RD^{2}BENCH: TOWARD DATA-CENTRIC AUTOMATIC R&D

Haotian Chen,[*] Xinjie Shen,[*] Zeqi Ye, Xiao Yang,[†] Xu Yang, Weiqing Liu, Jiang Bian
Microsoft Research Asia
{v-haotchen,v-xinjieshen,v-zeqiye,xiao.yang,xuyang1,
weiqing.liu,jiang.bian}@microsoft.com

ABSTRACT

The progress of humanity is driven by those successful discoveries accompanied by countless failed experiments. Researchers often seek the potential research directions by reading and then verifying them through experiments. The process imposes a significant burden on researchers. In the past decade, the data-driven black-box deep learning method demonstrates its effectiveness in a wide range of real-world scenarios, which exacerbates the experimental burden of researchers and thus renders the potential successful discoveries veiled. Therefore, automating such a research and development (R&D) process is an urgent need. In this paper, we serve as the first effort to formalize the goal by proposing a **R**eal-world **D**ata-centric automatic **R**&**D Bench**mark, namely RD^{2}Bench. RD^{2}Bench benchmarks all the operations in data-centric automatic R&D (D-CARD) as a whole to navigate future work toward our goal directly. We focus on evaluating the interaction and synergistic effects of various model capabilities and aiding in selecting well-performing trustworthy models. Although RD^{2}Bench is very challenging to the state-of-the-art (SOTA) large language model (LLM) named GPT-4, indicating ample research opportunities and more research efforts, LLMs possess promising potential to bring more significant development to D-CARD: They are able to implement some simple methods without adopting any additional techniques. We appeal to future work to take developing techniques for tackling automatic R&D into consideration, thus bringing the opportunities of the potential revolutionary upgrade to human productivity.

1 INTRODUCTION

"I have not failed. I've just found 10,000 ways that won't work."

— Thomas Alva Edison

The advancement of human society and the enhancement of living standards are highly correlated with the development of technology (Smith, 1937; Ranis & Fei, 1961; Perez, 2003; Brynjolfsson & McAfee, 2014). Numerous truths and principles remain undiscovered in the world, awaiting experimental exploration (Shapere, 1964; Popper, 2005). Those few successful discoveries, accompanied by countless failed experiments, propel the frontiers of technology. Historically, scientific researchers, including Edison, have undertaken extensive experiments by conducting them manually. In the age of AI, the influence of data-driven solutions, such as machine learning (ML) systems, is rapidly expanding (Mikolov et al., 2013; Devlin et al., 2018; OpenAI, 2023b). These systems are known for their robust fitting capabilities and their "black box" nature, which significantly increases the experimental load on researchers and hinders the process of identifying and validating effective methodologies. This paper concentrates on this critical scenario, which we refer to as *Data-Centric Research and Development (R&D)*. To cope with the prohibitively expensive costs and the overwhelming volume of experiments required, we consider automating such an R&D process for

[*]Equally Contributed
[†]Corresponding Author

higher research efficiency by leveraging the strong language understanding and programming ability of the state-of-the-art (SOTA) large language models (LLMs) (Srivastava et al., 2023). The brief illustration of **D**ata-**C**entric **A**utomatic **R**&**D** (D-CARD) is shown in Figure 1.

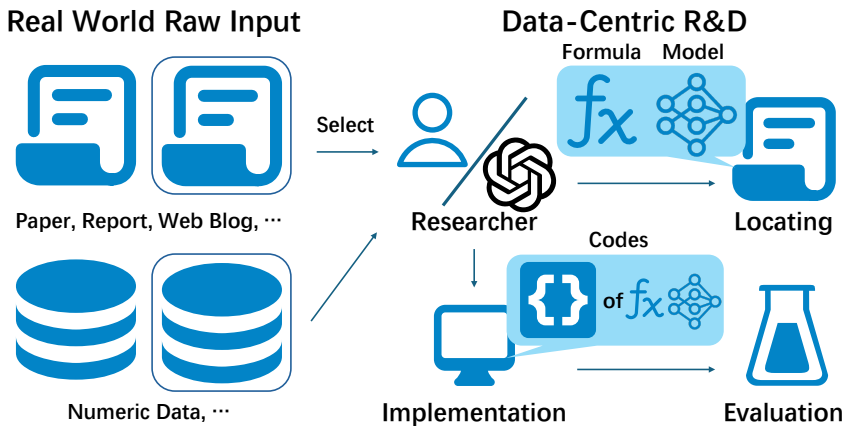

Figure 1: An overview of the R&D process. Researchers read papers and reports to extract the implementable methods (usually formulated as mathematical formulas or model architectures) for seeking potential research directions. Then, they correctly implement the methods to obtain the results for further analysis and development.

The first step towards automatic R&D is to formalize the task and provide a benchmark for identifying the potential effective methods and research directions. Intuitively, an outstanding methodology identified by the benchmark should possess (1) strong **language understanding ability** to identify the implementable methods or ideas (e.g., formulations and models) in the given raw information (e.g., papers, reports, websites, etc.) and (2) strong **implementation ability** to accurately implement the methods by programming and then obtain reliable experimental results. Previous work focuses on benchmarking the different aspects of the two abilities. Specifically, the language understanding ability of LLMs is partly evaluated through analyzing their performance on relation extraction (Wadhwa et al., 2023), question answering (Zhuang et al., 2023), and other natural language processing (NLP) tasks (Qin et al., 2023a). Meanwhile, the implementation ability of LLMs is partly tested through the benchmarks like SWE-Bench (Jimenez et al., 2023b), ToolBench (Qin et al., 2023c), ML-Bench (Liu et al., 2023b) and MetaTool (Anonymous, 2024), which study their ability of solving GitHub issues, using tools to program, and determining whether to use tools in a given scenario.

In this paper, we serve as the first effort to investigate the capabilities of the SOTA LLMs in tackling automatic R&D and propose a **R**eal-world **D**ata-centric automatic **R**&**D Bench**mark (RD^{2}Bench). The scenario studied by RD^{2}Bench possesses two unique and distinct characteristics that fundamentally differentiate it from others. First, RD^{2}Bench focuses on studying the real-world scenario where all the operations in R&D are automatic and evaluated as a whole, thus navigating the related future research efforts toward the goal of developing human technology more effectively. The real-world scenario requires more comprehensive and advanced model capabilities and exhibits new challenges. Second, we study the real-world automatic R&D in data-centric settings to navigate future work toward the urgent experimental exploration need brought by black-box data-driven models. Compared with existing benchmarks, RD^{2}Bench possesses two significant advantages:

(1) **RD^{2}Bench evaluates the interaction and synergistic effects of various model capabilities** instead of focusing on a single aspect of ability, which not only captures the frontier of SOTA LLMs but also bridges the gap between studying "individual ability" and "real-world synergistic effects of abilities". In automatic R&D, an ML system fails to complete the task even if it possesses both the strong information extraction ability and the strong programming or tool-using ability: While it succeeds in extracting methods and implementing them, it fails in selecting the appropriate data from the datasets or misunderstanding either the descriptions of data features or the requirements expressed by prompts. Additionally, exhaustively enumerating all the aspects for benchmarking is extremely challenging, which is overcame by RD^{2}Bench.

(2) **RD^{2}Bench tends to select well-performing trustworthy models** instead of those models that fail to learn rationales and causality yet possess outstanding performance. Specifically, ML systems easily achieve SOTA performance on previous benchmarks by shortcut learning or learning spurious correlations instead of learning rationales or causality (Mudrakarta et al., 2018; Geirhos et al., 2020; Cui & Athey, 2022; Wang et al., 2022; Chen et al., 2023). This renders a benchmark ineffective and misleading as it fails to accurately identify the well-performing trustworthy methods. For example, an ML system achieves SOTA performance on dog classification by merely recognizing grass (Zhang et al., 2021). RD^{2}Bench, on the contrary, eliminates such models by its high difficulty and large scope. The decision rules of models have to simultaneously satisfy at least four major requirements: (1) accurately and comprehensively extracting the implementable methods; (2) precisely selecting the method-specific data for computation; (3) correctly writing the code according to the logic expressed by methods and prompts; (4) successfully storing the correct results in a predefined format. Therefore, the decision rules of models selected by this benchmark are stable (work well in various situations), and thus getting closer to rationales and causality (Cui & Athey, 2022).

We evaluate the existing SOTA LLMs on RD^{2}Bench to expose the bottleneck of them and characterize the future research direction. RD^{2}Bench reveals new insights: (1) Among the popular LLMs, GPT-4 exhibits promising potency in dealing with the D-CARD task; (2) Detailed information of data descriptions significantly improves the performance of GPT-4; (3) The ability to query domain-specific knowledge is a basic requirement of D-CARD methods; (4) The more complex the method is, the more unstable the model performance is.

2 RD^{2}BENCH

Overall, our benchmark focuses on evaluating the finally implemented results according to the given raw information (e.g., papers, reports, websites, etc.). Moreover, we also provide human-annotated ground-truth information corresponding to the intermediate steps for debugging and more comprehensive evaluation, which not only aids in selecting the most trustworthy models by tracing back model operations in each step but also facilitates the research efficiency by clearly locating the problems. RD^{2}Bench selects those well-performing trustworthy models that are able to accurately calculate the final results with precise operation in each step. We introduce the details of our proposed RD^{2}Bench in the following sections. In section 2.1, we introduce the details of human-annotated ground truth and how we collect data to form RD^{2}Bench. Then, we elaborate on the two necessary steps, namely method extraction and method implementation, to perform R&D in section 2.2 and section 2.3. Finally, we detail our adopted metrics in section 2.4.

2.1 DATA COLLECTION

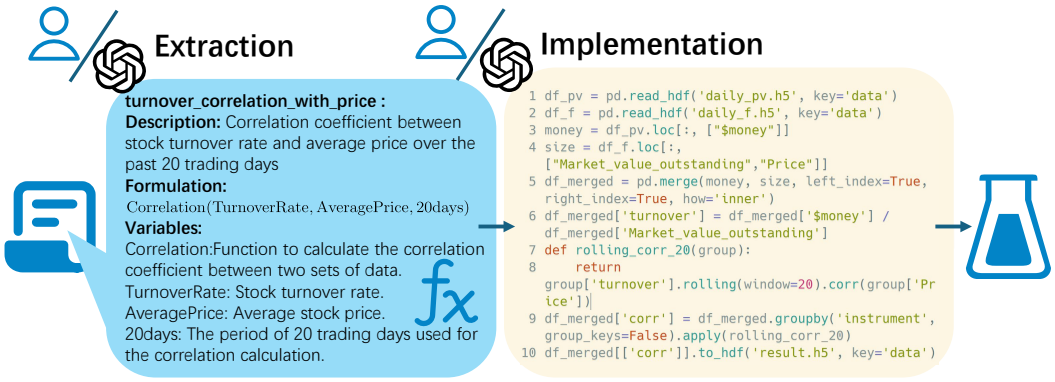

Figure 2: An example of formula implementation task.

As mentioned in the previous section, we are aiming to study the data-centric automatic R&D tasks and take formula implementation and model architecture implementation as the two major tasks included in this benchmark.

For formula implementation, we collect finance reports and stock trading data as real-world input. More specifically, the formula implementation task is to implement the factors function (usually mathematical formulas that take complex numeric input data about stock, company, and market as input and output a series of values with the time series) proposed in the report. To be more comprehensive, we include 11 factors from three major categories (fundamental, price-volume, high-frequency, denoted as Data I, II, III in following) and four difficulty levels (easy, medium, hard, and new factor). The difficulty level is manually labeled by domain experts according to the complexity of implementation (code implementation and idea understanding) and the novelty of the task. All ground truth implementations and formulas of the factors function are provided by us in the dataset. With the help of domain experts, the ground truth implementations and formulas are reliable and clear. One example of the formula implementation task is shown in Figure 2.

For model architecture implementation, we collect papers including (Gravina et al., 2023; Rossi et al., 2023; Rampášek et al., 2022; Lim et al., 2021; Yang et al., 2023a; Wang et al., 2024) and corresponding ground truth implementation codes using pytorch (Paszke et al., 2019) and torch_gemometric framework (Fey & Lenssen, 2019) in deep learning, more specifically, in graph representation learning field. The model architecture implementation task is to implement the new model architecture or layer proposed in the paper. We also manually label the difficulty level (easy, medium, hard) of the task based on the complexity of implementation (computational graphs and tensor operations) and the novelty of the task (brand new mechanism or new structure). All the ground truth implementations are provided by excellent researchers and engineers, which are reliable and truthful for evaluation. See the A.2 for more details about the dataset and the task.

2.2 METHOD EXTRACTION

This section evaluates the model's ability to recognize and extract information from formulas and to innovate within an automated R&D context. The model must identify actionable methods from extensive research data and extract all necessary information to realize these methods, which is critical for the subsequent code implementation.

This initial benchmark session tests the model's capability to accurately and comprehensively extract all conditions required for the methods it analyzes, and to develop the corresponding code implementation. The model should not proceed with incomplete methods but must fully understand and code for complete ones. We specify a format for information extraction that aligns with a manually labeled format or an appendix. We measure the model's extraction accuracy using the F1 score, considering both entity recognition and mention. For description accuracy, we compare the model's output with a manually annotated template, aiming for over 90% similarity. Developers can manually review this for precise evaluation. The model may encounter methods in the source material that are mentioned by name without detailed formulas or definitions. In such cases, the model can opt not to extract information or infer details based on the material's semantics. We encourage the latter approach to foster creativity and support future research that may introduce new methods and formulas. Currently, only methods explicitly named in the material are subject to this creative review, but future benchmarks may assess the model's ability to introduce entirely new concepts without prior mentions.

As most of our input data is encoded in the form of document files, we first use parsing tools to extract text content from files. Azure document intelligence API (4.0) is used for parsing reports and academic papers in PDF format.

2.3 METHOD IMPLEMENTATION

In this section, we evaluate the performance of LLM in the implementation of methods. Given all the necessary conditions provided to the model after the previous step, the model needs to select the necessary data and write code from scratch to implement the method with an informative and well-organized prompt. Details of the prompt are included in the dataset, which is also shown in A.3. We encourage model use of Python and popular data analysis and machine learning libraries.

2.4 METRICS

The goal of the benchmark is to evaluate the performance of LLM in data-centric R&D tasks, correctness and efficiency are the two major aspects we care about. To be more comprehensive and reveal more signals of performance, we propose multiple metrics for each task and each stage of the task. For formula implementation, we care about the average and maxima "running success rate", "format success rate", "Pearson correlation" and "value accuracy" across multiple independent attempts. To be simple, "avg.", "exe.", "form.", "corr.", "acc." denotes the average value, number of successful execution times, number of the matched result formats, the correlation, and the accuracy of corresponding values, respectively. See more details about the metrics calculation details in App A.1.

For model architecture implementation, we believe a successful implementation of a model should be consistent with the ground truth implementation as the model be viewed as a numeric function and combination of tensor transformations. Therefore, we propose these two metrics for the model architecture implementation task: tensor shape consistency rate (tsc.), tensor value consistency rate (tvc.). To be more specific, for each layer of the model, we calculate the consistency rate of the tensor shape and tensor value between the ground truth implementation and the implementation generated by the LLM. All the ground truth tensor value is determined by ground truth implementation codes with random Gaussian noise. Therefore, the formula for the two metrics is as follows, where S^i_{shape} and S^i_{value} are the consistency rate of tensor shape and tensor value in layer i, respectively, and d_i is the maximum length of the two tensors as the two tensors are $\mathbf{Z}_i$ and $\mathbf{Z}^*_i$, the ground truth and the generated tensor, respectively:

$$
\begin{aligned}
S^i_{\text{shape}}(\mathbf{Z}_i, \mathbf{Z}^*_i) &= \left(1 + \exp\left(\frac{\sum_{j=1}^{d} |\text{dim}(\mathbf{Z}_i)_j - \text{dim}(\mathbf{Z}^*_i)_j|}{d}\right)\right)^{-1}, \\
S^i_{\text{value}}(\mathbf{Z}_i, \mathbf{Z}^*_i) &= \left(1 + \exp\left(\frac{\sum_{j=1}^{d} |\mathbf{Z}^{(j)}_i - \mathbf{Z}^{*(j)}_i|}{d}\right)\right)^{-1}, \\
d &= \max(\text{len}(\text{dim}(\mathbf{Z}_i)), \text{len}(\text{dim}(\mathbf{Z}^*_i))),
\end{aligned}
\tag{1}
$$

while the shorter tensor is padded with zeros to match the length of the longer tensor. As the final score of the two metrics, we use the weighted sum of the consistency rate of all layers, weight increases with the depth of the layer and is summed as one.

$$
S_{\text{final}} = \frac{\sum_{i=1}^{n} S^i \cdot \gamma^i}{\sum_{i=1}^{n} \gamma^i}, \tag{2}
$$

where n is the number of layers in the model, γ is a tunable hyperparameter to control the weight increase, and we set $\gamma = 1.1$ in our experiments.

An example of the calculation is shown in Figure 3, using model LinkX (Lim et al., 2021) as an example. Meanwhile, we also include "average running success rate" as the basic metric for the model architecture implementation task, which is the same as the formula implementation task.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETTINGS

As we have numeric input and output in R&D tasks, we set numeric equability with 1e-6 as tolerance for the evaluation of the implementation of methods. We set the base models as GPT-4-turbo (OpenAI, 2023a), GPT-4-32k (OpenAI, 2023a), GPT-35-turbo-16k (OpenAI, 2023a) and Llama2 (Touvron et al., 2023) for the experiments. All the methods mentioned above, and their corresponding results are executed with Azure OpenAI API. There is no external data, resources, human feedback, or internet access involved in the experiments. We perform 20 independent attempts for each model and calculate the average and maximum value of each metric.

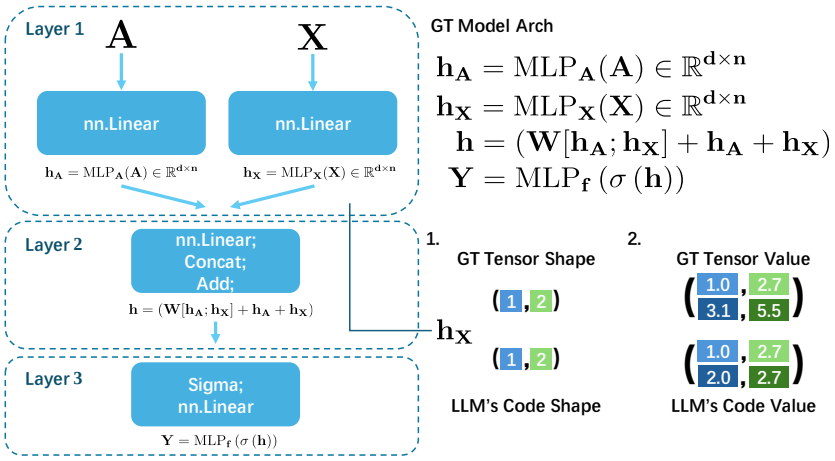

Figure 3: An example of metrics calculation for model architecture implementation task.

3.2 Results of Formula Implementation

In this section, we compare the performance of different models in the model architecture implementation task. We use the proposed metrics to evaluate the performance of the models. The results are shown in Table 1, Table 2, and Table 3. We observe that the GPT-4-turbo and GPT-4-32k achieve better performance than GPT-35-turbo-16k and LLaMa-2-70b in the model architecture implementation task. Overall experimental results indicate ample room for further research and the difficulty of the task and the challenges in automating R&D tasks. Specifically, we obtain the following four major findings revealed by the experimental results.

Category	Difficulty	Method Name	avg. exe.	avg. form.	avg. corr.	max. corr.	max. acc.	avg. acc.
Data I	simple	PEG	90.00%	0.00%	None	None	None	None
	medium	Turnover_STD_1M	80.00%	60.00%	0.314	0.839	0.001	0.000
	hard	turnover_correlation_with_price	20.00%	10.00%	0.371	0.936	0.000	0.000
	discovery	Liquidity_Factor	90.00%	55.00%	None	None	None	None
Data II	simple	One_Month_Volatility	75.00%	45.00%	0.770	1.000	0.000	0.000
	medium	Vol20	90.00%	15.00%	0.566	1.000	0.000	0.0
	hard	Alpha#70	15.00%	0.00%	None	None	None	None
Data III	simple	DailyRDvar	20.00%	0.00%	None	None	None	0.005
	medium	AdjRDvar	65.00%	0.00%	None	None	None	0.003
	hard	AdjRDskew	40.00%	5.00%	NaN	None	None	0.000
	discovery	minute_pv_corr	35.00%	5.00%	0.915	0.947	0.744	0.248

Table 1: The performance of GPT-4-turbo on formula implementation task.

Category	Difficulty	Method Name	avg. exe.	avg. form.	avg. corr.	max. corr.	max. acc.	avg. acc.
Data I	simple	PEG	75.00%	0.00%	None	None	None	None
	medium	Turnover_STD_1M	75.00%	75.00%	0.194	0.243	0.000	0.000
	hard	turnover_correlation_with_price	25.00%	15.00%	0.123	0.193	0.000	0.000
	discovery	Liquidity_Factor	85.00%	50.00%	None	None	None	None
Data II	simple	One_Month_Volatility	60.00%	40.00%	0.474	1.000	0.000	0.000
	medium	Vol20	20.00%	20.00%	0.527	1.000	0.000	0.000
	hard	Alpha#70	20.00%	0.00%	None	None	None	None
Data III	simple	DailyRDvar	35.00%	5.00%	None	None	0.008	0.001
	medium	AdjRDvar	35.00%	0.00%	None	None	0.001	0.000
	hard	AdjRDskew	40.00%	5.00%	NaN	None	None	0.000
	discovery	minute_pv_corr	20.00%	0.00%	0.434	0.858	0.002	0.001

Table 2: The performance of GPT-4-32k on formula implementation task.

LLM agents hold promising potential to tackle D-CARD. We can observe from Table 1 and Table 2 that GPT-4 possesses the ability to tackle some simple D-CARD cases without adopting any additional techniques. Specifically, GPT-4 achieves a high (more than 0.8) maximum correlation coefficient with the ground-truth results in implementing both simple and medium formulations. Especially, both GPT-4-turbo and GPT-4-32k achieve the maximum correlation value in implement-

ing simple One_Month_Volatility and Vol20, respectively. However, GPT-4 fails to precisely match the exact ground-truth values due to some minor mistakes, such as missing the domain common knowledge (e.g., using percent change rather than using difference when calculating the growth), mismatching the output format, and unnecessarily introducing additional computational operations.

Precisely understanding and selecting data requires more detailed data information in D-CARD. As shown in Table 1 and Table 2, we observe a special situation where GPT-4 significantly fails to implement a simple formulation while succeeds in implementing the harder ones. After analyzing its generated code, we find that GPT-4 confuses the different semantic meanings of data features due to their close natural language descriptions, which renders the subsequent calculation ineffective. For example, GPT-4 confuses the two terms named "volume" and "volatility" and always opts to use "volume" data when "volatility" is required. If we manually improve our initial prompt by adding a more detailed description, GPT-4 succeeds in understanding the semantic difference and obtains over 99% performance in the accuracy of values.

The ability to query domain-specific knowledge is a basic requirement of D-CARD methods. As we mentioned in the first finding, missing domain common knowledge impedes GPT-4 from calculating the precisely matched final results. Additionally, we find that the implementation of some operations in a formulation also requires domain-specific knowledge. For example, in the financial domain, it's clear enough for the financial practitioners to implement the operation named "IndNeutralize(x,g)" by merely giving the description "x cross-sectionally neutralized against groups g". However, in the code generated by GPT-4, it defines a function named "IndNeutralize(series, industry)" and leaves its content blank by merely adding a notation "Please replace this with your actual function definition".

The more complex the method is, the more unstable the model performance is. As shown in the columns of Table 1 and Table 2 named "avg. exe.", "avg. form.", and "avg. corr.", respectively, we can observe that the performance variance of GPT-4 is significantly higher with the complexity of formulations getting higher. In 20 times of execution, GPT-4 generates the successfully executed code in 18 times when implementing the medium Vol20 while only three times in implementing hard Alpha#70.

Category	**Difficulty**	**Method Name**	GPT-35-turbo-16k		LLaMa-2-70b	
			avg. exe.	**avg. form.**	**avg. exe.**	**avg. form.**
Data I	simple	PEG	0.00%	0.00%	0.00%	0.00%
	medium	Turnover_STD_1M	0.00%	0.00%	0.00%	0.00%
	hard	turnover_correlation_with_price	0.00%	0.00%	0.00%	0.00%
	discovery	Liquidity_Factor	0.00%	0.00%	0.00%	0.00%
Data II	simple	One_Month_Volatility	0.00%	0.00%	0.00%	0.00%
	medium	Vol20	0.00%	0.00%	0.00%	0.00%
	hard	Alpha#70	0.00%	0.00%	0.00%	0.00%
	simple	DailyRDvar	0.00%	0.00%	0.00%	0.00%
Data III	medium	AdjRDvar	0.00%	0.00%	0.00%	0.00%
	hard	AdjRDskew	5.00%	0.00%	0.00%	0.00%
	discovery	minute_pv_corr	5.00%	0.00%	0.00%	0.00%

Table 3: The performance of GPT-35-turbo-16k and LLaMa-2-70b on formula implementation task.

As shown in Table 3, the performance of GPT-35 and LLaMa2 is poor, even failing in execution codes. However, GPT-4 models shown in Table 1 and 2 have a much better performance. This indicates that the performance of the model in the data-centric R&D task is highly related to the model's pre-training and capacity. Therefore, we posit that continually training and improving the foundation model is a promising direction for future research in the field of data-centric R&D tasks.

3.3 Results of Model Architecture Implementation

In this section, we compare the performance of different LLMs in the model architecture implementation task and summarize the results in Table 4 and 5. As shown in the table, we can see the GPT-4-turbo, GPT-35-turbo-16k, and GPT-4-32K have similar running success rates, but differ variously in tvc. and tsc.. The LLaMa-2-70b has the lowest running success rate and other metrics. Notice that even though a significant gap still exists between GPT-35, LLaMa-2, and GPT-4,

it is much smaller than the gap in the formula implementation task. The overall running success rates are also higher than formula implementation task. We can conclude that we can have similar observations in the model architecture implementation task as in the formula implementation task.

Model Name	**Difficulty**	GPT-4-turbo					GPT-4-32K				
		avg. exe.	**avg. tsc.**	**avg. tvc.**	**max. tsc.**	**max. tvc.**	**avg. exe.**	**avg. tsc.**	**avg. tvc.**	**max. tsc.**	**max. tvc.**
PMLP	Easy	100.00%	1.00	1.00	1.00	1.00	100.00%	1.00	1.00	1.00	1.00
LinkX	Easy	100.00%	1.00	0.85	1.00	1.00	100.00%	0.90	0.90	1.00	1.00
VisNet	Hard	45.00%	0.29	0.09	0.37	0.49	45.00%	0.21	0.09	0.37	0.49
AntiSymmetric	Medium	80.00%	0.71	0.59	0.73	0.88	70.00%	0.56	0.66	0.66	0.88
GPSConv	Medium	75.00%	0.56	0.62	0.65	1.00	75.00%	0.53	0.62	0.65	0.72
DirGNNConv	Medium	100.00%	0.80	0.68	0.86	0.94	90.00%	0.65	0.62	0.82	0.91

Table 4: The performance of GPT-4-turbo and GPT-4-32k on model architecture implementation task.

Model Name	**Difficulty**	GPT-35-turbo-16k					LLaMa-2-70b				
		avg. exe.	**avg. tsc.**	**avg. tvc.**	**max. tsc.**	**max. tvc.**	**avg. exe.**	**avg. tsc.**	**avg. tvc.**	**max. tsc.**	**max. tvc.**
PMLP	Easy	100.00%	0.75	0.75	1.00	1.00	60.00%	0.45	0.55	1.00	1.00
LinkX	Easy	100.00%	0.60	0.34	1.00	1.00	30.00%	0.20	0.15	1.00	1.00
VisNet	Hard	5.00%	0.03	0.00	0.16	0.40	0.00%	0.00	0.00	0.00	0.00
AntiSymmetric	Medium	45.00%	0.16	0.21	0.61	0.22	0.00%	0.00	0.00	0.00	0.00
GPSConv	Medium	45.00%	0.24	0.19	0.45	0.42	0.00%	0.00	0.00	0.00	0.00
DirGNNConv	Medium	65.00%	0.56	0.29	0.71	0.42	0.00%	0.00	0.00	0.00	0.00

Table 5: The performance of GPT-35-turbo-16k and LLaMa-2-70b on model architecture implementation task.

3.4 LIMITATIONS

We also strongly want to include as many as possible of current state-of-the-art evolving tricks and techniques in the benchmark in further works, which will lead to a more comprehensive evaluation and reveal more signals of the performance of the models. We have primarily tried to apply tricks like "self-refine/correct/reflection" and "automatic curriculum" to improve the performance of LLM agents in this benchmark, and we find that the improvement is marginal in the performance of the models. For more details and a more comprehensive evaluation, we will include them in not-far-future works. We believe that the benchmark will be a valuable tool for the community to evaluate the performance of the models in the data-centric R&D tasks and to develop new models and techniques to address the challenges and opportunities in the domain.

4 RELATE WORK

4.1 LLM AS AUTONOMOUS AGENT

In the past few years, LLM has made great achievements in both academia and industry (OpenAI, 2023a; Touvron et al., 2023), and has achieved results that surpass the previous level in a number of classic tasks (Zhao et al., 2023). Research has shown that with the growth of data volume and model size (Zoph et al., 2022), LLM has emerged with stronger reasoning and other capabilities (Ouyang et al., 2022). These capabilities enable LLM to exhibit certain agent-like behaviors in some tasks such as using or creating tools (Qin et al., 2023b; Qian et al., 2023), planning (Yao et al., 2023; Brown et al., 2020), and memory. Therefore, more and more researchers have expressed their expectations for its human-like and overall capabilities, and have made preliminary explorations of

it as an independent agent (Wang et al., 2023a; Shinn et al., 2023). Multi-agent collaboration (Wu et al., 2023; Li et al., 2023) is also introduced to LLM for better accuracy and generalizability. Moreover, for reducing human efforts and automatically exploring, previous work focuses on autonomous LLM agents for general purpose are purposed (Yang et al., 2023b; Shen et al., 2023). Positive views further believe that the realization of AGI may come from the evolution of autonomous LLM and some inspiring examples have been released (Penov et al., 2024).

However, most research still focuses on limited scenarios that are given with clear and fixed questions and backgrounds. A recent work (Yang et al., 2023d) has attempted to introduce LLM to the R&D field and formalize the R&D process as a sequence of tasks. However, there is no easy-to-use benchmark for the community and current R&D tasks may be too general and can't reveal significant signals. In this work, we propose a benchmark for LLM in data-centric R&D tasks and provide a comprehensive evaluation.

4.2 Semi-Automatic R&D with Agents

Scientific research and development (R&D) is a time-consuming and important process. In the past, R&D has been mainly conducted by human researchers with countless failed experimental explorations and creative observation conclusions. Agents have been introduced to R&D to reduce human efforts and automatically explore. Recently, there have been attempts to partly automate R&D, including the automatic chemical synthesis planning (Boiko et al., 2023), automatic molecular design (Joshi & Kumar, 2021; Schneider, 2017; Boiko et al., 2023), automatic theorem proving (Wang et al., 2023b; Yang et al., 2023c). However, these attempts mainly focus on automatic searching for possible solutions and optimizations with symbolic representation (Lu et al., 2023) and heuristic techniques (Whalen, 2016), but less addressing long-horizon planning, implementation, and reasoning for the next step idea exploration. Moreover, the data-centric R&D tasks currently have not been explored in the community, and no benchmark has been proposed for the community. Previous works have applied LLM to real-world R&D tasks such as debugging issues (Tian et al., 2024; Jimenez et al., 2023a) or only focus on data-centric but not real-world R&D tasks (Liu et al., 2023a). In this work, we propose a benchmark for LLM in data-centric R&D tasks and evaluate the performance of LLMs.

5 Conclusion

In this paper, we serve as the first effort to tackle the real-world data-centric automatic R&D scenario in the hope of significantly improving the research efficiency of scientists and thus contributing to the revolution of human productivity. Specifically, we first propose RD^{2}Bench that benchmarks all the operations in D-CARD as a whole to navigate future work toward the ultimate goal of automating data-centric R&D directly. RD^{2}Bench focuses on evaluating the interaction and synergistic effects of various model capabilities and aiding in selecting the well-performing trustworthy models. Based on RD^{2}Bench, we find that although the most SOTA GPT-4 shows its promising potency in tackling D-CARD, there remains ample room for future work.

REFERENCES

Anonymous. MetaTool benchmark: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations*, 2024.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, December 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06792-0. URL http://dx.doi.org/10.1038/s41586-023-06792-0.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Erik Brynjolfsson and Andrew McAfee. *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. WW Norton & Company, 2014.

Haotian Chen, Bingsheng Chen, and Xiangdong Zhou. Did the Models Understand Documents? Benchmarking Models for Language Understanding in Document-Level Relation Extraction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6418–6435, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.354.

Peng Cui and Susan Athey. Stable Learning Establishes some Common Ground between Causal Inference and Machine Learning. *Nature Machine Intelligence*, 4(2):110–115, February 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00445-z.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2018.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut Learning in Deep Neural Networks. *Nature Machine Intelligence*, 2(11):665–673, November 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-00257-z.

Alessio Gravina, Davide Bacciu, and Claudio Gallicchio. Anti-symmetric DGN: a stable architecture for deep graph networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=J3Y7cgZOOS.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2023a. URL https://arxiv.org/abs/2310.06770.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*, 2023b.

Rajendra P. Joshi and Neeraj Kumar. Artificial intelligence for autonomous molecular design: A perspective. *Molecules*, 26(22):6761, November 2021. ISSN 1420-3049. doi: 10.3390/molecules26226761. URL http://dx.doi.org/10.3390/molecules26226761.

Huao Li, Yu Quan Chong, Simon Stepputtis, Joseph Campbell, Dana Hughes, Michael Lewis, and Katia P. Sycara. Theory of mind for multi-agent collaboration via large language models. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://api.semanticscholar.org/CorpusID:264172518.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021.

Yuliang Liu, Xiangru Tang, Zefan Cai, Junjie Lu, Yichi Zhang, Yanjun Shao, Zexuan Deng, Helan Hu, Zengxian Yang, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Zhengliang Li, Liang Chen, Yiming Zong, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and Mark Gerstein. Ml-bench: Large language models leverage open-source libraries for machine learning tasks, 2023a. URL https://arxiv.org/abs/2311.09835.

Yuliang Liu, Xiangru Tang, Zefan Cai, Junjie Lu, Yichi Zhang, Yanjun Shao, Zexuan Deng, Helan Hu, Zengxian Yang, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Zhengliang Li, Liang Chen, Yiming Zong, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and Mark Gerstein. ML-Bench: Large Language Models Leverage Open-source Libraries for Machine Learning Tasks, November 2023b.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. A survey of deep learning for mathematical reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.817. URL http://dx.doi.org/10.18653/v1/2023.acl-long.817.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. Did the Model Understand the Question? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1896–1906, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1176.

OpenAI. Gpt-4 technical report, 2023a. URL https://arxiv.org/abs/2303.08774.

OpenAI. GPT-4 Technical Report, March 2023b.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, March 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Franci Penov, Yohei Nakajima, Malik M Alnakhaleh, Alexander Dibrov, Shukri, Frank Chen, Anton Troynikov, David Byttow, John Cao, Felipe Schieber, Josh XT, FRM Minsu Yeom, CFA, Zain Hasan, zeel sheladiya, jmtatsch, Aidan Rauscher, Thiago Alves, jakvb, Jason Banich, Muhamed

AlGhzawi, Peter Banda, TungusSs, Lorenzo Fontoura, Joe Heitzeberg, Jay Scambler, Ikko Eltociear Ashimine, Cs4K1Sr4C, Mike Crawford, Michele Bellitti, and swyx.io. *yoheinakajima/babyagi*. 1 2024. URL https://github.com/yoheinakajima/babyagi.

Carlota Perez. *Technological Revolutions and Financial Capital*. Edward Elgar Publishing, 2003.

Karl Popper. *The Logic of Scientific Discovery*. Routledge, 2005.

Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.462. URL http://dx.doi.org/10.18653/v1/2023.findings-emnlp.462.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. Is ChatGPT a General-Purpose Natural Language Processing Task Solver?, February 2023a.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023b. URL https://arxiv.org/abs/2307.16789.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023c.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *Advances in Neural Information Processing Systems*, 35, 2022.

Gustav Ranis and John C. H. Fei. A theory of economic development. *The American Economic Review*, 51(4):533–565, 1961. ISSN 00028282.

Emanuele Rossi, Bertrand Charpentier, Francesco Di Giovanni, Fabrizio Frasca, Stephan Günnemann, and Michael Bronstein. Edge directionality improves learning on heterophilic graphs, 2023. URL https://arxiv.org/abs/2305.10498.

Gisbert Schneider. Automating drug discovery. *Nature Reviews Drug Discovery*, 17(2):97–113, December 2017. ISSN 1474-1784. doi: 10.1038/nrd.2017.232. URL http://dx.doi.org/10.1038/nrd.2017.232.

Dudley Shapere. The structure of scientific revolutions. *The Philosophical Review*, 73(3):383–394, 1964. ISSN 00318108, 15581470. doi: 10.2307/2183664.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, 2023. URL https://arxiv.org/abs/2303.17580.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Adam Smith. *The Wealth of Nations [1776]*, volume 11937. na, 1937.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, and Aditya Gupta. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Zhiyuan Liu, and Maosong Sun. Debugbench: Evaluating debugging capability of large language models, 2024. URL https://arxiv.org/abs/2401.04621.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, and Bhosale. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Somin Wadhwa, Silvio Amir, and Byron Wallace. Revisiting Relation Extraction in the era of Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15566–15589, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.868.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a. URL https://arxiv.org/abs/2305.16291.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, Jian Yin, Zhenguo Li, and Xiaodan Liang. Dt-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023b. doi: 10.18653/v1/2023.acl-long.706. URL http://dx.doi.org/10.18653/v1/2023.acl-long.706.

Tianlu Wang, Rohit Sridhar, Diyi Yang, and Xuezhi Wang. Identifying and mitigating spurious correlations for improving robustness in NLP models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1719–1729, Seattle, United States, July 2022. Association for Computational Linguistics.

Yusong Wang, Tong Wang, Shaoning Li, Xinheng He, Mingyu Li, Zun Wang, Nanning Zheng, Bin Shao, and Tie-Yan Liu. Enhancing geometric representations for molecules with equivariant vector-scalar interactive message passing. *Nature Communications*, 15(1), January 2024. ISSN 2041-1723. doi: 10.1038/s41467-023-43720-2. URL http://dx.doi.org/10.1038/s41467-023-43720-2.

Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic, 2016. URL https://arxiv.org/abs/1608.02644.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Chenxiao Yang, Qitian Wu, Jiahua Wang, and Junchi Yan. Graph neural networks are inherently good generalizers: Insights by bridging gnns and mlps. In *International Conference on Learning Representations (ICLR)*, 2023a.

Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions, 2023b. URL https://arxiv.org/abs/2306.02224.

Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023c. URL https://openreview.net/forum?id=g7OX2sOJtn.

Xu Yang, Xiao Yang, Weiqing Liu, Jinhui Li, Peng Yu, Zeqi Ye, and Jiang Bian. Leveraging large language model for automatic evolving of industrial data-centric r&d cycle, 2023d. URL https://arxiv.org/abs/2310.11249.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Xingxuan Zhang, Peng Cui, Renzhe Xu, Linjun Zhou, Yue He, and Zheyan Shen. Deep Stable Learning for Out-Of-Distribution Generalization. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5368–5378, Nashville, TN, USA, June 2021. IEEE. ISBN 978-1-66544-509-2. doi: 10.1109/CVPR46437.2021.00533.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023. URL https://arxiv.org/abs/2303.18223.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. ToolQA: A Dataset for LLM Question Answering with External Tools, June 2023.

Barret Zoph, Colin Raffel, Dale Schuurmans, Dani Yogatama, Denny Zhou, Don Metzler, Ed H. Chi, Jason Wei, Jeff Dean, Liam B. Fedus, Maarten Paul Bosma, Oriol Vinyals, Percy Liang, Sebastian Borgeaud, Tatsunori B. Hashimoto, and Yi Tay. Emergent abilities of large language models. *TMLR*, 2022.

A APPENDIX

A.1 FORMULA IMPLEMENTATION TASK METRICS CALCULATION DETAILS

As mentioned above, we have multiple metrics (the average and maxima score across multiple independent attempts, including "running success rate", "format success rate", "pearson correlation" and "value accuracy"). Assume the ground truth factor value is $\mathbf{Y}$ with length n (the length of the time series), and the generated factor value is $\mathbf{Y}^*$, the calculation of the metrics is as follows:

Running success is defined as successful execution. Any error occurs in python interpreter during the execution that stop the execution is considered as a failure. We calculate the ratio of the number of successful execution times to the total number of attempts, denoted as avg. exe.

Pearson correlation is the correlation between the ground truth factor value and the generated factor value.

$$\text{corr.} = \frac{\sum_{i=1}^{n}(\mathbf{Y}_i^* - \bar{\mathbf{Y}^*})(\mathbf{Y}_i - \bar{\mathbf{Y}})}{\sqrt{\sum_{i=1}^{n}(\mathbf{Y}_i^* - \bar{\mathbf{Y}^*})^2}\sqrt{\sum_{i=1}^{n}(\mathbf{Y}_i - \bar{\mathbf{Y}})^2}},$$

Format success is defined as successful format matching, which is the final output dataframe format is (datetime, factor_name). We calculate the ratio of the number of the matched result formats to the total number of attempts, denoted as avg. form.

Value accuracy is the accuracy of the generated factor value, which can be formulated as:

$$\text{acc.} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(|\mathbf{Y}_i^* - \mathbf{Y}_i| < t),$$

Please note that we set the tolerance t for the value accuracy as 1e-6 in this paper, which means two value is considered as equal if the absolute difference is less than 1e-6.

A.2 DATA COLLECTION DETAILS

As mentioned in the previous section, we collect papers including (Gravina et al., 2023; Rossi et al., 2023; Rampášek et al., 2022; Lim et al., 2021; Yang et al., 2023a; Wang et al., 2024) and corresponding codes using pyg (Fey & Lenssen, 2019), which are listed in the following table.

Paper	Type	Difficulty	GT Code
PMLP	Model	Easy	Link
LinkX	Model	Easy	Link
AntiSymmetric	Layer	Medium	Link
GPSConv	Layer	Medium	Link
DirGNNCOnv	Layer	Medium	Link
VisNet	Model	Hard	Link

Table 6: Papers and corresponding ground truth implementation codes for the model architecture implementation task

A.3 PROMPTS

The prompt for the model architecture implementation task is as follows:

```
The user is trying to implement some factors in quant investment,
    and you are the one to help write the python code.
The user will provide the source data in HDF5(H5) format which you
     can load using pandas.read_hdf. The file is located near your
     python code file which you can read from "./source_data.h5".
    After that, you will get a pandas dataframe with the following
     format:
```

```
open,close,high,low,volume,vwap,cap,IndClass.industry IndClass.
    sector,returns,date,instruments
2020-01-02,SH600000
    ,158.538132,158.538132,160.699432,158.283859,4060945.0,
159.431900,647446144.0,1.0,NaN
The explanation to the example column names:
1: returns: daily close-to-close returns
2: open, close, high, low, volume: standard definitions for daily
    price and volume data
3: vwap: daily volume-weighted average price
4: cap: market capitalization is the total value of a company's
    outstanding shares of stock
5: IndClass.industry and IndClass.sector: a generic placeholder
    for a binary industry classification such as GICS, BICS, NAICS
    , SIC, etc., in indneutralize(x, IndClass.level), where level:
     sector, industry, etc. Multiple IndClass in the same alpha
    need not correspond to the same industry classification.

The user will provide you a formulation of the factor, which
    contains some function calls and some operators. You need to
    implement the function calls and operators in python. Your
    code is expected to align the formulation in any form which
    means The user needs to get the exact factor values with your
    code as expected.

Your code should contain the following part: the import part, the
    function part, and the main part. You should write a main
    function name: "calculate_{function_name}" and call this
    function in "if __name__ == __main__" part. Don't write any
    try-except block in your code. The user will catch the
    exception message and provide the feedback to you.

User will write your code into a python file and execute the file
    directly with "python {your_file_name}.py". You should
    calculate the factor values and save the result into a HDF5(H5
    ) file named "result.h5" in the same directory as your python
    file. The result file is a HDF5(H5) file containing a pandas
    dataframe. The index of the dataframe is the date and
    instrument, and the single column name is the factor name,and
    the value is the factor value. The result file should be saved
     in the same directory as your python file.

To help you write the correct code, the user might provide
    multiple information that helps you write the correct code:
1. The user might provide you the correct code to similar factors.
     Your should learn from these code to write the correct code.
2. The user might provide you the failed former code and the
    corresponding feedback to the code. The feedback contains to
    the execution, the code and the factor value. You should
    analyze the feedback and try to correct the latest code.
3. The user might provide you the suggestion to the latest fail
    code and some similar fail to correct pairs. Each pair
    contains the fail code with similar error and the
    corresponding corrected version code. You should learn from
    these suggestion to write the correct code.

Please response the code in the following json format. Here is an
    example structure for the JSON output:
{
```

```
    "code": "The Python code as a string."
}
```

The prompt for the model architecture implementation task is as follows:

```
The user is trying to implement some models or layers in deep
    learning, specifically graph learning area, and you are the
    one to help write the python code.

Use pytorch and pyg (torch_geometric) framework to implement it.
    You can assume the input will contains node feature X [
    num_nodes, feature_dim], edge_index [2, num_edges],
    edge_feature [num_edges, num_edge_features], y [num_nodes, *]
    when is node-level targets or graph-level targets of shape
    [1, *], pos (node position matrix) [num_nodes, position_dim].

The user will provide you a formulation of the model/ layer. You
    need to implement it in python.

Your code should contain the following part: the import part, the
    function part, and the main part. You should write a main
    function name: "calculate_{function_name}" and call this
    function in "if __name__ == __main__" part. Don't write any
    try-except block in your code. The user will catch the
    exception message and provide the feedback to you.

User will write your code into a python file and execute the file
    directly with "python {your_file_name}.py".

Please response the code in the following json format. Here is an
    example structure for the JSON output:
{
    "code": "The Python code as a string."
}
```