A Neural Symbolic Model for Space Physics

Jie Ying[1,†], Haowei Lin[2,†], Chao Yue[3,†], Yajie Chen[4], Chao Xiao[5], Quanqi Shi[5], Yitao Liang[2], Shing-Tung Yau[6,7,*], Yuan Zhou[6,7,8,*], and Jianzhu Ma[9,10,*]

[1]Qiuzhen College, Tsinghua University, Beijing, China.
[2]Institute for Artificial Intelligence, Peking University, Beijing, China.
[3]School of Earth and Space Sciences, Peking University, Beijing, China
[4]Max-Planck Institute for Solar System Research, Göttingen, Germany.
[5]Institute of Space Sciences, Shandong University, Weihai, China
[6]Yau Mathematical Sciences Center, Tsinghua University, Beijing, China.
[7]Beijing Institute of Mathematical Sciences and Applications, Beijing, China.
[8]Department of Mathematical Sciences, Tsinghua University, Beijing, China.
[9]Department of Electronic Engineering, Tsinghua University, Beijing, China.
[10]Institute for AI Industry Research, Tsinghua University, Beijing, China.
[†]Equal contribution.
[*]Correspondence should be addressed to: styau@tsinghua.edu.cn, yuan-zhou@tsinghua.edu.cn, majianzhu@tsinghua.edu.cn.

ABSTRACT

Symbolic regression, a key problem in discovering physics formulas from observational data, faces persistent challenges in scalability and interpretability. We introduce PhyE2E, an AI framework designed to discover physically meaningful symbolic expressions. PhyE2E decomposes the symbolic regression problem into sub-problems via second-order neural network derivatives, and employs a transformer architecture to translate data into symbolic formulas in an end-to-end manner. The generated expressions are further refined via Monte-Carlo Tree Search and Genetic Programming. We leverage a large language model to synthesize extensive expressions resembling real physics, and train the model to recover these formulas directly from data. Comprehensive evaluations demonstrate that PhyE2E outperforms existing state-of-the-art approaches, delivering superior symbolic accuracy, fitting precision, and unit consistency. We deployed PhyE2E to five critical applications in space physics. The AI-derived formulas exhibit excellent agreement with empirical data from satellites and astronomical telescopes. We improved NASA's 1993 formula for solar activity and provided an explicit symbolic explanation of the long-term solar cycle. We also found that the decay of near-Earth plasma pressure is proportional to r^2 to Earth, with subsequent mathematical derivations validated by independent satellite observations. Furthermore, we found symbolic formulas relating solar EUV emission lines to temperature, electron density and magnetic field variations. The formulas obtained are consistent with properties previously hypothesized by physicists.

1 Introduction

The primary distinction between physics and other data sciences is the pursuit of the discovery of fundamental laws behind the world using concise symbolic formulas. The discovery of physics formulas is a lengthy process of trial and error. For instance, after about 40 attempts to match Mars data with various elliptical shapes, Kepler discovered that Mars' orbit was elliptical and proposed the three laws of Kepler[1]. Similarly, through meticulous experimentation of electric and magnetic phenomena, Faraday unveiled the laws of electromagnetic induction. He demonstrated that the intricate relationship between electricity and magnetism could be articulated through elegant, fundamental principles derived from empirical data[2].

Discovering laws in physics or other specialized fields often demands years of domain expertise. Numerous methods have been specifically developed and proposed to aid in the discovery of physics formulas and to enhance the understanding of the underlying principles[3, 4, 5]. Since AI has achieved tremendous success in multiple domains, a naturally arising question is whether we can leverage AI to automatically extract physical laws from experimental observation data to better understand our physical world. This task is known as symbolic regression (SR) within the AI field and has received widespread attention in recent years. Genetic algorithms (GAs) were first adopted to address SR problems by evolving a population of candidate symbolic expressions (typically represented as trees) to minimize a fitness function, which measures how well the expression fits the data[6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. Monte Carlo Tree Search (MCTS) predicts the expression by exploring a search tree of possible expressions, simulating paths down the tree by randomly selecting mathematical operations to extend expressions. For each path, it generates complete expressions and assesses fitness based on how well they approximate the target function[16, 17, 18, 19, 20]. Compared to GAs, MCTS offers more dynamic and fine-grained control over reward signals. Rewards can be customized to reflect the quality of partial solutions during tree exploration, facilitating a more efficient search for the optimal solution. Recent advances in deep neural networks have paved the way for the development of end-to-end methodologies. Unlike Genetic Algorithms (GAs) and Monte Carlo Tree Search (MCTS), which are symbolic search techniques that frequently struggle with the expansive search space, the end-to-end approach streamlines the process by eliminating the need for iterative searching and refinement. It predicts mathematical expressions as sequences of symbols (operators, variables, constants) from data, allowing symbolic expressions to be generated in a single forward pass through the neural network, which significantly improves speed, especially for large datasets and complex functions[21, 22, 23, 24, 25, 26]. End-to-end approaches require large datasets for training, yet the number of physical formulas discovered by humanity remains relatively limited. There are still many unknown areas in the physical world waiting for humanity to discover and formulate new laws and equations. Therefore, current data-driven end-to-end algorithms are limited in reasoning about simple mathematical equations, with only a few preliminary works successfully applied to real physical data[27, 28]. Additionally, symbols in physical law have physical unit systems, and the entire expression should ensure the correctness of these units, which is not yet considered by current computational methods.

To address these limitations, we propose a new framework, named PhyE2E, to achieve accurate symbolic regression for space physics. The PhyE2E framework includes the following key components. First, we fine-tuned a large language model (LLM) with existing physics formulas, enabling it to generate a diverse array of formulas that align with the statistical distribution of physics formulas. By harnessing the common knowledge acquired by LLMs from the internet, the generative model efficiently learns the underlying distribution from a small set of seed formulas, thereby overcoming the challenge of data sparsity in the training process. Second, we trained an end-to-end formula regression model based on the transformer model, which directly converts data matrices into symbolic formulas encoded in Polish representation[26, 24, 25]. With the aid of the LLM, our end-to-end model is trained with a large volume of formulas that “look physical” and adhere to consistent unit systems. Third, to reduce the search complexity, we developed a formula-splitting technique that is able to group variables without nonlinear (or logarithmically nonlinear) relationships, producing a series of simpler sub-formulas for a nested and more simplified symbolic regression task. This technique uses an oracle neural network to fit the data and then analyzes its second-order derivatives to uncover relationships among the input variables. Finally, we leveraged the state-of-the-art GAs and MCTS methods to further refine the predicted formulas. To evaluate our performance, we conducted comparisons on both an LLM-synthesized dataset and another real-world physics dataset AI-Feynman against GA-based, Transformer-based, and NN-based methods. Experimental results indicate that our method outperforms all others in terms of data fitting accuracy, the correctness of the mathematical form of the formulas, and unit accuracy. We further applied our model to various important applications in space physics, including predicting sunspot numbers, plasma sheet pressure, solar rotational angular velocity, emission line contribution functions from the Sun, and lunar tidal signals in the plasma layer. Compared to formulas proposed by physicists, the formulas derived from PhyE2E exhibit better generalizability, a more concise mathematical form, more precise physical units, and more importantly, provide physically meaningful insights and explanations for instrumental observations.

2 Results

Overview of PhyE2E. PhyE2E comprises an end-to-end transformer-based physical model designed to take observed data points as input and predict both the operators and the physical units involved in a formula directly (Fig. 1). A set of 264,000 synthetic formulas were generated by an LLM (OpenLLaMA2-3B) that had been fine-tuned using real physics formulas from the Feynman Symbolic Regression Database (FSReD)[29] (Fig. 1 top). We randomly selected 180,000 formulas from the synthetic dataset for training, reserving the remaining data for testing. The inference process of PhyE2E involves three stages: a variable interaction detection method to decompose the target formula into sub-formulas (Fig. 1 bottom left), end-to-end prediction of each sub-formula using the trained model (Fig. 1 middle), and GA and MCTS approaches to refine the predicted formulas (Fig. 1 bottom right). More specifically, we first fit the data using a standard multi-layer neural network, and used its Hessian matrix to determine the nonlinear interactions between all pairs of variables (Sec. 4.2 in Methods). The core of our model leverages a transformer architecture, synthesizing formulas as a prefix sequence with attached physical units (Sec. 4.3 in Methods). This approach incorporates both observed data points and physical prior knowledge about the target formula. In the final stage, we utilized GA and MCTS approaches to minimize the root-mean-square error (RMSE) of the predicted formula (Sec. 4.4 in Methods). This was achieved by using a grammar pool of context-free rules that incorporates basic operators like exp, as well as the most promising sub-formulas, which were automatically constructed within the search tree by PhyE2E.

Performance on the synthetic and AI Feynman datasets. We divided the synthetic dataset containing 264,000 formulas into training, validation, and test sets with a ratio of 80%, 10%, and 10%. We ensure that all validation and test formulas are unseen during training by removing formulas that are either identical, or become completely equivalent to the formulas in the training set after simple mathematical transformations (Sec. 4.5 in Methods). We compared PhyE2E with 15 state-of-the-art symbolic regression baseline models, including 4 GA-based models (PySR[30], GP-GOMEA[10], Operon[31], and GPLearn[11]), 3 Transformer-based models (TPSR[26], EndToEnd[24], NeSymReS[25]), 2 LLM-based models (LaSR[32], LLM-SR[33]), and 6 NN-based models (uDSR[34], PhySO[35], AIFeynman[29, 36], ParFam[37], KAN[38, 39], BSR[40]). The technical details of running these models are provided in Supplementary Notes 3. The detailed comparisons with PySR under different configurations are provided in Supplementary Notes ???. We also included two variants of PhyE2E in our comparison, including versions without using the formula decomposition module (D&C) and the MCTS refinement module (MCTS). We evaluated the performance of all the models on 6 metrics[41] including symbolic accuracy, average accuracy($R^2 > 0.99$), unit accuracy, complexity, relative complexity to the ground truth formulas and elapsed times (Sec. 4.6 in Methods).

First, we evaluated whether the physics formulas synthesized by the LLM were consistent with the real physics formulas from the Feynman Symbolic Regression Database (FSReD). It can be observed that the formulas generated by the LLM closely match the distribution of real physics formulas in terms of the number of variables, formula complexity, depth, and operator types measured by the Jensen-Shannon divergence (D_{JS}) (Sec. 4.6 in Methods, Fig. 2a). Then, we studied the symbolic accuracy of the formulas generated by the model, that is, whether the mathematical forms of the formulas correspond to the true formulas used to generate the data. PhyE2E(D&C+MCTS) exceeds the second-best model PySR by 26.48%, outperforms the best NN-based model, uDSR, by 31.75%, the best LLM-based model, LaSR, by 37.89%, and the best Transformer-based model, TPSR, by 39.83% (Fig. 2b). The generated formulas by PhyE2E also demonstrate a more powerful ability to fit data compared to other methods. PhyE2E(D&C+MCTS) outperforms all the state-of-the-art approaches by at least 20.00% in terms of Avg.Acc.($R^2 > 0.99$). Regarding the accuracy of (physical) units, we observe that PhyE2E achieves 99.27% of accuracy, leading all the other approaches. The performance drops to 93.30% by including the D&C and MCTS modules. This decline is due to the absence of unit constraints during the D&C and MCTS refinement stage. The most compatible baseline of unit accuracy is PhySO, which reaches a comparable 89.70% with a strictly unit constraint during its search process[35].

The data-fitting capabilities of the formulas can be enhanced by increasing their complexity. According to Occam's Razor, complex formulas tend to have weaker generalizations compared to simpler ones and lose their interpretability in a physical context. Therefore, in addition to studying the formulas' ability to fit data, we also focus on the complexity of the formulas generated by different models. Our model produces formulas with lower model complexity compared to the formulas generated by other models. For 42.17% of the test formulas, their depth is less than 3, suggesting that they predominantly represent linear relationships. For these low complexity formulas, our model successfully recovers the mathematical form of 98.02% of them (Fig. 2d). The relative complexity to the ground truth formula of PhyE2E is 2, which is 33.27% better than the second-best model PySR. By introducing the D&C and MCTS modules, PhyE2E increases the complexity of formulas by 6.79%, while still maintaining a similar complexity with PySR. Only a handful of other baseline models demonstrate the capability to predict formulas of appropriate complexity (e.g., PySR, GP-GOMEA, AIFeynman, PhySO), primarily because of their constraints on complexity or the inclusion of physical units. Others either generate formulas with high complexity (complexity>50) or formulas deviate far away from the target formula (relative complexity > 10), making it difficult to interpret in practice (Fig. 2b).

Next, we evaluated the performance of different models on the formulas collected from the Feynman Symbolic Regression Database (FSReD), referred to as the Feynman Dataset for brevity. The Feynman dataset contains 100 real-world physics formulas sampled from the seminal Feynman Lectures on Physics[42, 43, 44] covering core physics topics like classical mechanics, electromagnetism, and quantum mechanics. Although our model was not directly trained on formulas from the Feynman dataset, our training dataset was generated by LLM based on formulas from the Feynman dataset. To inhibit data leakage, we removed the formulas in the training dataset that were either identical, or became completely equivalent after simple mathematical transformations compared to those in the Feynman dataset. A comprehensive list of the test formulas is provided in Supplementary Table S1.

First, we observe that all computational methods, including ours, show similar performance on the Feynman dataset and on our synthetic data, which demonstrates that the distribution of formulas generated by the LLM is essentially consistent with those from the Feynman dataset. In terms of symbolic accuracy, PhyE2E(D&C+MCTS) exceeds the best classical model, PySR, by 10.09%, and the best NN-based model, uDSR, by 18.49%, the best Transformer-based model, TPSR, by 21.77%, the best LLM-based model, LaSR, by 29.35%. Although PySR outperforms standard PhyE2E in numerical precision ($R^2 > 0.99$) with an accuracy of 84.96%, it is still surpassed by PhyE2E with the D&C and MCTS modules. The complexity of the models generated by PhyE2E remains low. The relative complexity to the ground truth formula of PhyE2E is 2.98, 3.67% higher than the best model PySR. The relative complexity for PhyE2E is lower than 5.75 and the complexity is lower than 16.85, which is essentially the best among all the baseline models (Fig. 2c). To further investigate the relationship between performance and formula complexity, we calculated the symbolic accuracy as a function of the number of variables, complexity, and the number of unary and binary operations. We found that our method had a significant advantage over other methods for formulas with high complexity. When the complexity is larger than 20, our model outperformed the second-best methods 67.11% and 32.73% on the two datasets, respectively (Fig. 2d). We divided both datasets into three subsets of varying difficulty based on the similarity of mathematical formulas compared to those in training datasets (0.95-1.0 as easy, 0.80-0.95 as medium, 0.00-0.80 as hard), and then systematically calculated the symbolic accuracy of each method. We found that our method had a significant advantage on the medium and hard datasets (Sec. 4.5 in Methods, Fig. 2d).

Among all the modules, the divide-and-conquer (D&C) module plays a crucial role in simplifying the search space in our framework. Consider a formula $f = m\sqrt{B_1^2 + B_2^2 + B_3^2}$ from the Feynman dataset, our D&C module first determined that the three variables B_1, B_2 and B_3 under the square root do not interact, which indicates that the operators between them can only be addition or subtraction (Supplementary Fig. S1a). Therefore, the original symbolic regression task decomposes into three parts $g_i = m\sqrt{B_i^2 + C_i}$ ($i = 1, 2, 3$), each of which is processed individually by the end-to-end module. The predicted formulas for g_1, g_2 and g_3 are later aggregated into one complete formula (Supplementary Fig. S1a). The D&C module reduces the complexity of a formula by splitting a set of variables that have no nonlinear interactions locally within a certain formula. This also explains why we find that our method has a considerable advantage over other methods as the complexity of the formulas increases.

However, the risk associated with this methodology is that if the decomposition is incorrect, some of the segments will contain incorrect variables. Therefore, we carefully studied the performance of different D&C strategies. We implemented 4 different strategies, including Single Pattern (detects the interaction of additive and multiplicative), Multi-Pattern (identifies all interactions such as additive terms under sine functions), Fixed Threshold and Adaptive Threshold (Sec. 4.2 in Methods). We evaluated the performance of different strategies by assessing how many formulas could be correctly decomposed, partially correctly decomposed, and completely incorrectly decomposed on the test set of the synthetic dataset. We found that the Adaptive Threshold strategy provided a flexible approach for interaction detection, resulting in a substantial improvement in complete accuracy by 50.61%. The Multi-Pattern strategy facilitates diverse types of interactions and effectively reduces the proportion of absolutely wrong formulas from 6.50% to 5.03% (Supplementary Fig. S1b). Overall, adopting the Multi-Pattern and Adaptive Threshold strategies results in the highest number of accurate formula decompositions, which was also the strategy we used in our framework.

To discover physics formulas rather than purely mathematics formulas, another important technique is the consistent units of physics quantities[35, 36]. We further retrain two additional models using the same architecture but one without units decoding strategy and another without any physical priors, thereby excluding the unit decoding strategy as well (Sec. 4.3 in Methods). These three models are evaluated using three accuracy metrics on the synthetic dataset (Supplementary Fig. S1d). We found that the unit prior plays an important role especially when dealing with a small amount of input data. In an extreme case with only five input data points, the symbolic accuracy of PhyE2E improved to 39.83%, compared to 26.06% without units decoding strategy and 8.97% without any physical priors. Another observation is that the units accuracy is notably enhanced by the incorporation of physical priors. The unit accuracy improved by 25.90% compared to the PhyE2E without units decoding strategy, and by 56.70% compared to the PhyE2E without any physical priors in the five-data-point case. This improvement is also observed in the case with 50 data points, where the unit accuracy increased by 4.47% and 12.23%, respectively.

Performance of sunspot number prediction. Next, we applied our trained model to multiple applications in space physics. Our goal is to find formulas that are more accurate in prediction and simpler in mathematical form than existing physics formulas. We directly applied our trained model to these real-world applications instead of performing any fine-tuning operations. We started by studying the pattern of changes in the sunspot number (SSN) over time. The Sun serves as the primary source of energy for the entire Earth system. Predicting sunspot numbers is essential for forecasting space weather events that can impact both satellite operations and terrestrial communication systems. These forecasts are also pivotal in climate studies, aiding in modeling the impact of solar variability on the Earth's climate. The accurate prediction of sunspot activity is also crucial for managing the effects of geomagnetic storms on the technological infrastructure, which can lead to significant disruptions and damage. Although the 11-year solar cycle is determined through direct observations of sunspot numbers for the past four centuries, scientists want to know whether there is a longer cycle in solar activity which could influence the climate of Earth (Fig. 3a). Therefore, in this task, in addition to predicting the sunspot number, we also focus on how to derive the long-term cyclical variations of solar activity from the predicted physics formulas.

We first collected the SSN data from the Sunspot Index and Long-term Solar Observations (SILSO) over the last 400 years[45]. The most widely adopted formula is the one proposed by Hathaway et al.[46], which is still being used in recent studies to analyze data from the last 30 years[47]. This formula modeled the sunspot numbers $R(t)$ using different sets of parameters for different cycles (Fig. 3b). Although it can accurately fit the data for each cycle, it cannot be generalized from one cycle to another, making it incapable of predicting future SSNs or revealing the longer cycle of solar activity. To summarize a symbolic formula for SSN, we selected the SSN data from year 1855 to 1976 which containing 11 cycles and 1,450 data points as input of our model. The main components of the denominator in our formula are a squared sine term and a squared cosine term, while the numerator consists of a squared sine term (Fig. 3c top). The Pearson correlation between the predicted SSN and the measured ones for the next four complete cycles (year 1976 to 2019) reaches 0.72. For the upcoming cycle (2019-present), PhyE2E predicts the peak value to be 177.40 and occurs on October 10, 2024 (Fig. 3c top). It is worth noting that no data were used to fine-tune or retrain our model. We did not use all the data to generate the formula because we needed to examine the generalizability of the formula obtained by the model. The generalizability of the model and the generalizability of the formulas predicted by our model should be evaluated separately. We tried generating formulas with different amounts of data, and the resulting formula forms remained largely consistent (Supplementary Tables S7, S8).

We compared the formula generated by PhyE2E with those generated by other SR models. Among all the models, only our model can be directly applied to the data for formula inference without any retraining or fine-tuning. Therefore, we retrained all the other models to be compared on data from year 1855 to 1976 and tested their performance on data after year 1976 to fairly compare with our model. We first examined the formulas generated by different SR models. We observed that, except for AIF and GP-GOMEA, all other methods produced formulas containing trigonometric functions capable of generating periodic outputs. Formulas from BSR, EndToEnd, KAN, NeSymReS, PhySO, and TPSR all yield identical values for each cycle, which clearly contradicts our observations and common sense. The formulas generated by GPLearn, Operon, ParFam, uDSR, and our method are capable of generating periodic outputs with different values for each cycle (Fig. 3c bottom). We further examined the formulas generated by PySR under different operator sets and different constraints variants (Supplementary Tables ???, ???), and take the best PySR model with the highest Avg-R score into comparison. However, the five other models generated excessively complex formulas, making it impractical to parse their physical meaning or ascertain the long-term cyclical patterns of solar activity (Supplementary Tables S9, S10). Next, we quantified the performance of the formulas generated by these models on the test sets from year 1976 to 2019 using two metrics: 1) the correlation between the formula's predictions and the measured data within each individual cycle, then averages these correlations across multiple cycles (avg-R); 2) the correlation across multi-cycles (multi-R). Our simpler formulas significantly outperformed these complex formulas for both metrics. Specifically, the avg-R and multi-R of our method outperform the second-best methods by at least 76.01% and 58.57%, respectively (Fig. 3d).

To further validate the accuracy of the formula we obtained, we focused on the SSN data before the 1700s. Due to technological limitations, there are no SSN data directly observed from telescopes prior to 1749[45]. Therefore, we collected solar modulation levels reconstructed from atmospheric ^{14}C concentrations from the annual rings of thousand-year-old trees as an approximation of the SSN measurements[48]. We adopted the same smoothing strategy as reported in Brehm et al., 2021 and found that the Pearson correlation between solar modulation from tree rings and the SSN measured by the telescopes is 0.886 after the 1700s, which verifies their close relationship (Fig. 3f). Then, we compared the SSN generated by our formula and solar modulation data before the 1700s. Their Pearson correlation is 0.561 before 1300, 0.653 during 1300 to 1700 and 0.501 after 1700 (Fig. 3e), which further demonstrates the predictive capacity of our formulas on previously unseen data. Lastly, since there are three sine/cosine functions in our formula, it is natural for us to derive three cycles from these trigonometric functions. The shortest cycle is 10.91 years, which aligns with observations of solar activity widely accepted by the research community. The second longest cycle is 59.27 years, which coincides exactly with the 60-year cycle of the ancient Chinese astronomical calendar system of Heavenly Stems and Earthly Branches. The longest cycle is 204.93 years, which we speculate is the cycle of the solar system operating within a larger planetary system (Fig. 3e). Although this result requires further confirmation through additional astronomical observational data, it is the first conjecture directly derived from symbolic formulas. The constants of our formula are in Supplementary Table S2.

Performance of plasma pressure prediction. Plasma pressure is a macroscopic parameter that plays an important role in plasma dynamics and the generation of electric currents. Increasing plasma pressure gradients in the radial direction causes the stretching of magnetic field lines and enhances perpendicular currents flowing azimuthally. The azimuthal plasma pressure gradient generates field-aligned currents, resulting in the bending of magnetic field lines (Fig. 4a). Wang et al. proposed a formula that describes how equatorial plasma pressure varies with its position relative to Earth using Geotail and the NASA mission Time History of Events and Macroscale

Interactions During Substorms (THEMIS) data. However, their formula involves exponential terms and 9 constants for night-side equatorial isotropic plasma pressure, making it difficult to derive meaningful physical interpretations [49, 50](Fig. 4b). To derive a simpler formula, we divided the same equatorial plasma pressure data according to the azimuthal angle, using the data from the near-side of the Earth as input for our model and the data from the far-side to assess the performance of our formula. As we increased the number of input data points, feeding the PhyE2E model with progressively farther data from Earth, the average mean square error (MSE) decreased rapidly (Fig. 4c, Supplementary Table S11, S12). PhyE2E achieves an average MSE of 7.04×10^{-3} with only 10% of the data provided, demonstrating strong generalization capabilities with the small dataset as input. As more data was provided, the accuracy of our model continued to improve, eventually reaching 6.63×10^{-4}, resulting in more precise predictions for the far side of Earth regions (Fig. 4e). PhyE2E also outperforms all other baseline models in terms of fitting accuracy (MSE) and model complexity, delivering the most accurate and simplest prediction formula, while other models are unable to provide accurate predictions in certain areas. The formula by Wang et al. cannot accurately predict the plasma pressure for the far side of Earth regions, while the EndToEnd method fails to provide accurate predictions for the near side of Earth regions (Figs. 4d). Note that this problem involves two variables, so there are two possible scenarios: one where r and θ can be decomposed into two sub-formulas, and another where decomposition is not possible. We generated one formula for each scenario, compared their MSE on the training dataset, and selected the formula with the lower MSE as the final prediction. In this case, the formula derived using the decomposition method outperformed the alternative. The formula we predicted has a more concise mathematical form compared to the formula proposed by Wang et al. (Fig. 4b). It reveals to us that the decay of the near-Earth plasma pressure is proportional to the square of the distance r to the Earth's center, whereas in the formula proposed by Wang et al., the plasma pressure has an exponential relationship with r. More importantly, we can derive certain physical facts that align with the observational data from this new formula. Specifically, if the plasma pressure decays with the square of r and it is also known that the magnetic pressure decays with the sixth power of r. Then, according to the formula plasma beta = magnetic pressure (Pth)/plasma pressure (Pb), we can infer that plasma beta increases with the fourth power of r, which can be confirmed by observational data from another independent study[51]. Among the formulas obtained by the other methods, only the EndToEnd approach[24] produced a formula that is inversely proportional to the square of r. However, its mathematical form is more complex compared to our formula (Supplementary Table S11). The constants of our formula are in the Supplementary Table S3.

Performance of solar rotational angular velocity prediction. The Sun's magnetic field is generated by the plasma motion within its interior. Angular velocity of solar rotation varies at different latitudes, and the magnetic field lines are stretched and twisted (Fig. 4f). Differential rotation is a significant factor in the solar cycle for the prediction of magnetic fields and sunspots. Understanding solar differential rotation helps to improve the prediction of solar activities, which is important for predicting and mitigating the impact of space weather events on satellites and human activities in space. Differential rotation also provides key insights into the structure and dynamics of the solar interior by comparing rotation speeds at different latitudes to those predicted by comprehensive numerical models[52, 53]. One of the most widely adopted formulas decribing the relationship between the solar differential rotation and solar latitude was derived by Snodgrass et al.[54] In this work, they assumed that solar differential rotation was symmetric with respect to the equator. Such an assumption often fails especially during periods of high levels of solar activity. In addition, this formula was fitted by using the measurements at low latitudes of the Sun, but observations at high latitudes are still missing, which limits the suitability of the model near the solar poles. For this task, the challenge for other AI models is that the limited amount of data makes it impossible for them to train the model and predict the formulas.

PhyE2E derived a simpler and more accurate formula with a simple trigonometric term using the data from Snodgrass et al. in Magnetic Rotation of the Solar Photosphere[54]. The largest difference between this formula and the one generated by PhyE2E lies in their difference in the trigonometric periodicity, leading to a steeper prediction for polar regions, rather than a flat one (Fig. 4g). We further reduced the number of training data points and found that PhyE2E could predict the same formula with only 14 data points as input, rapidly achieving an MSE of 1.31×10^{-4}, which outperforms other models (Fig. 4g, Supplementary Table S13, S14). In contrast to the oscillations seen in other models, PhyE2E exhibits exceptional consistency and robustness, providing stable predictions with varying amounts of input data. The constants of our model can be found at the Supplementary Table S4. The predictions for all the baseline models are quite similar in non-polar regions, but they start to diverge in the polar regions (Fig. 4h). Regarding predictions at high latitudes, Hotta et al.[52] overcame the "convective conundrum" through the supercomputer Fugaku and successfully reproduced solar-like differential rotation. We selected the simulation data from the north and south polar regions and compared them with the baseline models. The formula derived from PhyE2E performed the best in both polar regions, with correlations of 0.9814 and 0.9740, outperforming all baseline models including the formula proposed by Snodgrass et al[54]. In addition, we applied our model to different heights in the solar atmosphere, using data from various spectral lines[55] in the photosphere (Si I and Fe I) and the chromosphere (Hα) (Fig. 4i). Similar formulas are derived across all the spectral lines with remarkable consistency, suggesting that the differential rotation speed within the Sun follows a regular and predictable pattern(Fig. 4j).

Performance of contribution function of emission lines. Emission lines in the extreme ultraviolet spectrum of the Sun, such as Fe X lines, are often used to observe the solar corona (Hinode/EIS, Solar Orbiter/EUI) (Fig. 5a). Predicting the contribution functions of these lines helps in plasma diagnostics such as temperatures, densities, and magnetic fields, which is essential to understand solar phenomena such as flares and coronal mass ejections. The EUV emission lines can also be formed in other types of astrophysical targets, including stellar coronae, galactic nuclei, and supernovae. Understanding the formation of the emission lines can provide insight into the physical conditions and processes of these targets. Given its role as a critical component of fundamental atomic databases applicable to a wide range of studies, considerable efforts have been made to calculate the contribution functions of emission lines (CHIANTI[56, 57] and NIST atomic database[58]). The contribution functions could be approximated by solving complex quantum mechanical equations involving detailed calculations of electron transitions, collisions, and radiative transfer, which is usually a computationally expensive process. Therefore, a challenge in physics is whether we can accurately estimate the contribution function using easily observable data, including the temperature and electron density around the Sun. Currently, there is no physics formula that accurately reveals how the temperature and electron density around the Sun influence the contribution levels of the emitted spectral lines.

To address this problem, we downloaded the Fe X 174 and Fe X 175 line data from the CHIANTI database[56, 57], and uniformly sampled 2,500 data points in an electron density range of $10^8 - 10^{10}$ and a temperature range of $5 \times 10^5 - 5 \times 10^6$°C. Data were segmented into low and high temperature regions using a cutoff of 2.8×10^6°C. Instead of fine-tuning or retraining our model, we took the data from the low-temperature region as input to generate the formula and test the prediction performance of the formula in the high-temperature region. In the high-temperature region, PhyE2E achieved a significantly lower MSE, with the magnitude of the MSE being two orders of magnitude smaller than the other baseline models (Fig. 5b left). Both the Fe X 174 and 175 emission lines are highly temperature-dependent, with a relatively smaller influence from the electron density. To address this issue, we further investigated the ratio of these two lines, which serves as a powerful diagnostic tool for probing the effects of electron density on the intensity. Compared to the prediction of the two spectral lines individually, the prediction of the ratio of the two lines carries more physical significance and is also more challenging. Accurate predictions of the individual spectral lines do not guarantee that their ratio can be predicted accurately (Figs. 5b,c,d). In this task, our formula achieves an MSE of 3.10×10^{-3}, which is three orders of magnitude lower than the second best method EndToEnd (Fig. 5b, middle).

Next, we examined the physical significance of the formulas generated by different methods. First, the complexity of our formula is not the lowest, but it is the only formula that has the correct physical units among all the baseline methods (Figs. 5b right, Supplementary Table S15, S16). In the solar corona, the processes of ionization and recombination, as well as collision and excitation, can be considered decoupled[59]. In our formula, the electron density and temperature also exist in a decoupled form. The dependence of the electron density following the mathematical form $(n + c_1)/(n + c_2)$ is widely accepted by the space physics community[60]. It captures the behavior where the intensity increases with electron density at low values but saturates at higher densities because of collisional de-excitation. The temperature term of our formula is composed of the combination of a power-law term and an exponential term. The power-law term dominates at low temperatures, capturing the increase in intensity as more electrons gain sufficient energy to excite the ions; the exponential decay term dominates at high temperatures, reflecting the rapid decrease in intensity due to ionization to higher states. This combination of a power-law term and an exponential term was also adopted by Raymond et al.[61] The constants of our formula are in the Supplementary Table S5. For the formulas generated by other methods, some have excessive complexity, making them difficult to interpret, while others have incorrect physical units or are overly simplistic. For instance, the formula produced by PhySO is simple, but does not include the electron density term (Supplementary Table S15).

Performance of lunar tide signal of plasma layer. The Earth's magnetospheric electric fields, including corotation and convection electric fields, are crucial for understanding the behavior of charged particles and maintaining the stability of the magnetosphere (Fig. 5e). These fields are responsible for the movement and energization of charged particles, which in turn affect space weather and the interaction between the solar wind and Earth's magnetic field. Prior to 2023, it was commonly accepted in the scientific community that the electric fields at a specific near-Earth location were solely influenced by the Earth's position relative to the Sun and the distance to the Earth's center. A recent work indicated that due to the effects of lunar tidal forces, the electric fields were also related to the Earth's position relative to the Moon [62]. However, due to the complexity of the problem, physicists have been unable to provide a physical formula that links these three important factors: electric fields with the distance to the Earth's center, the relative positions of the Earth and the Moon, and the Earth's position relative to the Sun, which can be represented as Lunar Local Time (LLT), Magnetic Local Time (MLT) and L-shell, respectively.

To address this problem, we collected ~20,000,000 data measured by the Van Allen Probes satellites between L values of 3–6 from January 2013 to May 2019 from the RBSP/EFW official website (http://www.space.umn.edu/rbspefw-data/), and used PhyE2E to generate a formula to predict Radial Electric Field, denoted as E_r, from LLT, MLT and L-shell values. Due to the large volume and high redundancy of data, we divided the entire three-dimensional space near the Earth into $50 \times 50 \times 50$ grids, and then calculated the average value of the Radial Electric Field within each grid. We randomly sampled 80% of the grids as input for our model and also used them as training data for other baseline models. The remaining data were adopted as test data to evaluate the performance of all models. Based on the adaptive threshold for the decomposition of the formula, we found that there are no coupling relationships among these three variables. Therefore, we decomposed the original symbolic regression problem into three sub-problems, each containing only one variable. Then, we predicted each uni-variate function using the end-to-end model (Sec. 4.3 in Methods). Without fine-tuning or re-training of the model, the formula generated by our model has an MSE lower than the second-best method by 53.37%, with complexity reduced by 75.91% (Fig. 5f). We examined the effects of LLT and MLT on the Radial Electric Field separately and found that our formula provides a good approximation for the original data. The prediction of our formula is much smoother than the measured data, due to the data smoothing applied within each grid (Fig. 5g). Compared to the formulas generated by other methods, the formula produced by our model captures multiple physical principles, making it more physically meaningful. First, among all the models, only our model provides an asymmetric prediction between the dayside and nightside of the Earth, suggesting that the radial electric field (E_r) on the dayside decays more rapidly in the radial direction (L-shell), while on the nightside, the radial electric field(E_r) decays faster in the non-radial direction (MLT). Second, since the radial electric field (positive direction towards Earth) is derived from the calculation of the electric field's y-component, it exhibits periodic variation with Magnetic Local Time (MLT), and the period is 12 hours[63], which is consistent with the periodicity of the cosine function of MLT in our formula (12.13 hours). Third, our formula indicates that E_r decays with the square of the L-shell, which is consistent with theoretical calculations. According to the ideal magneto-hydrodynamic (MHD), the corotational electric field E could be derived as $E = -\Omega_E B_0 / L_{shell}^2$, where Ω_E and B_0 are Earth's rotational angular velocity and Earth's surface magnetic field, respectively. The constants of our model could be found in Supplementary Table S6. The complexity of our formula is not the lowest among all the models because the pattern of this physical application is complicated (Fig. 5b, Supplementary Table S17, S18). Among methods with lower complexity, only the formulas from BSR and PySR exhibit periodicity in LLT or MLT, and only PySR captures the inverse relationship between E_r and L-shell. However, the BSR formula does not include the crucial L-shell variable, and the PySR formula lacks the LLT variable (Supplementary Table S17).

3 Discussion

Existing symbolic regression research primarily employs search methods based on Monte Carlo Tree Search (MCTS) and Reinforcement Learning (RL). These methods often struggle to accurately predict formulas with a large number of variables or complex operational relationships between variables. To discover the correct formulas within a limited time, most of the MCTS approaches require prior knowledge to achieve an initialization close to the true solution. In contrast, our method can decompose formulas without knowing their specific forms, significantly reducing the complexity of symbolic regression. For each decomposed sub-problem, we utilized an end-to-end approach, tokenizing the data and directly translating it into formula strings using a transformer. Among all SR methods, our approach offers a ready-to-use model and is the only one that does not require retraining or fine-tuning on physical data.

One characteristic of space physics is that there is no data noise on the planet scales or on the microscopic particle scale. However, if the model is to be generalized to other areas of physics, such as condensed matter physics and fluid dynamics, the effects of noise inherent in the data must be considered. Since the data is free from noise, the model must learn to deduce the operational relationships between physical variables based solely on the data provided. Therefore, we can still leverage large language models using this method of simulating the generation of physics formulas for data augmentation. This principle is similar to that of AlphaZero[64], which does not require human game records but can learn to play Go through AI-versus-AI games. This is because AI only needs to learn optimal strategies in various complex situations, rather than necessarily mimicking human players' thought processes and habits. Currently, our current model cannot handle operations such as integration and differentiation, which means that a significant portion of physics formulas based on partial differential equations cannot be resolved. We believe that the data augmentation and formula decomposition techniques are still applicable in partial differential equations.

4 Methods

We now detail the components of our system, starting with the generative model for synthetic physics formulas. Next, we present the core framework, which includes an end-to-end symbolic regression model integrated with a Divide-and-Conquer (D&C) strategy for decomposing complex formulas into simpler sub-formulas. We then describe the process to refine the formula predicted by the end-to-end model, encompassing Monte Carlo Tree Search (MCTS) and Genetic Programming (GP) refinement. Finally, we discuss the details of how to construct test data and evaluation metrics.

4.1 Generative model for synthetic physics formulas

To generate synthetic formulas resembling real physics formulas, we fine-tuned the pre-trained OpenLLAMA-2-3B language model[65] using the AI Feynman dataset[29], which is a benchmark collection of mathematical formulas representing real-world physical laws and relationships, consisting of 100 formulas. A two-stage fine-tuning strategy was devised to address the challenge of limited training data and to integrate prior knowledge of physical unit systems into the training process. In the first stage, the OpenLLAMA-2-3B model was fine-tuned on the AI Feynman training set. The fine-tuned model was then employed to generate 50,000 synthetic formulas. These formulas were evaluated for consistency with physical unit systems, resulting in approximately 8,000 formulas that adhered to unit consistency. The second stage involved reassigning weights to the 8,000 unit-consistent formulas generated in the first stage. This weighting was designed to ensure that the statistical distribution of the synthetic formulas, such as the number of variables, formula depth, and operator frequency, aligned with those of the real physics formulas in the AI Feynman dataset. Finally, the language model was further fine-tuned using the weighted distribution of the 8,000 formulas.

Specifically, the fine-tuning was performed using the DeepSpeed ZeRO Stage 2 optimizer within the HuggingFace Transformer framework[66], with a learning rate of 3×10^{-5}. The training prompt followed the format: "`Generate a physics formula: {formula}`". Mathematical formulas in the prompt were represented in plain text using their natural mathematical forms. The notation of variables for physical quantities in the formulas adhere to the standard conventions used in the Feynman Dataset. To generate synthetic formulas, the model was prompted with the same instruction. The hyperparameters of the generative model were configured to balance diversity and quality in the generated formulas: the temperature was set to 2.0, efficient sampling was enabled (`do_sample = True`), and the maximum length of the generated sequences was restricted to 64 tokens. To evaluate the consistency of synthetic formulas with physical unit systems (in the first stage), the formula's expression tree[67] was constructed, and the units of each subtree were computed in a bottom-up manner, starting from the leaves and moving toward the root. During this process, the following checks were performed: (1) for any sub-tree in the form of $A + B$ or $A - B$, the units of A and B were required to be the same, (2) for any sub-tree in the form of $\sin(A)$, $\cos(A)$, or $\exp(A)$, the unit of A was required to be null. For instance, the formula "acceleration + velocity / time" is valid while the formula "acceleration + velocity" or "sin(acceleration + velocity / time)" is invalid due to unit mismatch. No constraints were imposed on operations such as multiplication, division, or square root, although these operations produce derived units that may affect the validity of their parent expressions. For example, "sqrt(acceleration / time)" yields a unit equivalent to that of velocity, so the formula "sqrt(acceleration / time) + velocity" satisfies the unit consistency requirement for addition. In contrast, "sqrt(acceleration / time) + velocity / time" fails this criterion. To reassign weights to a set of formulas (in the second stage), a linear program (LP) was formulated. The LP variables represent the weights assigned to each formula, subject to the constraints that the weights must be non-negative and sum up to 1. The objective was to minimize the total variation distances between the statistical distributions (e.g., number of variables, formula depth, operator frequency) of the weighted synthetic formulas and those of the AI Feynman dataset. These total variation distances were expressed as linear combinations of the LP variables. Additionally, a regularization term was included in the LP objective to ensure that the weighted distribution did not deviate excessively from the original uniform distribution.

4.2 The divide-and-conquer strategy

Many physics formulas exhibit intrinsic simplicity and symmetry, with variables often interconnected through straightforward addition or multiplication. Inspired by this observation, we proposed a divide-and-conquer strategy to decompose the target formula into a summation (or multiplication) of simpler sub-formulas by estimating inter-variable relationships. To achieve this, we first train an oracle neural network to fit the data, then use the hessian matrix to identify the inner nonlinear relationship between variables. These relationships guide the decomposition strategy, breaking the target formula into several simpler sub-formulas, which are then predicted independently and subsequently combined to reconstruct the target formula. For instance, consider the mathematical formula $f(x_1, x_2, x_3, x_4) = x_1 x_2 + x_3 \log(x_4)$. It is straightforward to verify the second derivatives between the group $\{x_1, x_2\}$ and the group $\{x_3, x_4\}$ are zero, i.e., $\partial^2 f / \partial x_i \partial x_j = 0, \forall i \in \{1, 2\}, \forall j \in \{3, 4\}$, which mathematically indicates that the target formula can be decomposed into two sub-formulas $f_1(x_1, x_2) = x_1 x_2$ and $f_2(x_3, x_4) = x_3 \log(x_4)$. Our divide-and-conquer strategy is based on a generalization of the underlying mathematical principle of the above example. Below, we first introduce the inner-variable relationships, followed by the decomposition strategy for sub-formulas and the resampling technique for data points. Finally, we present the aggregation theorem for the back aggregation step.

4.2.1 The oracle neural network and estimation of inter-variable relationships

Given the data $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ with N evaluations of the target formula $f(\boldsymbol{x})$ ($N = 200$ in our experiments), an oracle neural network $\tilde{f}_\theta(\boldsymbol{x})$ was first trained to approximate $f(\boldsymbol{x})$ at any input point $\boldsymbol{x}$. Here, θ are the parameters of the oracle neural network, which consists of 5 hidden layers. The first 3 layers each contained 128 tanh neurons, while the last 2 layers each contained 64 tanh neurons. The network was trained for 400 epochs, with an initial learning rate of 0.1 that decayed tenfold every 100 epochs. The following definition characterizes the inter-variable relationship to be estimated for decomposing the target formula.

Definition 1. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a uni-variate operator. Two features $i, j \in \{1, 2, \dots, n\}$ of a target formula $f : \mathbb{R}^n \to \mathbb{R}$ are said to be* σ-separable *if there exist sub-formulas f_1 and f_2 such that f can be expressed as:*

$$f(\boldsymbol{x}) \equiv \sigma(f_1(\boldsymbol{x}_{-i}) + f_2(\boldsymbol{x}_{-j})),$$

where $\boldsymbol{x}_{-i}$ is the $(n-1)$-dimensional vector obtained by removing x_i from $\boldsymbol{x}$, and $\boldsymbol{x}_{-j}$ is defined analogously.

The uni-variate operator σ cannot be generalized to multi-variate operator, as our method relies on the invertibility of σ. When the operator σ is invertible, the following lemma, whose proof is provided in Supplementary Sec. 5.1, provides an equivalent condition to check whether two features are σ-separable in a twice-differentiable formula.

Lemma 1. *Let the uni-variate operator $\sigma : \mathbb{R} \to \mathbb{R}$ and the target formula $f : \mathbb{R}^n \to \mathbb{R}$ be twice differentiable. Suppose σ is strictly monotonic, then two features $i, j \in \{1, 2, \dots, n\}$ are σ-separable if and only if for all $\boldsymbol{x} \in \mathbb{R}^n$,*

$$\frac{\partial^2 \sigma^{-1} \circ f}{\partial x_i \partial x_j}(\boldsymbol{x}) = (\sigma^{-1})''(f(\boldsymbol{x})) \cdot \frac{\partial f}{\partial x_i}(\boldsymbol{x}) \cdot \frac{\partial f}{\partial x_j}(\boldsymbol{x}) + (\sigma^{-1})'(f(\boldsymbol{x})) \cdot \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}) = 0. \tag{1}$$

Practical implementation. In our experiment, we tried the uni-variate operators $\sigma \in \{\text{id}, \text{sqrt}, \text{inv}, \text{arcsin}, \text{arccos}, \log, \text{sqrt} \circ \log, \text{inv} \circ \log, \text{arcsin} \circ \log, \text{arccos} \circ \log\}$ to perform the divide-and-conquer strategy and predict the target formula via the end-to-end model (Sec. 4.3). Note that the operators that involve logarithm effectively separate the target formula into the multiplication of two sub-formulas according to Definition 1. The prediction based on different uni-variate operators were collected and fed into the MCTS and GP module for further refinement (Sec. 4.4).

However, even with access to the oracle neural network $\tilde{f}_\theta$, we cannot directly verify the condition in Lemma 1 because it required evaluating all the points in $\boldsymbol{x} \in \mathbb{R}^n$ and the second-order derivative of the approximate function $\tilde{f}_\theta(\boldsymbol{x})$ tends to be noisy. In our algorithm, this condition was verified approximately by sampling a subset of points and employing a majority rule to mitigate the effects of approximation noise. Specifically, for the set of training data points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$ ($N = 200$), two features i and j are determined to be σ-separable if, for a threshold parameter $\epsilon > 0$, it holds that

$$J_{i,j}(\sigma, \tilde{f}_\theta) \overset{\text{def}}{=} \underset{1 \le k \le N}{\text{median}} \left\{ \left| \frac{\partial^2 \sigma^{-1} \circ \tilde{f}_\theta}{\partial x_i \partial x_j}(\boldsymbol{x}_k) \right| \right\} \le \epsilon. \tag{2}$$

The choice of the threshold parameter ϵ is crucial to the accurate identification of σ-separable pairs. We first employ the *fixed thresholding* strategy, treating $\epsilon = \epsilon(\sigma)$ as a constant for each uni-variate operator σ. This fixed constant is determined using a data-driven approach for each σ. More specifically, we randomly sampled 1,000 target formulas h_k and trained the corresponding oracle neural networks $\tilde{h}_k(\boldsymbol{x}_k|\theta_k)$. For each feature pair (i, j), we identified their σ-separability by

Lemma 1 and calculated the corresponding $J_{i,j}(\sigma, \tilde{h}_k)$. The threshold was then chosen to maximize the number of feature pairs among the 1,000 additional target formulas that were classified correctly in terms of σ-separability.

On the other hand, a single constant value for a fixed ϵ may not work well for all target formulas f, as different functions can exhibit vastly different scales of derivatives. This scale can even vary if f is multiplied by a large constant factor. To address this issue, the technique of *adaptive thresholding* was also proposed to automatically select the threshold based on the derivative scale, thereby improving the numerical stability of the estimation of σ-separable pairs. More specifically, let $\epsilon_0 = \min_{1\le i<j\le n} J_{i,j}(\sigma, \tilde{f}_\theta)$. We then define $\epsilon_1 = \alpha_{\min}\epsilon_0$ and $\epsilon_2 = \alpha_{\max}\epsilon_0$ as the minimum and maximum thresholds, respectively. In our algorithm, we set $\alpha_{\min} = 2$ and $\alpha_{\max} = 10$. For each $\epsilon \in [\alpha_{\min}, \alpha_{\max}]$, the set of σ-separable pairs (hereafter referred to as the *σ-separable set*) was estimated as

$$Q_\epsilon(\sigma, \tilde{f}_\theta) \overset{\text{def}}{=} \left\{(i,j) \;\middle|\; 1 \le i < j \le n, J_{i,j}(\sigma, \tilde{f}_\theta) \le \epsilon\right\}. \tag{3}$$

Finally, the class of σ-separable sets was defined as

$$\mathcal{Q} \overset{\text{def}}{=} \{Q_\epsilon(\sigma, \tilde{f}_\theta) \mid \epsilon \in [\alpha_{\min}, \alpha_{\max}]\}. \tag{4}$$

It is straightforward to verify that $|\mathcal{Q}| \le n(n-1)/2$. Therefore, applying the divide-and-conquer strategy based on each $Q \in \mathcal{Q}$ is computationally feasible. Each application of this strategy derived a prediction of the target formula. These formulas were evaluated and the best one was selected. In the following subsections, we will explain how the divide-and-conquer strategy operates for any estimated σ-separable set Q: it begins by inducing a set of feature sets, followed by predicting the sub-formulas for these feature sets, and ultimately integrates them into the final prediction of the target formula.

4.2.2 The division of the target formula

We have discussed the σ-separable relationship between pairs of features. The following definition about the global division of the target formula and features is crucial to our divide-and-conquer strategy.

Definition 2. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a uni-variate operator. For any feature subset A_i of $A = \{1, 2, \ldots, n\}$, denote by $\boldsymbol{x}_{A_i}$ the vector obtained by restricting $\boldsymbol{x}$ to A_i, i.e., if $A_i = \{j_1, j_2, \ldots, j_k\}$, then $\boldsymbol{x}_{A_i} = (x_{j_1}, x_{j_2}, \ldots, x_{j_k})$. A class of subsets $\{A_i\}_{i=1}^m$ is said to be a* σ-division *of f if none of the subsets is contained in another subset and there exist m sub-formulas $f_1, f_2, \ldots, f_m$ such that f can be expressed as:*

$$f(\boldsymbol{x}) \equiv \sigma(f_1(\boldsymbol{x}_{A_1}) + f_2(\boldsymbol{x}_{A_2}) + \ldots + f_m(\boldsymbol{x}_{A_m})), \tag{5}$$

where each f_i is a function of $\boldsymbol{x}_{A_i}$.

It is quite straightforward to derive a σ-division from σ-separable pairs, detailed as follows. First, the σ-separable relationships between all features are identified as described in Sec. 4.2.1. Then, the algorithm starts with an initial (and trivial) σ-division $\mathcal{A} = \{\{1, 2, ..., n\}\}$ and iteratively refines it. At each iteration, the algorithm selects a σ-separable pair of features j_1 and j_2 that has not been considered. For each feature subset $A \in \mathcal{A}$ such that $\{j_1, j_2\} \subseteq A$, $\mathcal{A}$ is updated as follows:

$$\mathcal{A} \leftarrow (\mathcal{A} - \{A\}) \cup \{A - \{j_1\}, A - \{j_2\}\}.$$

This process is done after all σ-separable feature pairs have been considered. By the following Lemma 2, whose proof can be found in Supplementary Sec. 5.1, this iterative procedure guarantees to yield a valid σ-division.

Lemma 2. *Let uni-variate operator $\sigma : \mathbb{R} \to \mathbb{R}$ be strictly monotonic and $f : \mathbb{R}^n \to \mathbb{R}$ be the target formula that is twice differentiable. Suppose we have accurately obtained the set of all σ-separable feature pairs of f. Then, for each iteration number $\ell \geq 1$, the division obtained by the above algorithm after the ℓ-th iteration, denoted by $\{A_k^\ell\}_{k=1}^{m_\ell}$, is a σ-division of f.*

Algorithm 1 is provided in Supplementary Sec. 4 to formally describe the above procedure to derive σ-separable feature pairs via the fixed and adaptive thresholding techniques, as well as to construct a σ-division based on the σ-separable pairs.

4.2.3 Evaluation of surrogate sub-formulas and back aggregation for the target formula

Once a σ-division $\{A_i\}_{i=1}^m$ is derived, it would be most natural to recursively perform symbolic regression to predict the sub-formulas $\{f_i\}$ and reconstruct the target formula f according to Eq. (5). However, there are two challenges to this approach. First, we do not have evaluation data ($\{(\boldsymbol{x}, y = f_i(\boldsymbol{x})\}$) for each sub-formula f_i. Moreover, the sub-formula f_i are not even unique. For example, if $f(x_1, x_2, x_3) = \sigma(f_1(x_1, x_2) + f_2(x_1, x_3))$, then it can also be expressed as $f(x_1, x_2, x_3) = \sigma((f_1(x_1, x_2) - x_1) + (f_2(x_1, x_3) + x_1))$, where $f_1' = f_1 - x_1$ and $f_2' = f_2 + x_1$ are also a set of valid sub-formulas for the σ-division $\{\{x_1, x_2\}, \{x_1, x_3\}\}$.

To address the above challenges, we turn to predict the *surrogate sub-formulas* $\{g_i\}_{i=1}^m$, defined as follows. First, an arbitrary $\boldsymbol{z} \in \mathbb{R}^n$ is chosen, where in the experiment, $\boldsymbol{z}$ was sampled from the standard multivariate Gaussian distribution. Then, given the σ-division $\{A_i\}_{i=1}^m$, for each $i \in \{1, 2, \ldots, m\}$, we define $g_i : \mathbb{R}^{A_i} \to \mathbb{R}$ as

$$g_i(\boldsymbol{x}_{A_i}) = f(\boldsymbol{x}_{A_i}, \boldsymbol{x}_{\overline{A_i}} = \boldsymbol{z}_{\overline{A_i}}), \tag{6}$$

where $\overline{A_i} = [n] - A_i$.

Each surrogate sub-formula g_i might be quite different from the corresponding sub-formula f_i. For instance, consider $f(x_1, x_2, x_3) = f_1(x_1, x_3) + f_2(x_2, x_3)$ where $f_1(x_1, x_3) = \sin(x_1 x_3)$ and $f_2(x_2, x_3) = \cos(x_2 x_3)$. According to the definition, we have $g_1(x_1, x_3) = \sin(x_1 x_3) + \cos(c_2 x_3)$, and $g_2(x_2, x_3) = \cos(x_2 x_3) + \sin(c_1 x_3)$. Note that each g_i not only contains f_i, but also introduces extra non-trivial terms. The following theorem, the proof of which can be found in Supplementary Sec. 5.2, provides a way to eliminate the extra terms and aggregate the surrogate sub-formulas $\{g_i\}_{i=1}^m$ to reconstruct the target formula f.

Theorem 3. *Suppose the uni-variate operator $\sigma : \mathbb{R} \to \mathbb{R}$ is strictly monotonic. Let $\mathcal{A} = \{A_i\}_{i=1}^m$ be a σ-division of the target formula f and $\{g_i\}_{i=1}^m$ be defined as in Eq. (6) based on a vector $\boldsymbol{z}$. For each $I \subseteq [m]$, denote*

$$\mathcal{A}_I = \cap_{i \in I} A_i.$$

Then, it holds that

$$f(\boldsymbol{x}) = \sigma\left(\sum_{\emptyset \neq I \subseteq [m]} \frac{(-1)^{|I|-1}}{|I|} \sum_{i \in I} \sigma^{-1} \circ g_i(\boldsymbol{x}_{\mathcal{A}_I}, \boldsymbol{x}_{A_i - \mathcal{A}_I} = \boldsymbol{z}_{A_i - \mathcal{A}_I})\right). \tag{7}$$

Practical implementation. For the i-th surrogate sub-formula g_i, the data $D^{(i)} = \{\boldsymbol{x}_k^{(i)}, y_k^{(i)}\}_{k=1}^N$ were constructed as follows:

$$\boldsymbol{x}_k^{(i)} = (\boldsymbol{x}_k)_{A_i}, \qquad\qquad y_k^{(i)} = \tilde{f}_\theta(\boldsymbol{x}_{A_i} = \boldsymbol{x}_k^{(i)}, \boldsymbol{x}_{\overline{A_i}} = \boldsymbol{c}_{\overline{A_i}}),$$

where $\boldsymbol{x}_k$ is the k-th data point in the original data D. Then, each g_i where predicted by the end-to-end model (as will be described in Sec. 4.3). Finally, the target formula f was predicted by aggregating the surrogate sub-formulas according to Eq. (7).

4.3 The end-to-end model

Architecture. Kamienny et al.[24] established a transformer-based end-to-end model for symbolic regression. We design a similar transformer that includes 4 encoder layers and 16 decoder layers, forming an asymmetric architecture[68]. Each layer consists of 16 attention heads and an embedding dimension of 512. We remove positional embeddings to ensure permutation invariance of the input data.

Physical priors (e.g. physical units) play a crucial role in governing the structure and plausibility of physics formulas[35, 69]. In PhyE2E, we integrate four types of physical priors, including the physical units of input and output variables, formula complexity, candidate operators and candidate constants. Besides the evaluations at N data points, we also append a set of h candidate physical priors to the input of our model.

To encode the observed data points, symbolic formulas, and physical priors, we construct a vocabulary that includes tokens for float numbers, operators, variables, and physical units. Each float number is decomposed into three tokens representing its sign, mantissa (between 0 and 9999), and exponent (from E-100 to E+100)[68]. For physical units, we adopt five commonly used base units: Meter (m), Second (s), Kilogram (kg), Kelvin (K), and Volt (V)[29], which are slightly different from the International System of Units (SI). In this way, a physical unit can be encoded into five tokens. For dimensionless constants and formulas, we simply assign each base unit with value 0. Each data point is encoded into $(n + 1) \times 3$ tokens, where each of the n features and the evaluation is encoded into 3 tokens. Each of the h physical priors is represented using specialized vocabulary tokens that encode the variables, physical units, operators, and float numbers. Specifically, we use $1 + 5$ tokens to represent a variable with its physical unit, and $3 + 5$ tokens to represent a constant with its physical unit. All data point and physical prior sequences are padded with the `<pad>` token to ensure a uniform fixed length $L = 33$. They are finally concatenated together to form the input of our model.

Additionally, to help the model better understand a consistent physical unit system, we propose a novel unit decoding strategy for the model's output. Specifically, we incorporate the physical units of each operator and variable within the formula into the output sequence. While formulas are represented as prefix expressions composed of operators and variables, we enhance this representation by associating each operator and variable with its corresponding physical unit, so that our model outputs a sequence of `(operator/variable, physical unit)` pairs. Notably, we do not impose explicit unit consistency constraints during generation. Instead, by explicitly inferring the physical units of the formula in a step-by-step manner within the output sequence, the model is guided to learn the underlying physical principles that govern variable relationships automatically. This approach enables consistent and meaningful unit inference directly from data.

Training details. The training data consisted of the data evaluated by 200,000 synthetic physics formulas generated in Sec. 4.1. Each formula was evaluated at $N = 200$ points sampled from the standard multivariate Gaussian distribution, conditioned on the fact that the formula is properly defined at the point, which follows the previous work[24].

We used the cross-entropy loss on the next-token prediction with the Adam optimizer, starting with a learning rate of 10^{-7}, gradually increased to 2×10^{-4} during the first 10,000 steps, and subsequently decayed proportional to the inverse square root of the step count[70]. A validation set comprising 20,000 synthetic formulas is held out, and our model is trained for 100 epochs, processing 500 million formulas in total, until the validation accuracy stabilizes. Each batch consists of 10,000 tokens, with formulas grouped by similar lengths to minimize padding overhead. The training was performed on a single NVIDIA A100 GPU with 80GB of memory over about one day.

Constant optimization. The end-to-end model only returned a "formula skeleton" where the constant parameters in the formula remained unoptimized. We adopted the BFGS algorithm[71] to further refine the constants in the formula skeleton. Specifically, given a formula f, we denote by $\boldsymbol{c}$ the constant parameters in the formula. Based on the data $\{(\boldsymbol{x}_k, y_k)\}_{k=1}^N$, the BFGS algorithm was invoked to find

$$\arg\min_{\boldsymbol{c}} \sum_{k=1}^{N} [f(\boldsymbol{x}_k; \boldsymbol{c}) - y_k]^2.$$

4.4 MCTS and GP refinement

MCTS. The standard Monte Carlo Tree Search (MCTS) process consists of four steps:[20] 1) selection, where the best candidate formulas are identified based on their performance; 2) expansion, where new symbolic formulas (which may be incomplete) are generated by adding one more operator from the incomplete formula; 3) simulation, where the incomplete formulas are randomly simulated and then evaluated based on their predictive accuracy; and 4) back-propagation, where the results are propagated back to update rewards and visit times of each MCTS node. We employed MCTS to refine the target formula obtained by the end-to-end model. Specifically, random sub-trees were removed from the expression tree of the target formula, and MCTS was invoked with the resulting incomplete formula to search for a better target formula. Moreover, following the work of Sun et al.[20], we constructed a grammar pool to reduce the search space of MCTS. Specifically, this grammar pool consisted of basic operators ($+$, $-$, $\times$, $/$, $\sin(x)$, $\cos(x)$, $\exp(x)$) and the operators extracted from the predicted results of the end-to-end model. During the expansion steps, MCTS was restricted to select operators only from the grammar pool to construct a complete symbolic formula.

Genetic programming. Our genetic programming (GP) refinement module followed the work of PySR[30], where we used the results from the end-to-end model and MCTS as the initial populations for the GP algorithm. During each round of evolution, each formula in the population undergoes random mutations, including appending, changing, or deleting specific operators in the formula. Crossover operations were also performed between individuals to combine their expressions and create new formulas. The entire GP algorithm optimizes the formulas for 40 rounds. The optimization process stops early if the MSE of one of the generated formulas achieves 10^{-6} during the process. Finally, a Pareto front of the formulas was generated. The best formula from this set was selected based on the criterion of $\mathrm{MSE} \times 0.99^{\text{complexity}}$ to serve as the final solution, considering a trade-off between both accuracy and complexity.

4.5 Details of test data

To ensure that no data leakage occurred, we strictly split the training and test datasets. While it is not straightforward to identify mathematically equivalent formulas via symbolic derivations, we chose to measure the similarity between two formulas based on numerical evaluations. Specifically, we define the similarity between two formulas, f_i and f_j, with the same number of features, using the averaged R^2 score:

$$\mathrm{sim}(f_i, f_j) \overset{\text{def}}{=} 1 - \frac{1}{2}\left[\frac{\sum_{k=1}^{N'}(f_i(\boldsymbol{x}_k) - f_j(\boldsymbol{x}_k))^2}{\sum_{k=1}^{N'}(f_i(\boldsymbol{x}_k) - \hat{f}_i))^2} + \frac{\sum_{k=1}^{N'}(f_i(\boldsymbol{x}_k) - f_j(\boldsymbol{x}_k))^2}{\sum_{k=1}^{N'}(f_i(\boldsymbol{x}_k) - \hat{f}_j))^2}\right],$$

where $\hat{f}_i \overset{\text{def}}{=} [\sum_{k=1}^{N'} f_i(\boldsymbol{x}_k)]/N'$, $N' = 40$, and the $\boldsymbol{x}_k$'s are independently sampled from the standard multivariate Gaussian distribution. The similarity is set to 0 if there exists an $\boldsymbol{x}_k$ such that exactly one of $f_i(\boldsymbol{x}_k)$ and $f_j(\boldsymbol{x}_k)$ is undefined. The similarity between two formulas with a different number of features is also set to 0. Observe that two mathematically equivalent formulas achieve the maximum possible similarity 1.

We selected 3,000 formulas such that the pairwise similarity is less than 0.99 to form our test set. For the training dataset, we only removed the symbolically identical formulas while not requiring the pairwise similarity to be away from 1. This is because formulas that are mathematically equivalent but with different symbolic forms may help the model understand the mathematical equivalences. Moreover, such formulas might originate from different physical scenarios and have different physical units, thus still represent distinct entities. The formulas in the test set were further divided into three difficulty levels based on their maximum similarity with the formulas in the training dataset. Formulas with a similarity greater than 0.95 were defined as the ones that were easy to predict, those with a similarity between 0.8 and 0.95 were considered as medium difficulty, and formulas with a similarity less than 0.8 were considered as hard.

4.6 Evaluation metrics

We evaluate the performance of our model and other baseline models using the following metrics.

- *Symbolic accuracy.* This evaluation metric was introduced by Cava et al.[41] Let the target formula be f. The symbolic accuracy of a predicted formula g is 1 if there exists a constant such that $f - g \equiv c$ or $f = c \cdot g$ ($c \neq 0$ in the second case); otherwise, the symbolic accuracy is 0.

- *Numerical precision.* This metric is used to evaluate the numerical precision of data points given by predicted formula. We utilize R^2-score for the predicted formula g on testing data points $\{\boldsymbol{x}_k^{\text{test}}, y_k^{\text{test}}\}_{k=1}^{N}$ ($N = 200$) which are sampled from the same distribution as the training data points:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i^{\text{test}} - g(\boldsymbol{x}_i^{\text{test}}))^2}{\sum_{i=1}^{N}(y_i^{\text{test}} - \bar{y})^2}$$

  where $\bar{y} \overset{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} y_i^{\text{test}}$.

- *Accuracy of physical units.* This metric aims to assess the capability of a model to generate formulas with consistent physical units. The accuracy is 1 if all operations in the formula work with compatible physical units and the physical unit of the result is the same as the target formula; otherwise, the accuracy of physical units is 0.

- *Formula complexity.* The complexity of a formula f is defined to be the number of operators, variables and constant parameters in the formula. We also define the relative complexity of a prediction g is the difference (in absolute value) of the complexity of g and that of the target formula f.
- *Formula depth.* The depth of a formula f is the maximum depth of its expression tree.

5 Data availability

AI Feynmen data can be downloaded from Feynman Symbolic Regression Database (https://space.mit.edu/home/tegmark/aifeynman.html). The formula generated by the LLM, including the training and test datasets can be downloaded[72] from https://figshare.com/articles/dataset/PhyE2E_datas/29615831/1. The sunspot number data could be downloaded from the Sunspot Index and Long-term Solar Observations website (https://www.sidc.be/SILSO/datafiles). The plasma pressure data could be downloaded from Geotail and Time History of Events and Macroscale Interactions During Substorms (THEMIS) website (https://themis.igpp.ucla.edu/overview_data.shtml). The solar rotational angular velocity data could be found at table presented by Snodgrass et al. 1983 in Magnetic Rotation of the Solar Photosphere[54]. We collected the contribution function of emission lines data from the CHIANTI website (http://www.chiantidatabase.org). Lunar tide signal data was downloaded from RBSP/EFW official website (https://www.space.umn.edu/rbspefw-data/).

6 Code availability

Codes for running PhyE2E including both training and test modules are accessible at https://github.com/Jie0618/PhysicsRegression with a permanent version available[73] via Zenodo at https://doi.org/10.5281/zenodo.16305086. The pre-trained PhyE2E can be downloaded at https://figshare.com/articles/dataset/PhyE2E_datas/29615831/1.

References

1. Cranmer, M. D. *Interpretable Machine Learning for the Physical Sciences*. Ph.D. thesis, Princeton University (2023).
2. Pearce Williams, L. Faraday's discovery of electromagnetic induction. *Contemporary Physics* **5**, 28–37 (1963).
3. Brunton, S. L., Proctor, J. L. & Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings national academy sciences* **113**, 3932–3937 (2016).
4. De Florio, M., Kevrekidis, I. G. & Karniadakis, G. E. Ai-lorenz: A physics-data-driven framework for black-box and gray-box identification of chaotic systems with symbolic regression. *Chaos, Solitons & Fractals* **188**, 115538 (2024).
5. Ahmadi Daryakenari, N., De Florio, M., Shukla, K. & Karniadakis, G. E. Ai-aristotle: A physics-informed framework for systems biology gray-box identification. *PLOS Computational Biology* **20**, e1011916 (2024).