AutoMMLab: Automatically Generating Deployable Models from Language Instructions for Computer Vision Tasks

Zekang Yang[1], **Wang Zeng**[1], **Sheng Jin**[2,1*], **Chen Qian**[1], **Ping Luo**[2], **Wentao Liu**[1*]
[1]SenseTime Research and Tetras.AI
[2]The University of Hong Kong
{yangzekang, zengwang, jinsheng, qianchen, liuwentao}@tetras.ai, pluo@cs.hku.hk

Abstract

Automated machine learning (AutoML) is a collection of techniques designed to automate the machine learning development process. While traditional AutoML approaches have been successfully applied in several critical steps of model development (*e.g.* hyperparameter optimization), there lacks a AutoML system that automates the entire end-to-end model production workflow for computer vision. To fill this blank, we propose a novel request-to-model task, which involves understanding the user's natural language request and execute the entire workflow to output production-ready models. This empowers non-expert individuals to easily build task-specific models via a user-friendly language interface. To facilitate development and evaluation, we develop a new experimental platform called AutoMMLab and a new benchmark called LAMP for studying key components in the end-to-end request-to-model pipeline. Hyperparameter optimization (HPO) is one of the most important components for AutoML. Traditional approaches mostly rely on trial-and-error, leading to inefficient parameter search. To solve this problem, we propose a novel LLM-based HPO algorithm, called HPO-LLaMA. Equipped with extensive knowledge and experience in model hyperparameter tuning, HPO-LLaMA achieves significant improvement of HPO efficiency.

Code — https://github.com/yang-ze-kang/AutoMMLab
Extended version — https://arxiv.org/abs/2402.15351

Introduction

Machine learning (ML) has achieved considerable successes in recent years and an ever-growing number of disciplines rely on it. However, this success crucially relies on human ML experts to perform tasks including data preparation, model algorithm selection, model training with hyperparameter tuning and model deployment. The rapid growth of machine learning applications has created a demand for off-the-shelf machine learning methods that can be used easily and without expert knowledge. To this end, automated machine learning, or AutoML, has been developed in literature. AutoML aims to minimize the workload of human experts in the ML development process, making AI algorithms available and easily accessible even for non-AI experts.

*Corresponding Author

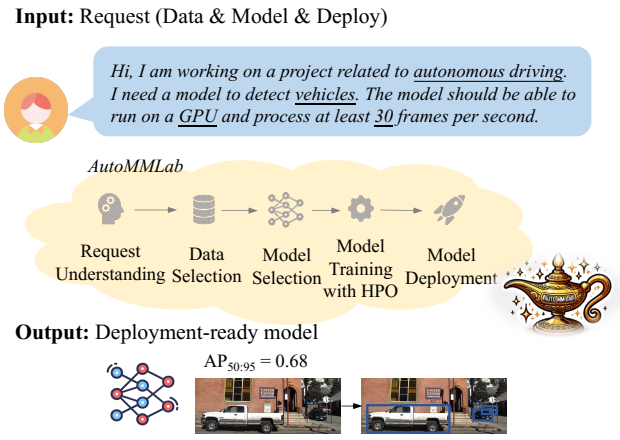


Figure 1: AutoMMLab autmatically creates deployable models from user's language instructions.

Recent public computer vision (CV) communities (*e.g.* OpenMMLab) have provided a wide variety of toolboxes in various research areas such as image classification (MMPretrain (Contributors 2023)), object detection (MMDetection (Chen et al. 2019)), semantic segmentation (MMSegmentation (Contributors 2020a)), and pose estimation (MMPose (Contributors 2020b)). Considering large language models (LLMs) have exhibited exceptional ability in language understanding, generation, interaction, and reasoning, we are inspired to employ LLMs to connect AutoML with public CV communities (*e.g.* OpenMMLab) for automating the *whole* CV model production workflow via a natural language based interface.

In this paper, we introduce a novel comprehensive task called request-to-model. Unlike traditional AutoML task that focus on one particular step in the model production workflow (*e.g.* neural architecture search or hyperparameter optimization), request-to-model extends the reach of AutoML to evolve towards fully automated model production through a user-friendly language interface. This empowers individuals to leverage the capabilities of AI without extensive expertise, driving progress in various fields.

To facilitate the development of the request-to-model task, we propose a LLM-empowered platform called Au-

toMMLab, which can serve as a testbed for developing and evaluating the request-to-model algorithms. As shown in Figure 1, given language-based user requirements, our AutoMMLab platform will schedule each module to execute the entire workflow and finally output production-ready models for computer vision tasks. AutoMMLab connects diverse datasets, CV models, training pipelines and deployment tools to facilitate the solution of numerous CV tasks. The whole workflow consists of 5 major modules, including request understanding, data construction, model construction, model training with hyperparameter optimization, and model deployment.

Thanks to the modular design of AutoMMLab, we also develop the LAMP (Language-instructed Automated Model Production) benchmark for research on the language-instruction based model production. The LAMP is the *first* benchmark for evaluating the request-to-model AutoML algorithms in computer vision, which enables the community to explore key components in the end-to-end request-to-model pipeline.

Hyperparameter optimization (HPO) aims to choose a set of optimal hyperparameters that yields an optimal performance based on the predefined objective function. Thanks to its wide applications, it becomes one of the most important components for AutoML. Traditional approaches (*e.g.* bayesian optimization) mostly rely on trial-and-error, leading to inefficient parameter search. To solve this problem, we propose a novel LLM-based hyperparameter optimization algorithm, called HPO-LLaMA. Equipped with rich knowledge of model training, HPO-LLaMA can skillfully search for the optimal model training hyperparameters with significantly reduced numbers of trials.

Our main contributions can be summarized as follows:

- We propose the novel request-to-model task, automating the whole model production workflow. We present the *first* request-to-model AutoML platform for computer vision tasks, called AutoMMLab. By integrating AutoML and language interface, it enables non-expert users to easily build and deploy CV models, unlocking the potential of AI for a wider audience.
- Based on the AutoMMLab platform, we build a benchmark termed LAMP for evaluating end-to-end prompt-based model production, and also studying each component in the whole production pipeline.
- We propose a novel LLM-based hyperparameter optimization algorithm, called HPO-LLaMA. To the best of our knowledge, HPO-LLaMA is the first supervised fine-tuned LLM specifically designed for HPO.

Related Works

Hyperparameter Optimization (HPO). Grid search and random search (Bergstra and Bengio 2012) are commonly employed methods for hyperparameter optimization (HPO). Grid search divides the search space into regular intervals and evaluates all choices to select the best-performing one, while random search selects the best choice from a set of randomly sampled hyperparameters. Bayesian optimization (Lindauer et al. 2022) is another efficient HPO method, where a surrogate model such as random forest and Gaussian process is constructed to learn from past function evaluations and choose promising future candidates. Gradient-based optimization methods (Pedregosa 2016; Franceschi et al. 2017) further improve it by employing gradient information. In contrast to these traditional approaches that rely solely on trial-and-error, our proposed HPO-LLaMA leverages a large language model with extensive knowledge and experience in model hyperparameter tuning, resulting in significant improvement of HPO efficiency.

AutoML libraries and systems. While several AutoML pipeline libraries and systems have been introduced, most of them specialize in specific components of the pipeline and are mainly designed for AI developers. Early AutoML frameworks like Auto-WEAK (Thornton et al. 2013) and Auto-Sklearn (Feurer et al. 2015) concentrated on optimizing traditional machine learning pipelines and their associated hyperparameters. More recent frameworks like Auto-PyTorch (Zimmer, Lindauer, and Hutter 2021) and AutoKeras (Jin, Song, and Hu 2019) have shifted their focus towards searching for deep learning models. There are also comprehensive AutoML solution. For example, Microsoft's NNI[1] toolkit automates feature engineering, neural architecture search, hyperparameter tuning, and model compression specifically for deep learning. Vega (Wang et al. 2020) provides an extensive pipeline that encompasses data augmentation, HPO, NAS, model compression, and full training. Google's AutoML[2] suite offers a user-friendly collection of AutoML tools that streamline the entire machine learning pipeline. However, they still require intervention and effort from users with expertise.

LLM in AutoML. Large Language Models (LLMs) are large-scale neural networks with billions of parameters that are pre-trained on massive amounts of data to achieve general-purpose language understanding and generation. LLMs such as GPTs (Brown et al. 2020; OpenAI 2023), PaLM (Chowdhery et al. 2022) and LLaMAs (Touvron et al. 2023a,b), have demonstrated impressive capabilities in comprehension and generation of natural language. Some studies directly explore GPT's capabilities on AutoML including feature engineering (Hollmann, Müller, and Hutter 2023), neural architecture search (NAS) (Zheng et al. 2023). For example, GENIUS (Zheng et al. 2023) proposes to use GPT-4 as a "black box" optimiser to tackle the problem of NAS through an iterative refinement process, (Zhang et al. 2023) demonstrates that in hyperparameter optimization tasks, LLM-based methods outperform random search and Bayesian optimization. More recently, (Huang et al. 2024) and (Guo et al. 2024) leverage LLMs to iteratively optimize the baseline method for a specific task, resulting in a better-performing method. (Hong et al. 2024) leverages LLMs to automatically analyze structured data but lacks a self-improvement strategy to refine the results. (Viswanathan et al. 2023) presents a demo that provides a natural language task description to LLMs, and uses it to train a special-purpose NLP model that is conducive to de-

[1]https://github.com/microsoft/nni

[2]https://cloud.google.com/automl

ployment. Different from the work mentioned above, Our propose AutoMMLab system has several distinct features. First, it automates the whole model production pipeline, from understanding requirements to deploying the model, while supporting a wide range of mainstream computer vision tasks, including classification, detection, segmentation, and pose estimation. Second, instead of relying solely on prompt engineering, we developed a requirements understanding dataset and a hyperparameter optimization dataset, leveraging supervised fine-tuning of LLMs to enhance their domain-specific capabilities. Third, it can recommend optimal hyperparameters tailored to specific requests.

AutoMMLab

Overview

The overview of AutoMMLab platform is illustrated in Figure 2. It directly takes language-based model request from users as the input, and automatically schedule and execute the entire workflow to produce deployable models. The whole pipeline of AutoMMLab consists of 5 main stages, including request understanding, data selection, model selection, model training with hyperparameter optimization (HPO), and model deployment. The Request Understanding stage understands the language-based model request and schedules the model production workflow, generating a structured configuration. The configuration contains detailed data information, model constraints and deployment requirements. AutoMMLab then construct the training data inside built-in dataset zoo and select the most appropriate model that meets the model constraints. When the data and model are ready, AutoMMLab execute model training with hyperparameter optimization (HPO) to obtain the final trianed model. In the model deployment stage, AutoMMLab apply MMDeploy (Contributors 2021) to deploy the model based on the user's hardware requirements.

Request Understanding

Request understanding aims to automatically and accurately understand the user's request, schedule the model production workflow, and generate the JSON-format configuration. Since it is non-trival for general-purpose LLMs to succeed in request understanding solely through prompting, we also train a task-specific LLM (RU-LLaMA) to achieve this goal.

GPT-assisted Training Data Generation. Tuning LLMs typically require numerous training data (Liu et al. 2023b,a), which is exceptionally costly and time-consuming to collect with human crowd-scouring. Inspired by the success of recent LLMs in data generation (Wang et al. 2022), we leverage GPT-4 to build our data generation pipeline. For each task type, (*i.e*. image classification, object detection, semantic segmentation, and keypoint estimation), we first ask 5 professional computer vision researchers to manually design 100 diverse request-config pairs. These human annotations are used as seeds in in-context-learning to query GPT. In our pipeline, we first design prompts to ask GPT to generate diverse requests, and then use carefully designed prompts to query GPT to parse the requests and generate corresponding configurations. In our preliminary experiments, we ablated the use of GPT-3.5 and GPT-4 and empirically found that GPT-4 can consistently produce higher quality data. Therefore, we use GPT-4 in our data generation pipeline and generate 1,000 request-config pairs. Finally, the generated data samples are carefully checked and manually corrected by a group of professional annotators.

RU-LLaMA. LLM has learned massive knowledge in the pre-training stage and have strong general question and answer capabilities. However, its usually require supervised fine-tuning on instruction-tuning dataset to better activate its domain-specific capabilities. We construct a instruction fine-tuning dataset based on the dataset generated by GPT-4. Following the mainstream methods of fine-tuning large language models in specific domains, we use LoRA (Hu et al. 2022) technology to fine-tune the LLM. Specifically, we utilize LoRA to fine-tune Llama-2 (Touvron et al. 2023b), and obtained RU-LLaMA that is skilled in parsing user's requirements into JSON-format configurations.

Data Selection

Data selection stage aims to use the JSON-format configuration parsed by RU-LLaMA to automatically retrieve and build request-related training dataset from the dataset zoo. **Dataset Zoo.** To facilitate dataset preparation, we construct a comprehensive dataset zoo encompassing a wide variety of public datasets (Krizhevsky, Sutskever, and Hinton 2017; Lin et al. 2014; Cordts et al. 2016). Each dataset is complemented by a data card that contains the meta information appertaining to the dataset intricately. We also take into account AI safety considerations, by ensuring openness, transparency, and reliability of data. **Dataset Selection Pipeline.** We design an elaborate pipeline to automatically retrieve pertinent images from the dataset zoo. First, we extract the relevant data cards from the dataset zoo based on the specified task category. Next, if the user requests the specific dataset, we compute the fuzzy match scores, quantifying the similarity between the user-designated name and the name of data cards. These cards are subsequently reordered in a descending pattern, according to their scores. Then the images that meet the requirements are sequentially collected from each dataset. Meanwhile, we use WordNet (Miller 1995) to identify semantic relationships between objects.

Model Selection

Model selection automates the process of choosing the most suitable model from the model zoo. **Model Zoo.** We first construct a comprehensive model zoo where each model is accompanied by a detailed model card and the pre-trained model weights. Each model card incorporates attributes such as the model's name, structure (*e.g*. network depth), parameters, floating point operations per second (FLOPs), inference speed, and performance. We have considered AI safety issues when constructing the model zoo. We try to keep the model source open and transparent, and all models are pre-trained with reliable public data. **Model Selection Pipeline.** We also design a elaborate pipeline able to automatically determine the most suitable model in model zoo, tailored to the user's model-specific requirements. First, we extract the task-related model cards from the model zoo according to

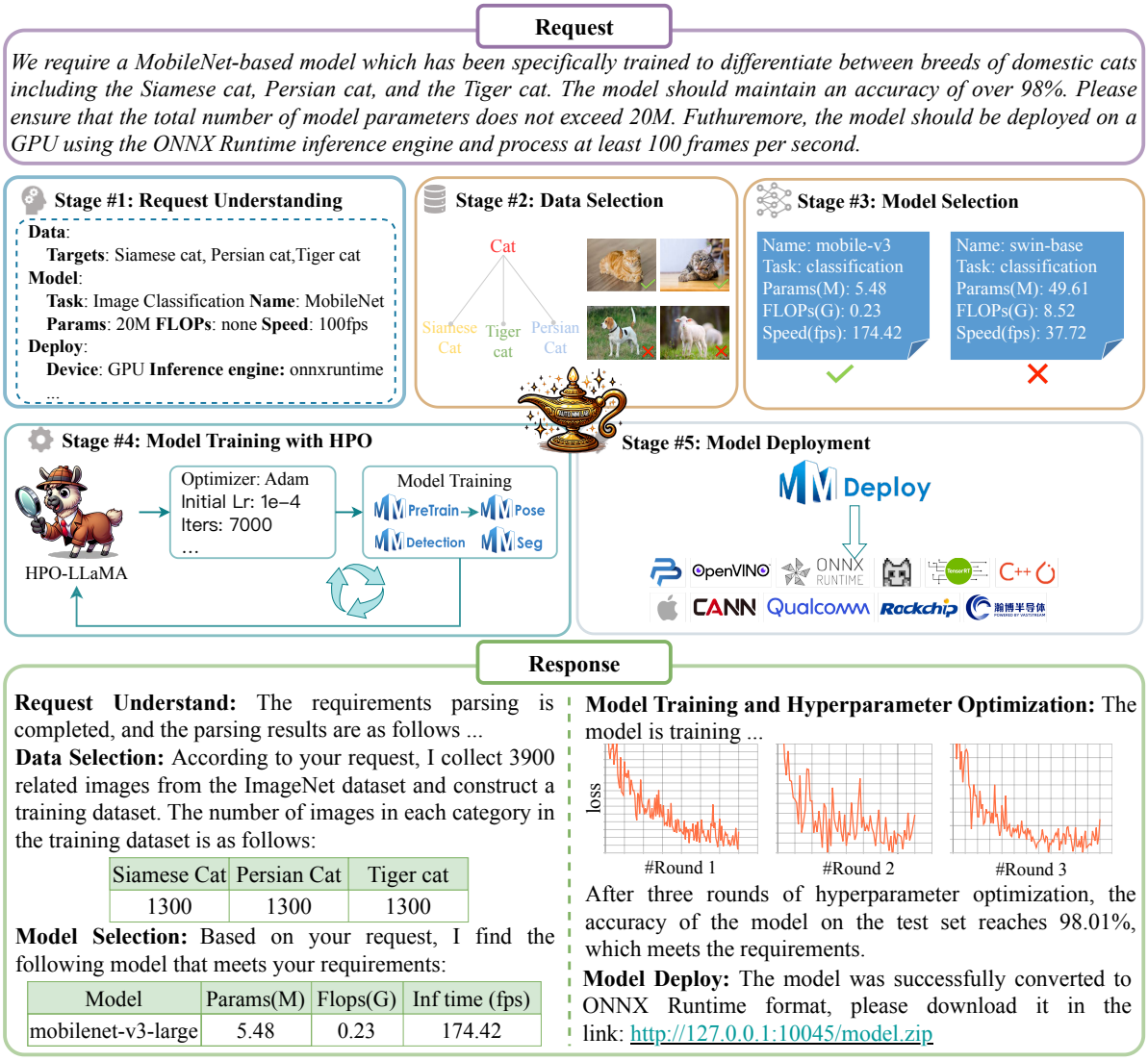


Figure 2: Overview of AutoMMLab. The workflow of AutoMMLab consists of five stages.

predicated task categorization specified in parsed config. If the user has preference to specific model, then calculate the fuzzy matching score between the card's name containing the model structure information and the user's specified requirements. Next, we filter out model cards that did not meet the constraints imposed by users regarding the model's parameters, FLOPs, or the inference speed. Finally, we select the optimal model from the remaining model cards based on fuzzy matching scores and model performance.

Model Training with HPO

Hyperparameter optimization (HPO) or hyperparameter tuning is the problem of choosing a set of optimal hyperparameters for model training, which is crucial for the enhancement of the model performance. Hyperparameters are configuration settings that are not learned during training, but are predefined to control the learning process (*e.g.* learning rate and batch size). AutoMMLab supports various types of HPO baselines, including classic methods (*e.g.* random sampling) and LLM-based methods.

Model Deployment

Model deployment in computer vision means integrating a trained computer vision model into a real-world production environment. Model deployment mainly involves model conversion, graph structure optimization, and model quantization. Our AutoMMLab platform is equipped with MMDeploy (Contributors 2021), a comprehensive model deployment toolbox, for one-click end-to-end model deployment. It supports a growing list of models that are guaranteed to be exportable to various backends (*e.g.* ONNX, NCNN, OpenVINO). The model converters enable converting OpenMM-

Lab models into backend models that can be run on target devices. The resulting package is also with a inference SDK, supporting data preprocessing, model forward and postprocessing modules during the model inference phase.

LAMP Benchmark

Large language models (LLMs) have shown amazing emergent abilities in recent studies, however, there lacks a comprehensive benchmark to evaluate the capability of recent LLMs for AutoML. To fill in this blank, we introduce the LAMP (Language-instructed Automated Model Production) benchmark. The benchmark provides a common ground for evaluating and developing the AutoML platform especially for language-instruction based model production.

Testing Data Collection. The benchmark covers four basic computer vision tasks, *i.e*. image classification, object detection, semantic segmentation, and keypoint detection. To ensure the quality of the test data, we ask 5 professional computer vision researchers to manually design 20 diverse CV model requests for each task. In total, we collect 80 unique user instructions and each instruction is accompanied with a ground truth configuration and a test dataset. To enable comparisons of different LLMs for request understanding, we also provide the corresponding ground-truth configuration for each instruction.

Configuration Definition. We define three dimensions of model requirements in the configuration file: "data", "model", and "deploy". "Data" contains the user's data-related requirements, including: application scenarios, target objects, data modalities, etc. "Model" contains the requirements related to the model, including: task type, model constraints, and the performance that the model is expected to achieve. The constraints of the model may include parameter size, floating point operations per second (FLOPs) and inference speed. "Deploy" contains the user's deployment requirements for the model, including whether to use GPU, which inference engine to use, etc. Note that all values in the configuration can be empty or "none".

We establish three types of evaluation protocols for our LAMP benchmark:

• **Evaluation of Request Understanding.** The evaluation of request understanding assesses the ability of LLMs to parse user's requests into configurations, adhering to the predefined JSON format. We measure performance based on the accuracy of parsing key-value pairs in the configuration. These pairs can be categorized into item-type pairs and list-type pairs, depending on the data format. In the case of list-type pairs, such as target objects, the pair is considered correctly parsed only if all items in the list are parsed accurately. Note that we apply post-processing and fuzzy matching before calculating the parsing accuracy. Two evaluation metrics are used to gauge the request understanding capabilities: (1) Key-level accuracy calculates the average accuracy of each key-value pair. (2) Req-Level accuracy calculates the accuracy of understanding the entire request.

• **Evaluation of Hyperparameter Optimization.** LAMP allows researchers to assess the effectiveness of various Hyperparameter Optimization (HPO) algorithms across a wide range of tasks. Different evaluation metrics are utilized for different tasks, with the mean values of the generated model serving as the primary evaluation metric for HPO. For image classification, the top-1 accuracy is adopted. For object detection, the standard mean Average Precision (mAP) is reported. In the case of semantic segmentation, mean Intersection over Union (mIOU) is employed. For keypoint estimation, mAP based on Object Keypoint Similarity (OKS) is used. Simultaneously calculate the mean value and standard deviation of four tasks for overall evaluation.

• **End-to-end Evaluation.** End-to-end evaluation focuses on the functional quality of the model generated by the request-to-model platform and is assessed using a grading scale ranging from 'F' for total failure to 'P' for perfect conformity to users' specifications: 'F' for total failure, scoring 0 points. The platform either fails to generate a functional model. 'W' for workable model, scoring 1 point. The model is runnable, but it may not fully meet all the users' requirements, such as lower accuracy. 'P' for perfect model, scoring 2 points. The model perfectly matches the users' expectations. There are four types of tasks. The full score for each task is 40, and the total full score is 160.

HPO-LLaMA

We propose to tackle the challenging HPO task by tuning a LLM-based optimizer, called HPO-LLaMA. Figure 3 demonstrates an overview of our proposed HPO-LLaMA workflow. Initially, HPO-LLaMA begins by receiving a nature language description of the HPO problem in conjunction with request-specific details relating to the data and model. Following this, it generates a specific hyperparameter setting tailored to the request. Subsequently, HPO-LLaMA acquires the outcomes from the model trainings conducted using the specified hyperparameter setting on the test data and then endeavors to recommend an optimally enhanced hyperparameter setting.

Search Space We carefully constructed a compact yet expressive search space for HPO, which includes: optimizer type, initial learning rate, learning rate decay policy, weight decay value, batch size, and the training iterations. Considering the heterogeneity of the four tasks, a bespoke search space is delineated for each individual task.

Training HPO-LLaMA We first adopt GPT-assisted data generation method (introduced in Request Understanding) to generate 100 diverse training requests for each of the four CV tasks. And for each training request, we stochastically select 20 distinct hyperparamter settings from the search space. Consequently, we extensively execute model training for 8000 experiments. The outcomes of these experiments are compiled into a substantial dataset, comprising triplets of "request-hyperparameter-performance". we further conduct a comprehensive evaluation to elucidate the correlation between hyperparameters and model performance. This is primarily to substantiate the appropriateness of the predetermined selection for hyperparameter space. Please refer to supplementary for details.

The triplets of "request-hyperparameter-performance" contains rich knowledge about model training and hyperpa-

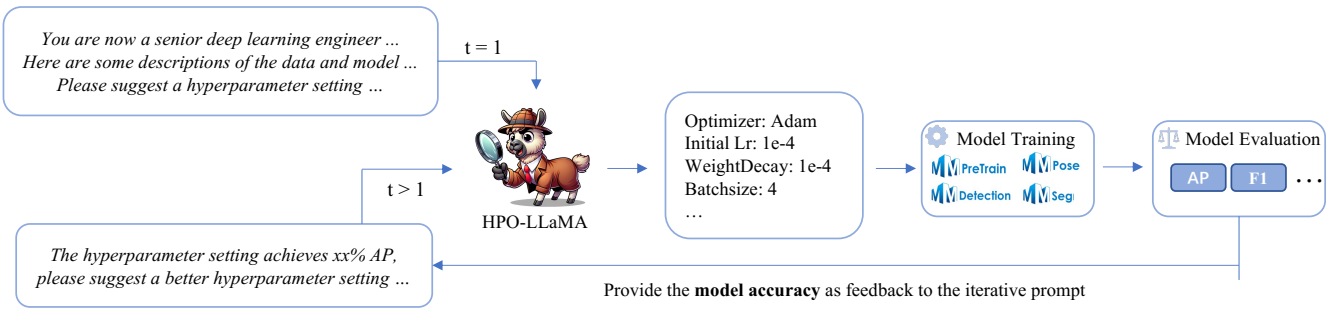


Figure 3: Overview of HPO-LLaMA.

rameter tuning. We utilize the triplet data to construct both single-round and multi-round dialogues, thereby empowering HPO-LLaMA to activate the capability of predicting efficient hyperparameters from these dialogues. Specifically, when constructing the first round of dialogue, the contextual prompts includes the hyperparameter search space, along with a detailed description of the data and the model. The hyperparameter settings to be predicted are derived from the top-k optimal hyperparameter settings corresponding to the request. In multiple rounds of conversations, the contextual prompt provided to HPO-LLaMA includes historical prompt of previous conversations and the performance of the model trained on the test set for the last time the model predicted hyperparameters. The predicted hyperparameter settings are derived from top-k of all hyperparameter settings corresponding to the request and are the best performing hyperparameter setting in current multi-round dialogue. Finally, we construct the training dataset for up to three dialogue rounds and selectively sample a subset from the entire pool of potential multi-turn conversation data. We then use LoRA to fine-tune LLaMA-7B and get our HPO-LLaMA. Considering the limited volume of data in a single-round dialogue, we mix dialogue rounds ranging from one to three together to train the HPO-LLaMA.

Experiments

Request Understanding

Baseline Models. We built several baseline methods by giving LLMs carefully designed prompts. The prompts comprise an instruction and, optionally, a few demonstrations of the anticipated behavior. LLaMA2-7B without instruction fine-tuning cannot accurately follows instructions, so we choose LLaMA2-7B-Chat with instruction fine-tuning as our baseline. As GPTs are currently the most successful LLMs for general language tasks, we opt to employ GPT-3.5-turbo and GPT-4 through API calls as baseline models to evaluate the request understanding ability. We also evaluate the request understanding ability of PaLM2.

Main Results. Table 1 presents the outcomes of various methods evaluated on LAMP utilizing the evaluation metrics delineated in the section of LAMP Benchmark. The findings manifest that LLaMA2-7B-Chat, which did not use the dataset we constructed for fine-tuning, is not able to completely correctly parse the request. Both RU-LLaMA and GPT4 demonstrate exciting capability in understanding requests. RU-LLaMA outperforms PaLM2, GPT-3.5-turbo, and GPT-4 with a 7B-parameter model for both key-level and req-level evaluations.

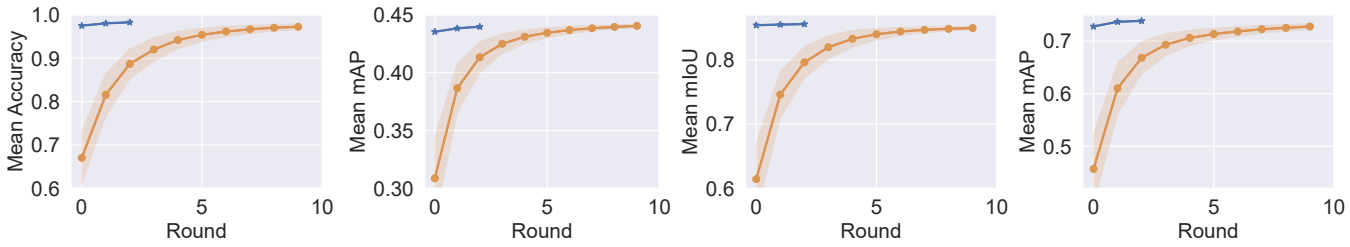
Model	Key-Level			Req-Level
	Item	List	Total	
LLaMA2-7B-Chat	85.71	50.00	77.78	0
PaLM2	96.79	88.13	94.86	63.75
GPT-3.5-turbo	96.43	95.63	96.25	72.50
GPT-4	97.50	93.13	96.53	80.00
RU-LLaMA	**98.57**	**96.88**	**98.20**	**86.25**

Table 1: Evaluation of request understanding (RU). Best results are marked in bold.

Hyper-parameter Optimization (HPO)

Baselines: (1) Random Sampling. In the realm of HPO, random sampling is usually employed as an important baseline. Specifically, we uniformly sample values from our predefined hyperparameter search space. We perform sampling for 10 rounds and subsequently identify the optimal configuration to serve as our baselines. Nevertheless, we observed considerable variance across individual trials resulting from this sampling approach. To address this, we repeated the 10-round process 1,000 times and reported the average and standard deviation. **(2) Bayesian.** We compare with two traditional Bayesian HPO methods, using random forest (BayesianRF) and Gaussian process (BayesianGP) as surrogate models. For each method we perform five rounds of optimization. **(3) LLM prompting.** We compare with several popular LLMs including LLaMA2-7B-Chat, PaLM2, GPT-3.5-turbo, and GPT-4. Specifically, we directly prompt these general-purpose LLMs with some few-shot examples to produce hyperparameter settings. Note that we used the same prompt as our HPO-LLaMA for fair comparisons.

Main Results. Figure 4 shows that our proposed HPO-LLaMA demonstrates a remarkable ability for hyperparameter optimization. It significantly outperforms the random sampling baseline on all four representative tasks. It is worth noting that with only one single round trial, our HPO-LLaMA already produces good hyper-parameters. In Table 2, we compare our proposed HPO-LLaMA with



(a) Image Classification (b) Object Detection (c) Semantic Segmentation (d) Keypoint Detection

Figure 4: HPO results of HPO-LLaMA (blue) and random sampling (orange) baselines on four tasks.

Model	#R	Cls.	Det.	Seg.	Kpt.
BayesianRF	5	0.618±0.287	0.291±0.211	0.847±0.044	0.069±0.136
BayesianGP	5	0.761±0.264	0.280±0.208	0.848±0.041	0.081±0.150
LLaMA2-7B	1	0.839±0.213	0.128±0.164	0.291±0.409	0±0
PaLM2	1	0.964±0.056	0.367±0.196	0.845±0.067	0.719±0.079
GPT-3.5-turbo	1	0.849±0.214	0.364±0.194	0.852±0.044	0.204±0.160
GPT-4	1	0.861±0.188	0.434±0.147	0.803±0.194	0.096±0.158
HPO-LLaMA	1	<u>0.975±0.028</u>	<u>0.435±0.148</u>	<u>0.854±0.042</u>	<u>0.728±0.051</u>
HPO-LLaMA	3	**0.983±0.020**	**0.440±0.150**	**0.856±0.043**	**0.738±0.053**

Table 2: Evaluation of HPO. #R means the number of iteration rounds. The mean and standard deviation on for task are exhibited. Best results are marked in bold, second best results are underlined.

RU	HPO	Cls.	Det.	Seg.	Kpt.	Total
LLaMA2-7B	LLaMA2-7B	0	0	0	0	0
PaLM2	PaLM2	14	25	27	15	81
GPT-3.5-turbo	GPT-3.5-turbo	24	24	25	11	84
GPT-4	GPT-4	17	27	29	14	87
RU-LLaMA	HPO-LLaMA	**31**	**31**	**32**	**18**	**112**

Table 3: End-to-end evaluation on LAMP benchmark.

Bayesian HPO methods using a 5-round evaluation protocol, as well as several popular LLMs with a 1-round evaluation protocol, demonstrating the effectiveness of our approach. In our experiment, we observed that relying on the few-shot capability of LLaMA2-7B-Chat to suggest hyperparameters doesn't consistently yield the anticipated output, and the effectiveness of the suggested hyperparameters is poor. An intriguing finding is that after meticulously designing prompts for GPT-3.5-turbo and GPT-4, they exhibit commendable abilities in proposing hyperparameter settings in tasks relating to image classification, object detection, and semantic segmentation. However, in contrast, for the relatively uncommon task of keypoint detection, they fail to suggest useful hyperparameter settings. The HPO-LLaMA model proposed in our study demonstrates the capacity to provide useful hyperparameter recommendations across all four computer vision tasks, and achieves the best performance. At the same time, as the number of rounds increases, HPO-LLaMA can continue to search for better hyperparameter settings.

End-to-end Evaluation

Table 3 reports the results of end-to-end evaluation on LAMP benchmark. There are four types of model training tasks, *i.e.* image classification (Cls.), object detection (Det.), semantic segmentation (Seg.) and keypoint detection (Kpt.). We employ various LLMs for request understanding (RU) and hyperparameter optimization (HPO) respectively. We find that our AutoMMLab empowered by RU-LLaMA and HPO-LLaMA significantly outperform GPT-3.5 and GPT-4 with a 7B-parameter model, scoring 112/160 in the overall evaluation and achieves the best performance on all four computer vision tasks.

Conclusion

This paper presents AutoMMLab, an innovative AutoML system that harnesses the power of LLMs to fully automate the model development process for computer vision tasks, guided by natural language instructions. We develop RU-LLaMA for understanding the user's requests and scheduling the whole workflow. Furthermore, we propose HPO-LLaMA, a novel LLM-empowered method for effective hyperparameter optimization. Additionally, we present the LAMP (Language-instructed Automated Model Production) Benchmark to facilitate the evaluation of end-to-end prompt-based model production. We equip our AutoMMLab with various LLMs and test on LAMP, demonstrating the superiority of our proposed RU-LLaMA and HPO-LLaMA. We hope our work could have broad applicability and inspire further research in related areas.

References

Bergstra, J.; and Bengio, Y. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, K.; Wang, J.; Pang, J.; Cao, Y.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Xu, J.; Zhang, Z.; Cheng, D.; Zhu, C.; Cheng, T.; Zhao, Q.; Li, B.; Lu, X.; Zhu, R.; Wu, Y.; Dai, J.; Wang, J.; Shi, J.; Ouyang, W.; Loy, C. C.; and Lin, D. 2019. MMDetection: Open MMLab Detection Toolbox and Benchmark. arXiv:1906.07155.

Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H. W.; Sutton, C.; Gehrmann, S.; et al. 2022. Palm: Scaling language modeling with pathways. arXiv:2204.02311.

Contributors, M. 2020a. MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark. https://github.com/open-mmlab/mmsegmentation.

Contributors, M. 2020b. OpenMMLab Pose Estimation Toolbox and Benchmark. https://github.com/open-mmlab/mmpose.

Contributors, M. 2021. OpenMMLab's Model Deployment Toolbox. https://github.com/open-mmlab/mmdeploy.

Contributors, M. 2023. OpenMMLab's Pre-training Toolbox and Benchmark. https://github.com/open-mmlab/mmpretrain.

Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; and Schiele, B. 2016. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3213–3223.

Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.

Franceschi, L.; Donini, M.; Frasconi, P.; and Pontil, M. 2017. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, 1165–1173. PMLR.

Guo, S.; Deng, C.; Wen, Y.; Chen, H.; Chang, Y.; and Wang, J. 2024. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. arXiv:2402.17453.

Hollmann, N.; Müller, S.; and Hutter, F. 2023. CAAFE: Combining Large Language Models with Tabular Predictors for Semi-Automated Data Science. In *1st Workshop on the Synergy of Scientific and Machine Learning Modeling @ ICML2023*.

Hong, S.; Lin, Y.; Liu, B.; Liu, B.; Wu, B.; Zhang, C.; Wei, C.; Li, D.; Chen, J.; Zhang, J.; et al. 2024. Data interpreter: An llm agent for data science. arXiv:2402.18679.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Huang, Q.; Vora, J.; Liang, P.; and Leskovec, J. 2024. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. In *Forty-first International Conference on Machine Learning*.

Jin, H.; Song, Q.; and Hu, X. 2019. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 1946–1956.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *ECCV*.

Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Ruhkopf, T.; Sass, R.; and Hutter, F. 2022. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54): 1–9.

Liu, H.; Li, C.; Li, Y.; and Lee, Y. J. 2023a. Improved Baselines with Visual Instruction Tuning. arXiv:2310.03744.

Liu, H.; Li, C.; Wu, Q.; and Lee, Y. J. 2023b. Visual Instruction Tuning. arXiv:2304.08485.

Miller, G. A. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11): 39–41.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Pedregosa, F. 2016. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, 737–746. PMLR.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847–855.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023a. Llama: Open and efficient foundation language models. arXiv:2302.13971.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288.

Viswanathan, V.; Zhao, C.; Bertsch, A.; Wu, T.; and Neubig, G. 2023. Prompt2Model: Generating Deployable Models from Natural Language Instructions. arXiv:2308.12261.

Wang, B.; Xu, H.; Zhang, J.; Chen, C.; Fang, X.; Kang, N.; Hong, L.; Zhang, W.; Li, Y.; Liu, Z.; Li, Z.; Liu, W.; and Zhang, T. 2020. VEGA: Towards an End-to-End Configurable AutoML Pipeline. arXiv:2011.01507.

Wang, Y.; Kordi, Y.; Mishra, S.; Liu, A.; Smith, N. A.; Khashabi, D.; and Hajishirzi, H. 2022. Self-instruct:

Aligning language model with self generated instructions. arXiv:2212.10560.

Zhang, M. R.; Desai, N.; Bae, J.; Lorraine, J.; and Ba, J. 2023. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Zheng, M.; Su, X.; You, S.; Wang, F.; Qian, C.; Xu, C.; and Albanie, S. 2023. Can GPT-4 Perform Neural Architecture Search? arXiv:2304.10970.

Zimmer, L.; Lindauer, M.; and Hutter, F. 2021. Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3079 – 3090.