

DOLPHIN: Moving Towards Closed-loop Auto-research through Thinking, Practice, and Feedback

Jiakang Yuan^{♣,♠}, Xiangchao Yan[♠], Bo Zhang^{♠,‡,✉}, Tao Chen^{♠,✉},
Botian Shi[♠], Wanli Ouyang[♠], Yu Qiao[♠], Lei Bai^{♠,✉}, Bowen Zhou[♠]

♣Fudan University, ♠Shanghai Artificial Intelligence Laboratory

 <https://alpha-innovator.github.io/Dolphin-project-page/>
 <https://github.com/Alpha-Innovator/Dolphin>

Abstract

The scientific research paradigm is undergoing a profound transformation owing to the development of Artificial Intelligence (AI). Recent works demonstrate that various AI-assisted research methods can largely improve research efficiency by improving data analysis, accelerating computation, and fostering novel idea generation. To further move towards the ultimate goal (*i.e.*, automatic scientific research), in this paper, we introduce DOLPHIN, a closed-loop LLM-driven framework to enhance the automation level of scientific research. DOLPHIN first generates novel ideas based on feedback from previous experiments and relevant papers ranked by the topic and task attributes. Then, the generated ideas can be implemented using a code template refined and debugged with the designed exception-traceback-guided local code structure. Finally, DOLPHIN automatically analyzes the results of each idea and feeds the results back to the next round of idea generation. Experiments are conducted on the benchmark datasets of different topics and a subset of MLE-bench. Results show that DOLPHIN can continuously improve the performance of the input topic in a loop. We highlight that DOLPHIN can automatically propose methods that are **comparable to the state-of-the-art** in some tasks such as 3D point classification.

1 Introduction

The rapid evolution of AI (Achiam et al., 2023; Anthropic, 2024; Yang et al., 2024a; Dubey et al., 2024) has profoundly transformed various fields (Yao et al., 2024; Yan et al., 2025; Luo et al., 2022), accelerating scientific research processes like scientific data collection and processing (Han et al., 2024; Yan et al., 2025), computation (Jumper et al., 2021; Chen et al., 2023), and innovation (Qi et al.,

✉ Corresponding Authors, ‡ Project Lead

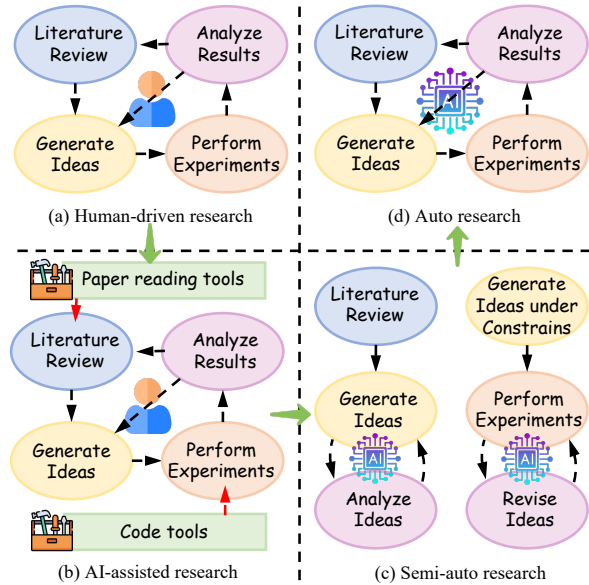


Figure 1: Comparisons of the four stages in the evolutionary trajectory towards auto-research including (a) Entirely human-driven research, (b) AI-assisted research, (c) Semi-automatic research, and (d) Auto-research.

2024). Under this trend, the research paradigm is shifting from completely human-driven research to AI-assisted research (Assafelovic, 2023). More recently, the continuous upgrading of LLMs has promoted the evolution of AI-assisted research to automatic scientific research (Lu et al., 2024; Si et al., 2024).

The evolutionary trajectory from human-driven research to automatic research consists of four stages as shown in Fig. 1. **1) The entirely human-driven stage** requires humans to complete all aspects including idea generation and experiments. **2) In the AI-assisted research stage**, researchers use LLMs-based tools (Assafelovic, 2023; Team, 2023a) to improve the research efficiency. For example, GPT-researcher (Assafelovic, 2023) can help us to decompose complex tasks and generate research reports using LLMs. **3) The semi-automatic research stage** enables automation in

certain processes of scientific research. For example, recent works (Si et al., 2024; Li et al., 2024; Su et al., 2024) leverage LLMs to automatically generate ideas for different topics by drawing on relevant works. 4) The fourth stage of AI-promoting scientific research is *auto-research stage*, where AI automatically handles the entire research process from conception to completion. Recently, AI-Scientist (Lu et al., 2024) introduced an open automatic research framework automating key tasks including idea generation, experimental verification, and academic paper writing.

Despite the encouraging progress made in existing works, auto-research still faces key challenges:

First, accurately assessing the effectiveness of AI-generated ideas remains a major hurdle. Most studies (Si et al., 2024; Li et al., 2024; Su et al., 2024) rely on either human evaluation or LLMs to evaluate the quality of generated ideas. However, *merely focusing on the novelty of an idea itself does not adequately reflect its effectiveness in experimental validation*. While works like AI-Scientist (Lu et al., 2024) incorporate experimental validation, they use simple, the self-constructed datasets, limiting meaningful comparisons with existing methods in the same field.

Second, another limitation of prior works (Si et al., 2024; Li et al., 2024; Su et al., 2024) lies in the absence of a feedback mechanism between the experimental validation and the idea generation – a process that is fundamental to human research. Human researchers refine their ideas and approaches iteratively based on experimental outcomes, which serves as a crucial pathway for improving the quality of research ideas.

To address these challenges and facilitate further progress towards automatic scientific research, in this work, we propose DOLPHIN, a closed-loop auto-research framework which is composed of three key stages in the research cycle (*i.e.*, idea generation, experimental verification, and results feedback). In each research loop, DOLPHIN first generates ideas based on the previous experimental results and the retrieved relevant papers which are filtered by judging the topic relevance and task attribute relevance between the retrieved papers and the input topic. Then, the proposed idea can be further implemented using a code template refined and debugged by the designed traceback-guided debugging module, which analyzes the local code structure related to the error-traceback to streamline the debugging process. Finally, DOLPHIN au-

tomatically analyzes the outcomes of successfully executed experiments, providing guidance for the next iteration of idea generation.

To experimentally validate the generated scientific ideas, we selected commonly-used public benchmarks across different tasks and modalities such as ModelNet40 (Wu et al., 2015), CIFAR-100 (Krizhevsky et al., 2009), and SST-2 (Socher et al., 2013). The results demonstrate that DOLPHIN generates ideas that outperform the selected baselines like PointNet (Qi et al., 2017a), WideResNet (He et al., 2016), and BERT-base (Devlin et al., 2019). We were **supervised** to observe that DOLPHIN is capable of proposing more concise method implementations while surpassing the performance of current human-designed state-of-the-art approaches in certain fields, as reported in Tab. 8. Besides, experiments on tasks from MLE-bench reveal that DOLPHIN can integrate with code generation pipelines like AIDE and further support version updates for technologies or code.

Our contribution can be summarized as follows:

- We propose DOLPHIN, a closed-loop auto-research framework covering three key stages in the research cycle, including generating ideas, performing experiments, and feedback.
- To improve the efficiency of auto-research, we propose task-attribute-guided paper ranking and exception-traceback-guided debugging process to improve the quality of generated ideas and the successful rate of code execution, respectively.
- Experimental results on benchmark datasets show that DOLPHIN can generate high-quality ideas and conduct validations in the loop.

2 Related Works

2.1 Open-ended Scientific Research

Recent studies (Qi et al., 2023; Wang et al., 2024b; Yang et al., 2024b; Qi et al., 2024; Lu et al., 2024; Hu et al., 2024) have demonstrated that Large Language Models (LLMs) have the potential to generate novel research ideas, a finding that has sparked widespread discussion in academia. Li et al. (2024) generate ideas based on the analysis of the paper chain. Si et al. (2024) find that LLM-generated ideas are innovative but their feasibility needs improvement. Some works (Wang et al., 2024b; Hu et al., 2024) improve the novelty of ideas using iterative optimization strategies. In addition, several works (Qiu et al., 2023; Zhou et al., 2024) try

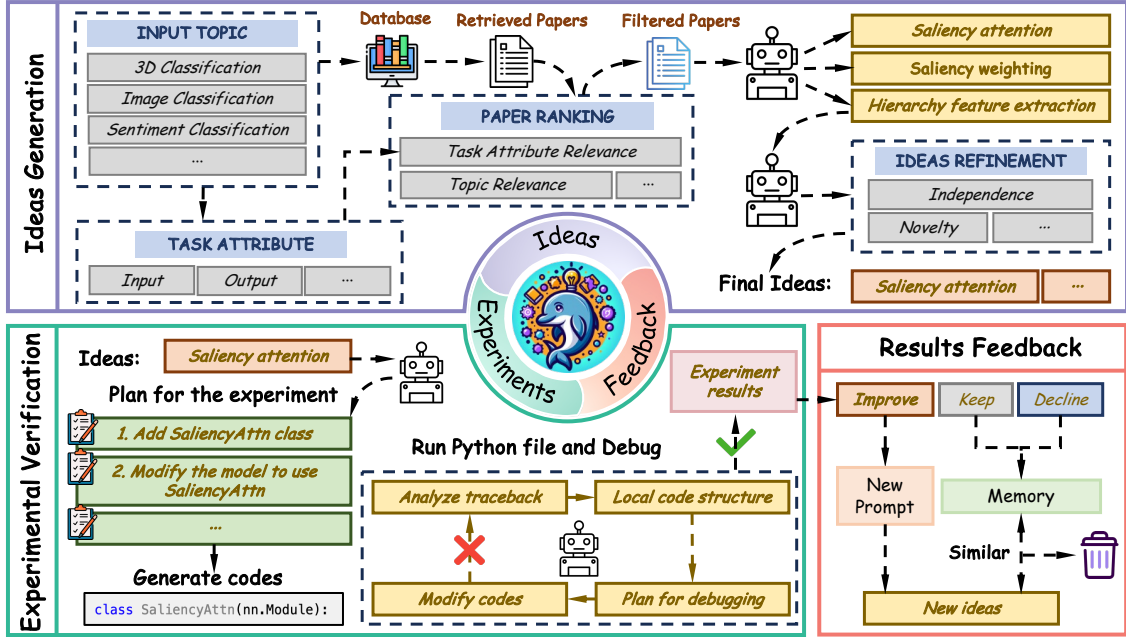


Figure 2: DOLPHIN first generates a set of ideas based on the retrieved papers. After filtering ideas, experimental plans will be generated for these filtered ideas. Then, codes can be generated and debugged using the proposed error-traceback-guided debugging process. Finally, the results of successfully executed experiments will be auto-analyzed and reflected into the next round of ideas generation.

to use LLMs to generate hypotheses. For example, Yang et al. (2023) generate hypotheses from web corpus and Wang et al. (2023) generate hypotheses from literature. However, these works lack empirical verification of their practical effectiveness. Recently, AI-Scientist (Lu et al., 2024) and Agent Laboratory (Schmidgall et al., 2025) introduce the end-to-end framework that automates the process from idea generation to experimental execution and paper writing. However, the experimental validation of its idea generation remains limited, lacking evaluation conducted on real-world datasets or scenarios. Furthermore, the framework lacks a feedback mechanism that connects experimental validation to idea generation, unlike human researchers who iteratively refine their hypotheses based on experimental outcomes.

2.2 Scope-limited Scientific Research.

Several studies have effectively applied LLMs to specific scientific discovery tasks. For example, AutoML-GPT (Zhang et al., 2023b) leverages LLMs for hyperparameter tuning by combining model and data descriptions as prompts to predict training logs. AgentHPO (Liu et al., 2024) introduces a creator-executor framework to iteratively optimize hyperparameters based on historical trials. Similarly, MLCopilot (Zhang et al., 2023a) constructs an experience pool from historical data to enable LLMs-based hyperparameter prediction.

EvoPrompting (Chen et al., 2024) improves the in-context prompting examples of LLMs to achieve effective code-level neural architecture design. In contrast to these scope-limited approaches, our method focuses on more open-ended scientific discovery, spanning from idea generation to experimental validation, and achieving a fully closed-loop research lifecycle.

3 Methods

In this section, we introduce DOLPHIN, a closed-loop auto-research framework as shown in Fig. 2, which is mainly composed of **an idea generation process**, **an experiments verification process**, and **a result feedback process**. The closed-loop means that the experimental results will be fed back into the idea generation process and the above three processes form a research cycle. In the idea generation process, DOLPHIN generates ideas based on the feedback and retrieved papers that are filtered by a designed task-attribute-guided paper ranking process. Then, DOLPHIN formulates experimental plans and proceeds to generate and debug code using a specifically designed error-traceback-guided debugging process. Finally, the results will be analyzed and utilized as feedback for the next cycle of ideas generation. The following sections detail the ideas generation process, experiments verification process, and results feedback process in Sec. 3.1, Sec. 3.2, and Sec. 3.3 respectively.

3.1 Ideas Generation Process

“A good beginning is half done.” As the beginning of the research cycle, high-quality ideas are crucial to the entire research. A promising approach to generating high-quality ideas is to imitate the behaviors of human researchers. They typically first conduct literature reviews and then generate ideas based on the literature (Randolph, 2019) and previous experimental experience (detail in Sec. 3.3). DOLPHIN typically divided the idea generation process into two steps including 1) paper retrieval and ranking, and 2) ideas generation and filtering.

Paper Retrieval and Ranking. To generate high-quality ideas, the first step is to retrieve papers that are relevant to the topic. Given a research topic (e.g., 3D classification), DOLPHIN begins by searching for relevant papers using Semantic Scholar API¹, obtaining the essential information such as titles and abstracts. However, the initially retrieved papers are not always directly related to the input topic, which can limit their usefulness in generating subsequent ideas. For instance, if the input topic is 3D classification, some retrieved papers might pertain to 3D detection (Wang et al., 2024c). Although these topics are interconnected, they typically focus on different challenges. As a result, it is necessary to filter out papers that are irrelevant to the specific topic.

To this end, we design a task-attribute-guided paper ranking process that aims to assign a higher score to the paper relevant to the input topic and task attribute. In detail, DOLPHIN ranks the retrieved papers based on two main criteria: 1) relevance to the input topic, and 2) alignment of task attributes with those of the input topic. The task attributes typically define a task, including model inputs, model outputs, and other characteristics. Specifically, the LLM is first utilized to extract the task attributes of the input topic and then prompt it to score (i.e., 1-10) each retrieved paper based on the designed criteria. The detailed prompts can be found in our appendix. After scoring, DOLPHIN filters out papers with scores below 8, using the remaining papers as the references of the subsequent ideas generation process.

Ideas Generation and Filtering. After obtaining retrieved papers, the next step is to generate ideas based on the references. DOLPHIN begins by prompting the LLM to generate N novel and non-redundant ideas, each comprising a title, a brief ex-

periment plan, and a summary. However, these generated ideas are not consistently novel, and some ideas are similar to one another. As a result, directly performing experiments on such ideas will cost substantial time and computational resources, further reducing research efficiency. To address this, we introduce a further idea-filtering procedure that filters out non-novel or redundant ideas.

To be specific, DOLPHIN first examines the independence of ideas to ensure non-redundancy. Given N ideas $[I_1, I_2, \dots, I_N]$, DOLPHIN first extract the sentence-level embedding $[f_1, f_2, \dots, f_N]$ based on the summary of each ideas. Then, an idea bank $\mathbf{B}$ is constructed to store embeddings of the ideas that have been checked to be independent. $\mathbf{B}$ is initialized as empty in the first loop and initialized with previous ineffective ideas in the following loops which will be introduced in Sec. 3.3. When examining the i -th idea, DOLPHIN calculates its cosine similarity with existing ideas stored in $\mathbf{B}$ and discards the current idea when the max similarity exceeds a predetermined threshold (i.e., 0.8). After filtering out redundant ideas, the remaining N' ideas will undergo the novelty check. Following AI-Scientist (Lu et al., 2024), DOLPHIN simply prompt the LLMs to decide whether the idea is novel based on the searched papers by Semantic Scholar API¹. Only ideas identified as novel and independent will proceed to the subsequent experimental verification process.

3.2 Experimental Verification Process

“Jump in and get your feet wet.” Experimental verification is crucial in the research cycle as it serves as the most effective way to confirm the effectiveness of proposed ideas. Most existing works (Li et al., 2024; Su et al., 2024) assess the novelty of AI-generated ideas via LLMs or humans. However, without closed-loop experimental verification, the effectiveness of these ideas remains uncertain. In contrast, DOLPHIN screens out truly effective ideas through an experimental verification process.

Given an idea generated by the ideas generation process (Sec. 3.1) and reference codes, DOLPHIN first prompts the LLM to generate detailed experimental plans and then modifies the reference codes according to the idea and the generated plans. After modified codes, the experiment will automatically proceed. However, we find that the execution success rate is relatively low since LLMs encounter significant challenges in modifying code with complicated nested relationships (e.g., between class

¹<https://www.semanticscholar.org/product/api>



Figure 3: Debugging with traceback-guided local code structure.

and function), while ensuring complete error-free execution. This will further reduce the efficiency of verifying ideas and research.

Based on our observation, we design an exception-traceback-guided debugging process as shown in Fig. 3, aiming at assisting the LLMs in comprehending code logic with local code structure. Specifically, to generate the code structure related to the code errors, DOLPHIN first extracts information in exception traceback, including function name, line, and code, since traceback contains the nested information between functions. Note that DOLPHIN only focuses on custom codes, excluding library function calls. Then, DOLPHIN prompts the LLM to generate the code structure under the guidance of extracted exception traceback information. After that, the LLM analyzes the exception traceback and local code structure to make necessary modifications, enabling automatic code execution after these adjustments. The debugging process will be repeated until either successful execution is achieved or the predetermined maximum number of debugging times is reached. Finally, all successfully implemented ideas will undergo comprehensive analyses in the next phase.

3.3 Results Feedback Process

“Experience is the best teacher.” Human researchers often analyze experimental results to further propose new ideas or improve existing ideas, since in-

sights from previous experiments can be leveraged to effectively enhance the quality of subsequent idea generation. However, recent works either implement feedback mechanisms within the isolated idea generation process (Li et al., 2024) or lack feedback mechanisms entirely (Lu et al., 2024). To address this limitation, DOLPHIN analyzes experimental results and incorporates the findings into the subsequent round of ideas generation process.

DOLPHIN first divides the experimental results into three categories (*i.e.*, **improvement**, **maintenance**, and **decline**) compared to the performance of the reference codes. Our goal is to discourage the development of ideas that have previously led to stagnant or declining performance, while actively promoting the creation of innovative concepts or iterations based on past ideas that enhance the model performance. In detail, DOLPHIN incorporate the embeddings of summaries from ideas that maintain or improve the performance into **B** defined in Sec. 3.1. In this way, ideas will be filtered out if they are similar to previous ideas that cannot improve the performance and avoid redundant verification of the ineffective ideas. Besides, DOLPHIN incorporates performance-enhancing ideas into the idea generation prompt for the next loop. Detailed prompts are shown in Appendix A.

4 Experiment

4.1 Experimental Setups

Tasks and Datasets. We conduct auto-research on three topics including image classification, 3D classification, and sentiment classification. *For 2D classification task*, we evaluate our method on CIFAR-100 (Krizhevsky et al., 2009), which is widely-used in computer vision. *For 3D classification task*, we use ModelNet40 (Wu et al., 2015) which is a 3D CAD dataset and consist of 40 categories. *For the sentiment classification task*, we use Stanford Sentiment Treebank (SST-2) dataset (Socher et al., 2013). We also conduct experiments on MLE-bench (Chan et al., 2024), which is designed to evaluate the capability of models in handling real-world ML tasks on Kaggle competitions. Further details of datasets can be found in Appendix B.1.

Implementation Details. For the ideas generation process, we use gpt-4o-2024-08-06 (OpenAI, 2024) as our LLM agent. The total number of retrieved papers is set to 50 and only papers with scores higher than 8 will be treated as references for the ideas generation process. We generate 20

Tasks	ModelNet40		CIFAR-100	SST-2
	OA (%)	mAcc. (%)	Top-1 Acc. (%)	Acc. (%)
Baseline	89.2 (PointNet)	86.2 (PointNet)	79.5 (WRN-28-10)	-
Baseline [†]	91.0 (PointNet)	87.6 (PointNet)	81.2 (WRN-28-10)	91.0 (BERT-base)
Avg. Improvement	92.0 (+1.0)	88.7 (+1.1)	81.8 (+0.6)	91.8 (+0.8)
Max Improvement	93.9 (+2.9)	91.1 (+3.5)	82.0 (+0.8)	92.5 (+1.5)
Human designed	93.8 (GPSFormer)	91.8 (GPSFormer)	82.2 (ResNeXt)	93.1 (BERT-large)
Number ideas	5 / 40	5 / 40	6 / 40	6 / 40

Table 1: Experimental verifications on 3D point classification, 2D image classification, and sentiment classification tasks. Number ideas refers to the number of ideas that can achieve performance gains. [†] denotes the results of our implementation. Avg. Improvement and Max Improvement represent the average and maximum improvement of all ideas that can improve the baseline performance.

ideas in each loop and the threshold of the independence filtering is set to 0.8. We use sentence-transformer/all-roberta-large-v1 (Reimers and Gurevych, 2020) to extract the summary embedding of each idea. For the experimental verification process, we use deepseek-v2.5 (Zhu et al., 2024) deployed by ollama (Team, 2023b) as our code agent. The maximum number of debugging attempts is set to 5 for experimental efficiency. Following AI-Scientist (Lu et al., 2024), we use self-reflection to first eliminate some syntax errors before executing the program and use aider as the framework to call LLM agents. The same hyperparameters and models employed in the ideas generation process are utilized in the results feedback.

4.2 Main Results

We evaluate DOLPHIN’s capabilities across various tasks covering point clouds, images, and language modalities. In this section, we conduct experiments on each task for two loops (*i.e.*, 40 ideas).

Results on 3D Point Classification. We conduct experiments on 3D classification task using PointNet (Qi et al., 2017a) as our baseline model. As illustrated in Tab. 1, a total of 5 ideas achieve performance gains in two loops, with an average improvement of 1.0% OA, which shows that DOLPHIN can generate and verify effective ideas. Besides, the maximum improvement can achieve 93.9% OA, which largely improves the performance compared with the human-designed baseline (*i.e.*, 91.0% achieved by PointNet). More excitingly, such results achieved by auto-research obtain comparable performance to the current SoTA method (*i.e.*, 93.8% achieved by GPSFormer (Wang et al., 2024a)). This method is carefully designed by human researchers based on Transformer architecture. We would like to emphasize that for a fair comparison, we compare the 3D methods without pre-training and the voting mechanism.

Results on 2D Image Classification. Further, we

conduct experiments on the image classification task, using WRN-28-10 (Zagoruyko, 2016) as our baseline model. As shown in Tab. 1, the average improvement and max improvement are 0.6% and 0.8%, respectively. Notably, the idea generated and performed automatically by DOLPHIN can obtain comparable performance to hand-crafted methods such as ResNeXt (Liu et al., 2022) (*e.g.*, 82.0% compared to 82.2%). It should be noted that Transformer-based methods such as ViT (Dosovitskiy et al., 2020) are not included in our comparison, due to their heavy dependence on large-scale pre-training, which, at this stage, requires significant resources to validate their effectiveness.

Results on Sentiment Classification. To verify the effectiveness of DOLPHIN on different modalities, we also perform experiments on sentiment classification task. We conduct experiments on SST-2 (Socher et al., 2013) and report the classification accuracy. In Tab. 1, we fine-tune the pre-trained BERT-base (Devlin et al., 2019) as our baseline model. It can be seen that DOLPHIN also generates and performs effective ideas (*e.g.*, 1.5% improvement on SST-2) on sentiment classification, reducing the performance gap between BERT-base (*i.e.*, 92.5%) and BERT-large (*i.e.*, 93.1%).

Results of tasks in MLE-bench. We further conduct experiments on tasks from MLE-bench (Chan et al., 2024) as shown in Tab. 2. First, DOLPHIN can flexibly integrate with code generation frameworks such as AIDE (Dominik Schmidt, 2024) and Agent Laboratory (Schmidgall et al., 2025). For example, based on the code template generated by AIDE, DOLPHIN further improves the performance on tasks such as insult detection (*i.e.*, 81.0% $\rightarrow$ 84.7%) and achieves **gold medal** level performance on Kaggle. Additionally, DOLPHIN performs version updates for technologies or code. For example, based on previous code from Kaggle, DOLPHIN can achieve further accuracy breakthroughs (*e.g.*, 95.3 $\rightarrow$ 96.2 on tabular playground).

Tasks	Domain	Code src.	Prev. score	Score
Detecting insults in social commentary		AIDE	81.0	84.7
Tabular playground series dec 2021		Kaggle	95.3	96.2
Jigsaw toxic comment classification		Kaggle	94.7	97.2

Table 2: Results of tasks in MLE-bench. Code src. denotes the source of the code template (AIDE: auto-generate by AIDE, Kaggle: Kaggle public code).

Method	Novelty	Cost (Avg.)
Naive generation	8 / 20	\$0.106
Generation with naive retrieval	13 / 20	\$0.187
Ours (task attribute filtering)	19 / 20	\$0.184

Table 3: Results of ideas generation process. The novelty is evaluated by gpt-4o-2024-08-06. Cost (Avg.) is the cost per idea including paper retrieval, ideas generation, and novelty check.

4.3 Further Analyses

Analysis on Ideas Generation Process. To evaluate the effectiveness of the ideas generation process, we conduct further ablation studies by comparing naive generation, generation with retrieved papers, and our proposed method. Naive generation refers to the direct use of LLMs to generate ideas based on the seed idea and reference codes, similar to the approach used by AI-Scientist (Lu et al., 2024). Generation with retrieved papers involves directly searching papers based on the topic and filtering them by their relevance to the input topic. As shown in Tab. 3, the naive generation yields the poorest results, with more than half of the ideas being judged as not novel. Furthermore, the quality of generated ideas is significantly improved when using naive retrieved papers, as this approach more closely aligns with the way human researchers generate ideas. However, as mentioned in Sec. 3.1, this approach tends to retrieve irrelevant papers and will mislead LLMs. As indicated in the first line of Tab. 4, some papers primarily focus on 3D detection or point cloud completion, where the design approach of the model is entirely different from that of point classification. This phenomenon can be well handled by *the designed paper ranking process*. As illustrated in Tab. 3 and Tab. 4, the number of novel ideas significantly increased from 8/20 to 19/20, while the proportion of papers related to irrelevant topics substantially decreased. This improvement can be attributed to the availability of more relevant reference papers during the ideas generation process. Note that the higher occurrence of the keyword “segmentation” is due to that, many studies concurrently perform both point classification and segmentation tasks. In addition,

Keywords	Classification	Detection	Segmentation	Completion
Naive	82	17	38	16
Filter (ours)	109	4	43	0

Table 4: For the 3D classification task, the frequency of each keyword is determined from the retrieved papers, focusing only on those words that appear in the abstracts and titles of papers scoring above 8 points in the ranking process. “Naive” and “Filter” refer to naive retrieval and filtering based on task attributes.

the average cost per idea is shown in Tab. 3. It can be seen that the cost of each idea is very small. Besides, the cost of generating each idea is relatively higher when retrieving papers. This is mainly due to the retrieval process and the longer prompt required for idea generation (*i.e.*, both the title and abstract are fed into the LLMs as references).

Analysis on Experimental Verification Process.

The success rate of experiment execution is crucial for improving research efficiency. We further conducted studies on the experimental verification process. As illustrated in Tab. 5, we conduct experiments on three approaches: **1)** directly feed the exception traceback to LLM, **2)** extract the local code structure based on exception traceback, and then feed the local code structure and traceback to LLM, and **3)** extract local code structure according to the information derived from the exception traceback, and then feed the local code structure and traceback to LLM.

Firstly, we find that the successful execution rate is relatively low (*e.g.*, 4 / 15) when directly feeding the traceback into LLM for debugging since the LLM cannot fully understand the complicated nested relationships in the codes. For example, when a dimension mismatch error occurs between networks and feature dimensions, LLMs can easily locate where the error occurs. However, since the feature might be obtained through multiple nested modules, LLMs fail to correct the network dimension. Then, by adding the local code structure according to exception traceback, the success rate does not significantly improve. This is because the exception traceback contains lots of information about called libraries, which makes LLMs generate code structures irrelevant to our custom codes. Further, by guiding LLMs to generate code structures with information extracted from traceback, the execution rate can be largely improved (*i.e.*, 33.3%→50.0%). This is due to the extracted information containing custom code information related to the exception, enabling LLMs to focus on relevant functions and variables. Note that to improve

L.C.S.	Traceback	Successful execution		
		Loop 1	Loop 2	Loop 3
✗	✗	4 / 15	5 / 13	5 / 14
✓	✗	3 / 15	5 / 13	6 / 14
✓	✓	7 / 15	6 / 13	8 / 14

Table 5: Results of successful execution rate. L.C.S. represents local code structure. Traceback denotes using information extracted from exception traceback. The denominator is the number of ideas after the novelty and independence check.

Loop	Loop 1	Loop 2	Loop 3	Total
Improvement rate	2 / 7	3 / 6	4 / 8	9 / 21
Cost (Avg.)	0.184	0.203	0.218	0.201

Table 6: Performance in different loops. The denominator is the number of successfully executed ideas.

efficiency, we only allow a maximum of 5 debugging iterations in our experiments.

Analysis on Results Feedback Process. To demonstrate the advantages of the closed-loop framework, we further analyze results on different loops. As shown in Tab. 6, we find that the quality of generated ideas without feedback is relatively low. The reasons can be divided into two folds: **1)** Repeated ideas may be generated without a feedback process, resulting in redundant verification of the same idea and decreased experimental efficiency. **2)** In the absence of feedback, the model cannot learn what kind of ideas are effective for the specific task.

In contrast, DOLPHIN effectively solves these challenges through a closed-loop approach, demonstrating progressive improvement in idea quality as the number of iterations increases (*e.g.*, 2/7 improvement rate in Loop 1 $\rightarrow$ 4/8 improvement rate in Loop 3). Further, these ideas that improve the performance are different between each round, which further improves the research efficiency and shows the effectiveness of DOLPHIN. Besides, the average cost slightly increases as the iterations continue, since the results will be fed back into the next round of ideas generation process.

4.4 Case Studies

We illustrate our approach with an example drawn from 3D point classification task as shown in Fig. 9. It can be seen that DOLPHIN can generate codes corresponding to the idea and only in this way, the generated ideas can be effectively verified. Tab. 7 presents the comparison between AI-generated approach (*i.e.*, PointNet-CSR obtained by DOLPHIN)

Method	Accuracy (Avg. class)	Overall accuracy
<i>Human designed methods</i>		
PointNet (Qi et al., 2017a)	86.2	89.2
PointNet++ (Qi et al., 2017b)	-	91.9
DGCNN (Wang et al., 2019)	90.2	92.9
PointNetXt (Qian et al., 2022)	90.8	93.2
OctFormer (Wang, 2023)	-	92.7
GPSFormer (Wang et al., 2024a)	91.8	93.8
<i>Methods obtained by DOLPHIN (auto-research)</i>		
PointNet-CSR	91.1	93.9

Table 7: Accuracy on ModelNet40. The results are obtained from 1024 points without voting.

Diff.	DGCNN (Human)	PointNet-CSR (DOLPHIN)
Idea	1) Architecture-level 2) With learnable parameters 3) Repeated blocks	1) Module-level 2) Without learnable parameters 3) Single module
Impl.	Multi-layer Edge with high complexity	Single contextual semantic reasoning module with low complexity
Results	1) 90.2% mAcc., 92.9% OA 2) $\sim$ 20.86s per epoch	1) 91.1% mAcc., 93.9% OA 2) $\sim$ 6.12s per epoch ($>$ 3x faster)

Table 8: The differences between DGCNN (Wang et al., 2019) proposed by human and PointNet-CSR proposed using DOLPHIN.

and previous human-designed methods. Idea automatically generated and performed by DOLPHIN can outperform most human-designed methods and achieve comparable performance to current SoTA (Wang et al., 2024a). Furthermore, as shown in Tab. 8, we carefully investigate the existing works on 3D classification task up to the submission date, identifying the work most relevant to PointNet-CSR (AI-generated 3D work), as illustrated in Fig. 9. The detailed comparison of the idea, implementation, and result can be found in Tab. 8, showing that PointNet-CSR can achieve better and faster performance through a more concise architecture. Please refer to Fig. 9 and Appendix B.2 for more comparisons between human-designed works and DOLPHIN-generated works.

5 Conclusion and Future Outlook

DOLPHIN evaluates idea quality through experimental verification, improving it in a closed-loop fashion. Besides, beyond conducting quantitative experiments that demonstrate DOLPHIN’s capability to generate solutions and results comparable to human-designed approaches, we also conducted in-depth case studies to evaluate the novelty of the ideas and the implementation efficiency of the codes generated by DOLPHIN. These quantitative and qualitative evaluations we conducted for DOLPHIN are essential for gaining further insight into the potential and value inherent in DOLPHIN.

In the future, we envision DOLPHIN further ad-

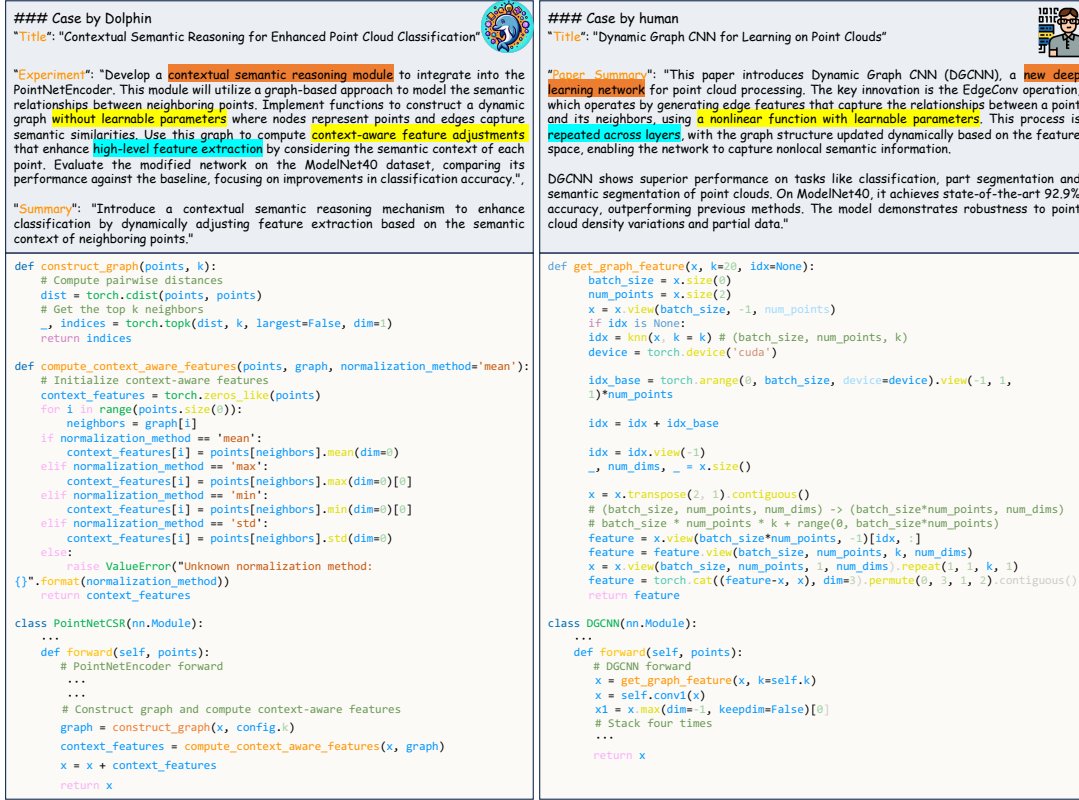


Figure 4: Case studies for the ideas and codes generated by DOLPHIN (Left) and human researcher (Right).

vancing AI-driven automated scientific research. By harnessing its ability to generate novel ideas in a closed-loop system, we also aspire for DOLPHIN to foster the development of groundbreaking ideas inspired by cross-disciplinary knowledge, ultimately providing innovative solutions for complex scientific challenges.

Limitations

Although DOLPHIN can generate and implement effective ideas in a loop, it still has inherent limitations. First, in the idea generation stage, LLMs retain historical knowledge obtained from training data, which may cause knowledge leakage when generating ideas. Besides, only the abstract and title are used in the idea generation process, which may result in LLM not being able to understand the technical details in the article and the logic between the articles. However, the feedback-driven idea generation process in DOLPHIN can be flexibly combined with methods that address these concerns. In addition, the code capabilities of LLMs are not sufficient to understand complex codes such as project-level codes, resulting in DOLPHIN being unable to verify complex tasks currently.

Ethics Statement

This work focuses on the development of a closed-loop framework for auto-research, aiming to accelerate the research cycle. The proposed method relies on publicly available datasets, research papers, and models, ensuring compliance with copyright and intellectual property laws. While the framework is intended to aid human research, we encourage users to critically evaluate the generated results to ensure that they adhere to ethical research practices and mitigate any potential limitations, such as bias or incomplete ideas.

Acknowledgement

This work was supported by National Key Research and Development Program of China (No. 2022ZD0160101), Shanghai Natural Science Foundation (No. 23ZR1402900), Shanghai Science and Technology Commission Explorer Program Project (24TS1401300), Shanghai Municipal Science and Technology Major Project (No.2021SHZDZX0103). The research was also supported by Shanghai Artificial Intelligence Laboratory, a locally commissioned task from the Shanghai Municipal Government, the Shanghai Municipal Science and Technology Major Project, and Shanghai Rising Star Program (Grant No.230D1401000).

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. URL: https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.
- Assafelovic. 2023. gpt-researcher. URL: <https://github.com/assafelovic/gpt-researcher>.
- Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*.
- Angelica Chen, David Dohan, and David So. 2024. Evo-prompting: language models for code-level neural architecture search. *Advances in Neural Information Processing Systems*, 36.
- Kang Chen, Tao Han, Junchao Gong, Lei Bai, Fenghua Ling, Jing-Jia Luo, Xi Chen, Leiming Ma, Tianning Zhang, Rui Su, et al. 2023. Fengwu: Pushing the skillful global medium-range weather forecast beyond 10 days lead. *arXiv preprint arXiv:2304.02948*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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*, pages 4171–4186.
- Yuxiang Wu Dominik Schmidt, Zhengyao Jiang. 2024. Aide. URL: <https://www.weco.ai/blog/technical-report>.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Tao Han, Zhenghao Chen, Song Guo, Wanghan Xu, and Lei Bai. 2024. Cra5: Extreme compression of era5 for portable global climate and weather research via an efficient variational transformer. *arXiv preprint arXiv:2405.03376*.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Xiang Hu, Hongyu Fu, Jinge Wang, Yifeng Wang, Zhikun Li, Renjun Xu, Yu Lu, Yaochu Jin, Lili Pan, and Zhenzhong Lan. 2024. Nova: An iterative planning and search approach to enhance novelty and diversity of llm generated ideas. *arXiv preprint arXiv:2410.14255*.
- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589.
- Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images.
- Long Li, Weiwen Xu, Jiayan Guo, Ruochen Zhao, Xinxuan Li, Yuqian Yuan, Boqiang Zhang, Yuming Jiang, Yifei Xin, Ronghao Dang, et al. 2024. Chain of ideas: Revolutionizing research in novel idea development with llm agents. *arXiv preprint arXiv:2410.13185*.
- Siyi Liu, Chen Gao, and Yong Li. 2024. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*.
- Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery. *ArXiv*, abs/2408.06292.
- Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. 2022. Biogpt: generative pre-trained transformer for biomedical text generation and mining. *Briefings in bioinformatics*, 23(6):bbac409.
- OpenAI. 2024. Hello gpt-4o. <https://openai.com/index/hello-gpt-4o/>.
- Biqing Qi, Kaiyan Zhang, Haoxiang Li, Kai Tian, Sihang Zeng, Zhang-Ren Chen, and Bowen Zhou. 2023. Large language models are zero shot hypothesis proposers. *arXiv preprint arXiv:2311.05965*.
- Biqing Qi, Kaiyan Zhang, Kai Tian, Haoxiang Li, Zhang-Ren Chen, Sihang Zeng, Ermo Hua, Hu Jinfang, and Bowen Zhou. 2024. Large language models as biomedical hypothesis generators: A comprehensive evaluation. *arXiv preprint arXiv:2407.08940*.

- Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.
- Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30.
- Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. 2022. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. *Advances in neural information processing systems*, 35:23192–23204.
- Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, et al. 2023. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. *arXiv preprint arXiv:2310.08559*.
- Justus Randolph. 2019. A guide to writing the dissertation literature review. *Practical assessment, research, and evaluation*, 14(1):13.
- Nils Reimers and Iryna Gurevych. 2020. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In *EMNLP*.
- Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*.
- Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. 2024. Can llms generate novel research ideas? a large-scale human study with 100+ nlp researchers. *arXiv preprint arXiv:2409.04109*.
- Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.
- Haoyang Su, Renqi Chen, Shixiang Tang, Xinzhe Zheng, Jingzhe Li, Zhenfei Yin, Wanli Ouyang, and Nanqing Dong. 2024. Two heads are better than one: A multi-agent system has the potential to improve scientific idea generation. *arXiv preprint arXiv:2410.09403*.
- Copilot Team. 2023a. copilot. URL: <https://github.com/features/copilot>.
- Ollama Team. 2023b. ollama. URL: <https://ollama.com/>.
- Changshuo Wang, Meiqing Wu, Siew-Kei Lam, Xin Ning, Shangshu Yu, Ruiping Wang, Weijun Li, and Thambipillai Srikanthan. 2024a. Gpsformer: A global perception and local structure fitting-based transformer for point cloud understanding. In *European Conference on Computer Vision*. Springer.
- Peng-Shuai Wang. 2023. Octformer: Octree-based transformers for 3d point clouds. *ACM Transactions on Graphics (TOG)*, 42(4):1–11.
- Qingyun Wang, Doug Downey, Heng Ji, and Tom Hope. 2023. Scimon: Scientific inspiration machines optimized for novelty. *arXiv preprint arXiv:2305.14259*.
- Qingyun Wang, Doug Downey, Heng Ji, and Tom Hope. 2024b. SciMON: Scientific Inspiration Machines Optimized for Novelty. In *ACL*.
- Yidong Wang, Qi Guo, Wenjin Yao, Hongbo Zhang, Xin Zhang, Zhen Wu, Meishan Zhang, Xinyu Dai, Min Zhang, Qingsong Wen, et al. 2024c. Autosurvey: Large language models can automatically write surveys. *arXiv preprint arXiv:2406.10252*.
- Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12.
- Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920.
- Xiangchao Yan, Shiyang Feng, Jiakang Yuan, Renqiu Xia, Bin Wang, Bo Zhang, and Lei Bai. 2025. Surveyforge: On the outline heuristics, memory-driven generation, and multi-dimensional evaluation for automated survey writing. *arXiv preprint arXiv:2503.04629*.
- An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- Zonglin Yang, Xinya Du, Junxian Li, Jie Zheng, Soujanya Poria, and E. Cambria. 2024b. Large Language Models for Automated Open-domain Scientific Hypotheses Discovery. *ACL Findings*.
- Zonglin Yang, Xinya Du, Junxian Li, Jie Zheng, Soujanya Poria, and Erik Cambria. 2023. Large language models for automated open-domain scientific hypotheses discovery. *arXiv preprint arXiv:2309.02726*.
- Shunyu Yao, Qingqing Ke, Qiwei Wang, Kangtong Li, and Jie Hu. 2024. Lawyer gpt: A legal large language model with enhanced domain knowledge and reasoning capabilities. In *Proceedings of the 2024 3rd International Symposium on Robotics, Artificial Intelligence and Information Engineering, RAIIIE*

'24, page 108–112, New York, NY, USA. Association for Computing Machinery.

Sergey Zagoruyko. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. 2023a. Mlcpilot: Unleashing the power of large language models in solving machine learning tasks. *arXiv preprint arXiv:2304.14979*.

Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. 2023b. Automl-gpt: Automatic machine learning with gpt. *arXiv preprint arXiv:2305.02499*.

Yangqiaoyu Zhou, Haokun Liu, Tejes Srivastava, Hongyuan Mei, and Chenhao Tan. 2024. Hypothesis generation with large language models. *arXiv preprint arXiv:2404.04326*.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

Outlines for Appendix

In Appendix, we provide additional details and qualitative results on the following aspects:

- Appendix A: Further details of DOLPHIN including prompts, qualitative results for each step, and so on:
 - Appendix A.1: Ideas generation process.
 - Appendix A.2: Experimental verification process.
- Appendix B: Additional details and results of performing experiments using DOLPHIN:
 - Appendix B.1 The selected datasets and implementation on different tasks:
 - * Appendix B.1.1: Image classification task.
 - * Appendix B.1.2: 3D classification task.
 - * Appendix B.1.3: Sentiment classification task.
 - * Appendix B.1.4: MLE-bench.
 - Appendix B.2: Code implementation differences between human-designed and Dolphin-designed.
- Appendix C: More analyses and further works:
 - Appendix C.1: The extensibility of the research task related to idea generation.
 - Appendix C.2: Analyses on future works.

A Further details of Dolphin

In this section, we provide more details about DOLPHIN including prompts, qualitative results of some processes, and algorithms of some processes. In the following section, we will give more details of the ideas generation process, experimental verification process, and results feedback process, respectively.

A.1 Ideas Generation Process

We provide prompts used in paper retrieval, paper ranking, and idea generation process in Fig. 5. We partially refer to the prompt design of previous works (Lu et al., 2024; Si et al., 2024). As depicted in our manuscript, the quality of retrieved papers is important to idea generation. Here, we give an example to further illustrate the impact. Given the topic ‘3D classification’, naive retrieval will result

Algorithm 1: Independence Check Process

Input: List of ideas $\mathbf{I}$, previous paper memory bank $\mathbf{B}$, sentence embedding model $\mathcal{S}$, threshold τ .

Output: Idea independence of each paper $\mathbf{R}$.

```

for idea summary  $s$  in  $\mathbf{I}$  do
  if  $\text{len}(\mathbf{B}) == 0$  then
     $\mathbf{R}.\text{append}(\text{True})$ 
  else
    Extract summary embedding:
     $f_s = \mathcal{S}(s)$ .
    Compare  $f_s$  with summary
    embeddings in  $\mathbf{B}$  by
     $\text{sim} = f_s \cdot \mathbf{B}^T \in \mathbf{R}^{1 \times \text{len}(\mathbf{B})}$ .
    if  $\max(\text{sim}) < \tau$  then
       $\mathbf{R}.\text{append}(\text{True})$ 
    else
       $\mathbf{R}.\text{append}(\text{False})$ 
return Independence list  $\mathbf{R}$ ,  $\text{len}(\mathbf{R}) == \text{len}(\mathbf{I})$ 

```

in lots of papers related to 3D object detection. As a result, we have identified several ideas that are more closely related to detection (*e.g.*, region proposal PointNet), which are significantly influenced by the detection task.

Further, we provide the algorithm of independence check in Algorithm 1. To show the effectiveness of the independence check process, we show an example in Fig. 6. It can be seen that although the name and title of the idea are totally different from each other, the technologies used in the two ideas are almost the same. Our idea independence check process can effectively filter the repeated ideas, further improving the auto-research efficiency.

A.2 Experimental Verification Process

Fig. 7 shows the local code structure generation prompt, which needs first to extract the exception traceback information. Further, to show how this information can guide the LLM in generating the local code structure, we provide an example for better illustration. As shown in Fig. 8, the LLM tend to copy the original reference code which is useless in the following debugging process.

B Additional Details and Results of Experiments

In this section, we provide a comprehensive overview of the implementation details and dataset information used in our main text.

B.1 Details of Selected Tasks

B.1.1 Image Classification Task

Dataset: CIFAR-100. The CIFAR-100 dataset (Krizhevsky et al., 2009) includes colored natural images with a resolution of 32×32 pixels, categorized into 100 distinct classes. The training and test sets contain 50,000 and 10,000 images, respectively. We adopt a standard data augmentation scheme (*i.e.*, RandomCrop, RandomHorizontalFlip, RandomRotation).

Implementation Details. We use WRN-28-10 (Zagoruyko, 2016) as our baseline. We partially refer to the codebase². Our training process employs the SGD optimizer with the CosineAnnealing scheduler. The initial learning rate is set to 0.1, and we train the model for 200 epochs with a batch size of 128. We apply a weight decay of 5e-4 and a Nesterov momentum with a coefficient of 0.9.

B.1.2 3D Classification Task

Dataset: ModelNet40. ModelNet40 (Wu et al., 2015) is a synthetic object dataset that contains 12,311 3D CAD models covering 40 categories. The standard training/validation set of ModelNet40 carries 9843/2468 point clouds.

Implementation Details. We use PointNet (Qi et al., 2017a) as our baseline. Following PointNet (Qi et al., 2017a), we uniformly sample 1024 points on each object. We use the random scale, random dropout, and point shift during training and train the model for 200 epochs. The initial learning rate is set to 1e-3. We use Adam optimizer (weight decay=1e-4) and step learning rate decay (step size=20, gamma=0.7). Our implementation partially refers to codebase³.

B.1.3 Sentiment Classification Task

Dataset: SST-2. Stanford Sentiment Treebank (SST) (Socher et al., 2013) contains 11,855 one-sentence movie reviews extracted from Rotten Tomatoes. SST contains 215,154 unique manually labeled texts of varying lengths.

²<https://github.com/bmsookim/wide-resnet.pytorch>

³https://github.com/yanx27/Pointnet_Pointnet2_pytorch/tree/master

Implementation Details. Our code of sentiment classification tasks refers to the codebase⁴. We fully fine-tune the BERT-base model for the classification task for 5 epochs with the learning rate 2e-5. The batch size is set to 32. We use the early stop mechanism during training (*i.e.*, stop training if the accuracy of current epoch is lower than that of the previous epoch).

B.1.4 MLE-bench

MLE-bench (Chan et al., 2024) is designed to evaluate the capability of AI agents’ ML engineering. It is composed of 75 ML competitions from Kaggle. In our manuscript, we conduct experiment on 3 tasks in MLE-bench including detecting insults in social commentary challenge, tabular playground series dec 2021 challenge, and jigsaw toxic comment classification challenge.

B.2 Case Studies

We provide several cases that are automatically generated and evaluated by DOLPHIN as shown in Fig. 10, Fig. 11, and Fig. 12. We show the ideas and modified codes in figures and the performance in the corresponding caption, respectively.

C Further Analysis and Future Work

C.1 The Extensibility for the Research Task

Recently, lots of works have explored automatic idea generation (Li et al., 2024; Si et al., 2024). One limitation is that the novelty of an idea can only be judged through human scoring or large model scoring. However, in real scientific research, we need ideas that can lead to performance breakthroughs that can not be judged without experiments. DOLPHIN which includes ideas generation, experimental verification, and results feedback process can serve as an evaluation protocol. In the future, it can be combined with auto-idea generation works to assess the effectiveness of the idea generation.

C.2 Analysis on Future Works

DOPLINE achieves the first closed-loop automatic research framework, we still hope that DOPLINE will possess stronger auto-research capabilities. For example, our ultimate goal is to utilize the capabilities of large models to integrate multi-disciplinary knowledge which is hard to be achieved by human researchers. To achieve this goal, we still

⁴<https://github.com/YJiangcm/SST-2-sentiment-analysis>

need to make efforts in the following aspects: **1)** develop more powerful code models that can understand and modify project-level code, and **2)** retrieve multi-disciplinary papers that may be related to the given topic.

Task Attributes Extraction

You are a researcher doing research on the topic of {topic_description}. You should define the task attributes of the topic for better searching relevant papers, for example, the model input and output. Formulate the task attributes as AttributeName("attribute"), for example the input and output are formulated as: Input("input"), Output("output").

Paper Searching

You are a researcher doing literature review on the topic {topic_description}. You should propose some keywords for using the Semantic Scholar API to find the most relevant papers to this topic. Formulate your query as: KeywordQuery("keyword").

Just give me one query, with the most important keyword, the keyword can be a concatenation of multiple keywords (just put a space between every word) but please be concise and try to cover all the main aspects. Your query (just return the query itself with no additional text)

Paper Searching

You are a helpful literature review assistant whose job is to read the below set of papers and score each paper. The criteria for scoring are:

- (1) The paper is directly relevant to the topic of: {topic_description}. Note that it should be specific to solve the problem of focus, rather than just generic methods.
- (2) The {task_attribute_name_1}, {task_attribute_name_2} ... of the proposed method in this paper is same with {task_attribute_1}, {task_attribute_2}, ... Note that if the task attributes are not match, the paper should get a low score.

The papers are: {paper_list}.

Please score each paper from 1 to 10. Write the response in JSON format with "paperID: score" as the key and value for each paper.

Ideas Generation

{task_description}

Here are some relevant papers on this topic just for your background knowledge: \n {rag_reference}. \n

The above papers are only for inspiration and you should not cite them and just make some incremental modifications. Instead, you should make sure your ideas are novel and distinct from the prior literature.

<experiment.py>\n {code} \n </experiment.py> \n

Here are the ideas that you have already generated: \n {prev_ideas_string}. \n

The following ideas have been proved to be effective: \n {prev_ideas_string_pos} \n

Come up with the next impactful and creative idea for research experiments and directions you can feasibly investigate with the code provided.

Note that you will not have access to any additional resources or datasets.

Make sure any idea is not overfit the specific training dataset or model, and has wider significance.

Respond in the following format:

THOUGHT: \n <THOUGHT>

NEW IDEA JSON: `` `json \n <JSON> \n `` ` \n

In <THOUGHT>, first briefly discuss your intuitions and motivations for the idea. Detail your high-level plan, necessary design choices and ideal outcomes of the experiments. Justify how the idea is different from the existing ones.

In <JSON>, provide the new idea in JSON format with the following fields:

- "Name": A shortened descriptor of the idea. Lowercase, no spaces, underscores allowed.
- "Title": A title for the idea, will be used for the report writing.
- "Experiment": An outline of the implementation. E.g. which functions need to be added or modified, how results will be obtained, ...
- "Summary": A sentence to summarize the idea according to the title and experiment above.

Be cautious and realistic on your ratings. This JSON will be automatically parsed, so ensure the format is precise.

Figure 5: Prompts of paper retrieval, paper ranking, and ideas generation.


```

#### Idea 1
{
  "Name": "contextual_saliency",
  "Title": "Contextual Saliency Mechanism for Enhanced Point Cloud Classification",
  "Experiment": "Integrate a contextual saliency module within the PointNetEncoder. Implement a lightweight neural network to predict saliency weights for features based on their context within the point cloud. Modify the feature extraction process to incorporate these saliency weights dynamically, emphasizing contextually important features. Evaluate the modified network on the ModelNet40 dataset, comparing its performance against the baseline PointNet, focusing on improvements in classification accuracy and robustness to input variations.",
  "Summary": "Introduce a contextual saliency mechanism in PointNet to enhance classification by dynamically weighting features based on their contextual importance within the point cloud.",
  "independence": true,
}

#### Idea 2
{
  "Name": "perceptual_importance",
  "Title": "Perceptual Importance for Enhanced Point Cloud Classification",
  "Experiment": "Develop a perceptual importance module to integrate into the PointNetEncoder. Implement a mechanism to predict perceptual weights for features based on their contextual significance and perceptual cues within the point cloud. Modify the feature extraction process to incorporate these perceptual weights dynamically, enhancing focus on perceptually important features. Evaluate the modified network on the ModelNet40 dataset, comparing its performance against the baseline PointNet, focusing on improvements in classification accuracy and interpretability through perceptual relevance.",
  "Summary": "Introduce a perceptual importance mechanism in PointNet to enhance classification by dynamically weighting features based on their perceptual significance and context within the point cloud.",
  "independence": false,
}

```

Figure 6: An example of independence check.

```

#### Code Structure Prompt

You are an expert code analyst specializing in exception detection, debugging, and error handling patterns. Your task is to thoroughly analyze the provided code with a focus on potential errors below: \n {error_messages} \n

You need to focus on error-related aspects of code and analyze their relations. The following functions and codes may highly related to the error which is extracted from the traceback. \n {function_code} \n

Note that you do not need to modify the code in this step and just need to give the exception-related code structure.

#### Debugging Prompt

You are an expert code debugger specializing in structural analysis and error diagnosis. Your task is to debug the code based on the following error message: \n {error_messages} \n

Previously, you have analyzed the error-related code structure as follows: \n {code_structure} \n

You need to first analyze the error message and list all the possible reasons and code modification plan of the error. Then, modify the code based on the plan. You can refer to the code structure obtained from the previous analysis.

```

Figure 7: Prompts of local code structure and debugging.

```
##### You are an expert code analyst specializing in error detection, debugging, and error handling patterns. Your task is to thoroughly analyze the provided code with a focus on potential errors below:
##### Traceback (most recent call last):
##### File "experiment.py", line 345, in <module>
#####   loss, logits = model(points, target)
##### File ".../python3.9/site-packages/torch/nn/modules/module.py", line 1130, in _call_impl
#####   return forward_call(*input, **kwargs)
##### File "experiment.py", line 220, in forward
#####   x, trans, trans_feat = self.backbone(x)
##### File ".../python3.9/site-packages/torch/nn/modules/module.py", line 1130, in _call_impl
#####   return forward_call(*input, **kwargs)
##### File ".../python3.9/site-packages/torch/nn/modules/container.py", line 139, in forward
#####   input = module(input)
##### File ".../python3.9/site-packages/torch/nn/modules/module.py", line 1130, in _call_impl
#####   return forward_call(*input, **kwargs)
##### File ".../python3.9/site-packages/torch/nn/modules/conv.py", line 307, in forward
#####   return self._conv_forward(input, self.weight, self.bias)
##### File ".../python3.9/site-packages/torch/nn/modules/conv.py", line 303, in _conv_forward
#####   return F.conv1d(input, weight, bias, self.stride,
##### RuntimeError: Given groups=1, weight of size [64, 6, 1], expected input[64, 64, 1026] to have 6 channels, but got 64 channels instead
#####
##### You need to focus on error-related aspects of code and analyze their relations. Note that you do not need to modify the code in this step and just need to give the error-related code structure.
```

Error-Related Code Structure

```
1. **PointNetEncoder Class Initialization:**
...
python
class PointNetEncoder(nn.Module):
    def __init__(self, global_feat=True, feature_transform=False, in_channels=3):
        super(PointNetEncoder, self).__init__()
        self.stn = STN3d(in_channels)
        self.conv_layer1 = nn.Sequential(
            nn.Conv1d(in_channels, 64, 1),
            nn.BatchNorm1d(64),
            nn.ReLU(inplace=True),
            nn.Conv1d(64, 64, 1),
            nn.BatchNorm1d(64),
            nn.ReLU(inplace=True)
        )
        self.conv_layer2 = nn.Sequential(
            nn.Conv1d(64, 64, 1),
            nn.BatchNorm1d(64),
            nn.ReLU(inplace=True)
        )
...
python
def forward(self, x):
    B, D, N = x.size()
    trans = self.stn(x)
    x = x.transpose(2, 1)
    if D > 3:
        feature = x[:, :, :3]
        x = x[:, :, :3]
    x = torch.bmm(x, trans)
    if D > 3:
        x = torch.cat([x, feature], dim=2)
    x = x.transpose(2, 1)
    complexity_scores = self.compute_complexity_scores(x.transpose(2, 1))
    1) detach().cpu().numpy()
    x = self.adjust_kernel_size(x, complexity_scores)
    x = self.conv_layer1(x) # Error occurs here
...

2. **PointNetEncoder Forward Pass:**
...
python
def forward(self, x):
    B, D, N = x.size()
    trans = self.stn(x)
    x = x.transpose(2, 1)
    if D > 3:
        feature = x[:, :, :3]
        x = x[:, :, :3]
    x = torch.bmm(x, trans)
    if D > 3:
        x = torch.cat([x, feature], dim=2)
    x = x.transpose(2, 1)
    complexity_scores = self.compute_complexity_scores(x.transpose(2, 1))
    1) detach().cpu().numpy()
    x = self.adjust_kernel_size(x, complexity_scores)
    x = self.conv_layer1(x) # Error occurs here
...
```

```
3. **Model Class Forward Pass:**
...
python
class Model(nn.Module):
    def __init__(self, in_channels=3, num_classes=40, scale=0.001):
        super().__init__()
        self.mat_diff_loss_scale = scale

        self.backbone = PointNetEncoder(global_feat=True, feature_transform=True,
                                         in_channels=in_channels)
        self.cls_head = nn.Sequential(
            nn.Linear(1024, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 256),
            nn.Dropout(p=0.4),
            nn.BatchNorm1d(256),
            nn.ReLU(inplace=True),
            nn.Linear(256, num_classes)
        )

    def forward(self, x, gts):
        x, trans, trans_feat = self.backbone(x) # Calls PointNetEncoder forward
        x = self.cls_head(x)
        x = F.log_softmax(x, dim=1)
        loss = F.nll_loss(x, gts)
        mat_diff_loss = feature_transform_regularizer(trans_feat)
        total_loss = loss + mat_diff_loss * self.mat_diff_loss_scale
        return total_loss, x
...

4. **Training Loop:**
...
python
for epoch in tqdm(range(config.max_epoch), desc='training'):
    model.train()
    cm = ConfusionMatrix(num_classes=len(datasets['train'].classes))
    for points, target in tqdm(data_loaders['train'], desc=f'epoch {cur_epoch}/{(config.max_epoch)}'):
        points = points.data.numpy()
        points = data_transforms.random_point_dropout(points)
        points[:, :, 0:3] = data_transforms.random_scale_point_cloud(points[:, :, 0:3])
        points[:, :, 0:3] = data_transforms.shift_point_cloud(points[:, :, 0:3])
        points = torch.from_numpy(points).transpose(2, 1).contiguous()

        points, target = points.cuda(), target.long().cuda()

        loss, logits = model(points, target) # Calls Model forward

        self.conv_layer3 = nn.Sequential(
            nn.Conv1d(64, 128, 1),
            nn.BatchNorm1d(128),
            nn.ReLU(inplace=True)
        )
        self.conv_layer4 = nn.Sequential(
            nn.Conv1d(128, 1024, 1),
            nn.BatchNorm1d(1024)
        )
        self.global_feat = global_feat
        self.feature_transform = feature_transform
        if self.feature_transform:
            self.fstn = STNkd(k=64)
...
```

Figure 8: Code structure without extracted traceback information.

#### Case by Dolphin  "Title": "Contextual Semantic Reasoning for Enhanced Point Cloud Classification" "Experiment": "Develop a contextual semantic reasoning module to integrate into the PointNetEncoder. This module will utilize a graph-based approach to model the semantic relationships between neighboring points. Implement functions to construct a dynamic graph without learnable parameters where nodes represent points and edges capture semantic similarities. Use this graph to compute context-aware feature adjustments that enhance high-level feature extraction by considering the semantic context of each point. Evaluate the modified network on the ModelNet40 dataset, comparing its performance against the baseline, focusing on improvements in classification accuracy." "Summary": "Introduce a contextual semantic reasoning mechanism to enhance classification by dynamically adjusting feature extraction based on the semantic context of neighboring points." <pre>def construct_graph(points, k): # Compute pairwise distances dist = torch.cdist(points, points) # Get the top k neighbors _, indices = torch.topk(dist, k, largest=False, dim=1) return indices def compute_context_aware_features(points, graph, normalization_method='mean'): # Initialize context-aware features context_features = torch.zeros_like(points) for i in range(points.size(0)): neighbors = graph[i] if normalization_method == 'mean': context_features[i] = points[neighbors].mean(dim=0) elif normalization_method == 'max': context_features[i] = points[neighbors].max(dim=0)[0] elif normalization_method == 'min': context_features[i] = points[neighbors].min(dim=0)[0] elif normalization_method == 'std': context_features[i] = points[neighbors].std(dim=0) else: raise ValueError("Unknown normalization method: {}".format(normalization_method)) return context_features class PointNetCSR(nn.Module): ... def forward(self, points): # PointNetEncoder forward ... # Construct graph and compute context-aware features graph = construct_graph(x, config.k) context_features = compute_context_aware_features(x, graph) x = x + context_features return x</pre>	#### Case by human  "Title": "Dynamic Graph CNN for Learning on Point Clouds" "Paper Summary": "This paper introduces Dynamic Graph CNN (DGCNN), a new deep learning network for point cloud processing. The key innovation is the EdgeConv operation, which operates by generating edge features that capture the relationships between a point and its neighbors, using a nonlinear function with learnable parameters. This process is repeated across layers with the graph structure updated dynamically based on the feature space, enabling the network to capture nonlocal semantic information. DGCNN shows superior performance on tasks like classification, part segmentation and semantic segmentation of point clouds. On ModelNet40, it achieves state-of-the-art 92.9% accuracy, outperforming previous methods. The model demonstrates robustness to point cloud density variations and partial data." <pre>def get_graph_feature(x, k=20, idx=None): batch_size = x.size(0) num_points = x.size(2) x = x.view(batch_size, -1, num_points) if idx is None: idx = knn(x, k=k) # (batch_size, num_points, k) device = torch.device('cuda') idx_base = torch.arange(0, batch_size, device=device).view(-1, 1, 1)*num_points idx = idx + idx_base idx = idx.view(-1) _, num_dims, _ = x.size() x = x.transpose(2, 1).contiguous() # (batch_size, num_points, num_dims) -> (batch_size*num_points, num_dims) # batch_size * num_points * k + range(0, batch_size*num_points) feature = x.view(batch_size*num_points, -1)[idx, :] feature = feature.view(batch_size, num_points, k, num_dims) x = x.view(batch_size, num_points, 1, num_dims).repeat(1, 1, k, 1) feature = torch.cat((feature-x, x), dim=3).permute(0, 3, 1, 2).contiguous() return feature class DGCNN(nn.Module): ... def forward(self, points): # DGCNN forward x = get_graph_feature(x, k=self.k) x = self.conv1(x) x1 = x.max(dim=-1, keepdim=False)[0] # Stack four times ... return x1</pre>
--	--

Figure 9: Case studies for the ideas and codes generated by DOLPHIN (Left) and human researcher (Right).



Case 1

"Title": "Latent Space Exploration for Enhanced Point Cloud Classification"

"Experiment": "Enhance the PointNet architecture with a latent space exploration module using an autoencoder framework. Map point cloud data to a latent space, perform exploration using auxiliary objectives such as clustering or reconstruction, and identify meaningful latent structures. Combine these latent features with the original point cloud representation to enrich feature extraction. Evaluate the modified network on the ModelNet40 dataset, comparing its performance against the baseline PointNet, focusing on improvements in classification accuracy by leveraging latent structures."

"Summary": "Introduce a latent space exploration mechanism in PointNet using an autoencoder to enhance classification by discovering and integrating hidden structures within point clouds."

```
class PointNetEncoder(nn.Module):
    def __init__(self, global_feat, feature_transform, in_channels):
        ...
        self.encoder = nn.Sequential(
            nn.Linear(1024, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, 128)
        )
        self.decoder = nn.Sequential(
            nn.Linear(128, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 1024)
        )

    def forward(self, x):
        ...
        latent = self.encoder(x)
        reconstructed = self.decoder(latent)
        ...

class Model(nn.Module):
    ...
    def forward(self, points)
        ...
        reconstruction_loss = F.mse_loss(reconstructed, original_x)
        total_loss = cls_loss + mat_diff_loss * self.mat_diff_loss_scale + reconstruction_loss
```

Figure 10: Idea and codes generated by DOLPHIN which achieves 92.34% OA and 89.54% mAcc. on ModelNet40 (+1.34% OA and +1.94% mAcc. compared to our baseline).

Case 2

"Title": "Contrast Enhancement Module for Improved Point Cloud Classification"



"Experiment": "Develop a Contrast Enhancement Module within the PointNetEncoder. Implement a mechanism to compute contrast levels between features in the point cloud and amplify differences by boosting high-contrast regions. Design functions to integrate this enhanced feature contrast into the existing feature extraction process. Evaluate the modified network on the ModelNet40 dataset, comparing its performance against the baseline PointNet, focusing on improvements in classification accuracy and robustness."

"Summary": "Introduce a Contrast Enhancement Module in PointNet to improve classification by amplifying feature contrasts, enhancing model robustness and accuracy."

```
class PointNetEncoder(nn.Module):
    ...
    def compute_contrast(self, x):
        # Compute contrast levels between features in the point cloud
        mean = torch.mean(x, dim=-1, keepdim=True)
        std = torch.std(x, dim=-1, keepdim=True)
        contrast = (x - mean) / (std + 1e-8)
        # Apply adaptive contrast normalization
        normalized = self.adaptive_contrast_normalization(contrast)
        return normalized

    def amplify_contrast(self, x):
        # Compute dynamic alpha based on local contrast distribution
        contrast = self.compute_contrast(x)
        std_dev = torch.std(contrast, dim=-1, keepdim=True)
        alpha = 1 + (std_dev / torch.mean(std_dev))
        amplified = torch.where(contrast>0, contrast*alpha, contrast/alpha)
        # Apply adaptive contrast normalization
        normalized = self.adaptive_contrast_normalization(amplified)
        return normalized

    def forward(self, x):
        # Apply dynamic contrast enhancement
        x = self.amplify_contrast(x)
        x = self.conv_layer3(x)
        # Apply adaptive contrast normalization
        x = self.adaptive_contrast_normalization(x)
        x = self.conv_layer4(x)

    def adaptive_contrast_normalization(self, x):
        # Normalize feature maps based on local contrast distribution
        mean = torch.mean(x, dim=-1, keepdim=True)
        std = torch.std(x, dim=-1, keepdim=True)
        normalized = (x - mean) / (std + 1e-8)
        return normalized
```

Figure 11: Idea and codes generated by DOLPHIN which achieves 92.30% OA and 88.96% mAcc. on ModelNet40 (+1.30% OA and +1.36% mAcc. compared to our baseline).

Case 3

"Title": "Bio-Inspired Filter Module for Enhanced Image Classification"



"Experiment": "Integrate a Bio-Inspired Filter Module (BIFM) within the WideResNet architecture. Develop a set of edge and pattern detection filters inspired by simple and complex cells in the human visual cortex, using Gabor filters as a starting point due to their similarity to visual cortex receptive fields. Organize these filters hierarchically and integrate them with existing convolutional layers to enhance feature extraction. Evaluate the performance of this augmented model on CIFAR-100, comparing it with the baseline WideResNet model."

"Summary": "Introduce a Bio-Inspired Filter Module in CNNs to enhance image classification by simulating the hierarchical and specialized processing of visual information observed in biological systems."

```
class GaborFilter(nn.Module):
    def __init__(self, kernel_size, sigma, frequencies, thetas):
        super(GaborFilter, self).__init__()
        self.kernel_size = kernel_size
        self.sigma = sigma
        self.frequencies = frequencies
        self.thetas = thetas
        self.gabor_kernels = nn.ParameterList([self._create_gabor_kernel(f, t)
                                                for f, t in zip(frequencies, thetas)])

    def _create_gabor_kernel(self, frequency, theta):
        x, y = np.meshgrid(np.arange(-self.kernel_size // 2 + 1, self.kernel_size // 2 + 1),
                           np.arange(-self.kernel_size // 2 + 1, self.kernel_size // 2 + 1))
        x_theta = x * np.cos(theta) + y * np.sin(theta)
        y_theta = -x * np.sin(theta) + y * np.cos(theta)
        kernel = np.exp(-0.5 * (x_theta**2 + y_theta**2) / self.sigma**2) *
                 np.cos(2 * np.pi * frequency * x_theta)
        return torch.tensor(kernel, dtype=torch.float32).unsqueeze(0).unsqueeze(0)

    def forward(self, x):
        filtered_outputs = [F.conv2d(x, kernel.to(x.device),
                                     padding=self.kernel_size // 2) for kernel in self.gabor_kernels]
        return torch.cat(filtered_outputs, dim=1)

class WideResNet(nn.Module):
    def __init__(self, depth, widen_factor, num_classes, dropout_rate):
        ...
        self.gabor_filter = GaborFilter(kernel_size=5, sigma=1.0,
                                         frequency=0.1, theta=0.0)

    def forward(self, x):
        out = self.gabor_filter(x)
        ... (Original codes)
```

Figure 12: Idea and codes generated by DOLPHIN which achieves 82.05% Acc. on CIFAR-100 (+0.85% Acc. compared to our baseline).