy approach (3) with discrepancy or, as is done in EPDE, by constructing the Pareto frontier over the discrepancy and complexity criteria. Both discrepancy computation and Patero frontier formation are performed as part of the fitness function computation or to generate a reward for the reinforcement learning agent.

There are also more robust measures. For a given surface $M(S, P)$, we try to restore the continuous function u that exactly generates the surface and then compare it with observations U. It, of course, requires the solution of the equation. We note that in this case, we do not need to consider jets J^r; instead, we begin working with the fibers u and no longer need to consider the differentials D_r. In that case, all surfaces are single-connected, i.e., the solution of the equation is unique, which is, of course, a limitation, but it is more robust than a discrepancy measure.

There are also some intermediate cases, such as PIC. Here we spatially handle jets, but temporally restore continuous paths. It could be considered as jet factorization and partial fiber projection.

3 Differential equation discovery LLM pipelines

In this paper, we focus on the differential equation discovery part. That means we do not use a solver to handle the equation, thereby avoiding tuning. Additionally, we do not focus on differentiation by itself - all differential fields are obtained equally for both evolutionary algorithms and LLMs. As a result, we pass only the observation data field and differentials to the algorithm to check its ability to form an equation with indirect equation quality assessment.

3.1 Input data field preparation

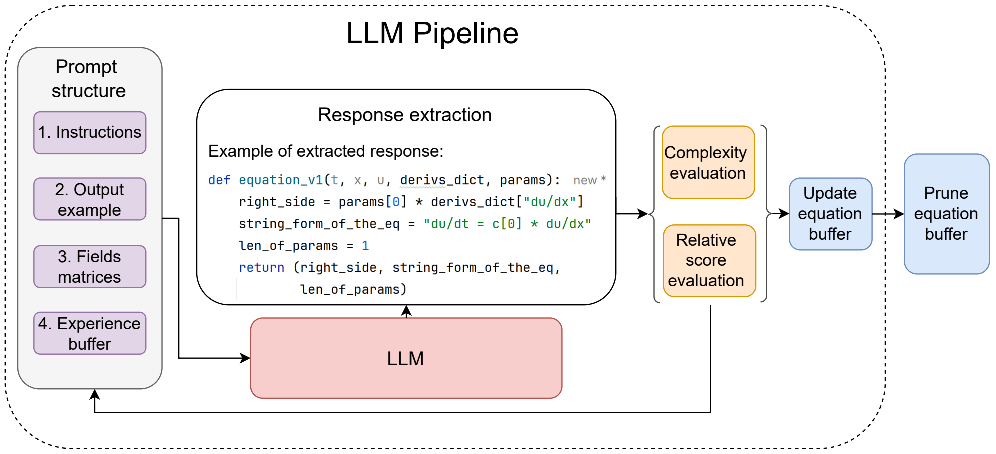
Presenting the raw data in its original form was infeasible due to the context window constraints of the LLM. After an extensive evaluation of alternatives — including visual LLMs, data transformations, and tensor decomposition—the most viable solution was found to be a significant dimensionality reduction. The original high-dimensional data was downsampled via interpolation to a manageable spatial resolution (approximately 20×20 to 30×30 grid points), which preserves the essential structural information while drastically reducing token consumption. We conducted a mini-research on how VLM handles physical data. Additionally, we conducted a mini-research on how text data should be supplied in the App. A.

3.2 LLM-generated equations pipeline overview

The inspiration for the algorithm came from [12], where they suggest leveraging LLMs' programming skills to compile the desired equation in the form of a Python function. Similar to [12], we also utilize the equation buffer so that the LLM is aware of which attempts improve the approximation.

In every other aspect, the proposed algorithm differs from the LLM-SR approach. All in all, it includes these stages (depicted in Fig. 1):

(1) Response generation;
(2) Equation extraction;
(3) Evaluation of the extracted equation;
(4) Recompilation of the prompt;
(5) Equation buffer pruning.



```
def equation_v1(t, x, u, derivs_dict, params):  new *
    right_side = params[0] * derivs_dict["du/dx"]
    string_form_of_the_eq = "du/dt = c[0] * du/dx"
    len_of_params = 1
    return (right_side, string_form_of_the_eq,
            len_of_params)
```

Figure 1: The pipeline of the LLM-based algorithm.

Response generation A pivotal factor in this step of the algorithm is prompt engineering. The prompt is divided into the following sections:

1. Instructions. They include problem statement, requirements, and restrictions.

2. A code snippet that defines an evaluator for the LLM-generated solutions.
3. Input data.
4. Experience buffer. Provides the LLM with a performance history of previously proposed equation structures. This buffer, updated iteratively, is implemented as a dictionary where keys are string representations of equations and values are their corresponding relative performance scores (discussed in detail in **Evaluation of the extracted solution** below).
5. An example of input data.

In reality, we use two prompts, depending on the current iteration of the LLM. The prompt for the first iteration is much simpler than those for the subsequent ones, although it also adheres to the structure described above. The second prompt is enhanced, with greater complexity, added constraints, and a refined problem statement.

Equation extraction This stage of the algorithm is responsible for extracting, refining, and correcting the solutions generated by the LLM. Despite explicit constraints defined in the prompt, LLM outputs can be unstable and often require post-processing to ensure structural validity and adherence to requirements.

This extraction pipeline significantly improves reliability but cannot guarantee a valid solution in every instance. To ensure overall algorithmic robustness, a failure mode is implemented where iterations containing irresolvable outputs are discarded, following the precedent set by LLM-SR [8].

Evaluation of the extracted solution The evaluation mechanism quantifies the quality of an extracted solution through two distinct scores: complexity (Alg. 1) and a relative score (Eq. 4). The relative score may be defined as a normalized Mean Absolute Error (MAE), assessing predictive accuracy. In contrast, the complexity score evaluates the structural intricacy of the equation based on the number and type of terms that comprise it.

In the proposed algorithm, the left-hand side term s_{left} is fixed, following the methodology established in the SINDy approach. This design choice was made to initially probe the capabilities of the LLMs under the assumption that the algorithm has correctly identified the balancing term. Each constructed equation is then assigned a normalized Mean Absolute Error (relative score) R, defined using the mean l_2 norm ($||\cdot||_2$) over all grid points. This score inversely represents quality, with values near 0 indicating high accuracy and a ceiling of 1000 representing the worst-case performance.

$$R = \frac{||M(S, P)||_2}{||s_{\text{left}}||_2} \cdot 1000 \tag{4}$$

The algorithm for complexity evaluation is formalized in Alg. 1. It operates by parsing each equation into its tokens and then assigning a complexity weight based on the token's class and power p. The scoring policy is defined as follows: derivative terms are weighted according to $\frac{(n+1)\cdot\beta_d}{2} \cdot p$, where n is the derivative order and β_d is a base cost for derivatives. Elementary functions (e.g., sin, cos) incur a cost of $\beta \cdot p$ plus the complexity of their inner terms. Finally, basic variables and constants contribute a cost of $\beta \cdot p$, where β is a base cost for simple tokens.

Recompilation of the prompt The prompt provided in App. B is dynamically updated at each iteration to incorporate the latest state of the experience buffer. This buffer serves as a cumulative record of solution performance, implemented as a dictionary where keys are string-based equation descriptors and values are their corresponding relative scores (i.e., normalized mean absolute error, or MAE). The complexity metric is intentionally omitted from this feedback to present the LLM with a single, unambiguous performance objective, as LLMs lack the inherent capability to interpret and optimize within a multi-dimensional fitness space natively.

Equation buffer pruning Following the completion of all iterations, a final refinement stage is applied to the accumulated solution buffer. This stage leverages the previously unused complexity metric to address a key limitation of the relative score: its high sensitivity to noise, which can cause equations with artifacts to outperform correct ones.

```
Data: List of terms - $terms$
Result: Complexity score
$\beta_d = 0.5$;
$\beta = 0.2$;
$complexity = 0$;
for $term$ in $terms$ do
    for $token$ in $term$ do
        $p = extract\_power(token)$;
        if token is derivative then
            $n = extract\_derivative\_order(token)$;
            $complexity = complexity + \frac{(n+1)\cdot\beta_d}{2} \cdot p$;
        else
            if token is function then
                $complexity = complexity + \beta \cdot p + eval\_complexity(inner\_terms)$;
            else
                $complexity = complexity + \beta \cdot p$;
            end
        end
    end
end
```

Algorithm 1: The pseudo-code of complexity evaluation

To mitigate this, we employ a two-step process. To eliminate the terms that capture noise, we enrich the solution space through a combinatorial expansion. With this method, one of the generated variants is bound to exclude the noisy term, making it highly probable that a correct version of the equation will be discovered.

All equations are then evaluated to form a two-dimensional Pareto front based on complexity and relative score. Finally, a knee detection algorithm identifies the optimal trade-off frontier. The solution space is pruned to retain only those equations lying on or below the calculated supporting line, and the length of the perpendicular distance from this line subsequently ranks these solutions.

4 Experiments

The purpose of these experiments is to evaluate the capability of the LLM for equation discovery. Three types of equations — Burgers', wave, and KdV–de Vries — are tested. The datasets used in these experiments are generated numerically. Detailed statements of the initial-boundary value problems and descriptions of the solution methods can be found in App. C. The outcomes are benchmarked against those obtained using the EPDE framework.

All experiments were performed using a `qwen-2.5-72b-instruct` LLM model and the latest EPDE version.

4.1 Experimental setup

For each experiment, we conduct thirty independent runs of both the LLM-EPDE and EPDE frameworks. We evaluate their performance on datasets with and without noise. It is crucial to assess how these frameworks handle noise, as real-world data often contains measurement noise. We use a common approach to add a Gaussian to the data:

$$\tilde{u} = u + \varepsilon \cdot std(u) \cdot N(0,1) \tag{5}$$

where u represents the original clean data, $\tilde{u}$ denotes the noisy data, $N(0,1)$ refers to the standard normal distribution, and ε indicates the magnitude of the noise.

The magnitudes vary in scale according to the input data. Consequently, each type of equation has a threshold magnitude above which the EPDE fails to identify the target equation more than twice in 30 runs. The noise levels are expressed relative to this threshold and are thus set to the limit noise level. The maximum permissible noise magnitudes are established as 5×10^{-3} for Burgers-type equations, 5×10^{-4} for the Korteweg-de Vries equation, and 2.5×10^{-3} for the Wave equation. It is worth noting that in experiments involving noisy data, each run is assigned a unique noise profile.

The performance of the algorithm is evaluated using several quality metrics: the discovery rate of the correct equation, the relative error between the coefficients of the identified equations and those of the theoretical model (ground truth), and the convergence time of the algorithms.

We note that we use strict discovery rate for example $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$ and $u(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x}) = 0$ are considered two different equations even though they are differ only on single trivial solution $u = 0$. The run becomes successful if at least one strictly correct equation is found.

When multiple solutions are obtained (in the Pareto frontier case), the structure of the equation is verified first. Only if the structure matches correctly is the relative error in coefficients calculated.

We measure the relative error of coefficients using the formula:

$$E(\hat{\xi}_i) = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{\xi}_i - \xi_i^*|}{|\xi_i^*|} \tag{6}$$

where N is the total number of terms in the equation, $\hat{\xi}_i$ s the coefficient identified in the discovered equation, and ξ_i^* represents the corresponding coefficient in the true equation.

Furthermore, the hyperparameters used in all experiments are detailed in the supplementary material in App. D. The exact prompts used are listed in App. B.

In the following, we show aggregated tables; more detailed experimental results are in App. E

4.2 Clean data performance comparison

In noise-free environments, the LLM-enhanced framework demonstrates superior equation discovery capabilities, achieving near-perfect identification rates of the underlying governing equations. The method exhibits particular strength in generating plausible equation forms from clean data patterns. In contrast, the conventional EPDE approach maintains a distinct advantage in parameter estimation precision, consistently producing coefficient values with significantly lower error margins. This performance divergence suggests a natural specialization: the LLM framework serves as an effective hypothesis generator for equation structures, while the EPDE methodology provides refined numerical optimization for parameter identification. Both approaches face limitations when confronting more complex equation forms, indicating persistent challenges in handling sophisticated mathematical representations.

Table 1: Comparison of performance of the frameworks with clean data

Dataset	EPDE		LLM	
	Discovery rate	Coefficient error	Discovery rate	Coefficient error
Burgers A	0.86	$\mathbf{1.09 \cdot 10^{-7}}$	**1.00**	$1.81 \cdot 10^{-4}$
Burgers B	0.27	$\mathbf{4.42 \cdot 10^{-4}}$	**0.50**	$2.26 \cdot 10^{-2}$
KdV	0.30	$\mathbf{1.52 \cdot 10^{-2}}$	**0.37**	$1.92 \cdot 10^{-2}$
Wave	0.60	$\mathbf{7.50 \cdot 10^{-4}}$	**0.93**	$6.14 \cdot 10^{-2}$

Since we use a strict discovery rate, EPDE in Table 1 shows a lower discovery rate performance, as we find some equations that differ from the correct ones with trivial or constant solutions. Also, since the process was deterministic, no error bounds were computed.

4.3 Noisy data performance comparison

Under noisy conditions, the comparative analysis reveals a pronounced trade-off in performance between structural identification robustness and parametric estimation accuracy. The LLM-incorporated framework exhibits remarkable resilience against noise contamination, maintaining high equation discovery rates even under substantial noise levels (up to 100 %). Noise level refers to the ratio of the maximal noise stated in the experimental setup section. This robustness suggests that the linguistic processing capabilities inherent in the LLM component provide effective regularization against noise interference during the equation structure detection phase as shown in Tab. 2.

Table 2: Comparison of discovery rates of the frameworks with noisy data

Noise level	Framework	Dataset			
		Burgers A	Burgers B	KdV	Wave
25%	EPDE	0.36	0.30	0.30	**1.00**
	LLM	**0.93**	**0.90**	**0.33**	0.03
50%	EPDE	0.23	0.13	**0.36**	**0.93**
	LLM	**1.00**	**0.83**	0.16	0.13
75%	EPDE	0.20	0.10	0.16	**0.33**
	LLM	**0.97**	**0.83**	**0.23**	0.10
100%	EPDE	0.06	0.03	0.06	**0.06**
	LLM	**0.93**	**0.97**	**0.30**	0.03

Conversely, the EPDE framework demonstrates consistently superior performance in coefficient estimation accuracy across scenarios with lower levels of noise as shown in Tab. 3. The evolutionary optimization approach appears better equipped for precise parameter identification once the equation structure is established, potentially due to the use of more sophisticated numerical optimization techniques. Notably, EPDE maintains exceptional performance on wave equations under all noise conditions, while simultaneously achieving lower coefficient errors across multiple systems.

These complementary strengths suggest promising avenues for integrating the framework. A hybrid methodology leveraging LLM-EPDE's structural discovery capabilities for initial equation identification, followed by EPDE's precision optimization for parameter refinement, could yield superior overall performance in noisy environments. This synergistic approach would combine the noise resilience of linguistic processing with the precision of evolutionary computation, potentially addressing the limitations observed in both individual frameworks.

Table 3: Comparison of coefficient errors (10^{-4}) of the frameworks with noisy data

Noise level	Framework	Dataset			
		Burgers A	Burgers B	KdV	Wave
25%	EPDE	**7.47±1.56**	**31.2±0.26**	**185±0.02**	**18.9±2.89**
	LLM-EPDE	24.5±0.23	135±0.20	245±1.32	2460±10.3
50%	EPDE	**25.7±16.1**	**17.4±1.32**	**180±0.06**	**83.9±10.8**
	LLM-EPDE	23.5±0.49	113±0.55	214±8.17	2690±697
75%	EPDE	77.5±7.04	895±3750	172±2.95	**182±30.3**
	LLM-EPDE	**22.1±0.79**	**77.6±0.81**	**42.0±7.40**	2470±29.6
100%	EPDE	140±10.7	66.1	**160±0.21**	**252±51.2**
	LLM-EPDE	**21.1±1.09**	**57±1.25**	192±21.2	2420

5 Conclusion

The trivial results are that LLM could be used to replace evolutionary optimization. It has its own advantages and drawbacks. With proper instruction, for example, it is able to generate compact forms as is partially done in PDE-READ. However, apart from structural optimization success, there is a numerical coefficient determination failure.

We show that LLM+EPDE form a practical, complementary pair: we pass a small field snapshot to an LLM to generate compact structural hypotheses, then pass the full dataset and a simple initial coefficient guess to EPDE for numerical differentiation, structure refinement, and coefficient fitting. This two-stage workflow narrows the search space and yields cleaner, more reliable discovered PDEs than either component alone. We did not evaluate the LLM for numerical differentiation and do not expect it to replace dedicated numerical modules, which remain necessary for accurate residual evaluation and coefficient estimation.

Acknowledgments and Disclosure of Funding

This work supported by the Ministry of Economic Development of the Russian Federation (IGK 000000C313925P4C0002), agreement No139-15-2025-010.

References

[1] Rudy, S. H., S. L. Brunton, J. L. Proctor, et al. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

[2] Brunton, S. L., J. N. Kutz. Promising directions of machine learning for partial differential equations. *Nature Computational Science*, 4(7):483–494, 2024.

[3] Lorsung, C., A. B. Farimani. Explain like i'm five: Using llms to improve pde surrogate models with text. *arXiv preprint arXiv:2410.01137*, 2024.

[4] Ivanchik, E., A. Hvatov. Knowledge-aware differential equation discovery with automated background knowledge extraction. *Information Sciences*, 712:122131, 2025.

[5] Chen, Y., Y. Luo, Q. Liu, et al. Symbolic genetic algorithm for discovering open-form partial differential equations (sga-pde). *Physical Review Research*, 4(2):023174, 2022.

[6] Sun, J., Y. Liu, Z. Zhang, et al. Towards a foundation model for partial differential equations: Multioperator learning and extrapolation. *Physical Review E*, 111(3):035304, 2025.

[7] Cranmer, M. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023.

[8] Shojaee, P., K. Meidani, S. Gupta, et al. LLM-SR: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

[9] Wang, R., B. Wang, K. Li, et al. Drsr: Llm based scientific equation discovery with dual reasoning from data and experience. *arXiv preprint arXiv:2506.04282*, 2025.

[10] Du, M., Y. Chen, D. Zhang. Discover: Deep identification of symbolically concise open-form partial differential equations via enhanced reinforcement learning. *Physical Review Research*, 6(1):013182, 2024.

[11] Stephany, R., C. Earls. Weak-pde-learn: A weak form based approach to discovering pdes from noisy, limited data. *Journal of Computational Physics*, 506:112950, 2024.

[12] Shojaee, P., N.-H. Nguyen, K. Meidani, et al. LLM-SRbench: A new benchmark for scientific equation discovery with large language models. *arXiv preprint arXiv:2504.10415*, 2025.

[13] Hao, Z., C. Su, S. Liu, et al. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training. *arXiv preprint arXiv:2403.03542*, 2024.

[14] Herde, M., B. Raonic, T. Rohner, et al. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

[15] Grayeli, A., A. Sehgal, O. Costilla Reyes, et al. Symbolic regression with a learned concept library. *Advances in Neural Information Processing Systems*, 37:44678–44709, 2024.

[16] Du, M., Y. Chen, Z. Wang, et al. Llm4ed: Large language models for automatic equation discovery. *arXiv preprint arXiv:2405.07761*, 2024.

A Initial tests on LLM's understanding of the data

The fundamental question of our research was whether Large Language Models (LLMs) could discern functional dependencies within numerical data fields, presented, in our case, as two-dimensional data, and to identify which class of LLMs is best suited for this task. Given the spatial nature of the data, where u is a matrix defined over discrete x and t, our initial hypothesis inclined towards visual LLMs (VLLMs), which are designed to process image data.

A series of preliminary experiments, however, demonstrated that these visual LLMs struggled significantly with the core requirement of the task. They exhibited a notable inability to accurately interpret the content of even basic visual representations of the data (see the subsection below). The models failed to reliably identify data values from the heatmaps, let alone discover the underlying mathematical relationships between variables.

In contrast, experiments with textual representations of the data revealed that even small-scale textual LLMs could often propose equation structures that approximated the underlying function. This critical result - that textual models showed a surprising aptitude for the provided task - justified our pivot to textual LLMs and encouraged the development of the current pipeline.

A detailed analysis of these experiments is provided in the following subsections.

A.1 Space perception tests on Visual LLMs

The tests were performed mainly on the heatmaps derived from functions $cos(C \cdot x)$, chosen for their clear periodic structure, with the exception of the last test which was based on a hypothesis that the problem lies in the nature of the images and not in characteristics of VLLM. The models evaluated were: `gemini-pro-vision`, `qwen-2-vl-72b-instruct`, `llama-3.2-90b-vision-instruct`.

The experimental design, illustrated in Fig. 2, systematically examined different potential failure modes:

- Test (a) and (b) assessed basic pattern recognition ability by varying the frequency of oscillation ($\cos(2.5x)$ and $\cos(10x)$).
- Test (c) hypothesized that the monochromatic color scheme of standard heatmaps might be a limiting factor and tested the same high-frequency function ($\cos(10x)$) with a color mapping.
- Test (d) served as a core control. This test was used as a primal indicator of models' ability to understand periodic structures while accounting for their training data distribution, which consists largely of human-recognizable scenes.

The image resolution was mostly set to 128×128 pixels. An exception was the control image in case (d), which was rendered at a higher resolution of 512×512 to ensure clarity. Furthermore, to systematically rule out resolution-based limitations, case (c) was tested across multiple scales: 128×128, 256×256, 512×512, and 1024×1024. This range of resolutions was selected to test the models' limits, with the baseline set to a low resolution of 128×128 to reflect the typical scale of our numerical datasets, which does not exceed 512×512 pixels.

The experimental results revealed significant limitations in the visual LLMs' capabilities. In case (a), they misclassified a cosine gradient as linear and could not correctly count two minima. In the higher-frequency case (b), all models underestimated the count of extrema (reporting 5 or less vs. a true count of 6-7 for each type of extrema). Altering resolution and adding a color mapping in case (c) produced no substantial improvement, with a faint positive effect only at the maximum tested size of 1024×1024 pixels, where the 7th maximum was sometimes noted. Lastly, in (d) case the models reported the existence of 20 to 30 elements on the image with the only exception of qwen-2-vl-72b-instruct, which correctly identified 25.

These experiments led us to conclude that visual LLMs are ill-suited for this specific task. Consequently, we pivoted to textual LLMs. While raw numerical data is also non-ideal for these models, we hypothesized that a transformation of the data into a suitable textual format could leverage their strengths in symbolic reasoning and pattern recognition.

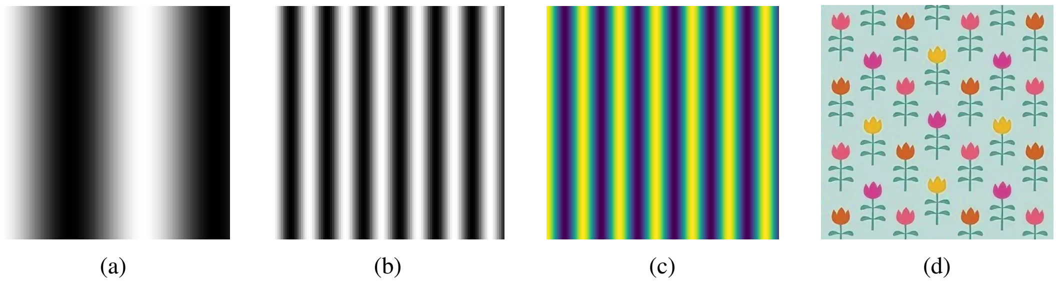


Figure 2: The input data for space perception tests on Visual LLMs: (a) A heatmap of $cos(2.5x)$, (b) A heatmap of $cos(10x)$, (c) A heatmap of $cos(10x)$ in colours, (d) An image containing an unambiguous periodic pattern of floral elements. The recognizability of these elements to a human observer establishes a baseline for expected model performance.

A.2 Space perception tests on textual LLMs

To evaluate the inherent pattern recognition capabilities of textual LLMs, we conducted initial experiments on one-dimensional data generated from the function $u(x) = \sin(2.5x)$ (see the prompt template in Appendix B.3). The models tested were `qwen2.5-72b-instruct` and `mixtral-8x7b-instruct`. Both models correctly identified the sinusoidal nature of the function, demonstrating their ability to recognize periodic patterns from numerical data. Although they made errors in estimating the precise oscillation parameters, their successful relation identification provides initial validation of our core hypothesis: that textual LLMs can serve as effective tools for extracting functional relationships from structured numerical data.

We subsequently extended our investigation to two-dimensional functions. The test case was designed to be partially periodic: $u(t, x) = 2\sin(2.5x) + 0.07t^2$. The models tested were `qwen2.5-72b-instruct`, `mixtral-8x7b-instruct`, and the larger `mixtral-8x22b-instruct`. The results were promising yet incomplete. The `qwen2.5-72b-instruct` model, for instance, correctly identified the sinusoidal component along the x-dimension in 9 out of 10 trials. While it never explicitly identified the quadratic term t^2, it consistently recognized the non-periodic, increasing trend along the t-dimension. This demonstrates a capacity for discerning composite spatial structures, albeit with limited parametric precision.

The other models yielded similar results, though `mixtral-8x7b-instruct` performed noticeably worse - as a rule, the model insisted on a polynomial structure, occasionally suggesting a sinusoidal function along x dimention; while the `mixtral-8x22b-instruct` performed on par with `qwen2.5-72b-instruct`, producing responses of equivalent quality and insight.

An essential aspect of our testing involved determining the optimal data representation for textual LLMs. We evaluated two distinct formats:

(1) Structured Tabular Data: A three-column format with headers "x, t, u", where each subsequent row represented a single data point (the prompt is given in Appendix B.4).

(2) Raw array data: The direct string representation of the two-dimensional NumPy array for u, provided in a row-major format (the prompt is showcased in Appendix B.5).

This comparison was crucial for assessing whether the models benefited from explicit feature structuring or could infer relationships from raw numerical arrays. The results demonstrated a significant advantage for the structured tabular format. When presented with the "x, t, u" table, the top-performing models (`qwen2.5-72b-instruct` and `mixtral-8x22b-instruct`) successfully identified the sinusoidal relation along the x-dimension in approximately 90% of cases (9/10 trials). In contrast, the same models achieved only a 60% success rate (6/10 trials) when the data was presented as a raw numerical array. This clear performance gap underscores the importance of feature-label structuring for enabling textual LLMs to perform spatial reasoning tasks.

B Prompts

B.1 Prompt for the zeroth iteration of the LLM pipeline

~~~
What is a possible function with the general equation form {full_form}
    that could be described with the set of points named points_set, that
    have the form of '{dots_order}'. Give an answer in the function
    equation_v1 constructing it in a way described by equation_v1 in the
    end.
Note that although the general equation form is {full_form}, the resulting
    equation may take on simpler forms, for ex., {left_deriv} = F(t,
    du/dx) or {left_deriv} = F(du/dx). Suggest some simple structure, that
    roughly describe the relationships in data, for example {left_deriv} =
    c[0] * du/dx.

Requirements:
1. Only output your reasoning and the code starting from "def
    equation_v1..." DO NOT recite the other functions (like loss_function
    evaluate etc.)

import numpy as np
from scipy.optimize import minimize

def loss_function(params, t, x, u, derivs_dict):
    u_pred = equation_v1(t, x, u, derivs_dict, params)[0]
    return np.mean((u_pred-derivs_dict["{left_deriv}"])**2)

def evaluate(data: dict) -> float:
    """ Evaluate the constructed equation"""
    inputs, derivs_dict = data['inputs'], data["derivs_dict"]

    # Optimize equation skeleton parameters
    loss_partial = lambda params: loss_function(params, *inputs,
        derivs_dict)
    params_initial_guess = np.array([1.0]*P)
    result = minimize(loss_partial, params_initial_guess, method='BFGS')
    optimized_params = result.x

    # Return evaluation score
    score = loss_function(optimized_params, *inputs, derivs_dict)
    return score if not np.isnan(score) and not np.isinf(score) else None


#/Input data

points_set =
{points_set}

#/end of input data

# An example of desired output:
```python
def equation_v1(t: np.ndarray, x: np.ndarray, u: np.ndarray, derivs_dict:
    dict(), params: np.ndarray):
    right_side = params[0] * derivs_dict["du/dx"]
    string_form_of_the_equation = "{left_deriv} = c[0] * du/dx"
    len_of_params = 1
    return right_side, string_form_of_the_equation, len_of_params
~~~

```
```

B.2 Prompt for the subsequent iterations of the LLM pipeline

```
What is a possible function with the general equation form {full_form}
    that could be described with the set of points named points_set, that
    have the form of '{dots_order}'? Give an answer in the function
    equation_v1 constructing it in a way described by the example in the
    end.
Your goal is to explore the equations space (in relation to their scores)
    and to examine any inexplicit interactions between the input variables
    (for ex. du/dx * u^2).
The dictionary exp_buffer stores previous attempts to find the equation
    evaluated with evaluate function. Refer to it in order to understand
    what is yet to be explored and what might be worth more exploration.
    The best score is 0.
Also, keep in mind, if it seems like t or x are involved in the equation
    do not forget that u and its derivatives are dependent on them, and
    thus the involvement of t and x might be expressed through u or its
    derivatives. Your goal is to find any possible inexplicit interactions.
Start by exploring simpler structures and then gradually move on to more
    complicated ones IF you see the need to do so.

Note that although the general equation form is {full_form}, the resulting
    equation may take on simpler forms (BUT IT DOESN'T HAVE TO!), like
    {left_deriv} = F(t, du/dx).
Make sure the suggested equation is dependent on at least one derivative,
    (e.g, in case of du/dt = F(t, x, u, du/dx), du/dx must be included).

Requirements:
1. First look at the exp_buffer and then suggest the equation, the string
    form of which is not already there!
2. Do not copy the equations from the exp_buffer!
3. Only give a simplified version of the equation in
    string_form_of_the_equation: always open the brackets, for ex. instead
    of 'du/dt = c[0] * (1 + du/dx) * t' return 'du/dt = c[0] * t + c[1] *
    du/dx * t'.
4. Higher order derivatives must be referenced as d^nu/dx^n or d^nu/dt^n,
    where n is an integer (for example, d^2u/dx^2 and NOT du^2/dx^2).
    Anything like du^n/dx^n refer to the multiplication of du/dx and
    should be written as (du/dx)^n or (du/dx)**n (same apply to du/dt).
5. Do not put {left_deriv} into the right side of the equation as a
    standalone term, you can though use it as part of a term: ..+
    {left_deriv} * u +.. for example

import numpy as np
from scipy.optimize import minimize

def loss_function(params, t, x, u, derivs_dict):
    u_pred = equation_v1(t, x, u, derivs_dict, params)[0]
    return np.mean((u_pred-derivs_dict["{left_deriv}"])**2)

def eval_metric(params, t, x, u, derivs_dict, left_side):
    u_pred = equation_v1(t, x, u, derivs_dict, params)[0]
    return np.mean(np.fabs(u_pred - derivs_dict[left_side]))

def evaluate(data: dict) -> float:
    """ Evaluate the constructed equation"""
    inputs, derivs_dict = data['inputs'], data["derivs_dict"]
```

~~~
    # Optimize equation skeleton parameters
    loss_partial = lambda params: loss_function(params, *inputs,
        derivs_dict)
    params_initial_guess = np.array([1.0]*P)
    result = minimize(loss_partial, params_initial_guess, method='BFGS')
    optimized_params = result.x
    # Return evaluation score
    score = eval_metric(optimized_params, *inputs, derivs_dict, left_side)
    return score if not np.isnan(score) and not np.isinf(score) else None


#/Input data

points_set =
{points_set}
exp_buffer = {{
}}

#/end of input data

# An example of desired output:
```python
def equation_v1(t: np.ndarray, x: np.ndarray, u: np.ndarray, derivs_dict:
    dict(), params: np.ndarray):
    right_side = params[0] * derivs_dict["du/dx"]
    string_form_of_the_equation = "{left_deriv} = c[0] * du/dx"
    len_of_params = 1
    return right_side, string_form_of_the_equation, len_of_params

```
~~~

B.3 1D case of textual LLMs' testing

```
What is a possible function (e.g. u(x) = x**2 + 5) that could be described
with this set of points, that have the form of "x u(x)":

0.00 0.00
0.21 0.50
0.42 0.87
0.63 1.00
0.84 0.86
1.05 0.49
1.26 -0.02
1.47 -0.52
1.68 -0.88
1.89 -1.00
2.11 -0.85
2.32 -0.47
2.53 0.03
2.74 0.53
2.95 0.88
3.16 1.00
3.37 0.84
3.58 0.46
3.79 -0.05
4.00 -0.54
```

B.4 2D case of textual LLMs' testing with structured tabular data

```
What is a possible function (e.g. u(x, t) = x**2 + 5t) that could be
    described with this set of points, that have the form of "t x u(t, x)":

0.00 0.00 0.00
0.00 0.21 1.00
0.00 0.42 1.74
0.00 0.63 2.00
0.00 0.84 1.72
0.00 1.05 0.98
0.00 1.26 -0.03
0.00 1.47 -1.03
0.00 1.68 -1.75
0.00 1.89 -2.00
0.00 2.11 -1.70
0.00 2.32 -0.95
0.00 2.53 0.07
0.00 2.74 1.06
0.00 2.95 1.77
0.00 3.16 2.00
0.00 3.37 1.69
0.00 3.58 0.92
0.00 3.79 -0.10
...
```

B.5 2D case of textual LLMs' testing with raw array data

```
What is a possible function (e.g. u(t, x) = x**2 + 5t) that could be
    described with this array, that represents the function u(t, x)":

[[ 0. 1. 1.74 2. 1.72 0.98 -0.03 -1.03 -1.75 -2. -1.7 -0.95
   0.07 1.06 1.77 2. 1.69 0.92 -0.1 -1.09]
 [ 0.02 1.02 1.76 2.02 1.74 1. -0.01 -1.01 -1.73 -1.98 -1.68 -0.93
   0.08 1.08 1.79 2.02 1.71 0.94 -0.08 -1.07]
...
```

C Equation problem statements

C.1 Burgers A

The initial-boundary value problem for Burger's equation is represented with Eq. 7.

$$\begin{gathered}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \\
u(0,t) = \begin{cases} 1000, t \geq 2 \\ 0, t < 2 \end{cases} \\
u(x,0) = \begin{cases} 1000, x \leq -2000 \\ -x/2, -2000 < x < 0 \\ 0, \text{otherwise} \end{cases} \\
(x,t) \in [-4000, 4000] \times [0,4]
\end{gathered} \tag{7}$$

The analytical solution to the problem presented in Eq. 7 is given in [1]. Data for the experiment were obtained with the discretization of the solution in the domain $(x,t) \in [-4000, 4000] \times [0,4]$ using 101×101 points.

C.2 Burgers B

The problem and data were provided by the authors of PySINDY[1]. The problem can be formulated in Eq. 8, where the boundary conditions were not reported. The solution was provided for the domain $(x,t) \in [-8,8] \times [0,10]$ using 256×101 discretization points.

$$\begin{gathered}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - 0.1\frac{\partial^2 u}{\partial x^2} = 0 \\
(x,t) \in [-8,8] \times [0,10]
\end{gathered} \tag{8}$$

C.3 Korteweg-de Vries

As in the case of Burgers' equation, the data and the problem (Eq. 9) were provided by the authors of PySINDY for the domain $(x,t) \in [-30,30] \times [0,20]$ using 512×201 discretization points.

$$\begin{gathered}
\frac{\partial u}{\partial t} + 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 \\
(x,t) \in [-30,30] \times [0,20]
\end{gathered} \tag{9}$$

C.4 Wave

The initial-boundary value problem for the wave equation is given in Eq. 10.

$$\begin{gathered}
\frac{\partial^2 u}{\partial t^2} - \frac{1}{25}\frac{\partial^2 u}{\partial x^2} = 0 \\
u(0,t) = u(1,t) = 0 \\
u(x,0) = 10^4 \sin^2 \frac{1}{10} x(x-1) \\
u'(x,0) = 10^3 \sin^2 \frac{1}{10} x(x-1) \\
(x,t) \in [0,1] \times [0,1]
\end{gathered} \tag{10}$$

[1]https://github.com/dynamicslab/pysindy

D Hyperparameters

Table 4: LLM hyperparameters

Hyperparameter	Dataset			
	Burgers A	Burgers B	KdV	Wave
Iterations	6	30	30	6
Derivative order	[2, 3]	[2, 3]	[2, 3]	[2, 3]
Best candidates	4	4	4	4

Due to the ongoing development of the EPDE framework, the results obtained with its newer versions may vary from those presented in this study. For these experiments, we use the hyperparameters presented in Table 5.

Table 5: EPDE hyperparameters

Hyperparameter	Dataset			
	Burgers A	Burgers B	KdV	Wave
Epochs	5	5	5	5
Population size	8	8	8	8
Boundary	20	20	20	20
Derivative order	[2, 3]	[2, 3]	[2, 3]	[2, 3]
Term number	5	5	5	5
Function power	3	3	3	3
Sparsity interval	(1e-5, 1)	(1e-5, 1e-2)	(1e-5, 1e-2)	(1e-5, 1)

E Detailed experiment boxplots

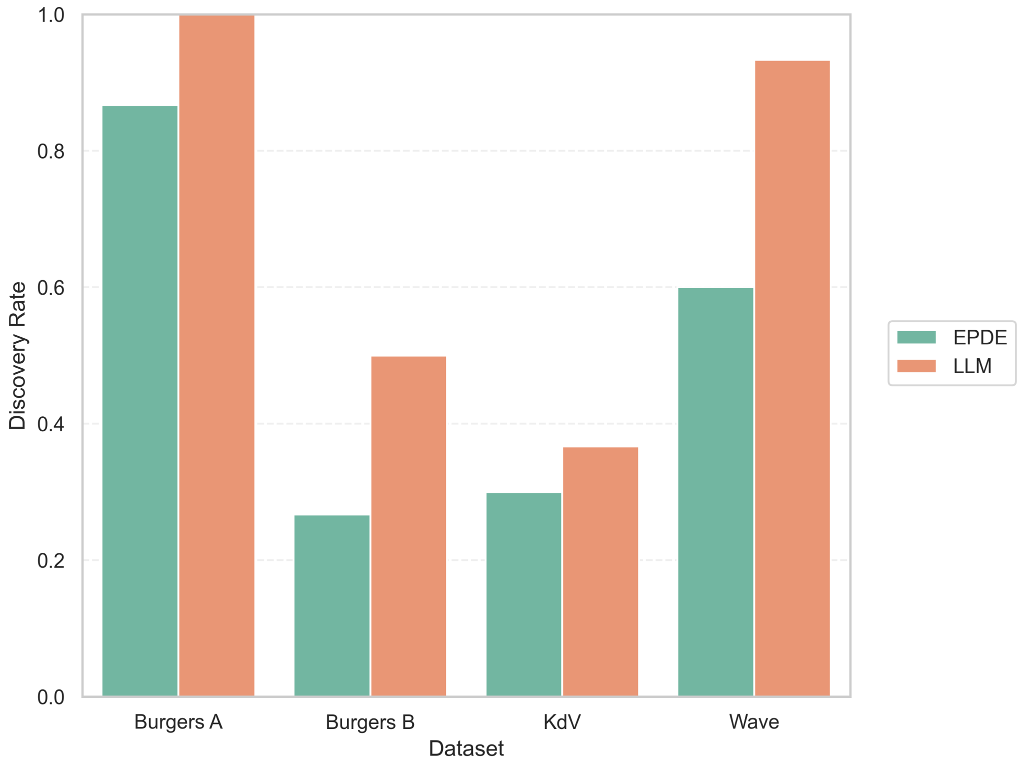


Figure 3: Comparison of discovery rates of the frameworks with clean data

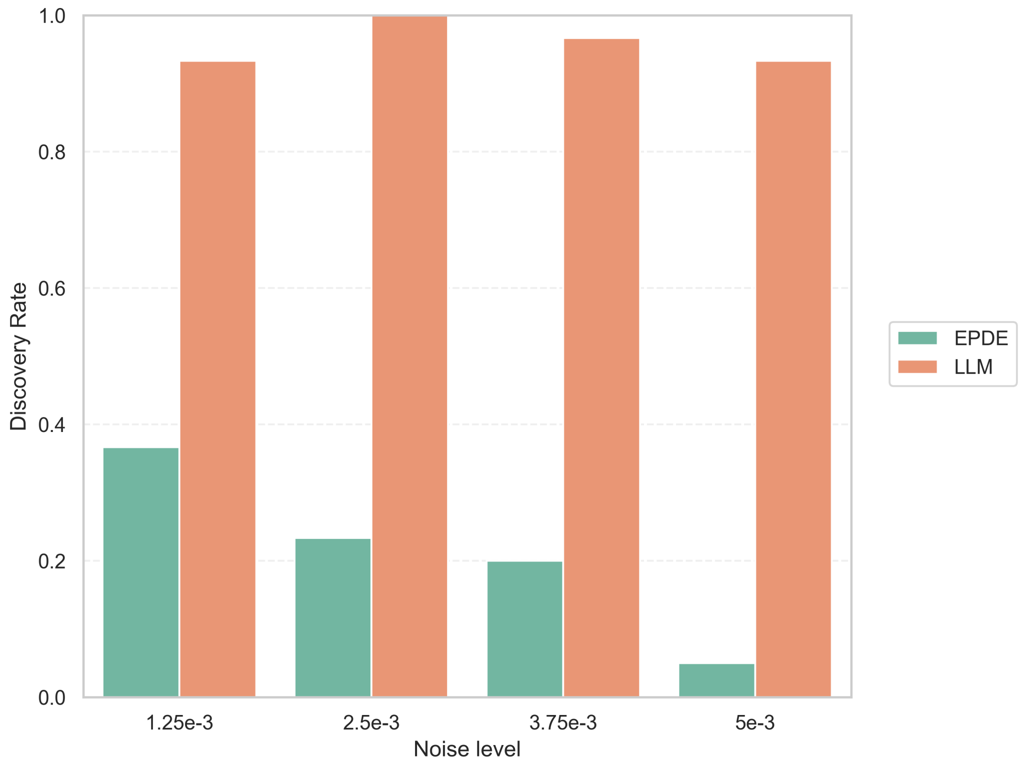


Figure 4: Comparison of discovery rates of the frameworks with noisy data – Burgers A

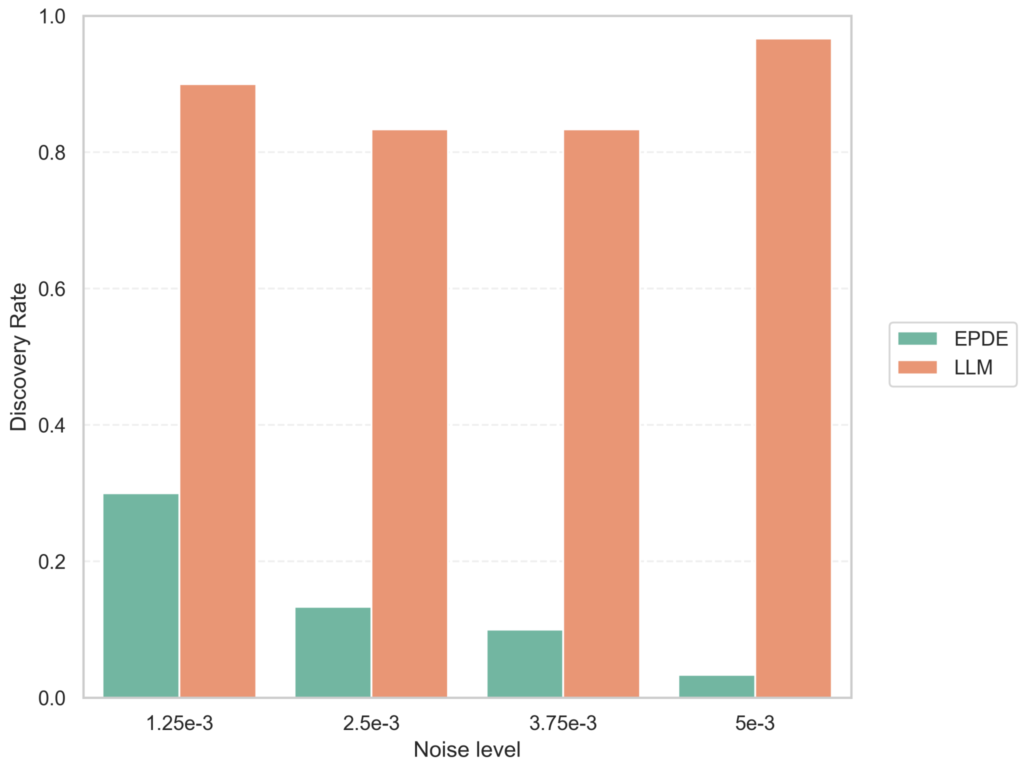


Figure 5: Comparison of discovery rates of the frameworks with noisy data – Burgers B

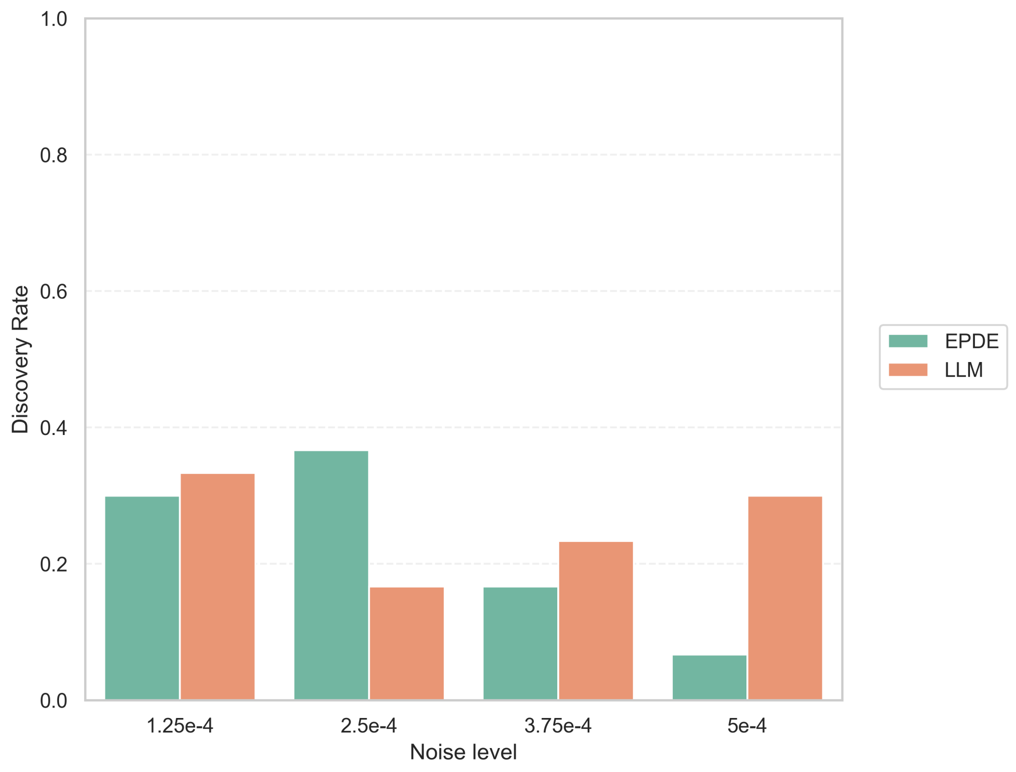


Figure 6: Comparison of discovery rates of the frameworks with noisy data – KdV equation

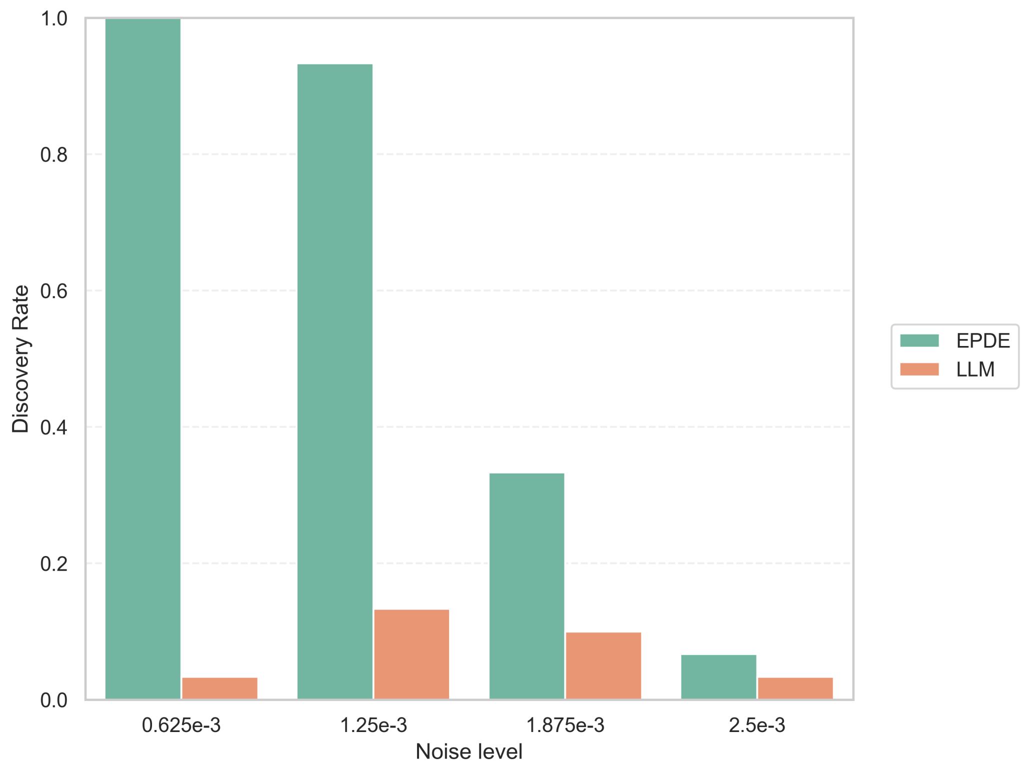


Figure 7: Comparison of discovery rates of the frameworks with noisy data – Wave equation

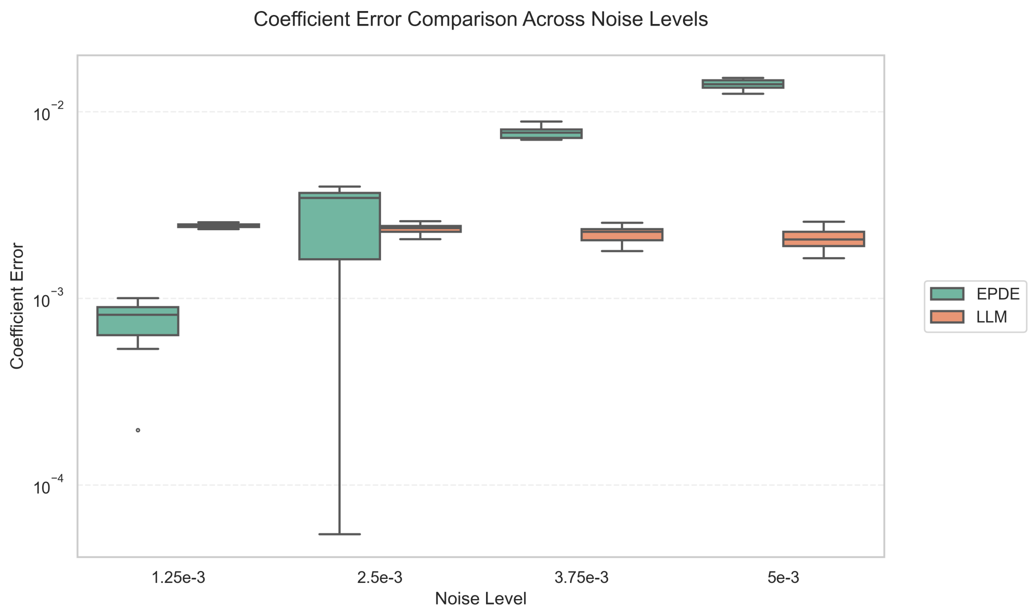


Figure 8: Comparison of coefficient errors of the frameworks with noisy data – Burgers A

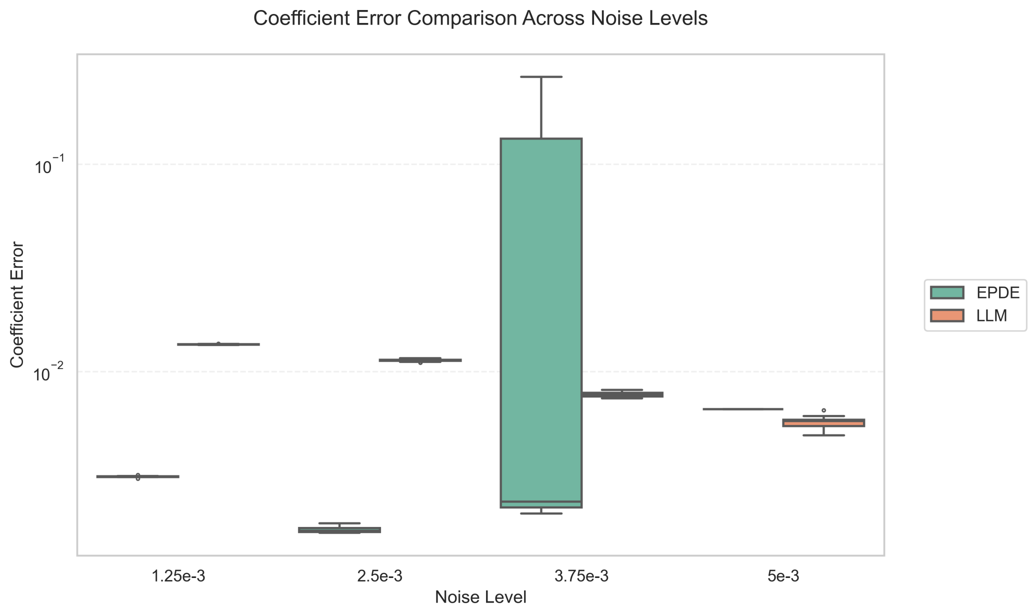


Figure 9: Comparison of coefficient errors of the frameworks with noisy data – Burgers B

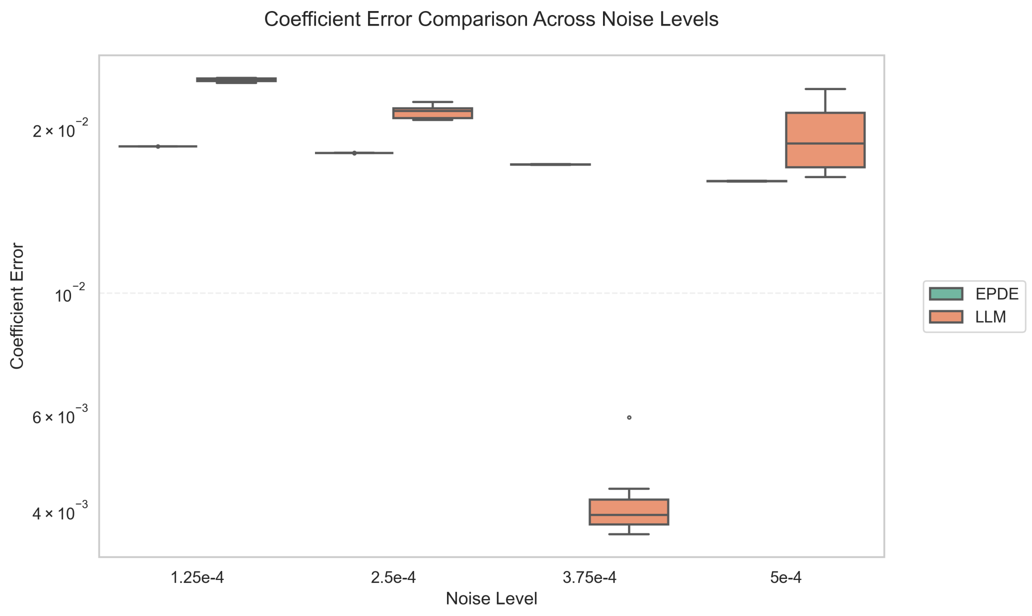


Figure 10: Comparison of coefficient errors of the frameworks with noisy data – KdV equation

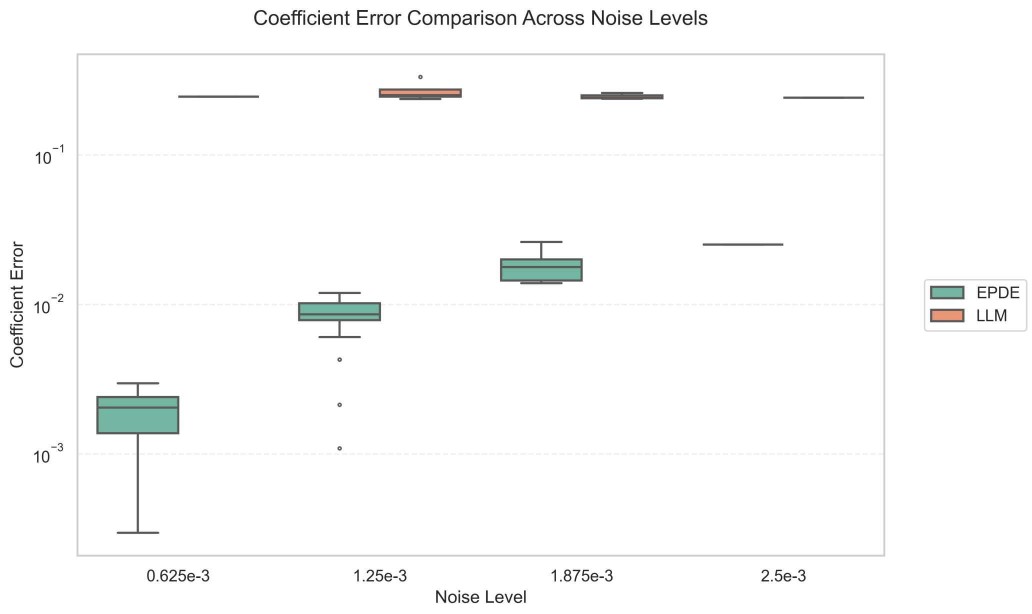


Figure 11: Comparison of coefficient errors of the frameworks with noisy data – Wave equation