SCIENCE CHINA
Information Sciences

• **Supplementary File** •

ProxyLLM: Augmenting LLMs with Proxy Models for Tool Utilization through Learning from LLMs

Xiaomao Zhou[1], Qingmin Jia[1], Yan Zhang[3*], Liwen Wang[3], Renchao Xie[1,2], Tao Huang[1,2] & Yunjie Liu[1]

[1]*Purple Mountain Laboratories, Nanjing 211111, China;*
[2]*School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China;*
[3]*China Unicom Research Institute, Beijing 100048, China*

Appendix A Experiments

Appendix A.1 Dataset and Evaluation Setup

Benchmark To the best of our knowledge, no existing benchmarks contain complex tools that require cooperation across different categories while involving long-term planning tool-use tasks. To address this gap, we develop two new test sets: ToolBench+ and ToolAlpaca+. For ToolBench+, based on the 3.4k tools across 49 categories in the original ToolBench, we prompt an LLM to divide these 49 categories into k clusters, simulating k proxy models. The LLM then selects tools from different clusters to combine into complex tasks. Subsequently, we engage human experts to curate and select high-quality generated data. ToolAlpaca+ is constructed using a similar methodology from ToolAlpaca [3]. In total, ToolBench+ and ToolAlpaca+ comprise 267 and 312 test cases, respectively, with an average requirement of 8.9 and 10.3 tools per task.

In addition, we construct a dataset, CPND, grounded in an actual Computing Power Network (CPN) system. This dataset encompasses a complete service process, including intent parsing, node selection, service deployment, operational monitoring, and service optimization. At each stage of this process, multiple functional components have been meticulously designed to facilitate a wide array of choices for the large-scale model to select and employ. The system provides over 60 component options distributed across 5 stages. Within the scope of this application scenario, we curate approximately 56 comprehensive data records. These records encapsulate not only the original user requirement specifications but also provide a detailed account of the component selections executed at each service phase. Each data record contains at least 8 functional component choices, ensuring the richness and diversity of the data.

Evaluation metrics. We assess the performance across two dimensions: accuracy and efficiency. In terms of accuracy, following previous work [4] [5], our evaluation integrates three key performance metrics: (1) Success Rate (SR%), quantifying the percentage of queries that are fully and successfully resolved; (2) Correct Path Rate (Path%), assessing the extent to which the model's selected tools match the predetermined optimal sequence; and (3) Correct Tool Precision (Pre%), evaluating the precision of the model's tool selections when compared against the benchmark tool sequence. As for efficiency, we track two key metrics: (1) the Average Number of LLM Involvement Steps (IS) required to complete a task, and (2) the Total System Parameter Count (TSPC) of the entire system.

Baselines We compare our method with a number of well-known baselines, including: (1) Chameleon, an LLM-based agent that directly generates multi-step plans for tool use and then sequentially executes the plan; (2) ReAct, which prompts LLM to generate the chain-of-thought and actions in an interleaved manner; (3) ToolLLM-DFSDT, which enhances LLMs with the Depth First Search-based Decision Tree (DFSDT) to select tools to solve a task. (4) AVATAR [6], which deliver insightful and comprehensive prompts to the LLM agent by contrastive reasoning between positive and negative examples t to effectively leverage tools. (5) Reflexion [7], which leverages verbal reinforcement to learn from past mistakes. (5) Self-Refine [8], which iteratively refines its policy using self-generated feedback without requiring additional supervised data or reinforcement learning. We also consider baselines with multi-agent architecture, including (1) RestGPT, which exploits the power of multiple LLMs and conducts a coarse-to-fine online planning mechanism to enhance the abilities of task decomposition and API selection. (2) ConAgents [9], which coordinates three specialized LLMs for tool selection, tool execution, and action calibration separately. (3)α-UMi [10], which decomposes the tool learning task into three distinct sub-tasks delegated to three small LLMs: planner, caller, and summarizer. For further comparison, we also establish two baseline models: ProxyLLM*heavy*, which exclusively utilizes LLMs as proxy models, and ProxyLLM*mid*, which leverages both LLMs and smaller models as proxy models in a coordinated manner.

Implementation Details

We employ various state-of-the-art LLMs as backbones in our method and all baselines, including ChatGPT [11], ToolLLaMA7B, ChatGLM4-9B [12], and GPT4 [13]. For the proxy model component, we implement a BERT-Base model for the

* Corresponding author (email: xiaomaozhou26@gmail.com)

tool proposer, utilizing a dataset of approximately 7.5k data points. The training protocol for this model involves a learning rate of 510^{-5}, a batch size of 32, and a maximum input length of 1024, executed over a maximum of 4000 training steps. For the DRL-based tool selector, we utilize a policy network architecture comprising a multilayer perceptron (MLP) with two hidden layers, each of size 128×128. Additionally, we incorporate a single self-attention head with a dimensionality of 128 to effectively encode the state representation. The DRL algorithm is implemented within the PyTorch framework and optimized using the Adam optimizer. The learning rate is set to $4e^{-5}$, complemented by a discount factor of 0.9. During training, the batch size is maintained at 64. The model undergoes a minimum of 10^5 steps of training, with evaluations conducted at every 500 time steps. The policy deviation tolerance is dynamically modulated via a linear schedule, initiating at an initial value of 0.26 and progressively decreasing in accordance with the remaining training progress.

Appendix A.2 Main Results

Table A1 demonstrates the experimental performances of all methods. We elaborate on our observations from the following perspectives. Firstly, Chameleon, ReAct, ToolLLM-DFSDT, and AVATAR all performed poorly on these three datasets. This is because the tasks in the datasets are relatively complex, often requiring multiple tool invocations. Relying solely on large models for planning and execution usually fails to ensure accuracy over longer chains of thought. Although AVATAR employs contrastive reasoning to enhance tool accuracy, it still cannot guarantee high performance in more complex tasks. Secondly, Multi-LLM based approaches, including RestGPT, ConAgents, alpha, and the ProxyLLM series, consistently outperform Single-LLM based approaches such as Chameleon, ReAct, ToolLLM-DFSDT, and AVATAR across all metrics. This is because Multi-LLM based approaches utilize multiple components to handle different tasks separately, with each component responsible for specific tasks like task planning and tool calling. While Reflexion and Self-Refine demonstrate performance on par with them, their effectiveness comes at the cost of either supplementary training or computationally expensive iterative optimization processes. Thirdly, the ProxyLLM series excels over other multi-LLM baselines due to its task-based division approach. This task-based division offers two major advantages: on one hand, it ensures that each individual proxy model can fully address the task assigned to it, thereby guaranteeing the integrity and coherence of task execution. On the other hand, since each proxy model focuses on a specific type of task, they can continuously learn and optimize to improve their performance in that specific area. Additionally, the ProxyLLM series offer greater flexibility and scalability, as each proxy model can be independently updated and optimized without the need to retrain the entire system. This design enables the ProxyLLM series to adapt more quickly to new tasks and data while maintaining high performance. Fourthly, the ProxyLLM series maintain consistent performance across diverse datasets. Notably, despite proxyLLM$_{heavy}$ utilizing LLMs as proxy models, it exhibits minimal additional improvement compared to the overall ProxyLLM series in various scenarios. Furthermore, in proxyLLM$_{mid}$, the synergy between LLMs and DRL-based small models as proxy models achieves performance parity with proxyLLM$_{heavy}$. These results highlight the effectiveness of our proposed method, enabling small models to achieve performance levels on par with LLMs for specific tasks. This not only proves the potential of our method in enhancing the performance of small models but also demonstrates its broad applicability and robustness across various scenarios and datasets.

To further demonstrate the efficiency of the proposed ProxyLLM, we compare the performance of different approaches in terms of IS and TSPC, where we adopt 2 proxy models in the ProxyLLM for a fair comparison. Figure A1 illustrates that the ProxyLLM series require significantly fewer IS than other methods. In other frameworks, LLMs must perform fine-grained task decomposition and directly interact with tools to solve problems step-by-step. In contrast, within the ProxyLLM, the large model only needs to decompose tasks and invoke proxy models to handle sub-tasks, with the majority of the workload being offloaded to the proxy models. The varying IS for different datasets in Figure A1(a) can be attributed to the necessity of LLM intervention when proxy models are unable to resolve tasks independently. Additionally, compared to other baselines, ProxyLLM excels in TSPC because it necessitates only the use of low-parameter proxy models, which can be trained to substitute for LLMs in task completion. For example, ConAgents and ProxyLLM $_{heavy}$ have a parameter count about three times that of ProxyLLM. This not only highlights the efficiency of our approach but also underscores the potential for cost reduction and enhanced scalability in practical applications.

Appendix A.3 Result Analysis

We further conduct a series of experiments to investigate the impact of different settings in our method.

Appendix A.3.1 *Overall Performance in different settings*

To evaluate the model's performance under varying numbers of tools, we conducted a series of comparative experiments with different tool quantities. Based on the constructed ToolBench+ dataset, we randomly increased the number of tools by one, two, and four times. As shown in the Table A2, as the number of tools increases, the performance of other baselines gradually deteriorates. For example, when the number of tools increases from 200 to 800, the SR of ReAct, RestGPT, and ConAgents decrease by 43.06%, 40.05%, and 29.02%, respectively. This decline can be attributed to two factors: first, the expanded choice space challenges the selection capabilities of LLMs, and second, the limited length constraint hinders the comprehensive description of tools. In contrast, ProxyLLM maintains stable and superior performance across various tool quantities. For example, when the number of tools increases from 200 to 800, the SR, Path, and Pre of ProxyLLM decrease by only 1.56%, 1.05%, and 1.39%, respectively. Notably, both ProxyLLM$_{heavy}$ and ProxyLLM$_{mid}$ also experience performance degradation as the number of tools increases. This is because different proxy models are responsible for specific subsets of tools within their respective domains and can learn how to use these tools through training. Furthermore, for proxy models, the knowledge of how to use tools is acquired through DRL-based interaction and LLM guidance, rendering

Table A1 Performance comparison of various approaches across different datasets. We employ various state-of-the-art LLMs as backbones in our method and all baselines, including ToolLLaMA7B, ChatGLM4-9B, GPT-4, and GPT-4o.

		ToolBench+			ToolAlpaca+			CPND		
Model	**Method**	**SR**	**Path**	**Pre**	**SR**	**Path**	**Pre**	**SR**	**Path**	**Pre**
ToolLLaMA-7B	Chameleon	19.77	49.58	48.98	12.25	40.54	33.96	24.11	37.16	25.52
	ReAct	22.87	42.75	43.83	13.65	31.73	39.37	27.64	27.56	34.98
	ToolLLM-DFSDT	32.62	42.23	44.52	16.11	40.54	28.60	30.89	38.11	31.04
	AVATAR	35.73	44.64	44.64	20.50	35.95	38.73	30.77	32.81	39.66
	Reflexion	46.45	50.92	49.31	39.54	42.33	45.18	42.12	42.45	43.21
	Self-Refine	42.24	48.52	47.68	35.26	37.46	45.53	40.05	39.84	43.71
	RestGPT	42.18	50.19	48.98	40.16	38.11	48.44	44.02	42.86	45.45
	ConAgents	48.17	55.55	53.29	41.32	59.86	49.82	48.47	52.70	44.57
	α-UMi	47.24	53.47	52.16	38.88	48.80	44.65	47.63	42.89	47.56
	$ProxyLLM_{heavy}$	**62.88**	62.23	70.11	**57.98**	**68.40**	**67.73**	58.39	**57.99**	**64.72**
	$ProxyLLM_{mid}$	62.62	62.89	**71.43**	56.67	65.85	65.73	**59.34**	54.25	63.71
	ProxyLLM	62.55	**63.10**	71.08	53.88	66.55	63.43	57.79	57.49	62.70
ChatGLM4-9B	Chameleon	29.33	68.87	64.91	17.65	52.86	49.53	34.10	46.77	38.34
	ReAct	34.87	56.84	66.14	19.38	41.31	58.46	37.16	41.90	54.92
	ToolLLM-DFSDT	50.34	61.38	65.26	23.56	53.08	40.73	42.46	54.92	47.30
	AVATAR	56.35	64.30	64.27	29.73	51.98	52.98	47.18	51.63	51.65
	Reflexion	62.23	69.76	65.67	49.08	65.95	62.10	60.99	59.84	59.63
	Self-Refine	61.27	69.21	65.51	45.13	64.12	60.31	58.39	58.21	58.46
	RestGPT	63.98	70.27	68.91	59.18	58.43	63.56	57.80	62.71	68.26
	ConAgents	70.78	**78.56**	75.87	63.03	72.19	71.69	67.54	70.95	58.89
	α-UMi	64.19	71.58	66.14	55.53	70.60	65.14	65.59	62.58	62.27
	$ProxyLLM_{heavy}$	74.74	75.53	82.22	**65.91**	74.61	**76.11**	**68.38**	**71.86**	**73.11**
	$ProxyLLM_{mid}$	**74.83**	76.95	**83.85**	65.72	74.39	76.00	65.73	69.36	71.53
	ProxyLLM	74.56	75.11	83.60	64.27	**75.14**	76.01	67.83	69.75	72.85
GPT-4	Chameleon	30.38	69.48	72.41	19.26	58.78	52.12	35.60	52.49	39.69
	ReAct	36.43	61.60	65.62	20.85	42.47	62.86	41.01	41.16	57.58
	ToolLLM-DFSDT	52.94	64.23	66.08	22.93	56.99	43.04	44.21	60.32	50.01
	AVATAR	54.87	69.73	66.29	31.69	54.81	54.42	49.22	52.97	54.08
	Reflexion	66.71	74.04	70.18	52.95	67.10	64.43	63.27	59.16	65.71
	Self-Refine	65.92	73.76	69.92	51.53	66.28	63.77	62.33	58.74	64.94
	RestGPT	70.75	71.44	74.50	60.15	63.05	70.40	64.73	70.12	69.62
	ConAgents	**78.63**	**80.46**	80.42	64.75	78.92	79.14	68.11	73.75	64.17
	α-UMi	70.65	75.48	71.48	60.03	71.19	67.77	67.95	61.22	69.59
	$ProxyLLM_{heavy}$	76.49	79.28	86.91	**68.67**	**80.45**	79.90	**70.56**	**74.57**	**77.53**
	$ProxyLLM_{mid}$	75.74	78.65	**87.98**	67.35	79.89	**81.56**	70.10	71.69	76.29
	ProxyLLM	76.84	78.92	87.01	68.27	79.68	80.21	69.32	72.00	75.39
GPT-4o	Chameleon	29.51	65.72	67.34	18.47	54.27	50.05	30.94	48.75	38.87
	ReAct	35.23	59.15	64.39	19.67	41.37	59.93	39.22	41.47	55.21
	ToolLLM-DFSDT	52.66	61.47	65.67	23.25	53.49	41.53	43.91	57.76	48.16
	AVATAR	55.48	69.49	64.78	31.01	52.95	53.92	48.89	52.41	52.04
	Reflexion	63.25	73.56	67.33	52.32	66.53	63.96	63.06	59.36	64.39
	Self-Refine	62.66	72.21	67.82	55.22	67.01	62.62	61.34	58.05	63.28
	RestGPT	69.58	71.24	73.29	60.1	61.94	68.44	64.14	68.65	69.49
	ConAgents	76.12	**78.39**	79.58	64.51	77.95	78.32	67.99	73.12	63.08
	α-UMi	69.84	74.92	70.85	59.42	71.05	67.31	67.78	61.68	68.51
	$ProxyLLM_{heavy}$	75.96	78.16	85.5	**67.84**	78.27	78.76	**69.91**	**73.76**	**76.22**
	$ProxyLLM_{mid}$	75.51	78.23	**86.95**	66.94	78.52	**80.17**	69.01	71.11	75.14
	ProxyLLM	**76.28**	78.33	86.12	67.54	**78.97**	79.51	68.95	71.61	74.86

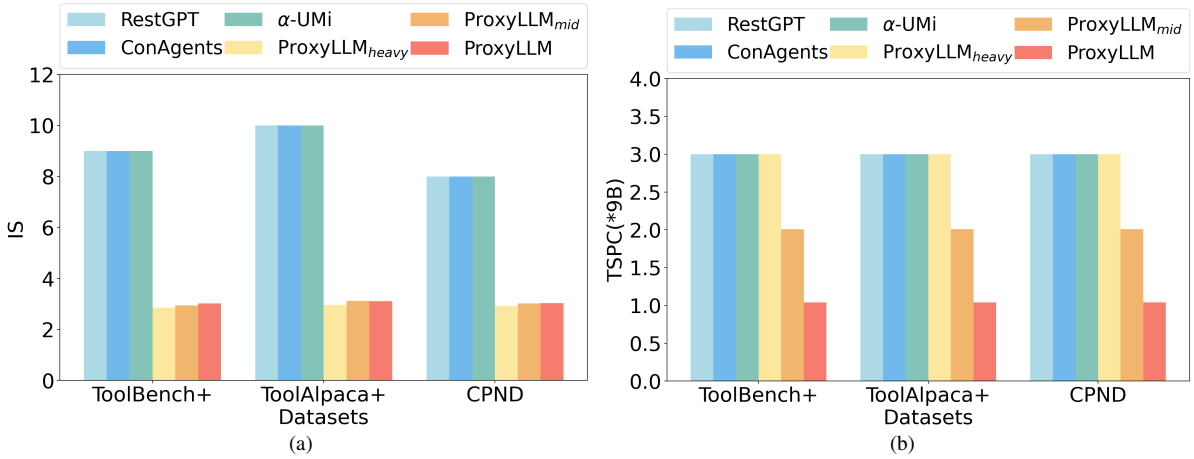

Figure A1 Performance comparison of different approaches in terms of IS and TSPC. (a) IS. (b) TSPC.

the quality of tool descriptions less impactful on the learning outcome. This highlights the robustness and adaptability of the ProxyLLM approach in handling diverse tool sets.

Table A2 Performance comparison of various approaches under varying numbers of tools. We report the average results of three experiments conducted under different conditions.

	100+ Tools			200+ Tools			400+ Tools			800+ Tools		
Method	**SR**	**Path**	**Pre**	**SR**	**Path**	**Pre**	**SR**	**Path**	**Pre**	**SR**	**Path**	**Pre**
Chameleon	31.93	72.23	68.12	30.12	68.54	65.37	28.29	58.32	56.83	17.15	34.04	32.89
ReAct	35.68	59.07	70.13	35.29	60.29	68.95	27.13	53.23	57.76	19.26	37.84	37.57
ToolLLM-DFSDT	52.37	63.25	69.25	51.83	61.92	69.02	46.28	56.02	58.74	25.32	42.02	42.76
AVATAR	59.24	66.18	65.32	58.34	64.82	63.23	54.37	60.03	55.43	27.84	45.13	44.23
Reflexion	68.66	76.32	74.41	67.58	74.56	72.7	56.25	60.11	57.76	37.45	43.42	45.38
Self-Refine	70.54	78.35	76.22	69.43	76.51	74.59	56.43	59.51	56.63	34.57	45.08	45.61
RestGPT	66.68	71.17	68.98	65.54	69.32	70.05	56.18	57.23	62.82	39.29	44.53	45.82
ConAgents	73.37	**81.39**	78.95	72.20	79.43	77.43	63.19	70.63	69.43	51.25	57.57	45.95
α-UMi	65.98	72.18	70.57	63.45	71.33	68.31	61.89	60.37	63.35	47.36	42.28	47.79
ProxyLLM$_{heavy}$	**75.17**	80.39	83.95	74.31	80.03	**82.47**	72.47	78.45	79.11	70.09	72.54	72.43
ProxyLLM$_{mid}$	75.04	79.18	83.97	**74.34**	**80.82**	82.39	72.73	78.36	78.85	71.72	77.45	77.41
ProxyLLM	74.87	79.26	**84.09**	74.24	79.12	82.28	**73.19**	**79.41**	**81.89**	**73.08**	**79.29**	**81.13**

Furthermore, we assess the performance of various models under different lengths and qualities of tool descriptions. For the original tool documentation, we utilize an LLM to compress it, for example, reducing its length by half. Alternatively, we simplify the tool documentation into concise instructions using EASYTOOL. As shown in Table A3, the average token counts of CPND for the original tool descriptions, their reduced versions, and those generated by EASYTOOL are 1896, 898, and 854, respectively. Figure A2 presents the comparison results for ReAct, ConAgents, ProxyLLM$_{heavy}$, and ProxyLLM. It is evident that ReAct and ConAgents are significantly affected by the quality of tool descriptions. When the descriptions are randomly shortened to half their original length, their SR and Path decrease substantially. Conversely, when EASYTOOL is used to optimize the tool descriptions, their SR and Path improve. Similarly, ProxyLLM$_{heavy}$'s performance is also influenced by the quality of tool descriptions, as it relies on LLMs as proxy models, which are susceptible to these changes during the selection process. In contrast, ProxyLLM maintains high SR and Path even with low-quality tool descriptions and further improves as the quality of the descriptions increases. This is because ProxyLLM is able to learn optimal tool invocation strategies through its training process.

Appendix A.3.2 *Performance of the Tool Proposer*

We leverage LLMs to synthesize dataset for knowledge transfer of tool proposal. To validate the effectiveness of the aforementioned method and the tool selection capability of the tool proposer, we conduct a series of comparative experiments. These experiments include variations in the number of k clusters used for dataset construction, different amounts of training data, different model sizes, and the use of labeled versus unlabeled data. Specifically, when using LLMs to generate datasets,

Table A3 The average token counts for original tool descriptions(Token_{doc}), their reduced versions($\text{Token}_{rand.}$), and those generated by EASYTOOL($\text{Token}_{Easytool}$).

Dataset	Token_{doc}	Token_{rand}	$\text{Token}_{Easytool}$
ToolBench+	2489	1237	863
ToolAlpaca+	2326	1268	829
CPND	1896	898	854

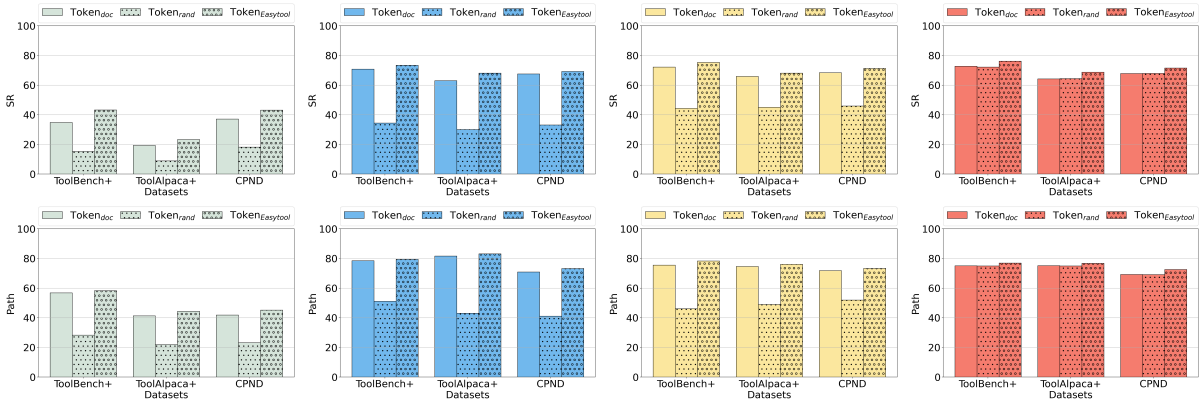

Figure A2 Performance of various models under varying tool description lengths and qualities. The first row represents SR results, the second row represents Path results. The first column represents ReAct, the second column represents ConAgents, the third column represents ProxyLLM_{heavy}, and the fourth column represents ProxyLLM.

we select k=3, 4, 6, 8, and 10 to create different datasets for model training, thereby comparing the tool selection capability under tasks of varying complexity. During the model training process, we compare the efficiency and effectiveness under different scenarios by limiting the amount of training data and using different pruning ratios for the BERT-based tool proposer. Additionally, we compare the training differences between using labeled and unlabeled data. Figure A3(a) shows that KD with rationales enables the tool selection capability of the tool proposer to remain relatively robust even as task difficulty increases, whereas standard KD experiences a decline in accuracy on complex tasks. Figure A3(b) demonstrates that when compared to standard KD, distilling with rationales achieves better performance with a substantially smaller number of training examples, significantly improving the data efficiency of learning task-specific models. Figure A3(c) illustrates that KD with rationales surpasses the performance of standard KD with a much smaller model size, drastically lowering the deployment cost. Figure A3(d) further shows that whether using labeled or unlabeled data, KD with rationales can achieve good results, while standard KD fails to perform well in scenarios with fewer labeled data.

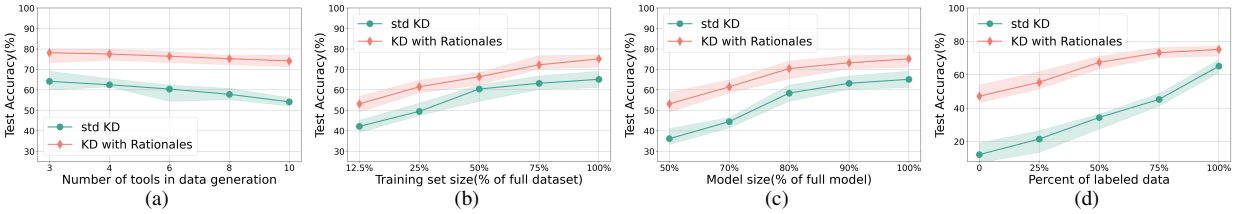

Figure A3 Performance of the Tool Proposer under various conditions: (a) Accuracy relative to the number of tools in data generation (task difficulty). (b) Accuracy as a function of training set size. (c) Accuracy as a function of model size. (d) Accuracy as a function of labeled data percentage.

Appendix A.3.3 *Performance of the Tool Decider*

To assess the effectiveness of the tool decider, we benchmark four algorithms: a value-based method (DQN [14]), a policy-gradient method (A2C), and an LLM-based method (LGDRL [15]). We record the average return during training in Figure A4, where the solid lines denote mean performance and the shaded regions indicate 95% confidence intervals. To ensure a fair comparison, all DRL methods involving expert guidance use the same LLM expert to provide guidance and all DRL methods are trained three times under the same seed sequence. In ToolBench+ depicted in Figure A45(a), our model demonstrates rapid improvement during the initial training phase and converges to the highest final return. In contrast,

DQN and A2C exhibit relatively slower learning speeds, suggesting instability near convergence. While other baselines eventually stabilize, their end-stage returns remain significantly lower than ours, indicating a substantial performance gap.In ToolAlpaca+ shown in Figure A4(b), our method achieves high returns early in the training process, stabilizing around 400 epochs and consistently maintaining the best performance thereafter. In comparison, DQN and A2C begin to converge after 700 and 800 epochs, respectively, ultimately reaching a suboptimal plateau. In CPND illustrated in Figure A4(c), our model maintains a leading position throughout the training process and converges to a near-maximal reward. LGDRL intermittently shows competitive performance but tends to be less efficient overall. Both DQN and A2C exhibit slow learning speeds and achieve a lower reward ceiling compared to our model.

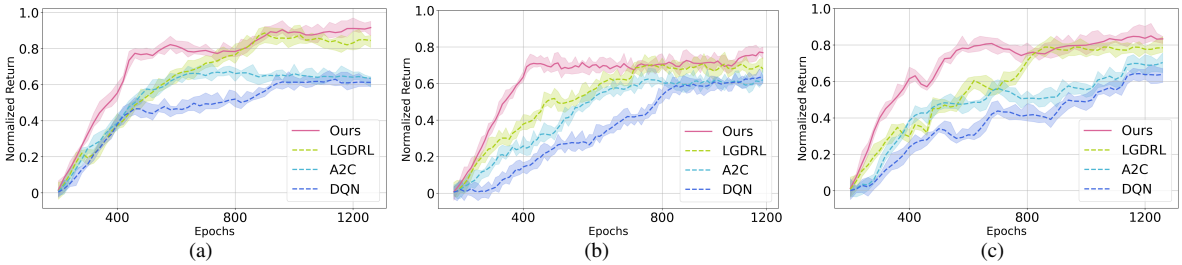

Figure A4 Performance comparison between our Tool Decider and traditional DRL training on three datasets: (a) ToolBench+, (b) ToolAlpaca+, and © CPND. The results highlight the effectiveness of our approach in these specific tasks.

Appendix A.3.4 *Analysis of Total Training Costs*

To validate the practicality of ProxyLLM, we conduct a comparative analysis of the training costs associated with our framework against those of fine-tuning a single, large-scale LLM (e.g., ChatGLM4-9B). As presented in Table. A4, our ProxyLLM demonstrates significant advantages across multiple dimensions. First, in terms of training strategy and model size, ProxyLLM adopts a distributed approach utilizing multiple lightweight proxy models (approx. 0.3M parameters each via LoRA). This stands in stark contrast to the monolithic LLM strategy, which requires fine-tuning a massive 9B parameter model (even with LoRA updates of ≈25M parameters). This drastic reduction in model size per task substantially lowers the barrier for hardware requirements. Second, regarding data efficiency, the specialized nature of our proxy models means each domain requires only a small dataset (e.g., 3K+ samples) to achieve expertise. Conversely, fine-tuning a single LLM to generalize across diverse tools necessitates a massive dataset (30K+ samples) to prevent catastrophic forgetting and ensure robust performance, incurring significant data curation costs. Third, ProxyLLM excels in convergence speed. By simplifying the optimization objective for each proxy, we observe a convergence rate approximately 5x faster than that of the end-to-end LLM. Furthermore, the architecture supports parallel training, allowing different proxies to be trained simultaneously, thereby reducing wall-clock time. Finally, ProxyLLM significantly reduces maintenance costs. Integrating a new tool requires only isolated retraining of the specific relevant proxy, whereas updating a monolithic LLM often requires expensive, full-scale re-tuning.

Table A4 Comparative analysis of training costs and efficiency between the Single LLM approach and ProxyLLM.

Model	Training Strategy	Model Size	Data per Domain	Parallel Training	Convergence Time	Update Cost
LLM	Unified	Single (FUll 9B, LoRA ≈25M)	Large(30K+)	No	Slow(≈1x)	High
ProxyLLM	Distributed	Multiple (≈0.3M/per)	Small(3K+)	Yes	Fast(≈5x)	Low

Appendix A.4 Ablation Studies

Appendix A.4.1 *Impact of Tool Candidate Value p*

We evaluate the impact of the number of tool candidates, p, on the final performance, which is the output size of the tool proposer. To this end, we gradually increase p from 1 to 20. As shown in Table A5, when p is 1, i.e., the tool proposer selects only one tool based on the task, no proxy decider is needed, and the performance of the ProxyLLM is poor. This is because the proxy model only acquires knowledge from the LLM through supervised learning, lacking DRL for interactive learning, which results in an incomplete knowledge base for problem-solving. When p is within the range of 5 to 10, we find that the performance of the proxy LLM is generally better. When p is larger, the performance of the ProxyLLM does not further improve, and the learning efficiency of the tool decider decreases. We believe the reasons are as follows: When p is small, the tool proposer's choices are relatively conservative, which may cause the proxy LLM to miss some potentially useful tools. As p increases, the tool proposer's choices become more diverse, providing more exploration opportunities for the proxy LLM. However, when p is too large, the tool decider needs to choose from a large number of tool candidates,

which increases the complexity and learning difficulty of its decision-making. Therefore, choosing an appropriate p value is crucial for balancing exploration and exploitation and improving the overall performance of the ProxyLLM.

Table A5 Performance comparison across varying numbers of tool candidates. We use the ChatGLM4-9B as the LLM in our experiments and report the average results of three tests.

p	Avg. Reward	Cvg. Epoch	ToolBench+			ToolAlpaca+			CPND		
			SR	Path	Pre	SR	Path	Pre	SR	Path	Pre
1	-	-	62.31	61.05	58.24	47.56	49.24	51.03	51.03	47.27	48.31
2	0.78	∼400	72.99	72.63	78.45	61.35	71.12	72.09	63.78	66.26	67.53
5	0.81	∼430	74.11	74.14	83.09	63.62	74.37	75.18	64.16	65.18	70.63
8	0.83	∼430	74.43	75.20	83.45	64.11	75.02	76.02	67.81	68.05	71.65
10	0.82	∼440	74.39	74.68	83.47	63.68	74.98	75.93	66.88	68.23	71.56
20	0.81	∼580	74.51	75.18	83.38	64.07	73.89	75.42	66.23	67.52	70.28

Appendix A.4.2 *Effect of Proxy Model Quantity*

To validate the impact of varying numbers of proxy models on the final performance, we conducted a series of comparative experiments employing different quantities of proxy models across various datasets. As illustrated in the Figure, the SR, Path, and Pre of $ProxyLLM_{heavy}$, $ProxyLLM_{mid}$, and proxyLLM gradually improve as the number of proxy models, denoted as k, increases. However, when k exceeds 5, employing additional proxy models leads to a decline in performance. This phenomenon occurs because multiple proxy models are tasked with handling different types of tasks. When k is within a certain range, they can quickly and effectively divide the task. Nevertheless, as k becomes larger, the LLM faces greater challenges in selecting the appropriate proxy model, and different proxy models often have overlapping capabilities, which further complicates the LLM's selection process. Additionally, as k exceeds 5, the IS value also rises, indicating that the LLM intervenes more frequently. This suggests that introducing more proxy models introduces a risk of uncertainty in selection. Furthermore, when more proxy models are used, the parameter counts of $ProxyLLM_{heavy}$ and $ProxyLLM_{mid}$ typically increase accordingly. In contrast, proxyLLM, which utilizes smaller models as proxy models, experiences minimal changes in parameter count. Therefore, when selecting the number of proxy models, it is crucial to balance the diversity of proxy models and the LLM's selection capability to avoid the uncertainties and performance degradation that may arise from introducing too many proxy models.

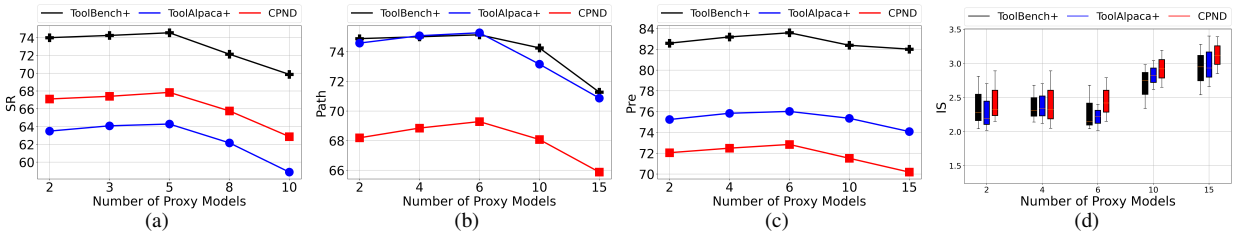

Figure A5 The impact of varying numbers of proxy models across different datasets. (a) SR changes as a function of proxy model numbers. (b) Path changes as a function of proxy model numbers. (c) Pre changes as a function of proxy model numbers. (d) IS changes as a function of proxy model numbers.

Appendix A.4.3 *Effectiveness of the LLM-Guided DRL*

We attribute the effectiveness of the LLM-Guided DRL to three key components: the self-attention module (SAM), the JS divergence constraint, and the Kendall's Tau coefficient constraint. To validate the impact of each component on the final performance, we construct baseline models by removing the corresponding parts, including ProxyLLM$_{w/oSAM}$, ProxyLLM$_{w/oJS}$, and ProxyLLM$_{w/oKT}$. As shown in the Table A6, each component contributes to an improvement in the final outcome. For example, the use of the SAM, JS divergence, and KT individually enhances the average reward by 5.06%, 7.79%, and 2.47%, respectively. Further analysis leads to the following conclusions: The self-attention module is employed for input embedding, enabling a more accurate reconstruction of the LLM policy. The JS divergence ensures that the actor's output probability distribution of actions is consistent with that suggested by the LLM, thereby guaranteeing the learning effectiveness and achieving a higher reward. The Kendall's Tau coefficient ensures that the order of preference for actions is consistent between the critic and the LLM, enhancing the learning speed and enabling rapid convergence by introducing additional knowledge.

Table A6 Performance comparison of various baselines across different datasets. We use the ChatGLM4-9B as the LLM in our experiments and report the average results of three tests

Method	Avg. Reward	Cvg. Epoch	ToolBench+			ToolAlpaca+			CPND		
			SR	Path	Pre	SR	Path	Pre	SR	Path	Pre
ProxyLLM$_{w/oSAM}$	0.79	∼450	71.35	70.02	78.93	60.01	69.54	71.16	61.54	64.87	68.35
ProxyLLM$_{w/oJS}$	0.77	∼450	73.87	73.02	80.83	61.35	71.12	72.09	63.78	66.26	67.53
ProxyLLM$_{w/oKT}$	0.81	∼470	74.18	74.23	83.17	63.59	74.31	75.23	64.29	65.24	70.56
ProxyLLM	0.83	∼430	74.56	75.11	83.60	64.27	75.14	76.01	67.83	69.25	72.85

Appendix A.4.4 *Impact of Proxy Model Construction Strategies*

We further test the impact of different proxy model construction strategies on the final performance. For the CPND dataset, which is grounded in an actual Computing Power Network (CPN) system, we consider a complete service process comprising five stages: intent parsing (S1), node selection (S2), service deployment (S3), operational monitoring (S4), and service optimization (S5). During the construction of proxy models, we experiment with various combinations of these five stages to form three proxy models. For instance, we test combinations like $\{(S1, S2), (S3), (S4, S5)\}$ and $\{(S1), (S2, S3, S4), (S5))\}$. Each proxy model managed all the tools under its corresponding stages. We evaluate all possible combinations and measure their performance. We present the top 5 and bottom 5 combinations along with their corresponding SR, Path, and Pre values. Each experiment is conducted three times, and the average results are reported. As shown in Table A7 and Table A8, the combination of $\{(S1, S2), (S3), (S4, S5)\}$ achieves the best performance, while combinations like $\{(S1, S3), (S2, S5), (S4)\}$ and $\{(S1), (S2, S5), (S3, S4)\}$ result in poorer performance. Upon analyzing the CPN system's business process, we observe that S1 and S2 have strong business relevance, S3 is relatively independent, and S4 and S5 are closely related. Therefore, combining S1 with S2 and S4 with S5 generally yields better results, while combining weakly related stages like S2 with S5 or S3 with S4 tends to lead to poorer outcomes. Consequently, when constructing proxy models, it is crucial to follow certain guidelines, such as business relevance and type similarity. We plan to explore these aspects further in our future research.

Proxy Models	RS	Path	Pre
{(S1,S2),(S3),(S4, S5)}	68.12	71.34	73.22
{(S1,S2,S3), (S4),(S5)}	68.05	70.87	72.93
{(S1),(S2),(S3,S4,S5)}	68.02	70.61	72.86
{(S1),(S2,S3,S4),(S5)}	67.82	69.52	72.16
{(S1,S2),(S3,S4),(S5)}	67.74	69.33	71.92

Table A7 The 5 combinations with the best performance, and their associated SR, Path, and Pre values.

Proxy Models	RS	Path	Pre
{(S1,S3),(S2,S4),(S5)}	67.12	68.31	69.12
{(S1,S5), (S2,S3),(S4)}	67.03	68.27	69.03
{(S1,S4),(S2),(S3,S5)}	66.64	67.61	68.86
{(S1),(S2,S5),(S3,S4)}	66.62	67.55	68.26
{(S1,S3),(S2,S5),(S4)}	66.47	67.43	68.25

Table A8 The 5 combinations with the lowest performance, and their associated SR, Path, and Pre values.

References

1 Yan J, Li P, Gao R, Li Y, Chen L. Identifying critical states of complex diseases by single-sample Jensen-Shannon divergence. Frontiers in Oncology. 2021, 11:684781.
2 Okoye K, Hosseini S. Correlation tests in R: pearson cor, kendall's tau, and spearman's rho. In: R Programming: Statistical Data Analysis in Research. 2024, 12: 247-277.
3 Tang Q, Deng Z, Lin H, Han X, Liang Q, Cao B, Sun L. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. arXiv preprint arXiv:2306.05301. 2023.
4 Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction. Neural Information Processing Systems: NeurIPS, 2023.
5 Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In Proceedings of the AAAI Conference on Artificial Intelligence: AAAI, 2024.
6 Wu S, Zhao S, Huang Q, Huang K, Yasunaga M, Cao K, Ioannidis V, Subbian K, Leskovec J, Zou JY. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. Advances in Neural Information Processing Systems. 2024, 37: 25981-6010.
7 Shinn N, Cassano F, Gopinath A, Narasimhan K, Yao S. Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 2023, 36, pp.8634-8652.
8 Madaan A, Tandon N, Gupta P, Hallinan S, Gao L, et. al. Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Systems, 2023, 36, pp.46534-46594.
9 Shi Z, Gao S, Chen X, Feng Y, Yan L, Shi H, Yin D, Ren P, Verberne S, Ren Z. Learning to use tools via cooperative and interactive agents. arXiv preprint arXiv:2403.03031. 2024.
10 Shen W, Li C, Chen H, Yan M, Quan X, Chen H, Zhang J, Huang F. Small LLMs Are Weak Tool Learners: A Multi-LLM Agent. arXiv preprint arXiv:2401.07324. 2024.
11 Roumeliotis K I, Tselikas N D. ChatGPT and Open-AI Models: A Preliminary Review. Future Internet. 2023, 15: 192.
12 GLM T, Zeng A, Xu B, Wang B, Zhang C, Yin D, Zhang D, Rojas D, Feng G, Zhao H, Lai H. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4, All Tools. arXiv preprint arXiv:2406.12793. 2024.
13 Baktash J A, Dawodi M. GPT-4: A Review on Advancements and Opportunities in Natural Language Processing. arXiv preprint arXiv:2305.03195. 2023.
14 Li J, Chen Y, Zhao X, Huang J. An Improved DQN Path Planning Algorithm. The Journal of Supercomputing. 2022, 78: 616-39.
15 Pang H, Wang Z, Li G. Large Language Model guided Deep Reinforcement Learning for Decision Making in Autonomous Driving. arXiv preprint arXiv:2412.18511. 2024.