

A. Details of CAORL

Prompts of context augmentation. The complete prompt of context augmentation scheme is illustrated belows.

```
# Instructions

Your role is to serve as a policy model for tool
planning. Given a task and a set of tools, you need to
select some tools and determine the execution plan of
tools that can be executed in sequential or parallel to
solve the task. Your goal is to generate the tool
plans that can optimize the task performance while
minimizing the execution costs.

# Tool Information

Each tool has its own functionality. Executing a tool
will incur some execution costs. Besides, the costs of
each tool may vary based on the size of inputs. The
following are the embedding features of each tool:

[Tool 1 Embeddings], ..., [Tool N Embeddings]

# Tool Cost Information

The cost features of each tool are as follows:

[Tool 1 Cost Embeddings], ..., [Tool N Cost Embeddings]

# Task Specifications and Input Attributes

Next, you will receive information about the task and
the attributes of the current inputs.
Task specifications: Given an low-resolution blurry
image, how to return a regular image?
Task input Attributes: {'has_image': true, 'image_size':
(520, 780), 'has_text': false, 'text_length': None}.
Now please generate a tool plan.
```

Learning algorithms. The details of the decision transformer [8] based learning algorithm in CAORL are described as follows. Formally, the LLM takes the historical return, state, and action sequences to predict the next action:

$$LLM(a_i | \mathcal{I}; \{R_{i-w}, s_{i-w}, a_{i-w}, \dots, R_i, s_i\}), \quad (\text{A.1})$$

where R_i, s_i, a_i represent the return, state and action at timestep i , respectively. \mathcal{I} denotes the input context described in Figure 4 and §3.2.1, while parameter w defines the context window to facilitate the LLM to learn the action distribution. The rationale behind this algorithm is to train the LLM to learn the distribution of actions conditioned on specific states and returns, enabling the LLM to generate actions to achieve the desired return after training [8, 53]. In particular, during the inference stage, we specify a target return indicating the desired performance to trigger the LLM to generate actions. In practice, we set the target return as the maximum return observed in the training plan dataset.

Note that instead of directly feeding states, actions, and returns into the LLM, we design three separate linear layers to project them into embedding features, followed by layer normalization [4]. Additionally, each embedding vector is added with a learned positional embedding based on its corresponding timestep.

B. Details of OpenCATP

B.1. Tool Set

Following OpenAGI [15], the tools OpenCATP for task solving consist of 10 open-source domain expert models of different functions. These models include: sentiment analysis [3], text summarization [27], machine translation [41], image classification [10], object detection [6], image colorization [62], image super-resolution [9], image denoising [59], image deblurring [59] and image captioning [11].

B.2. Evaluation Tasks

As mentioned in §4, OpenCATP includes 87 sequential tasks and 24 non-sequential tasks, each with 100 data samples. The sequential tasks in OpenCATP are constructed from OpenAGI [15], one of the most popular LLM sequential tool planning datasets. As for non-sequential tasks, we use the back-instruct method [46] for task construction. Specifically, we construct diverse non-sequential tasks by following steps: (1) create a tool graph based on the toolkit and dependencies between tools; (2) extract a sub-graph representing a non-sequential plan; (3) prompt the GPT-4 to generate the descriptions of a task that can be solved by this plan. Table B.1 provides several concrete examples of various types of tasks in OpenCATP.

B.3. Evaluation Metrics

Plan execution prices. OpenCATP utilizes execution prices to comprehensively reflect the overall costs of tool plans, inspired by the Function as a Service (FaaS) platforms. FaaS is a category of cloud computing that charges consumers in an event-driven manner, where consumers only need to pay for the time and resources used to run their functions, with no prices paid when the functions are idle [44]. This “pay-as-you-go” model can result in significant savings, especially for applications with intermittent usage or short-lived tasks, contributing to the market success of FaaS. According to recent statistics [23], the global market size of FaaS is estimated to be 17.70 billion USD in 2024 and will reach 44.71 billion USD by 2029.

To accurately charge consumers for running their functions, FaaS platforms like AWS Lambda [26] have established a mature pricing model based on execution time and resource consumption. The effectiveness of this model has been validated over years in the FaaS industry, which motivates us to adopt execution prices to represent the overall costs of tool plans. Hence, we draw inspiration from the AWS Lambda [26] and design a pricing model for running






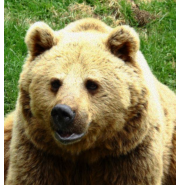




Task	Input Sample	Ground Truth	Example Plan
<i>Sequential Tasks</i>			
<i>Image-in-Image-out Task:</i> Given a noisy blurry grayscale image, how to return the regular image step by step?			
<i>Image-in-Text-out Task:</i> Given a noisy blurry grayscale image, how to return the caption in English step by step?		A woman stands in the dining area at the table.	
<i>Non-Sequential Tasks</i>			
<i>Image-in-Text-out Task:</i> Given a blurry image, how to (1) return the label of the image in English, and (2) return the caption of the image in German step by step?		(1) Braunbär; (2) Ein kräftiger Grizzly Bär ist im Hintergrund mit Gras zu sehen.	
<i>Image-in-Image-Text-out Task:</i> Given a blurry noisy image, how to (1) return the regular image, and (2) return the label of the image in German step by step?		(1)  ; (2) Schaf.	

Table B.1. Examples of the sequential and non-sequential tasks and their corresponding data samples in OpenCATP.

a tool as follows:

$$\begin{aligned}
C_{price}(t) = & price_per_run + time \\
& \times (cpu_{cons} \times price_cpu_{cons}(t) \\
& + cpu_{inst} \times price_cpu_{inst} \\
& + gpu_{cons} \times price_gpu_{cons}(t) \\
& + gpu_{inst} \times price_gpu_{inst}).
\end{aligned} \tag{B.1}$$

Here, $time$ denotes the execution time in milliseconds of tool t . cpu_{cons} (gpu_{cons}) denotes the constant CPU (GPU) memory consumption in MB required to load the tool. cpu_{inst} (gpu_{inst}) denotes the instant CPU (GPU) memory consumption in MB caused by the computation of the tool. $price_cpu_{cons}$, $price_cpu_{inst}$, $price_gpu_{cons}$, $price_gpu_{inst}$ calculate the prices in USD associated with the respective memory consumptions. $price_per_run$ denotes the prices in USD charged for each execution of the tool. Table B.2 lists the settings of the pricing parameters³.

³Note that the existing FaaS platforms do not provide pricing strategy

Based on the tool prices, the execution prices of a tool plan p are defined the sum of prices of each tool in the plan:

$$C_{price}(p) = \sum_t C_{price}(t) \tag{B.2}$$

Plan execution time. For a sequential plan, the execution time is simply the sum of the execution time of each tool in the plan. For example, for plan ① in Figure 1, the execution time is $0.18 + 3.46 + 0.13 = 3.77$ (s). For a non-sequential plan that can be executed in parallel, the execution time is defined as the maximum of the sum of tool execution time in each branch of the plan. For example, for plan ④ in Figure 1, the execution time is $\max(0.18 + 0.29 + 0.16, 0.18 + 0.09) = \max(0.63, 0.27) = 0.63$ (s).

Task performance of plans. In OpenCATP, we mainly use ViT scores [15] and BERT scores [63] to calculate the per-

for GPU resources. We then set the GPU prices as three times the CPU prices, according to the article in <https://news.rice.edu/news/2021/rice-intel-optimize-ai-training-commodity-hardware>

Notations	Values (\$)
$price_per_run$	$2e-7$
$price_cpu_cons$	$2.1e-9$ if memory $\leq 128MB$
	$8.3e-9$ if memory $\leq 512MB$
	$1.67e-8$ if memory $\leq 1024MB$
	$2.5e-8$ if memory $\leq 1536MB$
	$3.33e-8$ if memory $\leq 2048MB$
	$5e-8$ if memory $\leq 3072MB$
	$6.67e-8$ if memory $\leq 4096MB$
	$8.83e-8$ if memory $\leq 5120MB$
	$1e-7$ if memory $\leq 6144MB$
	$1.167e-7$ if memory $\leq 7168MB$
	$1.333e-7$ if memory $\leq 8192MB$
$price_gpu_cons$	$1.5e-7$ if memory $\leq 9216MB$
	$1.667e-7$ if memory $\leq 10240MB$
$price_cpu_inst$	$3.02e-14$
$price_gpu_inst$	$6.3e-9$ if memory $\leq 128MB$
	$2.49e-8$ if memory $\leq 512MB$
	$5.01e-8$ if memory $\leq 1024MB$
	$7.5e-8$ if memory $\leq 1536MB$
	$9.99e-8$ if memory $\leq 2048MB$
	$1.5e-7$ if memory $\leq 3072MB$
	$2.001e-7$ if memory $\leq 4096MB$
	$2.499e-7$ if memory $\leq 5120MB$
	$3e-7$ if memory $\leq 6144MB$
	$3.501e-7$ if memory $\leq 7168MB$
	$3.999e-7$ if memory $\leq 8192MB$
$price_gpu_inst$	$4.5e-7$ if memory $\leq 9216MB$
	$5.001e-7$ if memory $\leq 10240MB$

Table B.2. Settings of the pricing parameters in Eq.(B.1).

formance scores of an executed tool plan on the target task. Specifically, ViT scores are used for tasks that involve image outputs, which measures the cosine similarity between the image generated by the plan and the ground-truth image. BERT scores are used for tasks that involve text outputs, which measures the cosine similarity between the generated texts and ground-truth texts.

Note that for non-sequential tasks requiring multiple outputs, we calculate the performance scores for each pair of plan outputs and ground-truth outputs, then average these scores to derive the final plan performance scores on the tasks. In particular, if a model produces a sequential plan for a non-sequential task (e.g., Figure 1①), we assign a 0 score for the missing output pair. For instance, the plan ① in Figure 1 will receive a 0 score for the caption output, as it does not generate any captions.

C. Details of Experiments

C.1. Training and Testing Sets

For training sets, we select 2,760 and 480 samples from OpenCATP for sequential and non-sequential planning, respectively. As for testing sets, we select 720 and 480 samples from various tasks which are not present in the training sets for the evaluation of sequential and non-sequential planning, respectively.

C.2. Implementation of CATP-LLM

The details of implementing CATP-LLM on OpenCATP platform in the experiments are described as follows.

Details of TPL. For TPL, we assign each tool in OpenCATP with a unique tool token and a dependency token. The dependency token of a tool means accepting the outputs provided by this tool as inputs, as depicted in Figure 3. We also introduce a special dependency token $\langle task \rangle$ indicating accepting the task data as inputs. Note that in our experiments, we focus on resource dependencies, *i.e.*, input-output dependencies. However, the concepts of TPL can be easily extended to other types of dependencies, such as order dependencies where there exist strict execution orders between tools.

Details of CAORL. As for CAORL, we use the execution prices and task performance defined in §B.3 to calculate the execution costs and plan performance in the reward function. Besides, to implement the context augmentation scheme, we need to derive the cost attributes of each tool. To achieve this, we categorize the input data in OpenCATP into $k = 4$ size levels using k-means clustering, with the optimal number of clusters determined by the elbow method. We then profile the execution time and CPU/GPU memory consumption of each tool across varying input sizes, following the method outlined in §3.2.1. Based on these statistics, we calculate the overall costs for each tool using the execution prices defined in Eq.(B.1), which ultimately yield the tool cost attributes.

Details of fine-tuning. We apply the data generation method described in §3.2 to create 1,200 sequential tool plans and 780 non-sequential tool plans as training data. We then fine-tune CATP-LLM using Llama2-7B as the default backbone with LoRA rank 64. The total number of trainable parameters, including LoRA weights and learning tokens in TPL, are less than 1M, accounting for no more than 0.2% of the total parameters. Besides, it only takes CATP-LLM 9.5h and 6.2h to converge for 2 epochs in the sequential and non-sequential scenarios, when fine-tuned over a single 32GB V100 GPU. Hence, the fine-tuning overhead of CATP-LLM is small.

C.3. Implementation of Baselines

Zero-shot learning. For zero-shot learning, we design two types of prompts for sequential planning and non-sequential planning, respectively. For sequential planning, we prompt the LLM to produce a tool sequence that can be executed sequentially to solve the target tasks. As for non-sequential planning, we instruct the LLM to generate a tool plan following the format similar to our TPL. This is because we find the LLM achieves poor performance in non-sequential planning without an appropriate format to generate non-sequential plans. Note that we also incorporate the average tool cost information into the prompts to instruct the LLM

to create low-cost tool plans. The average tool cost information is derived from offline profiling results described in §C.2. The prompts for zero-shot learning are shown below.

```
[Prompt for Sequential Planning]

# Instructions

You need to act as a policy model that, given a task
and a set of tools, determines the sequence of tools
that can be executed sequentially to solve the task.
Your goal is to optimize the task performance while
minimizing the execution costs.

# Tool Information

The information of each tool is provided as follows:

Object Detection: This tool identifies the names of
objects in an image. It is generally used for object
identification in the images. The input and output
types of this tool are image and text, respectively. It
can accept inputs from tools ''Image Super Resolution
'', ''Colorization'', ''Image Deblurring'', or ''Image
Denoising''.

Costs of Object Detection: On average, this tool takes
about 175.73 milliseconds to run. It consumes
approximately 352.19 MB of CPU memory and 449.37 MB of
GPU memory.

Image Deblurring: This tool can enhance the clarity of
blurry images. It can be used for tasks that require
improving image quality. Both the input and output
types are images. It can accept inputs from tools ''
Image Super Resolution'', ''Colorization'', or ''Image
Denoising''.

Costs of Image Deblurring: On average, this tool takes
about 667.42 milliseconds to run. It consumes
approximately 444.91 MB of CPU memory and 3498.11 MB of
GPU memory.

[...]

# Response Format

Provide a response in the similar format according to
the following example: ''Tool1, Tool2, Tool3''.
```

```
[Prompt for Non-Sequential Planning]

# Instructions

You need to act as a policy model that, given a task
and a set of tools, determines the execution plan of
tools that can be executed in sequential or parallel to
solve the task. Your goal is to generate the tool
plans that can optimize the task performance while
minimizing the execution costs.

# Tool Information

[Same as Sequential Planning]

# Response Format

Provide a response in the similar format according to
the following example: [''Tool1'', [''Task''], ''Tool2'', [''
Tool1''], ''Tool3'', [''Tool1'']]. The meaning of this
format is that the input data of Tool2 comes from the
outputs of Tool1. Besides, [''Task''] means that Tool1
depends on the input data provided by the task. Please
generate plans strictly according to this format.
```

Few-shot learning. The few-shot learning share similar prompts with zero-shot learning, except that we handcraft

several high-quality demonstrations as in-context examples to augment the LLM for plan generation. The prompts for few-shot learning are shown below.

```
[Prompt for Sequential Planning]

# Instructions

[Same as Zero-Shot]

# Tool Information

[Same as Zero-Shot]

# In-Context Example

Task: Given a low-resolution, noisy, blurry gray image,
how to return the regular image step by step?

Plan: Image Super Resolution, Image Denoising, Image
Deblurring, Colorization

[...]

# Response Format

[Same as Zero-Shot]
```

```
[Prompt for Non-Sequential Planning]

# Instructions

[Same as Zero-Shot]

# Tool Information

[Same as Zero-Shot]

# In-Context Example

Task: Given a low-resolution grayscale image, how to
(1) return the regular image, and (2) return the class
label of the image in English?

Plan: ['Colorization', ['Task'], 'Image Super
Resolution', ['Colorization'], 'Image Classification',
['Colorization']]

[...]

# Response Format

[Same as Zero-Shot]
```

HuggingGPT. We reuse the prompt of HuggingGPT [45] for planning in our experiments. Besides, we have made some small modifications on the prompt by providing the tool cost information of each tool.

HYDRA. We reuse the pipeline and the prompt of HYDRA[24] in our experiments. To ensure fairness, we restrict the tool set of HYDRA to those available in OpenCATP. What’s more, similar to HuggingGPT, we have also added the tool cost information in the prompt of HYDRA.

Instruction fine-tuning (IFT) and RLTF. We adapt our TPL and apply it on IFT and RLTF, as we find that they perform poorly especially in non-sequential planning if generating tool plans in natural language.

Note that the engines of GPT-3.5 and GPT-4 used for all prompt-based methods are gpt-3.5-turbo-0125 and gpt-

Method	Tool Planner	Mean Accuracy (%)	Average Number of Tools
CoT GPT-3.5	/	78.31	/
CoT GPT-4	/	83.99	/
Chameleon [34]	GPT-4	86.54	3.40
Published Results (Above) ▲			
CATP-LLM	Llama2-7B	85.86	2.41

Table D.1. Comparing CATP-LLM with other methods on ScienceQA. CoT GPT-3.5/4 answer questions without using tools.

Devices	Methods	Task Scores ↑	Exec. Price (\$) ↓	Runtime (s) ↓	QoP ↑
NVIDIA RTX 3090	<i>Sequential Planning</i>				
	HuggingGPT (GPT-4)	0.665	0.097	1.108	0.246
	Hydra (GPT-4)	0.603	0.061	0.654	0.247
	CATP-LLM (Llama2-7B)	0.652	0.072	0.748	0.262
	<i>Non-Sequential Planning</i>				
	HuggingGPT (GPT-4)	0.419	0.159	2.099	0.069
NVIDIA RTX 4090	Hydra (GPT-4)	0.332	0.068	1.180	0.106
	CATP-LLM (Llama2-7B)	0.566	0.138	0.761	0.161
	<i>Sequential Planning</i>				
	HuggingGPT (GPT-4)	0.667	0.087	1.228	0.257
	Hydra (GPT-4)	0.603	0.039	0.432	0.267
	CATP-LLM (Llama2-7B)	0.651	0.045	0.487	0.286
	<i>Non-Sequential Planning</i>				
	HuggingGPT (GPT-4)	0.419	0.103	1.413	0.118
	Hydra (GPT-4)	0.332	0.049	0.844	0.122
	CATP-LLM (Llama2-7B)	0.566	0.077	0.463	0.215

Table E.1. Evaluation on real-world commercial GPUs. Arrow ↑/↓ means higher/lower is better.

4-turbo, respectively.

D. Evaluate CATP-LLM on General Benchmark

To validate the effectiveness of CATP-LLM on other benchmark except OpenCATP, we implement CATP-LLM on ScienceQA [34], a widely adopted multimodal benchmark for multi-choice question answering. We reused the tools in ScienceQA developed by Chameleon [35], a few-shot learning-based tool planning method. As ScienceQA lacks implementation for cost measurement (*e.g.*, memory consumption), we use the number of tools in each plan as the indicator for tool planning costs. As shown in Table D.1, CATP-LLM outperforms CoT GPT-3.5 and GPT-4. Besides, CATP-LLM uses significantly fewer tools than Chameleon GPT-4 (a reduction of 29.11%) while its accuracy is only marginally lower than that of Chameleon GPT-4 (a decline of 0.68%). This highlights CATP-LLM’s advantage in cost-aware tool planning to substantially reduce tool costs without sacrificing performance. *This finding on ScienceQA is consistent with that on OpenCATP, which demonstrates the generalization ability of CATP-LLM.*

E. Evaluate CATP-LLM on Various Hardware Equipment

According to the cost definition in Equation (B.1), the cost measurement (*e.g.*, runtime and memory consumption) can be affected by hardware equipment. To validate the applicability of CATP-LLM across diverse hardware configurations, we test the plans of CATP-LLM and top baselines on various real-world commercial GPUs. We report the performance of these methods in Table E.1. We can

see that CATP-LLM strikes the better balance between task scores and execution costs, leading to the highest QoP. *This demonstrates the applicability of CATP-LLM in various hardware configurations.*