The University of Hong Kong

Department of

Statistics and Actuarial Science

DASC7600 Data Science Project

LinguAML: Automated Machine Learning with Integrated Large Language Models

Author
FEI Yiheng 3036044533
ZHANG Kai 3036044806
ZHANG Yupeng 3036044090
SUN Gengchen 3036044521

Supervisor
Dr. KONG Lingpeng

December 6, 2023

Contents

1 Introduction

1.1 Background

AutoML, short for Automated Machine Learning, traces its origins back to a novel concept introduced in the academic world in 2012 known as "Programming by Optimization" (PbO). This concept aimed to address the problem of manually tuning parameters in software development by delegating this task to machines. The idea was that if machines could autonomously tune parameters, it would significantly reduce human effort, allowing people to focus on more creative tasks. However, enabling machines to perform self-tuning remained a challenging problem. Consequently, after the concept of AutoML was introduced, it didn't gain much attention initially.

It wasn't until 2018, during the Google Cloud Next conference, that Google's Chief Scientist of AI and Machine Learning, Fei-Fei Li, announced Google AutoML Vision entering public testing and the launch of two new AutoML products: AutoML Natural Language and AutoML Translation. These products were designed to make Google's AI technology more accessible to individuals with limited knowledge of machine learning, thereby lowering the barrier to entry. Specifically, AutoML Natural Language could analyze the structure and meaning of text, making it capable of extracting information about individuals, locations, events, and more from text documents, news, or blog posts. AutoML Translation, on the other hand, employed the latest neural machine translation techniques to translate strings into any supported language. This marked the beginning of AutoML becoming more widely known and accessible to the public.

With the release of Google AutoML, the industry dubbed it a strategic shift for Google Cloud. Google Cloud, which had traditionally catered to machine learning and AI developers, was now expanding its services to a broader audience. In essence, Google AutoML became a "development tool." Even individuals with limited knowledge of AI could upload labeled data to the Google AutoML platform and obtain a customized machine learning model. The entire process, from data import to labeling and model training, could be accomplished through a user-friendly drag-and-drop interface.

Initially, Google AutoML's services only supported computer vision models, but Google indicated plans to support all standard machine learning models, including speech, translation, video, natural language processing, and more. In fact, when Google AutoML was released, companies like Disney were already testing the service. Mike White, CTO and Senior Vice President of Disney Consumer Products and Interactive Media, mentioned how Cloud AutoML's technology helped them label their products based on Disney characters, product categories, and colors. These labels were integrated into their search engine to enhance user experience in the shopDisney store.

Not only abroad but also within China, numerous companies were exploring AutoML. According to a June 2021 report by IDC China, the overall market size of AI public cloud services in China reached 2.41 billion RMB in 2020, accounting for 10.4% of the overall AI software market. It was projected that by 2025, AI public cloud services would account for 36.1% of the AI software market in China. In the landscape of China's AI public cloud services in 2020, Baidu AI Cloud, Alibaba Cloud, Tencent Cloud, and Huawei Cloud were the top four players. These companies introduced self-learning platforms in various technical domains as different forms of AutoML products.

Baidu, with years of accumulated expertise in AI, played a significant role in this landscape. Baidu's AutoML platform was named "EasyDL" and was tailored for deep learning model training and deployment. Prior to EasyDL, Baidu had developed the deep learning computing engine, PaddlePaddle, targeting professional AI algorithm engineers with some computer science and algorithmic knowledge. Additionally, Baidu had an AI open platform where users could pay to access Baidu's AI algorithm capabilities via provided APIs. However, this open platform couldn't cover all scenarios, leading to unmet customization demands. Hence, Baidu introduced EasyDL to address industry pain points and fulfill the need for customized deep learning models. Currently, EasyDL offers classic, professional, and retail versions, providing services like image recognition, text classification, sound classification, and video classification. According to official figures, EasyDL has garnered over 900,000 users and is applied in over 20 industry sectors, including retail, healthcare, agriculture, and industry. From a technological and application perspective, Baidu's EasyDL platform has consistently ranked among the top players in both domestic and international tech enterprises. Alibaba, Tencent, and other prominent Chinese tech companies also launched similar services to keep pace with developments.

Alibaba Cloud's Machine Learning PAI (Platform of Artificial Intelligence) is an all-in-one machine learning platform. PAI offers a range of industrial-grade model deployment solutions, from automated model parameter tuning to one-click deployment and online streaming computing services. PAI-AutoML supports several parameter tuning methods, such as custom parameters, grid search, random search, and evolutionary algorithms, catering to diverse tuning requirements. PAI's auto-tuning functionality enables both beginners without a deep understanding of algorithms and seasoned algorithm engineers to deploy models easily. Customizable parameter tuning replaces repetitive labor in parameter details.

Apart from major companies, independent AI firms in China have also launched AutoML platforms. For example, Fourth Paradigm has productized AutoML algorithms and introduced an automated AI productivity platform called Sage HyperCycle ML.According to Fourth Paradigm, HyperCycle ML simplifies the cumbersome process of building machine learning applications into four steps: behavior, feedback, learning, and application. Users can easily initiate continuous learning and prediction services for an automated machine learning circle by configuring simple settings, making AI products accessible to business professionals. Currently, Fourth Paradigm's HyperCycle ML platform is mainly used in precision marketing, delinquency prediction, anti-fraud, anti-money laundering, and similar fields. Key clients include Chinese banks like Industrial and Commercial Bank of China and Guangfa Bank.

Language Model Learning (LLM) is currently a hot research topic in the field of NLP (Natural Language Processing). LLM represents a new era in NLP, allowing computers to understand natural text in a way that closely resembles human cognition, achieved through attention mechanisms and probabilistic generation. Looking at the historical development of NLP, it can be divided into three stages:

- 1. Stage one (until the 1970s): Rule-based models dominated, relying on manually crafted rules for natural language processing but limited in handling large datasets.

- 2. Stage two (1970s-2000s): Statistical models, such as N-Gram, predicted word occurrence probabilities from a mathematical statistical perspective. These models had intuitive inference but were highly influenced by dataset quality, leading to issues like data sparsity.
- 3. Stage three (2000s to the present): Neural network-based models began learning like human brains. Before 2017, these models were smaller in scale, but with the release of Transformer in 2017, they started training on massive datasets, entering the era of large language models (LLMs). With some human intervention, model performance reached new heights.

2 Related Work

In this section, we will go through all relevant work in the field of the Large Language Model, as well as that related to Automated Machine Learning, and that which combines the techniques from both of the above fields.

2.1 Large Language Models

2.1.1 Generative Pre-trained Transformers (GPT-series)

The GPT (Generative Pretrained Transformer) model [1] is one of the state-of-the-art language generation models developed by OpenAI. It uses a multi-layer Transformer decoder-only architecture in the auto-regression way to generate natural language text.

The decoder-only block in GPT has the Mask Multi-Head Self-Attention mechanism which is used to help the model generate text by attending to relevant parts of the input sequence. The feed-forward neural network in each decoder-only layer applies a non-linear transformation to the output of the Mask Multi-Head Self-Attention mechanism, allowing the model to capture complex relationships between different parts of the input sequence. The layer normalization and residual connections in GPT help to improve the stability and performance of the model by normalizing the activation of each layer and mitigating the vanishing gradient problem, respectively.

Reinforcement Learning with Human Feedback (RLHF) is the technique that is used in the instructGPT which is also known as GPT3.5. This technique has enabled language models to learn about aligning complex human concepts and values. It could be divided in 3 steps:

1. Supervised-Fine Tuning (SFT): Using the prompt and corpus as input, supervised fine-tuning the GPT model with the desired output. The best SFT model will be selected in the second step by the Reward Model score on the validation set.

2. Reward Model (RM) Training: The purpose of this step is to get a scaling human-evaluate-like reward for training in RL. With the particular prompt, several model outputs are generated. Then a labeler will rank these outputs from best to worst and they will be used to train the reward model by utilizing the pairwise loss function:

$$Loss(\theta) = -\frac{1}{\binom{N}{2}} E_{(x,y_w,y_l)\sim D}[log(\sigma(r_\theta(x,y_w) - r_\theta(x,y_l)))] \tag{1}$$

which $r_\theta(x,y)$ is the scalar output of the reward model for prompt x and response y with parameters θ, y_w is the preferred completion out of the pair of y_w and y_l, and D is the dataset of human comparisons. For this pairwise loss function $-\log(\sigma(r_j - r_k))$, we could observe that as r_j becomes larger, the pairwise loss will become smaller; as r_k becomes larger, the pairwise loss will become larger. We already know that j is better than k, so this loss function will cause the model to pull r_j and r_k further and further apart, making r_j larger and r_k

smaller. Therefore, this r_i from RM is the scaling reward aligning the human values and concepts for the model output text i.

3. Reinforcement Learning: Let the Large Language Model learn about aligning complex human concepts and values. With a sample prompt, update the policy (selected LLM) through PPO algorithm according to the scaling reward score from trained RM in step 2. Then repeat the process on different samples of prompts, until the reward score is globally maximized.

Thus, GPT with the Transformer decoder is a powerful model for generating natural language text. It allows the model to capture complex relationships between different parts of the sequence and generate proper text following human instructions. We utilize it as one of our backbone candidates to accomplish the AutoML task with its semantic understanding, logical reasoning, content production and other capabilities.

2.1.2 Large Language Model Meta AI (LLaMA-series)

LLaMA [2] is the open-source large language model developed by MetaAI. It has the similar structure to GPT with decoder-only blocks. However, there are some improvements and replacements in the detailed structure:

- LLaMA uses Rotary Positional Embeddings(RoPE) [3] instead of absolute positional embedding. The core idea of RoPE is to realize relative positional encoding by means of absolute positional encoding, which can be said to have the convenience of absolute positional encoding. Also, it could represent the relative positional relationship between different tokens. Unlike the original Transformers paper, which adds positional embedding and token embedding, RoPE multiplies positional encoding and query (or key).

- LLaMA uses SwiGLU [4] activation function instead of ReLU. Because the SwiGLU could obtain the optimal value of log-perplexity compared to other activation function variants.

- LLaMA uses Root-Mean-Squared Layer-Normalization [5] instead of standard layer-normalization. To help improve the stability of training, LLaMA implements the normalization on the input of each transformer layer instead of the output. The main difference between RMS layer-Norm with standard layer Norm is that the part of subtracting the mean is removed. This layer-Norm simplifying could reduce the computation time by about $7\% \sim 64\%$.

2.1.3 Artificial General Intelligence systems (AGI-systems)

Artificial General Intelligence (AGI) is defined as AI systems to accomplish not specific tasks but become more general as the Intelligent Agents. Based on the powerful semantic understanding, logical reasoning, content production and other capabilities of Large language models, the AGI field is also booming, with a number of novel and innovative technologies and projects in the last few months:

- Chain of Thoughts (CoT): CoT [6] is the implementation of a series of intermediate reasoning steps. The core improvement of CoT is to write out these reasoning steps (manually constructed) in the Answer part of the example and then give the final answer. It stimulates the process of logical thinking. The logic is that you want the model to learn to output the reasoning steps step by step and then give the result.

- ReAct: ReAct [7] is essentially a paradigm that combines Reasoning and Acting. The reasoning process is shallow and easy to understand, containing only thought-action-observation steps, and it is easy to determine the correctness of the reasoning process. For Chain of Thought reasoning, it is a static black box, it does not use external knowledge, so the problem of fact hallucination and error propagation occurs in the reasoning process. Meanwhile, Act-only approach obviously does not utilize the reasoning ability of LLM. ReAct overcomes the problem of hallucination and error propagation in problematic CoT reasoning by interacting with external APIs to generate the human-like task-solving path, with further enhancement of interpretability.

- Reflection (Generative Agents): Generative Agents [8] is a project to create a virtual town with 25 agents living and interacting. These agents are based on the GPT4 with its powerful skills and it implements the memory, reflection, planning and reacting parts for each agent. For the reflection block, it contains higher level, more abstract thoughts generated by the agent. Since Reflection is also a type of memory, it is also included in observations when retrieved. The first step in reflection is to identify questions that can be asked based on the most recent experiences of the agents, and thus what to reflect on. The agents reflect not only on observations but also on further reflections based on the results of other reflections. Eventually, the agents generate a reflection tree: leaf nodes represent basic observations, while non-leaf nodes represent thoughts. As a non-leaf node becomes higher, the thought it represents becomes more abstract and advanced.

- HuggingGPT: HuggingGPT [9] uses LLM as a controller to manage and organize these expert models (HuggingFace's text-to-pic, pic-to-text, target detection models, etc.). LLMs first plan a list of tasks according to the user's request and then assign an expert model to each task. After the execution of the expert model is completed, LLM collects the results and returns them to the user. The overall architecture of HuggingGPT is that LLM as the core controller and the model as the executor. The execution of HuggingGPT is divided into four phases: 1) Task Planning: LLM decomposes the user's request into a collection of tasks and decides the order of execution and the resource dependency between tasks; 2) Model Selection: Based on the description of HuggingFace's model, LLM arranges suitable models to tasks; 3) Task Execution: the model executes the assigned task; 4) Response Generation: LLM collects the results of experts and generates a summary of workflow logs to return to the user.

2.2 Automated Machine Learning (AutoML)

For the traditional AutoML field, there exist approximate two part:

- Hyperparameter Optimization: There exist some blackbox optimization techniques based on model-free methods and also Bayesian optimization (e.g. Grid search, Random search, Bayesian search). Since pure black-box optimization is extremely costly due to the high computational demands of many modern machine learning models, the researchers also focus on modern multi-fidelity methods that use cheaper variants of black-box functions to approximate the quality of the hyperparameter settings [10].

- Meta-Learning: Meta-learning [10], also known as learning to learn, is the science of systematically observing the performance of different machine learning models on a variety of tasks. And then learning from that experience or metadata to learn new tasks at a much faster rate than other methods. This not only significantly speeds up and improves the design of machine learning pipelines or neural architectures, but also allows system to replace hand-designed algorithms with new data-driven approaches.

2.2.1 Auto-sklearn

Feurer et al. [11] proposed a project called Auto-sklearn which implements an efficient and robust automated machine learning framework based on the Python machine learning package scikit-learn.

In the realm of model selection, Auto-sklearn employs meta-learning, while for hyperparameter optimization, its core methodology is Sequential Model-based Algorithm Configuration (SMAC). SMAC is an optimization approach based on sequential model-based optimization (SMBO), utilizing historical data to construct a model for predicting performance, subsequently leveraging this model to guide the direction of future searches. Particularly apt for dealing with optimization problems characterized by high dimensionality and complex search spaces, SMAC is quintessential in the hyperparameter optimization of machine learning models. It models the performance of various parameter configurations, typically using Gaussian processes or random forests, and selects configurations projected to yield the most significant improvements in unexplored parameter spaces. In summary, Auto-sklearn amalgamates strategies from Bayesian optimization, meta-learning, and ensemble learning, effectively navigating the intricate hyperparameter space with SMAC. The synthesis of these methodologies renders Auto-sklearn a powerful and versatile AutoML tool, capable of generating superior machine learning models across a broad spectrum of datasets.

2.3 Traditional Hyperparameter Tuning Method

When training machine learning models, the selection of hyperparameters is crucial to obtaining high-efficiency performance. Traditionally, several methods, such as grid search, random search, and Bayesian optimization, have been frequently applied for this purpose. Grid search exhaustively traverses the entire space of hyperparameters and tests all possible combinations, and then selects the setting with the best performance. While with grid search, the global optimal solution is guaranteed to be found, it can be rather computationally expensive, especially in high-dimensional spaces where there exist numerous hyperparameters to tune [12]. As an alternative to grid search, random search, only samples hyperparameters from predefined distributions. It offers significant time savings, though at the potential risk of overlooking the globally optimal hyperparameters. However, it often yields satisfying results in practice, and it is widely applied [12]. Bayesian search is another popular choice. It searches the hyperparameters by constructing a probabilistic model of the function mapping from hyperparameters to a validation dataset score, which intelligently samples the hyperparameters that are likely to yield improved results [13].

2.4 AutoML with RL Frameworks

2.4.1 Hyp-RL

Using a reinforcement learning (RL) framework, Hyp-RL is a unique technique to hyperparameter tuning in machine learning models, as described in [14]. According to the authors, choosing which hyperparameter to test next in a series of decisions is an important aspect of hyperparameter tuning. To maximize hyperparameter selection, they simulate this problem using reinforcement learning, more especially Q-learning. Their system is based on Q-learning and is called Hyp-RL. This strategy learns from the performance of prior hyperparameter configurations to explore high-dimensional hyperparameter spaces. Their model's state representation comprises a history of assessed hyperparameters and their accompanying performances, as well as dataset meta-features. They used a meta-dataset made up of 50 UCI classification datasets to conduct their studies. The outcomes demonstrated that Hyp-RL performs better in hyperparameter optimization tasks than conventional techniques like grid search and random search. The main benefit of Hyp-RL is that, unlike Bayesian optimization techniques, it can learn an effective policy for choosing hyperparameters without the need for heuristic acquisition functions. This method performs well across a variety of datasets and is scalable. The primary drawback of this strategy is that it necessitates the computationally demanding training of an RL agent. Furthermore, the quality of the learned RL policy determines performance and can be affected by the dataset's and the hyperparameter space's complexity.

2.4.2 Meta-RL

Meta-RL is persented by [15] which explores a different approach to utilizing reinforcement learning for automated hyperparameter optimization (HPO) in machine learning. It suggests a task-aware representation for a context-based meta-reinforcement learning (meta-RL) model. The purpose of this model is to address the limited generalization and data inefficiency problems in HPO. The main breakthrough lies in the separation of task inference and control, which enhances the effectiveness of meta-training and expedites the acquisition of new tasks. For on-the-fly inference of task properties, including both dataset representation and task-solving experience, the method uses a task encoder. This encourages more astute exploration techniques. This task inference is then used by the policy learner to inform decisions. Additionally, the technique employs amortized meta-learning, which is said to be less complicated and quicker than gradient-based meta-training techniques, to train the RL agent. The experimental findings presented in this paper demonstrate that their approach can quickly and efficiently find the best hyperparameter configurations while requiring the least amount of computing power. They outperform other sophisticated optimization techniques as well as more conventional approaches like grid search and random search. The focus on task-specific data and the approach's flexibility in quickly adjusting to new tasks because of the task-aware representation make it novel. As a result, the HPO process is more effective and broadly applicable. One potential limitation of the approach could be its complexity, as it necessitates the understanding and fine-tuning of multiple components, including the task encoder and policy learner. Furthermore, the effectiveness of the meta-training procedure and the caliber of the task representation play a major role in this approach's success.

2.4.3 Comparison

While both papers leverage reinforcement learning for automating hyperparameter optimization, their methods, focuses, and innovations differ significantly, offering distinct advantages and facing unique challenges in the AutoML landscape.

Similarities

Both papers aim to enhance HPO using reinforcement learning. They share the common goal of overcoming the limitations of traditional HPO methods (like grid or random search) by using more sophisticated, learning-based approaches.

Differences

The first paper focuses on a Q-learning based approach (Hyp-RL), modeling hyperparameter tuning as a sequential decision-making problem. In contrast, the second paper uses a context-based meta-RL with a task-aware representation, focusing on disentangling task inference and control for efficient meta-training.

Strengths and Weaknesses

Hyp-RL's strength lies in its simplicity and direct approach to learning an efficient policy for hyperparameter selection, whereas the context-based meta-RL method excels in task adaptability and efficiency in learning for new tasks. However, Hyp-RL may struggle with complex hyperparameter spaces and diverse datasets, while the context-based meta-RL's complexity and reliance on the quality of task representation can be challenging.

2.4.4 LLM-Enabled Model Selection

Facing a traditional machine learning task, e.g., build a predictive model on a tabular dataset, one must firstly decide the specific model to apply based on the nature of the task. This process is usually called model selection. In current automated machine learning algorithms, the automation of model selection is usually done by constructing search algorithms based on the meta-feature similarities between different machine learning tasks [10]. In this process, it is feasible to involve LLMs in this process in replace of traditional search algorithms to improve the efficiency and quality of the decision. In this report, we will leverage prompt engineering methods to equip LLMs with the capability to decide the most suitable model for specific machine learning tasks based on meta-feature similarities. The overall structure of the model selection module can be illustrated in Fig. 1.

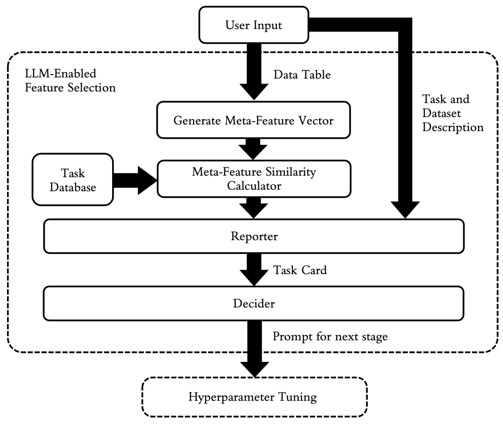

User Input
LLM-Enabled Feature Selection
Data Table
Task and Dataset Description
Generate Meta-Feature Vector
Task Database
Meta-Feature Similarity Calculator
Reporter
Task Card
Decider
Prompt for next stage
Hyperparameter Tuning

Figure 1: The working flow of the LLM-enabled model selection module

- Meta-feature similarity: Meta-feature similarity measures the similarity of the nature of different tasks. When people selecting models for specific machine learning tasks, they usually make their decision based on matching the nature of the task with the feature of the candidate machine learning models or similar tasks that they conducted previously. This indicates the idea of learning from experience. To quantify these experiences and make them understandable for LLMs, we can extract some descriptive statistics from the dataset. These features are called meta-features, which serve as features that distinguish different tasks from each other. Based on the meta-features vectors, we can calculate the meta-feature similarities by pre-defined algorithm, for instance, Euclid distance. Tasks with high meta-feature similarities are more possible to belong to the same class of problem with similar machine learning models. This criterion will highly facilitate the LLMs to make better decision based on past experiences. To achieve this, we will construct a database to store tasks along with the model used for the task, the description of dataset and features, the target of the tasks, and their meta-feature vectors. When a new task is passed to the module, the meta-feature similarity calculator will calculate the meta-feature similarity between the new task and all tasks in the database and return the tasks with top-5 meta-feature similarity in the database. This information will then be passed to the LLM agent for generating task card.

- Task card: The task card is a piece of well-structured prompt texts that contains all necessary information of the input new task. It serves as the identification of a task, which will help it distinguish itself from other tasks. It will be generated by the LLM agents involved in this module based on user inputs and the meta-feature similarities. The task card is of vital importance to improve the decision-making performance of LLM agents by fully utilizing their capability of understanding natural languages. The basic information about the task and the dataset will be passed to the LLM agents through task card for better understanding of the task and the structure of the dataset. Besides, the descriptions of the dataset and features contained in the task card will be used later in hyperparameter tuning (will be introduced in following sections).

- Selector agent construction: The model selection module is composed by two LLM agents. The first agent ("reporter") will directly take the user input and the result of the meta-feature similarity calculator, and utilize the information to construct the task card, and then pass it to the second agent for decision making. The second agent ("decider") will understand the task card and make a decision. After that, the decider will generate a piece of prompt text that combines the information from the task card and the information of the decided machine learning model. The latter includes the description of the model, the hyperparameter involved in this model, and the commonly used evaluating criteria for this model. The generated prompt will then be passed to the hyperparameter tuning module. The training of LLM agents is mainly through prompt engineering [16] [17]. We will compose stable prompt texts to initialize the LLM agents to act like the description above.

2.5 AutoML with LLMs

Recently, with the rise of LLMs, there have also been a number of researchers focusing on applying the powerful capabilities of LLM to achieve some breakthroughs in the field of AutoML.

Zhang et al. [18] propose a novel model called AutoML-GPT which uses GPT as a bridge between various AI models and optimized hyperparameters to dynamically train the ML models and accomplish different tasks. AutoML-GPT dynamically receives user requests from Model Card, Data Card and composes corresponding prompt paragraphs. It uses the prompt paragraphs to automatically conduct experiments, which the processing steps are:

1. Generate fixed-format prompt paragraphs from Model Card and Data Card.
2. Builds a training pipeline to process user descriptions on selected datasets and model architectures.
3. Generate performance training logs and tune hyperparameters.
4. Tuning models to auto-suggested hyperparameters.

The predicted training logs on the dataset record a variety of metrics and information which helps to understand the progress of model training, identify potential problems, and evaluate the effectiveness of the selected architecture, hyperparameters, and optimization methods.

There are also other works like: Zhang et al. [19] proposed a novel framework called MLCopilot, which leverages the state-of-the-art LLMs to develop ML solutions for novel tasks; HASSAN et al. [16] present a model to deal with four kinds of tasks: Data Visualization, Task Formulation, Prediction Engineering, and Result Summary and Recommendation.

3 Methodology

3.1 Paradigms

While in our project, we mainly set a focus on traditional machine learning and statistical models like SVM and random forests, etc. The underlying principles of reinforcement learning framework are adopted.

In framing hyperparameter tuning for a specific task as a reinforcement learning problem [20], the action, in this context, represents a vector of hyperparameters of a specific model for tuning. In terms of AutoML, the action space, in this case, coincides with the search space, and is denoted by

$$A = \{(\theta_1, \ldots, \theta_n) \mid \theta_i \in \Theta_i\}$$

where each θ_i is the hyperparameter from the space Θ_i. An RL agent is created to interact with the environment, which in return receives a state, represented by a sequence of actions and their associated rewards, which are determined by the model's performance metric with the selected hyperparameters. Formally,

$$s_t = ((a_0, r_0), \ldots, (a_t, r_t))$$

where s_t is the state at time t, and a_i and r_i are action and the associated reward from previous steps. The goal is to train the agent to learn a policy π^* that maximizes the expected discounted cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{t=0}^{T} R(s_t, a_t)]$$

where R is the reward function, and T is the maximum number of hyperparameter selections for training the current model. The suggested action by the policy network π_t during the training process is denoted by a_t^{RL}, which we should emphasize that is different from a_t. This is because the RL agent is envisioned to seamlessly collaborate with an LLM agent, details of which will be discussed in a later section.

Apart from adjusting hyperparameters in a trial-and-error fashion solely relying on the RL agent, it is believed that the practical meanings of hyperparameters are also of significant importance. The LLM is used here to capture this information since it is embedded in texts. The LLM agent, which we create to learn the meanings of hyperparameters, is then asked to select a setting of new hyperparameters based on the current state. This action taken by LLM agent is denoted as a_t^{LLM}.

3.2 Hyperparameter Tuning with RL Framework

3.2.1 The structure of Agent

For the Agent module in the RL Framework, we have the structure like the LSTM which could deal with the time-series information in the state, which is actually the previous L pairs of hyperparameters of the ML model (Family) and the corresponding accuracy (Reward). Meanwhile, the following part contains the module to generate the distribution parameters of the new Family hyperparameters which are actually the feed-forward networks. Then there is a sampling part for hyperparameters by the corresponding distribution. We use the sampled hyperparameters to train the ML models to get the accuracy which is actually the reward in the RL framework.The overall structure of the Agent has been shown in Fig. 2.

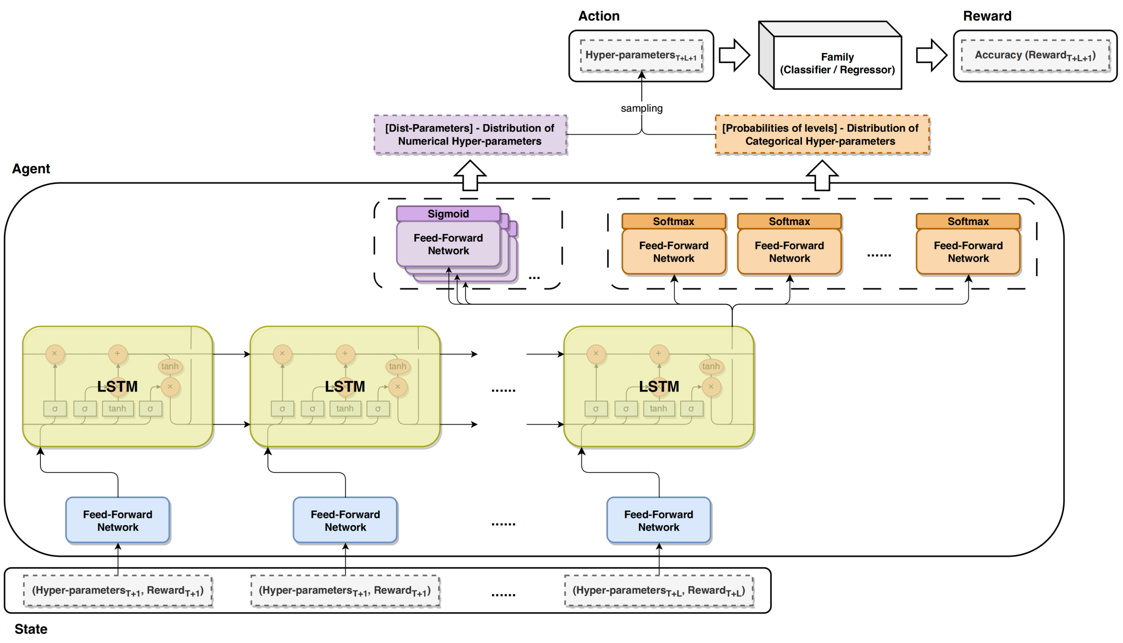

Action
Hyper-parameters$_{T+L+1}$
Family (Classifier / Regressor)
Reward
Accuracy (Reward$_{T+L+1}$)
sampling
[Dist-Parameters] - Distribution of Numerical Hyper-parameters
[Probabilities of levels] - Distribution of Categorical Hyper-parameters
Agent
Sigmoid
Feed-Forward Network
Softmax
Feed-Forward Network
LSTM
Feed-Forward Network
(Hyper-parameters$_{T+1}$, Reward$_{T+1}$)
(Hyper-parameters$_{T+L}$, Reward$_{T+L}$)
State

Figure 2: The structure of the Agent in RL Framework

Unlike the DL part design in previous work [14][15], we use the historical L hyperparameters and the reward pairs, which are defined as the state in our RL framework, and processed by a feed-forward network and LSTMs to make the agent capture the past tuning cases and the information of the performance for different hyperparameter combinations. Besides the advantage of using LSTM to process historical cases, which could keep track of important patterns and information from the past and be more effectively, this structure could also make our training process smoother and much quicker to get a satisfactory performance by providing sufficient information. Meanwhile, without utilizing only one previous sample to deliver the past tuning information, the model with a time window would make our model more robust without being particularly computationally expensive.

Then we construct two groups of feed-forward networks that generate the parameters of distributions for numerical and categorical hyperparameters respectively. For the numerical hyperparameter group, we choose three kinds of the distribution which are: *Normal, Cauchy and Beta*. By utilizing these three kinds of two-parameter distribution, for each numerical hyperparameter, we construct two feed-forward networks to generate their distribution parameters. For consistency and excluding the effects of scaling, we utilize a sigmoid function followed by each feed-forward network in the numerical group. For the categorical hyperparameters, we construct the specific feed-forward network for each of them with the dimension of the last hidden layer as the number of the levels in this hyperparameter. Certainly, it was followed by a softmax layer to normalize these outputs into a categorical probability distribution which is sum to 1. It should be noted that, all these distribution parameters will also be utilized when we calculate the log probabilities for gathering the PPO loss.

After gathering these distribution parameters, we sample the new hyperparameter based on the corresponding distributions. Sampling from the distribution to get the selected hyperparameters could bring the randomness and make the whole AutoML framework more robust. Because we could guarantee that the hyperparameter combinations with satisfactory performance are no accidental phenomena and after the performance (reward) converging, we could have high confidence to conclude that this is the ability boundary of current ML model on specific dataset.

Then, based on the sampled hyperparameter combinations, we could tune the families whatever classifier or regressor by these sampling hyperparameters and evaluate it on the validation set to get the accuracy which is actually the reward in the RL framework.

3.2.2 Gathering the Replay Buffer (Sampling Transactions)

Delivering an overview of the RL framework, the first step is to gather the replay buffer, which is actually sampling a large number of transactions for subsequent agent training. The transactions in the replay buffer actually contain not only states and actions but also the log-probablities and the advantage which is actually the difference between the current reward and the baselines which is calculated by the time series average algorithm. The agent and environment flow structure is similar to the common one in RL, which is agent takes an action and the environment gives the agent feedback, reward, and the current state of the agent. The overview of step one to sample the transactions has been shown in Fig. 3.

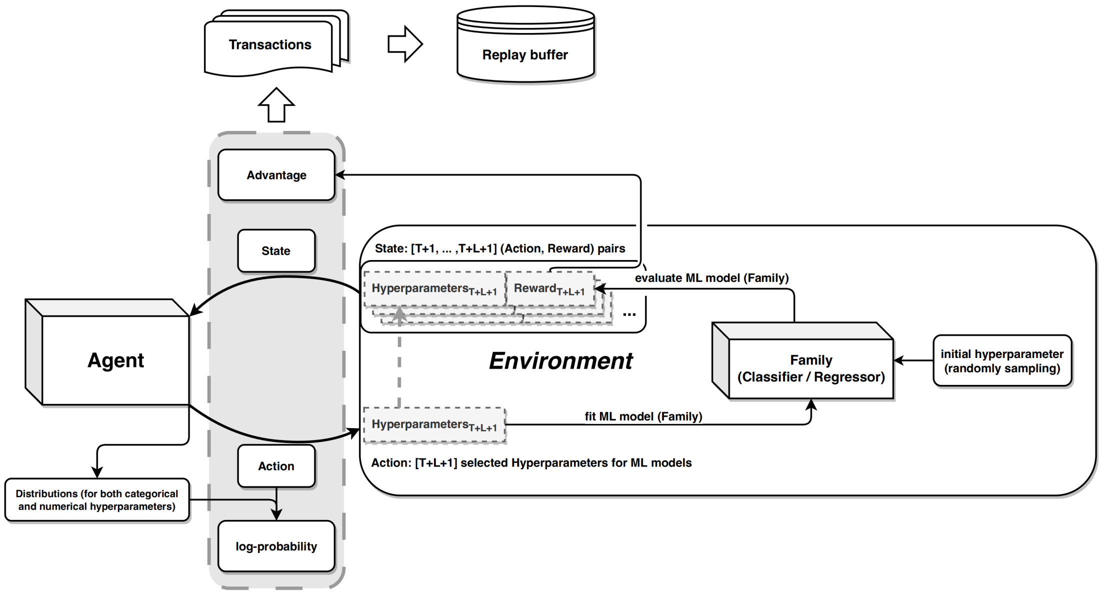

Transactions
Replay buffer
Advantage
State
State: [T+1, ... ,T+L+1] (Action, Reward) pairs
Hyperparameters$_{T+L+1}$
Reward$_{T+L+1}$
evaluate ML model (Family)
...
Agent
Environment
Family
(Classifier / Regressor)
initial hyperparameter
(randomly sampling)
Hyperparameters$_{T+L+1}$
fit ML model (Family)
Action
Action: [T+L+1] selected Hyperparameters for ML models
Distributions (for both categorical and numerical hyperparameters)
log-probability

Figure 3: Step-1 : Gathering the Replay Buffer (Sampling Transactions)

At the stage of initialization, the environment will first sample some hyperparameters to tune the ML models and evaluate them on the validation set to get the initial hyperparameters and the reward pairs. This random initialization not only guarantees the workflow is functioning properly but also brings the randomness for every episode to make the agent take the exploration actions instead of always being in the exploitation stage. Meanwhile, the advantage is calculated by the current reward. We first use the average algorithm for time series, like MA (Moving Average) and EMA (Exponential Moving Average) to calculate the baseline reward from the previous tuning cases and then gather the difference between the current reward and the baselines which is the advantage. The formula of the advantage with Moving Average algorithm is like:

$$advantage = reward_{T+L} - mean([reward_T, ..., reward_{T+L}])$$

The formula of the advantage with Exponential Moving Average algorithm is like:

$$\begin{aligned} advantage &= reward_{T+L} - EMA \\ EMA &= reward_{T+L} * \alpha + EMA * (1-\alpha) \end{aligned}$$

It needs to be emphasized that whatever initial states or the generated states and the advantages will be stored into the replay buffer for calculating the PPO loss.

Then, after gathering the initial state, the agent could forward to take the action, which means sampling (selecting) the new hyperparameters. As we have mentioned, the distributional parameters of the hyperparameters and the log-probabilities computed from these parameters, as agent forward, would be used in the workflow in the RL framework. These log-probabilities represent the possibility for these selected hyperparameters to be sampled from the corresponding distributions. The formula of log-probabilities is like:

$$\begin{aligned} \log P(\mathbf{a}_t|\mathbf{s}_t) = & \underbrace{\sum_i \left(-\ln \sigma_t^i - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\left(\frac{a_t^i - \mu_t^i}{\sigma_t^i}\right)^2 \right)}_{\text{continuous part}} \\ & + \underbrace{\sum_i \sum_{j=1}^{\#\text{ levels in category } i} \log p_t^{i,j} \mathbf{1}[a_t^i = \text{level } j \text{ in category } i]}_{\text{discrete part}} \end{aligned}$$

where μ_t, σ_t, $p_t^{i,j}$ are produced by the network, which are related to state $\mathbf{s}_t$. μ_t and σ_t are the two parameters of the normal distribution, and ${}_t^{i,j}$'s are the parameters belonging to the categorical distribution, which has the meaning of the probability when observing level j of category i.

The action and log-probabilities will be stored in the replay buffer for future PPO loss calculations.

After the agent takes the actions, sampling the new hyperparameters, the environment should take this hyperparameter combination to tune the Family (ML model) and evaluate them on the validation set to get the prediction performance which is the reward. Then the environment concatenates the previous L action and reward pairs to generate a new state. Therefore, the workflow closes the loop, and until the episode ends all the transactions from the current episode will stored into the replay buffer. For the normal Deep Learning model training process, the datasets are always settled which means we first collect and preprocess the data and then train the model on these datasets. However, the normal RL framework has different paradigms. For the normal RL frameworks, the Agent takes the action then the environment returns back corresponding reward and the current state, then the agent will be trained by these states and rewards, which means the dataset of the RL framework is generated by itself within the worlkflow between the agent and environment. Thus, here we use the replay buffer to store several episodes. This could make the whole process more efficient by training the agent on this replay buffer multiple times. So that, the agent can be fully trained and learned on the replay buffer with generated cases.

3.2.3 Training the Agent

With the replay buffer from step one, sampling transactions, we could train our agent by calculating and backpropagating PPO loss. As we have collected in the replay buffer, we have the log-probablitities(old), action, state and advantage. Then we utilize them to calculate the PPO loss and use the PPO loss to update the networks in the Agent. The overview of step two to train the Agent has been shown in Fig. 4.

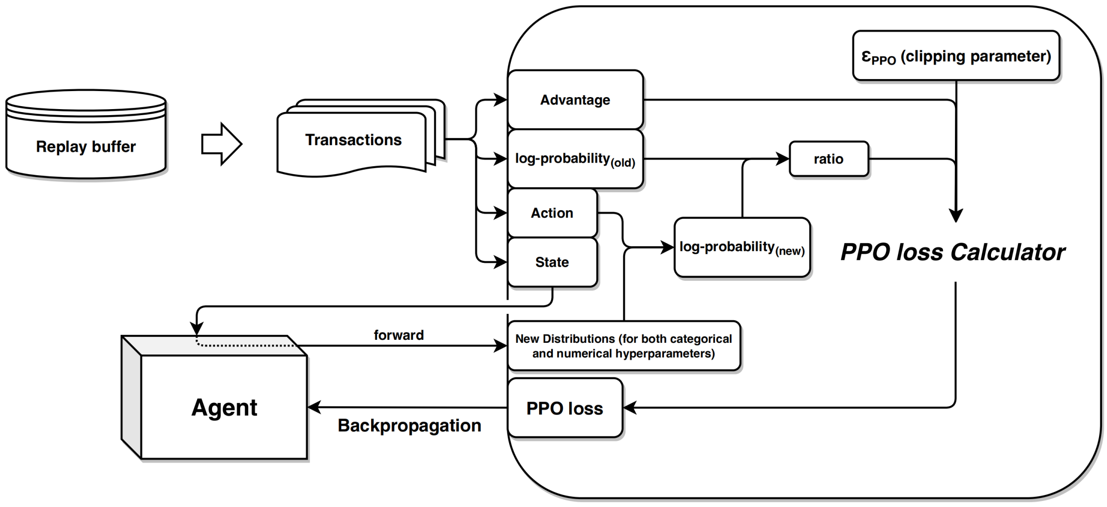


Figure 4: Step-2 : Training the Agent

Firstly, we use the current state which is the previous pairs of hyperparameters and accuracy of ML models to go through the model and generate the distribution parameters. After gathering the new distributions of both numerical and categorical hyperparameters, we could utilize action to calculate the new log-probablitity for these selected hyperparameters to be sampled from the new distributions.

Then we use the new log-probablitity and old log-probabilitiy to calculate the intermediate variable ratio, which is the difference between the log-probabilitis. If the ratio is larger than 0, it means the action in the new distribution is much more likely to occur. The formula of the ratio is like:

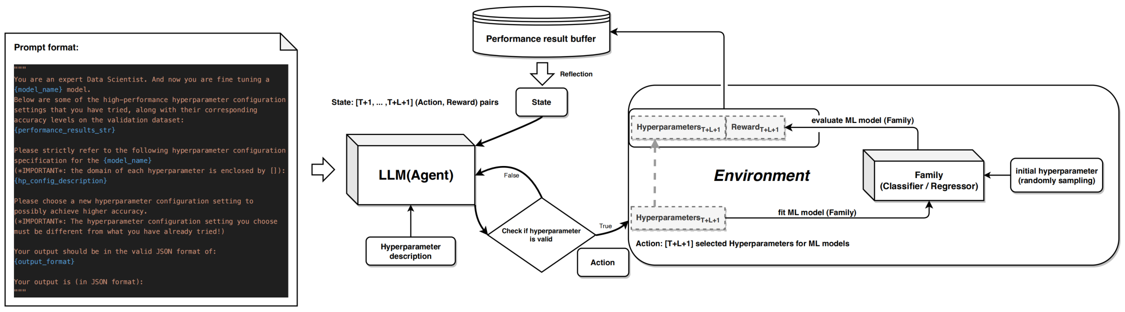
$$ratio = exp(\log P(\mathbf{a}_t|\mathbf{s}_t)_{new} - \log P(\mathbf{a}_t|\mathbf{s}_t)_{old})$$

Then the final PPO Loss was calculated by the log-probabilities ratio, advantage and the ϵ (which is the clipping parameter). The formula of the PPO loss is like:

$$\begin{aligned} PPO\_Loss = mean(-min(ratio * advantage, \\ clip(ratio, 1 - \epsilon, 1 + \epsilon) * advantage)) \end{aligned}$$

3.3 Hyperparameter Tuning with LLM integration

As the project name 'LinguAML', we would like to integrate the LLM into the Auto-Machine Learning system. Here we conduct a prompt format according to the performance result buffer and the reflection part for LLM to boost itself. The workflow is just like the RL framework with the action which is the selected hyperparameter by LLM and the evaluation result as the reward. However, we also add additional information to maximize the utilization of AGI and the prompt engineering ability with the hyperparameter description and also the historical cases provided. Meanwhile, we also deal with the hallucinations and instability with a double-checker module. The overview of LLM integration AutoML system has been shown in Fig. 5.



Prompt format:

```
"""
You are an expert Data Scientist. And now you are fine tuning a
{model_name} model.
Below are some of the high-performance hyperparameter configuration
settings that you have tried, along with their corresponding
accuracy levels on the validation dataset:
{performance_results_str}

Please strictly refer to the following hyperparameter configuration
specification for the {model_name}
(*IMPORTANT*: the domain of each hyperparameter is enclosed by []):
{hp_config_description}

Please choose a new hyperparameter configuration setting to
possibly achieve higher accuracy.
(*IMPORTANT*: The hyperparameter configuration setting you choose
must be different from what you have already tried!)

Your output should be in the valid JSON format of:
{output_format}

Your output is (in JSON format):
"""
```

Performance result buffer
Reflection
State: [T+1, ... ,T+L+1] (Action, Reward) pairs
State
LLM(Agent)
False
Check if hyperparameter is valid
True
Hyperparameter description
Action
Hyperparameters$_{T+L+1}$
Reward$_{T+L+1}$
evaluate ML model (Family)
Environment
Family (Classifier / Regressor)
initial hyperparameter (randomly sampling)
Hyperparameters$_{T+L+1}$
fit ML model (Family)
Action: [T+L+1] selected Hyperparameters for ML models

Figure 5: AutoML with LLMs

3.3.1 Reflection on Tuning Process

The concept of reflection plays an important role in agents powered by LLMs [21]. To boost the performance of our project with the idea of reflection, we prompt the LLM to make a summary of it learned experiences. Upon identifying a satisfactory hyperparameter configuration for a specific task, the LLM, which we shall call the reflection agent, reviews all undertaken actions, the resulting model performance, and the essence of the task. It is then prompted to compose a coherent reflection in a natural language consisting of a summary and gained insights from the entire tuning process. This mirrors the intuitive deliberations a data scientist would engage in the stage of post-analysis upon completion of a project.

Such reflections can enhance hyperparameter tuning processes, as introduced in previous sections. With the reflections, in the procedure of hyperparameters tuning, the LLM may develop a deeper understanding of all kinds of hyperparameters as well as the essence of tuning strategies.

3.4 Hyperparameter Tuning with Hybrid model

It is natural to have an idea to combine both the RL and the LLM models. Thus, we propose a hybrid framework that utilizes an epsilon-Greedy-like algorithm to select for each episode to use RL or LLM to select the hyperparameters. The overview of the hybrid framework has been shown in Fig. 6.

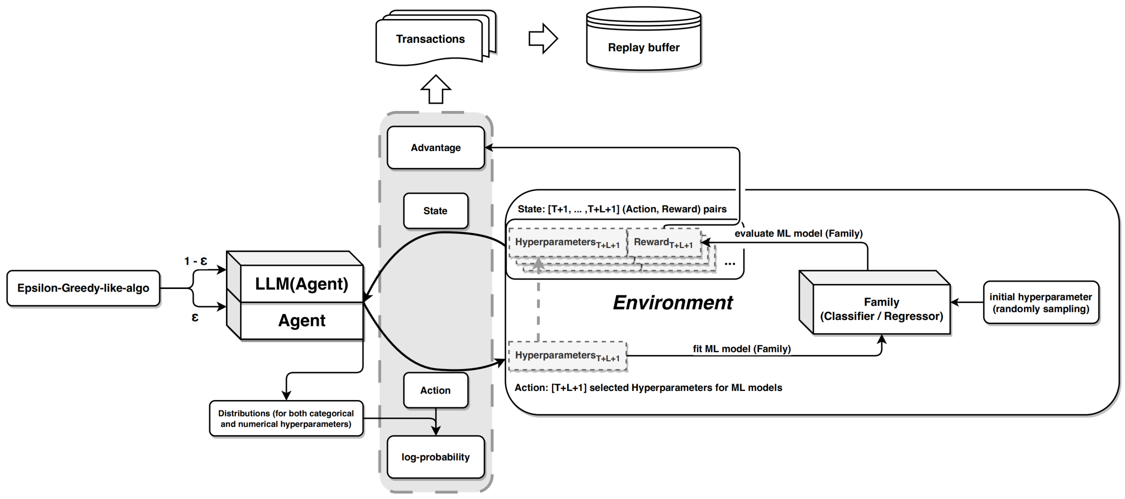

Transactions
Replay buffer
Advantage
State
Action
log-probability
Epsilon-Greedy-like-algo
1 - ε
ε
LLM(Agent)
Agent
Distributions (for both categorical and numerical hyperparameters)
State: [T+1, ... ,T+L+1] (Action, Reward) pairs
Hyperparameters$_{T+L+1}$
Reward$_{T+L+1}$
...
evaluate ML model (Family)
Environment
Family (Classifier / Regressor)
initial hyperparameter (randomly sampling)
Hyperparameters$_{T+L+1}$
fit ML model (Family)
Action: [T+L+1] selected Hyperparameters for ML models

Figure 6: AutoML with Hybrid model

4 Experiments

In this section, we will construct experiments on several classical machine learning datasets to examine the effectiveness and efficiency of our proposed new AutoML framework, LinguAML. We will focus on comparing our pure reinforcement learning based model, pure LLM based model and the hybrid model with a mature traditional AutoML framework, Auto-Sklearn, which has already shown convincing performance.

4.1 Baseline: Auto-sklearn

Auto-sklearn is an AutoML framework with python implementations developed around the scikit-learn machine learning library. It provides automatic model selection and corresponding hyperparameter tuning for novel machine learning tasks. The algorithms behind Auto-sklearn are mainly based on mathematical optimization techniques, including meta-learning and Bayesian optimization, which is different from LinguAML. On the other hand, Auto-sklearn does not rely on the practical information in textual form to find the optimal model with its best configuration. Such information including task requirement, dataset description, and the physical meanings of hyperparameters, which can be utilized by the LLM agent involved in LinguAML framework. Therefore, the comparison between LinguAML with Auto-sklearn is endowed with practical significance.

4.2 Experiment Settings

4.2.1 Model to Evaluate

We focus on testing the performance of LinguAML in tuning classification models on the current stage. Solving classification problems are one of the most important tasks in machine learning area, alongside with regression problems. The most prevailing machine learning models for classification problems, including Support Vector Machine Classifier (SVC), Random Forest Classifier (RF), Extreme Gradient Boosting Classifier (XGB), etc., have rich hyperparameters in both categorical form and numerical form. Most of the hyperparameters have their physical meanings which will be useful in prompting the LLM-enabled agents to learn the best way to adjust them, as what human being does. In this paper, we selected SVC and RF as the underlying model to compare LinguAML and Auto-sklearn.

4.2.2 Dataset

We conduct the experiments on 19 high quality datasets for classification problems selected from the UCI machine learning platform. These datasets are in various of sizes and covered a wide range of industrial areas. Thus, it is an ideal choice to examine the capability of LinguAML framework in general circumstances on these datasets.

Before the training phase, we divided each of the 19 datasets into training dataset, validation dataset, and testing dataset. The allocation of samples is in line with the distribution 60%-20%-20%. After that, we conducted data normalization and principal component decomposition to reduce the dimension of datasets. This procedure will eliminate the influence of different data scale and the risk of model under-fitting. During the training phase, both frameworks will utilize the training dataset to tune the hyperparameter of the underlying model and evaluate the accuracy on the validation dataset for model selection. After the training process, we will finally evaluate the output hyperparameter set on the testing datasets to examine the effectiveness of the two frameworks. For LinguAML, the validation accuracy evaluated on validation dataset during the training process is used as the reward of the agent for that transaction. This guaranteed the elimination of the risk of model overfitting.

The detailed information of all training dataset is listed in Table 1, where "records" refers to the number of records in the training dataset, "features" refers to the number of features after principal component decomposition, and "level" refers to the number of classes in the response variable.

Table 1: Training dataset information

Dataset	Size		Level
	records	features	
adult	28572	5	4
bank_marketing	4705	5	2
breast_cancer	166	5	2
breast_cancer_wisconsin_diagnostic	341	5	2
breast_cancer_wisconsin_original	409	5	2
car_evaluation	1036	5	4
credit_approval	391	5	2
dry_bean_dataset	8166	5	7
glass_identification	128	6	6
heart_disease	178	5	5
ionosphere	210	5	2
iris	90	4	3
mushroom	3386	5	2
predict_students_dropout_and_academic_success	2654	5	3
rice_cammeo_and_osmancik	2286	5	2
spambase	2760	5	2
statlog_german_credit_data	600	5	2
wine	106	5	3
wine_quality	3898	5	7

4.2.3 Configurations used in LinguAML

For every underlying model to tune, we established a pre-determined configuration file for the agent in LinguAML to consume. The configuration file including the settings of the LinguAML framework and the search field of hyperparameters of the underlying machine learning model. Detailed information of the key components in the configuration file is listed as follows:

- LinguAML settings
  - Framework to use: the LinguAML framework to use for this task. “rl” stands for pure reinforcement learning framework, “llm” is for pure LLM based framework, and “hybrid” refers to the LLM-enabled reinforcement learning framework.

  - Number of epochs: We set it to be 20 in our experiments. For pure LLM framework, the number of epochs is 400.
  - Look back period: the number of hyperparameter set generated in historical movement to be considered by the agents when deciding the next adjustment of the hyperparameters. Larger values allow the agent to refer to more historical records to avoid poorly performed decisions and move closer to good ones but will result in more occupation of system resources. In the experiments, we set the look back period to be 6 in line with both performance and efficiency considerations.
  - Fitting time limit: the limitation of time for each fitting of the underlying model on the target dataset. Hyperparameter set that will cause very long time to fit will not be considered as an efficient choice. In the experiments, we limit each fitting within 60 seconds.
  - Distribution family: the distribution to be approached for generating the next hyperparameter set. Normal distribution is used in our experiments.
  - Moving average algorithm: the algorithm to be used in calculating moving average for distribution construction. We deploy exponential moving average algorithm in our experiment.
  - Reply buffer size: The size of reply buffer of PPO algorithm. We use a replay buffer of size 50 in our experiments.
  - Performance result buffer size: The size of buffer storing the action-reward pairs generated by LLM agents. We use a performance result buffer of size 10 in our experiments.
  - LLM sample frequency: The frequency of the involvement of LLM in the sampling of reply buffer in generating the next transition in hybrid framework. It is set to be 0.3 in the experiments. We leverage OpenAI GPT-3.5 Turbo model with temperature 0.6 in our experiments.

- Search field of underlying machine learning Model
  - Random forest model: For hyperparameters in numerical form, we mainly follow the search field proposed by Wu, Liu and Chen [15]. Five key hyperparameters selected in our experiments with their search field are listed as follows: n_estimators: [100, 2000], max_depth: [3, 30], min_samples_split: [2, 100], min_samples_leaf: [1, 100], max_features: [0.1, 0.9]. For hyperparameters in categorical form, we only tune criterion in our experiments. The possible choices for it are namely “gini”, “entropy”, and “log_loss”.
  - Support vector machine classifier: We tune five hyperparameters for SVC model. They are namely: C: [0.01, 1e8], gamma: [1e-4, 1], tol: [0.001, 1], kernel: [“linear”, “poly”, “rbf”, “sigmoid”], decision_function_shape: [“ovo”, “ovr”].

4.3 Evaluation Metrics

We simply utilize the prediction accuracy evaluated on the testing dataset as the main evaluation metrics. This is the fundamental indicator of the effectiveness of an auto-ML framework. Higher testing accuracy implies higher capability in hyperparameter tuning tasks. On the other hand, we are also interested in the efficiency of the framework. Thus, the time or training epochs used to generate the result will also be considered as an ancillary criterion.

For LinguAML frameworks, we pick the first hyperparameter set that reached the best performance on validation dataset among all trials made by the agent during the 20-epoch training process from the training log as the output. We also record the number of epochs and time required to get this hyperparameter set for the purpose of efficiency test. However, since Auto-sklearn enables parallel computing, direct computing will not be feasible. Analysis of efficiency will be restricted on LinguAML itself.

5 Results and Discussion

The result of the testing accuracies of three LinguAML frameworks and Auto-sklearn framework are shown in Table 2.

Table 2: Testing accuracy of four AutoML frameworks

Model	Dataset	LInguAML			Auto-Sklearn
		RL	LLM	Hybrid	
	adult	0.5698	0.5691	0.5668	**0.5743**
	bank_marketing	0.8029	0.7940	0.8023	**0.8164**
	breast_cancer	**0.8182**	0.8000	**0.8182**	0.6964
	breast_cancer_wisconsin_diagnostic	**0.9737**	**0.9737**	**0.9737**	0.9649
	breast_cancer_wisconsin_original	**1.0000**	0.9854	**1.0000**	0.9781
	car_evaluation	**0.8410**	0.8295	0.8353	0.8006
	credit_approval	**0.8244**	0.7939	**0.8244**	0.8244
	dry_bean_dataset	0.9342	0.9328	**0.9346**	0.9232
	glass_identification	**0.9070**	0.8605	0.8837	0.7674
RF	heart_disease	0.5763	0.4746	0.5085	**0.6500**
	ionosphere	0.9286	0.9143	0.9143	**0.9296**
	iris	**1.0000**	**1.0000**	**1.0000**	0.9667
	mushroom	**0.9973**	**0.9973**	**0.9973**	**0.9973**
	predict_students_dropout_and_academic_success	0.7175	0.7085	0.7119	**0.7220**
	rice_cammeo_and_osmancik	**0.9423**	0.9409	**0.9423**	0.9199
	spambase	0.9250	0.9217	**0.9261**	0.9110
	statlog_german_credit_data	**0.7450**	0.7400	0.7400	0.7150
	wine	**0.9722**	**0.9722**	**0.9722**	**0.9722**
	wine_quality	**0.6398**	0.6265	0.6388	0.6051
	adult	0.5579	0.5608	0.0000	**0.5718**
	bank_marketing	0.8055	0.8036	0.8087	**0.8184**
	breast_cancer	**0.8364**	0.8000	0.8182	0.6964
	breast_cancer_wisconsin_diagnostic	0.9825	**0.9912**	**0.9912**	0.9825
	breast_cancer_wisconsin_original	**1.0000**	0.9708	**1.0000**	0.9781
	car_evaluation	**0.8902**	0.8353	0.8873	0.8439
	credit_approval	0.8321	0.8244	**0.8397**	0.8626
	dry_bean_dataset	0.9386	0.9364	**0.9394**	0.9288
	glass_identification	**0.9535**	0.8837	0.9302	0.7907
SVC	heart_disease	0.5424	0.5085	0.5424	**0.6667**
	ionosphere	**0.9286**	0.9143	0.9143	0.9014
	iris	**1.0000**	**1.0000**	**1.0000**	**1.0000**
	mushroom	**1.0000**	0.9814	**1.0000**	0.9982
	predict_students_dropout_and_academic_success	0.7153	0.7085	**0.7164**	0.7107
	rice_cammeo_and_osmancik	**0.9423**	0.9409	0.9409	0.9226
	spambase	0.9011	0.9076	**0.9120**	0.8827
	statlog_german_credit_data	**0.7550**	0.7300	0.7350	0.7250
	wine	0.9722	**1.0000**	**1.0000**	**1.0000**
	wine_quality	**0.6031**	0.5378	0.5929	0.5531

It can be observed from the table that, among all 19 experiment datasets, LinguAML frameworks provides better testing performance than Auto-sklearn in most of the cases. This result indicates that LinguAML framework have advantages over traditional mathematical hyperparameter optimization methods. Specifically, for random forest classifier, both the pure reinforcement learning framework and the hybrid framework of LinguAML out-performs Auto-sklearn on 14 out of 19 datasets, while the pure LLM framework have better performance on 13 datasets. For support vector machine classifier, the pure reinforcement learning framework, the pure LLM framework, and the hybrid framework out-performs Auto-sklearn on 14, 10, and 15 datasets, respectively. LinguAML frameworks guaranteed advantages on at least 50% of the datasets under both situations. Figure 7 shows the number of datasets on which each framework ranks top 2 among all frameworks.

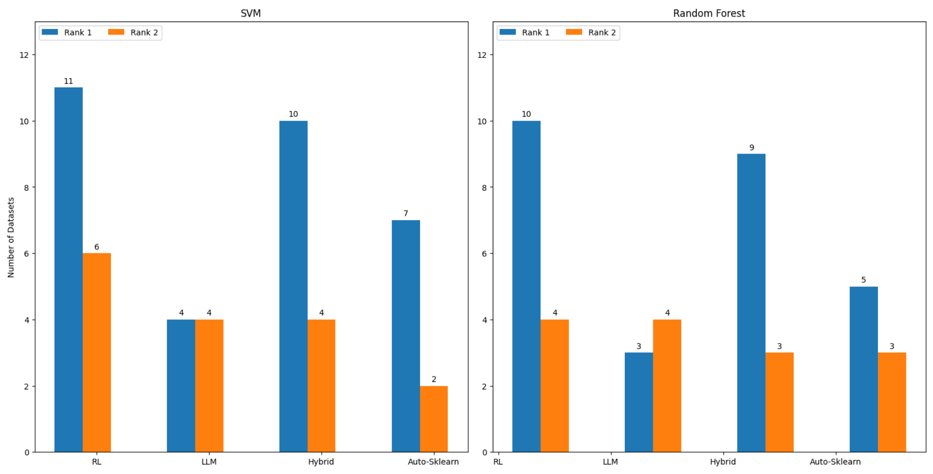


Figure 7: Rank of each AutoML frameworks

On the other hand, we would like to take some insights of the reason behind the testing results. The first concern is the size of the dataset. It can be observed that as the size of the dataset increasing, the performance rank of LinguAML frameworks dropped. This situation can be improved by increasing the allowed maximum model fixing time, in sacrifice of the tuning speed and computation resource. This indicates that the capability of LinguAML on large dataset still have space to improve.

The second concern is the effectiveness of involving LLMs in hyperparameter tuning. It can be observed from Table 1 that the performance of the hybrid framework has similar performance in beating Auto-sklearn to pure reinforcement learning. However, in several cases when the pure LLM framework shows advantages, the performance of the hybrid framework will also be improved. This indicates that the introduction of LLM have potential improvement in hyperparameter optimization. Besides, as is shown in Table 3, the time and epoch required to reach the optimal hyperparameter is generally shortened. This serves as evidence that the introduction of LLM is able to facilitate the process of decision of reinforcement learning agent, and thus shorten the training time.

Table 3: Epoch and time required to reach the optimal hyperparameter set

Model	Dataset	RL		LLM		Hybrid	
		Time (s)	Epoch	Time (s)	Epoch	Time (s)	Epoch
	adult	12718.418	5	0.000	1	363.564	1
	bank_marketing	7362.444	9	12.688	4	1408.178	3
	breast_cancer	428.262	3	50.768	27	418.202	3
	breast_cancer_wisconsin_diagnostic	691.702	4	3.396	2	72.984	1
	breast_cancer_wisconsin_original	3.214	1	11.178	5	2.014	1
	car_evaluation	203.398	1	75.098	38	21.140	1
	credit_approval	1459.708	7	0.000	1	454.332	3
	dry_bean_dataset	20660.530	16	524.362	114	7557.602	8
	glass_identification	2624.452	12	25.792	11	25.542	1
RF	heart_disease	1605.142	9	5.214	3	5.094	1
	ionosphere	1967.180	9	0.000	1	25.430	1
	iris	6.770	1	3.130	2	3.224	1
	mushroom	584.356	2	5.654	3	173.666	1
	predict_students_dropout_and_academic_success	4917.426	10	45.974	6	2817.212	8
	rice_cammeo_and_osmancik	2602.914	6	331.046	252	1935.078	4
	spambase	8066.862	13	181.316	254	3076.412	10
	statlog_german_credit_data	139.718	1	33.182	14	25.122	1
	wine	58.048	1	6.034	4	4.582	1
	wine_quality	6206.706	11	8017.912	274	6364.110	11
	adult	7285.834	2	0.000	1	0.000	1
	bank_marketing	3528.662	2	81.182	35	1533.070	2
	breast_cancer	12407.652	9	9.584	18	1736.870	4
	breast_cancer_wisconsin_diagnostic	1880.518	2	10.348	8	68.094	1
	breast_cancer_wisconsin_original	0.000	1	1.548	2	155.472	1
	car_evaluation	15774.764	11	3.290	7	1528.250	6
	credit_approval	2997.022	2	9.810	17	1857.380	5
	dry_bean_dataset	13231.566	7	6.538	8	9792.636	15
	glass_identification	11189.874	16	11.900	23	530.674	2
SVC	heart_disease	1312.690	1	1.780	2	184.684	1
	ionosphere	2461.920	2	3.200	3	66.116	1
	iris	0.000	1	1.504	2	3.366	1
	mushroom	71.136	1	33.558	72	63.440	1
	predict_students_dropout_and_academic_success	3269.124	2	186.394	221	1529.306	2
	rice_cammeo_and_osmancik	4796.334	2	21.910	56	3800.302	6
	spambase	5151.624	3	159.098	101	4220.556	9
	statlog_german_credit_data	14864.022	11	37.620	28	2310.952	5
	wine	2.138	1	1.342	2	8.682	1
	wine_quality	22004.650	15	4.530	5	1129.094	1

6 Conclusion

In conclusion, we have proposed a novel RL framework that utilizes the LSTM to handle the historical cases in agents, a feed-forward network to generate the distributional parameters and sampling the hyperparameters for the ML models. Then utilizing the replay buffer by generating a large number of transactions and using PPO loss to train the agent. Meanwhile, as the name of 'LinguAML' we also utilize the LLM model to try to complete the Auto-ML task by prompting it as an expert data scientist and also AGI system to accomplish reflection and avoid instability.

Next, we have evaluated the effectiveness of three LinguAML frameworks and the Auto-sklearn framework on 19 experiment datasets. The LinguAML frameworks show their superiority over conventional mathematical hyperparameter optimization techniques by outperforming Auto-sklearn in the majority of situations. In particular, the pure LLM framework performs better on 13 datasets, while the hybrid LinguAML framework and pure reinforcement learning framework beat Auto-sklearn on 14 of the 19 datasets for the random forest classifier. The pure reinforcement learning framework, the pure LLM framework, the hybrid framework, and the support vector machine classifier all beat Auto-sklearn on 10, 15, and 14 datasets, respectively. In both cases, LinguAML frameworks routinely perform better than 50% of the datasets.

Through observation, the amount of the dataset affects how well LinguAML frameworks perform; bigger datasets result in lower performance rankings. Although it comes at the expense of tuning speed and processing resources, this may be lessened by raising the maximum permitted model fixing time, indicating the potential for development in managing massive datasets. Meanwhile, the usefulness of using LLMs in tweaking hyperparameters is investigated. In most circumstances, the hybrid framework is on par with or even better than pure reinforcement learning, and it also gets better in situations when the pure LLM system performs better. This demonstrates how LLMs may be used for hyperparameter tuning. Furthermore, the addition of LLMs shortens the amount of time and epochs needed to achieve ideal hyperparameters, suggesting that they can hasten the decision-making and training processes of reinforcement learning agents.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners." [Online]. Available: http://arxiv.org/abs/2005.14165

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models." [Online]. Available: http://arxiv.org/abs/2302.13971

[3] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "RoFormer: Enhanced transformer with rotary position embedding." [Online]. Available: http://arxiv.org/abs/2104.09864

[4] N. Shazeer, "GLU variants improve transformer." [Online]. Available: http://arxiv.org/abs/2002.05202

[5] B. Zhang and R. Sennrich, "Root mean square layer normalization." [Online]. Available: http://arxiv.org/abs/1910.07467

[6] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models." [Online]. Available: http://arxiv.org/abs/2201.11903

[7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models." [Online]. Available: http://arxiv.org/abs/2210.03629

[8] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," version: 1. [Online]. Available: http://arxiv.org/abs/2304.03442

[9] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face." [Online]. Available: http://arxiv.org/abs/2303.17580

[10] "Automated machine learning: Methods, systems, challenges." [Online]. Available: http://link.springer.com/10.1007/978-3-030-05318-5

[11] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-sklearn 2.0: Hands-free AutoML via meta-learning." [Online]. Available: http://arxiv.org/abs/2007.04074

[12] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization."

[13] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," Aug. 2012.

[14] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme, "Hyp-RL : Hyperparameter optimization by reinforcement learning." [Online]. Available: http://arxiv.org/abs/1906.11527

[15] J. Wu, X. Liu, and S. Chen, "Hyperparameter optimization through context-based meta-reinforcement learning with task-aware representation," vol. 260, p. 110160. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950705122012564

[16] M. M. Hassan, A. Knipper, and S. K. K. Santu, "ChatGPT as your personal data scientist." [Online]. Available: http://arxiv.org/abs/2305.13657

[17] N. Hollmann, S. Müller, and F. Hutter, "LLMs for semi-automated data science: Introducing CAAFE for context-aware automated feature engineering." [Online]. Available: http://arxiv.org/abs/2305.03403

[18] S. Zhang, C. Gong, L. Wu, X. Liu, and M. Zhou, "AutoML-GPT: Automatic machine learning with GPT." [Online]. Available: http://arxiv.org/abs/2305.02499

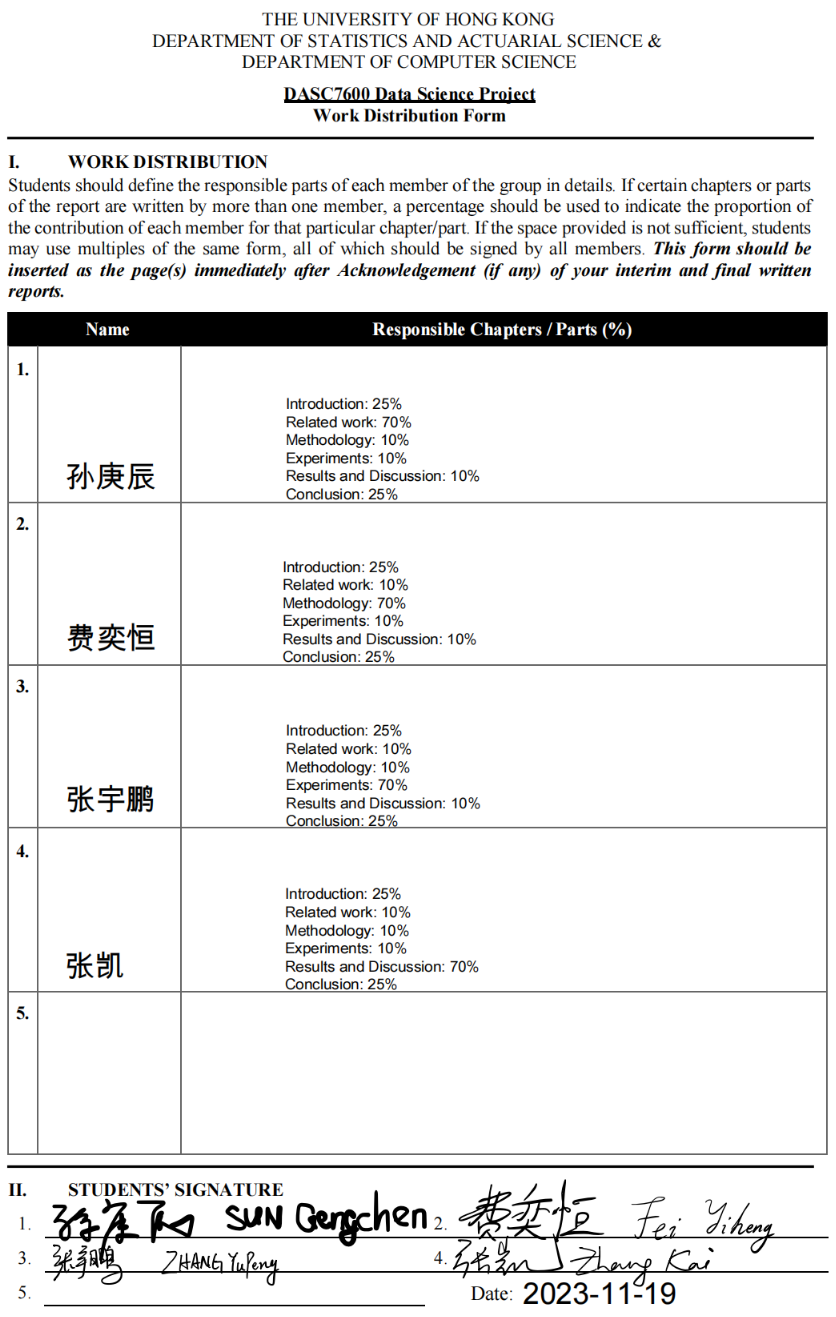
[19] L. Zhang, Y. Zhang, K. Ren, D. Li, and Y. Yang, "MLCopilot: Unleashing the power of large language models in solving machine learning tasks." [Online]. Available: http://arxiv.org/abs/2304.14979

[20] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," Feb. 2017.

[21] N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," 2023.

7 Appendices

7.1 Work Distribution



THE UNIVERSITY OF HONG KONG
DEPARTMENT OF STATISTICS AND ACTUARIAL SCIENCE &
DEPARTMENT OF COMPUTER SCIENCE

DASC7600 Data Science Project
Work Distribution Form

I. WORK DISTRIBUTION

Students should define the responsible parts of each member of the group in details. If certain chapters or parts of the report are written by more than one member, a percentage should be used to indicate the proportion of the contribution of each member for that particular chapter/part. If the space provided is not sufficient, students may use multiples of the same form, all of which should be signed by all members. ***This form should be inserted as the page(s) immediately after Acknowledgement (if any) of your interim and final written reports.***

	Name	**Responsible Chapters / Parts (%)**
1.	孙庚辰	Introduction: 25% Related work: 70% Methodology: 10% Experiments: 10% Results and Discussion: 10% Conclusion: 25%
2.	费奕恒	Introduction: 25% Related work: 10% Methodology: 70% Experiments: 10% Results and Discussion: 10% Conclusion: 25%
3.	张宇鹏	Introduction: 25% Related work: 10% Methodology: 10% Experiments: 70% Results and Discussion: 10% Conclusion: 25%
4.	张凯	Introduction: 25% Related work: 10% Methodology: 10% Experiments: 10% Results and Discussion: 70% Conclusion: 25%
5.		

II. STUDENTS' SIGNATURE

1. 孙庚辰 SUN Gengchen
2. 费奕恒 Fei Yiheng
3. 张宇鹏 ZHANG YuPeng
4. 张凯 Zhang Kai
5.

Date: 2023-11-19