



HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face

19기 NLP 심화세션

TOBIG'S 19기 권유진

Contents



19기 NLP 심화세션

TOBIG'S 19기 권유진

01 | Introduction

02 | HuggingGPT

03 | Experiments

04 | Limitations

05 | Code

01 | Introduction



19기 NLP 심화세션

TOBIG'S 19기 권유진

- AI의 발전으로 다양한 domain과 modality를 다룰 수 있음
 - 하지만 복잡한 task는 처리 불가능
 - ex) 예시 이미지의 소년과 같은 자세를 한 소녀가 책을 읽고 있는 이미지를 생성하고 목소리로 설명해줘.
- LLM(Large Language Model)이 NLP 분야에서 훌륭한 성과를 보임
 - 따라서 유저의 요청에 따라 LLM을 활용해 다양한 AI 모델을 연결해 task 수행
 - 거대한 text corpora로 사전 학습
 - Human Feedback을 통한 강화학습(RLHF)

01 | Introduction



19기 NLP 심화세션

TOBIG'S 19기 권유진

- 또한, 최근에는 다양한 연구 분야 등장
 - In-context learning, instruction learning, chain-of-thought prompting 등
 - In-context learning(context-aware learning): 정확도 향상을 위해 context나 주변 환경을 활용해 모델 학습
 - Instruction learning: 인간이 제공한 명확한 instruction을 학습(데이터에만 의존하는 기준 방법과 차이)
 - Chain-of-thought prompting: 언어 모델이 올바른 맥락의 답변을 생성하기 위한 기술
- 하지만 아직 LLM은 불완전
 - input, output의 형태가 한정
 - 복잡한 task는 다양한 sub-task들의 결합으로 이루어져 있는데, 이를 처리 불가
 - zero-shot, few-shot에 강하지만 fine-tuned model보다는 성능이 낮음
- 복잡한 AI task를 다루기 위해 여러 model들을 조정해야 함
 - LLM을 활용해 external models를 조정
 - 따라서 LLM과 external models 사이를 연결하는 middleware 선정 필요
- 즉, 해당 연구에서는 inter-model cooperation protocols를 제안

02 | HuggingGPT



19기 NLP 심화세션

TOBIG'S 19기 권유진

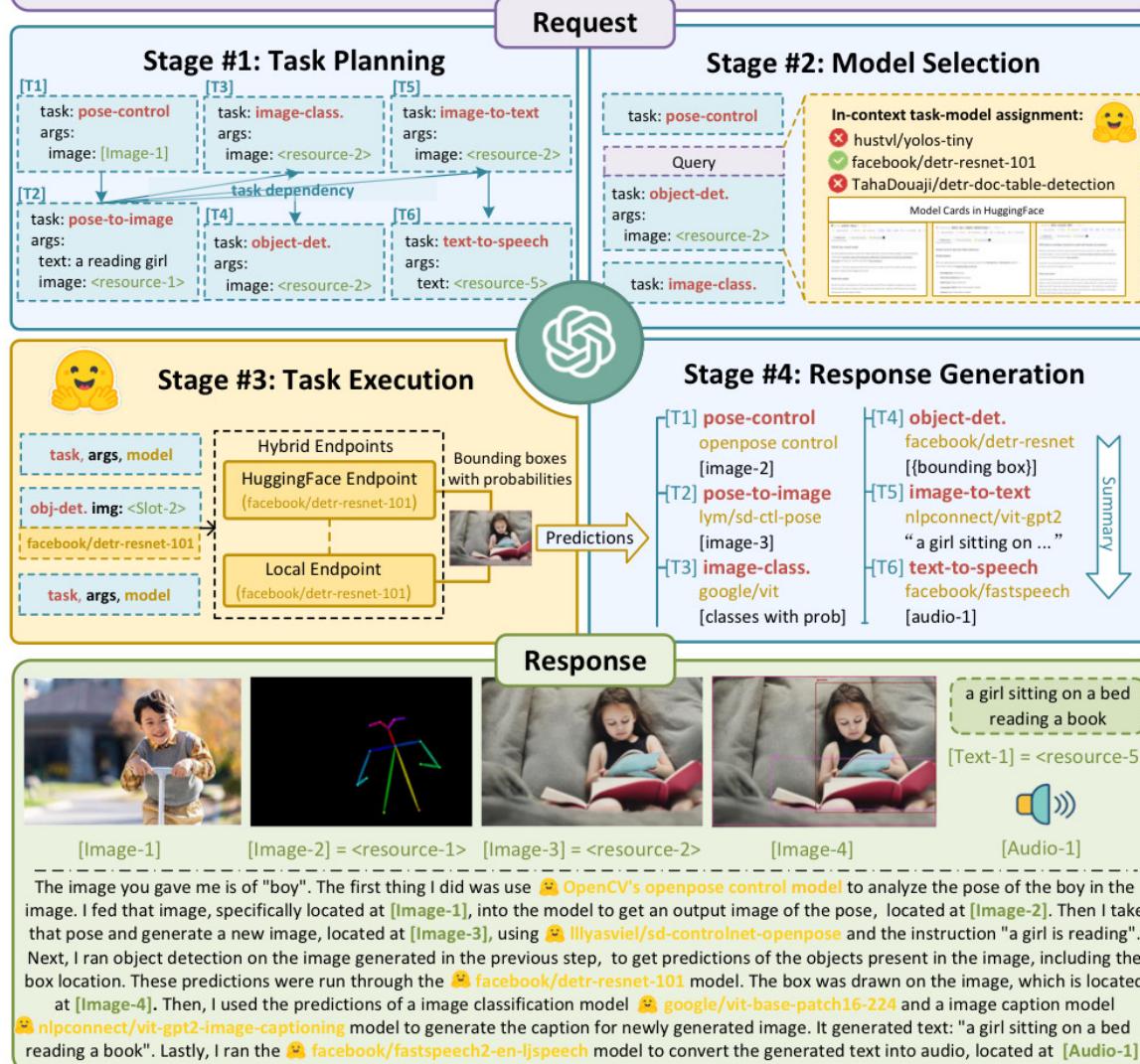
- 유저의 요청을 expert models로 연결해주는 interface
 - 다양한 modality, domain 활용 가능
- Model Descriptions를 기반으로 적절한 접근법 채택 가능
 - 별도의 모델 구조 변경 필요 없이 다양한 expert models 결합해 사용
- 즉, LLM을 controller, 다양한 expert models를 collaborative executor로 활용
- 과정
 - 유저의 요청을 parsing → multiple task로 분할 → task order 계획
 - parsing된 task를 expert models에 분배 (Model Selection)
 - expert models를 사용해 각 모델에 부여된 task 수행 → LLM에 로그 및 결과 전송
 - LLM이 로그와 결과를 요약해 유저에게 전달

02 | HuggingGPT



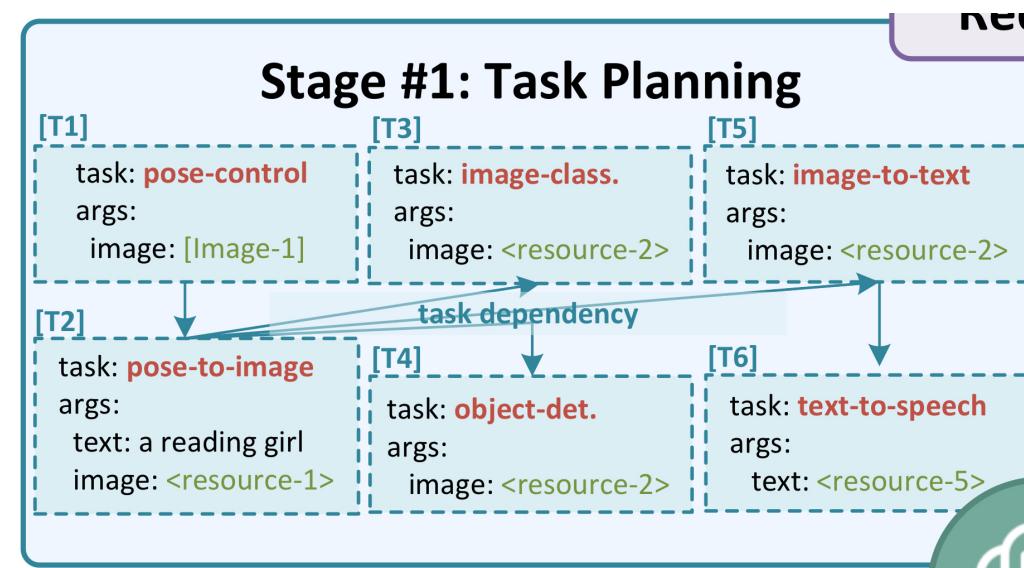
19기 NLP 심화세션
TOBIG'S 19기 권유진

please generate an image where a girl is reading a book, and her pose is the same as the boy in the image example.jpg. Then please describe the new image with your voice.



Task Planning

- 유저의 요청을 받아 구조화된 task sequence로 분할
 - 복잡한 요청은 여러 개의 task를 포함
 - Dependency와 수행 순서 결정
- specification-based instruction과 demonstration-based parsing을 활용



02 | HuggingGPT



19기 NLP 심화세션

TOBIG'S 19기 권유진

- specification-based instruction
 - task마다 uniform template을 부여
 - Slot Filling을 통해 LLM이 task parsing을 수행하도록 함
 - Task ID: task를 구분하기 위한 식별자
 - Task type: NLP, CV, Audio, Video 등 task 종류
 - Task dependencies: 해당 task를 수행하기 위해 사전에 수행되어야 하는 task
 - Task arguments: input의 형태(text, image, audio)
 - Instruction tuning과 human feedback을 통한 강화학습 수행
 - 따라서 LLM은 instruction을 따름
 - 결론적으로 HuggingGPT는 유저의 요청을 분석하고 적절한 task를 parsing하기 위해 task에 특화된 LLM 사용

Task	Args
Text-cls	text
Token-cls	text
Text2text-generation	text
Summarization	text
Translation	text
Question-answering	text
Conversational	text
Text-generation	text
Tabular-cls	text

Table 1: NLP tasks.

Task	Args
Image-to-text	image
Text-to-image	image
VQA	text + image
Segmentation	image
DQA	text + image
Image-cls	image
Image-to-image	image
Object-detection	image
Controlnet-sd	image

Table 2: CV tasks.

Task	Args
Text-to-speech	text
Audio-cls	audio
ASR	audio
Audio-to-audio	audio

Table 3: Audio tasks.

Task	Args
Text-to-video	text
Video-cls	video

Table 4: Video tasks.

- demonstration-based parsing
 - task parsing, planning을 더욱 잘하기 위해 in-context learning을 진행
 - LLM이 사용자의 의도와 조건을 잘 이해하도록 prompts에 demonstrations 제공
 - demonstration: task planning의 input(유저의 요청), output(예상되는 task sequence)
 - Demonstration에 task 사이의 dependency를 포함시켜 task 사이의 관계 및 수행 순서 학습
 - Chatbot에게 대화의 맥락 관리는 필수
 - 유저의 요청을 더욱 잘 이해시키기 위해 대화 로그를 활용
 - Instruction에 아래 paragraph 추가
 - *The chat logs are recorded as {{Chat Logs}}. From the chat logs, you can find the history resources for your task planning.*

02 | HuggingGPT



19기 NLP 심화세션

TOBIG'S 19기 권유진

Task Planning	Prompt
	<pre>#1 Task Planning Stage - The AI assistant can parse user input to several tasks: [{"task": task, "id": task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": URL, "video": URL}}]. The "dep" field denotes the id of the previous task which generates a new resource that the current task relies on. A special tag "<resource>-task_id" refers to the generated text image, audio and video in the dependency task with id as task_id. The task MUST be selected from the following options: {{ Available Task List }}. There is a logical relationship between tasks, please note their order. If the user input can't be parsed, you need to reply empty JSON. Here are several cases for your reference: {{ Demonstrations }}. The chat history is recorded as {{ Chat Logs }}. From the chat logs, you can find the path of the user-mentioned resources for your task planning.</pre>
	Demonstrations
	<p>Look at /exp1.jpg, Can you tell me how many objects in the picture?</p> <pre>[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/exp1.jpg"}}, {"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "/exp1.jpg"}}]</pre> <p>In /exp2.jpg, what's the animal and what's it doing?</p> <pre>[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/exp2.jpg"}}, {"task": "image-classification", "id": 1, "dep": [-1], "args": {"image": "/exp2.jpg"}}, {"task": "object-detection", "id": 2, "dep": [-1], "args": {"image": "/exp2.jpg"}}, {"task": "visual-question-answering", "id": 3, "dep": [-1], "args": {"text": "What's the animal doing?", "image": "/exp2.jpg"}}]</pre> <p>Given an image /exp3.jpg, first generate a hed image, then based on the hed image and a prompt: a girl is reading a book, you need to reply with a new image.</p> <pre>[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "examples/boy.jpg"}}, {"task": "openpose-control", "id": 1, "dep": [-1], "args": {"image": "examples/boy.jpg"}}, {"task": "openpose-text-to-image", "id": 2, "dep": [1], "args": {"text": "a girl is reading a book", "image": "<resource>-1"}}]</pre>



Model Selection

- task parsing 후, 각 task에 알맞는 모델 선택
- expert model의 description과 in-context task-model assignment mechanism을 통해 동적으로 task에 적절한 모델 선택
 - 더욱 유연하게 모델 선택
- model description
 - Hugging Face Hub에서 expert models를 활용
 - 개발자들이 남긴 자세한 model description도 함께 존재(model functionality, architecture, supported language, domain 등)

02 | HuggingGPT



19기 NLP 심화세션
TOBIG'S 19기 권유진

The screenshot shows the Hugging Face Models page. The left sidebar lists categories: Tasks (Libraries, Datasets, Languages, Licenses), Other, Multimodal (Feature Extraction, Text-to-Image, Image-to-Text, Text-to-Video, Visual Question Answering, Document Question Answering, Graph Machine Learning), Computer Vision (Depth Estimation, Image Classification, Object Detection, Image Segmentation, Image-to-Image, Unconditional Image Generation, Video Classification, Zero-Shot Image Classification), and Natural Language Processing (Text Classification, Token Classification, Table Question Answering, Question Answering, Zero-Shot Classification, Translation, Summarization, Conversational). The main area displays a grid of model cards. One card is selected: bert-base-uncased by @jonatasgrosman, updated Nov 17, 2022, size 62.4M, downloads 839. Other visible cards include jonatasgrosman/wav2vec2-large-xlsr-53-english, facebook/dino-vitb16, xlm-roberta-base, gpt2, microsoft/resnet-50, openai/clip-vit-large-patch14, facebook/convnext-large-224, and roberta-base.

The screenshot shows the Hugging Face Model card for the roberta-base model. The top navigation bar includes AutoTrain Compatible, arxiv:1907.11692, arxiv:1806.02847, License: mit, Model card, Files and versions, Community, Edit model card, Train, Deploy, Use in Transformers, and Examples. The Model card section for RoBERTa base model states: "Pretrained model on English language using a masked language modeling (MLM) objective. It was introduced in [this paper](#) and first released in [this repository](#). This model is case-sensitive: it makes a difference between english and English." A disclaimer notes that the model card was written by the Hugging Face team. The Model description section explains that RoBERTa is a transformers model pretrained on a large corpus of English data in a self-supervised fashion, using the Masked language modeling (MLM) objective. It describes how the model randomly masks 15% of words in input sentences and uses them to predict masked words. The text concludes that this allows the model to learn a bidirectional representation of the sentence. The Hosted inference API section shows examples like "Paris is the <mask> of France." The Compute section lists computation times for various inputs on Intel Xeon 3rd Gen Scalable CPU: capital (0.864), heart (0.056), Capital (0.028), center (0.015), city (0.004). The Datasets used to train roberta-base section lists wikipedia (Preview, 43.1k, 165) and bookcorpus.

02 | HuggingGPT



19기 NLP 심화세션

TOBIG'S 19기 권유진

- In-Context Task-Model Assignment

- single-choice problems로 접근
- prompt에 user query와 parsed task를 포함해 task에 가장 적절한 모델 선택
- 하지만 maximum context length 문제 존재
 - LLM model은 최대 token 수가 정해져 있으므로, 모든 관련 모델의 정보를 포함 불가
 - 이 문제를 해결하기 위해 task type에 알맞는 모델을 filtering 후 다운로드 순으로 순위를 매김
 - 그 중 top-K개를 선정
- 위 과정에서 선정된 모델들로 model selection 진행

uest

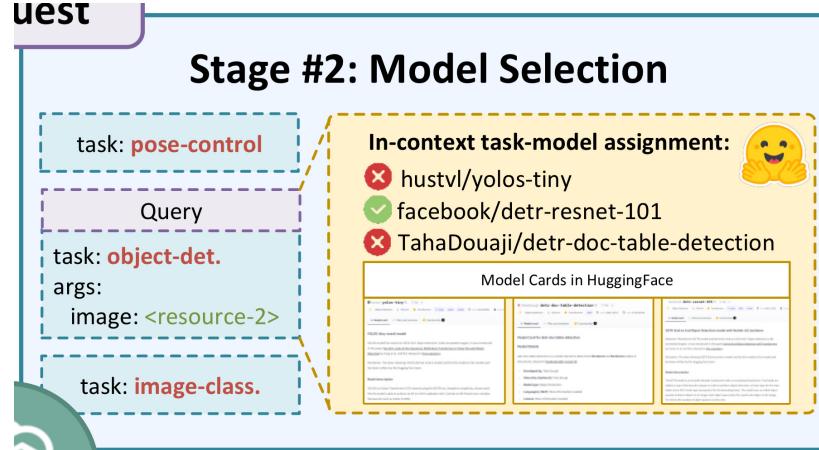
Stage #2: Model Selection

task: pose-control
Query
task: object-det.
args:
image: <resource-2>
task: image-class.

In-context task-model assignment:

- ✗ hustvl/yolos-tiny
- ✓ facebook/detr-resnet-101
- ✗ TahaDouaji/detr-doc-table-detection

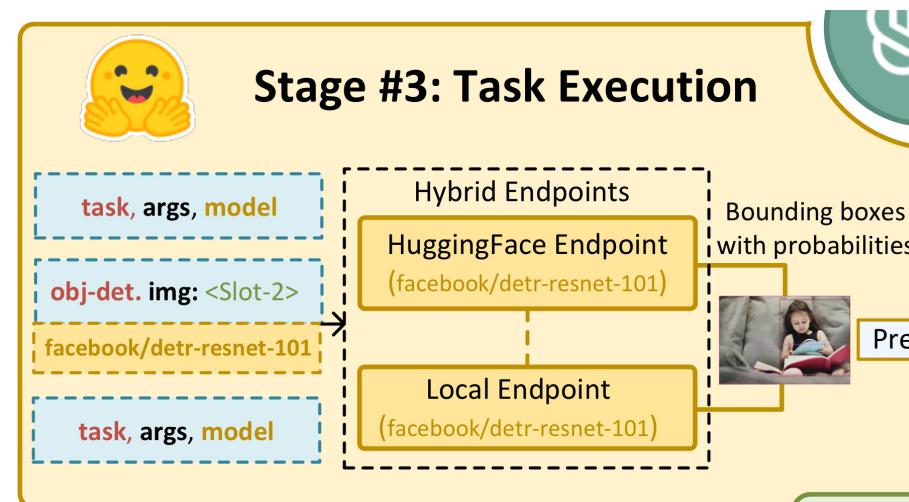
Model Cards in HuggingFace



	Prompt
	#2 Model Selection Stage - Given the user request and the call command, the AI assistant helps the user to select a suitable model from a list of models to process the user request. The AI assistant merely outputs the model id of the most appropriate model. The output must be in a strict JSON format: "id": "id", "reason": "your detail reason for the choice". We have a list of models for you to choose from {{ Candidate Models }}. Please select one model from the list.
Model Selection	<p>Candidate Models</p> <pre>{"model_id": model id #1, "metadata": meta-infos #1, "description": description of model #1} {"model_id": model id #2, "metadata": meta-infos #2, "description": description of model #2}</pre> <p>[Select top-K candidate models with the same task type as the current task]</p> <pre>{"model_id": model id #K, "metadata": meta-infos #K, "description": description of model #K}</pre>

Task Execution

- 각 task에 할당된 model들을 실행
 - task arguments를 input으로 받아 모델 계산 후, LLM에 로그와 결과 전송
- 계산 능력, 속도 향상을 위해 hybrid inference endpoints에서 모델 수행
 - inference endpoints: 모델이 수행되는 환경
- dependency가 없는 모델은 병렬 수행



02 | HuggingGPT



19기 NLP 심화세션

TOBIG'S 19기 권유진

- Hybrid Endponits

- Hugging Face를 유일한 inference endpoints로 사용하는 것이 이상적인 시나리오
- 하지만 몇몇의 상황에서는 local inference endpoints 사용이 필요
 - inference endpoints가 존재하지 않는 상황, network 접근이 제한된 상황 등
- 시스템을 안정적, 효율적으로 사용하기 위해 local에서 우선적으로 모델 구동
 - local은 빠르지만 지원하는 모델 수가 적음
- 따라서 local을 우선적으로 endpoints로 지정하고, 만약 local에서 수행이 불가능한 경우 Hugging Face endpoint에서 구동

02 | HuggingGPT



19기 NLP 심화세션

TOBIG'S 19기 권유진

- Resource Dependency

- 효과적으로 resource dependency를 활용해야 함
 - 하지만 미래에 할당할 resource는 task-planning 단계에서 알 수 없으므로 어려움
- 따라서 task planning stage에서 unique symbol <resource> 활용
 - Prerequisite task에서 생성된 resource를 <resource>-task_id로 구분
 - task_id: prerequisite task의 id
 - 해당 task를 참조하는 task의 task arguments를 일단 prerequisite task의 symbol을 사용
 - Task execution stage에서 동적으로 해당 symbol을 prerequisite task에서 생성한 resource로 대체

02 | HuggingGPT

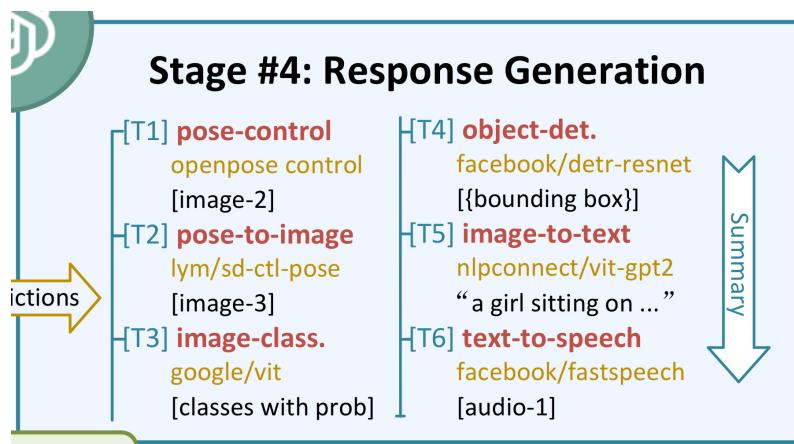


19기 NLP 심화세션

TOBIG'S 19기 권유진

Response Generation

- 이전 3단계 stage의 결과를 결합
 - List of planned tasks, Selected models, Inference results
- Inference results는 고정된 구조 보유
 - ex) Object Detection – bbox, 확률, Q&A: answer distribution
- LLM에 structured inference results를 입력해 human language 형태의 답변 생성



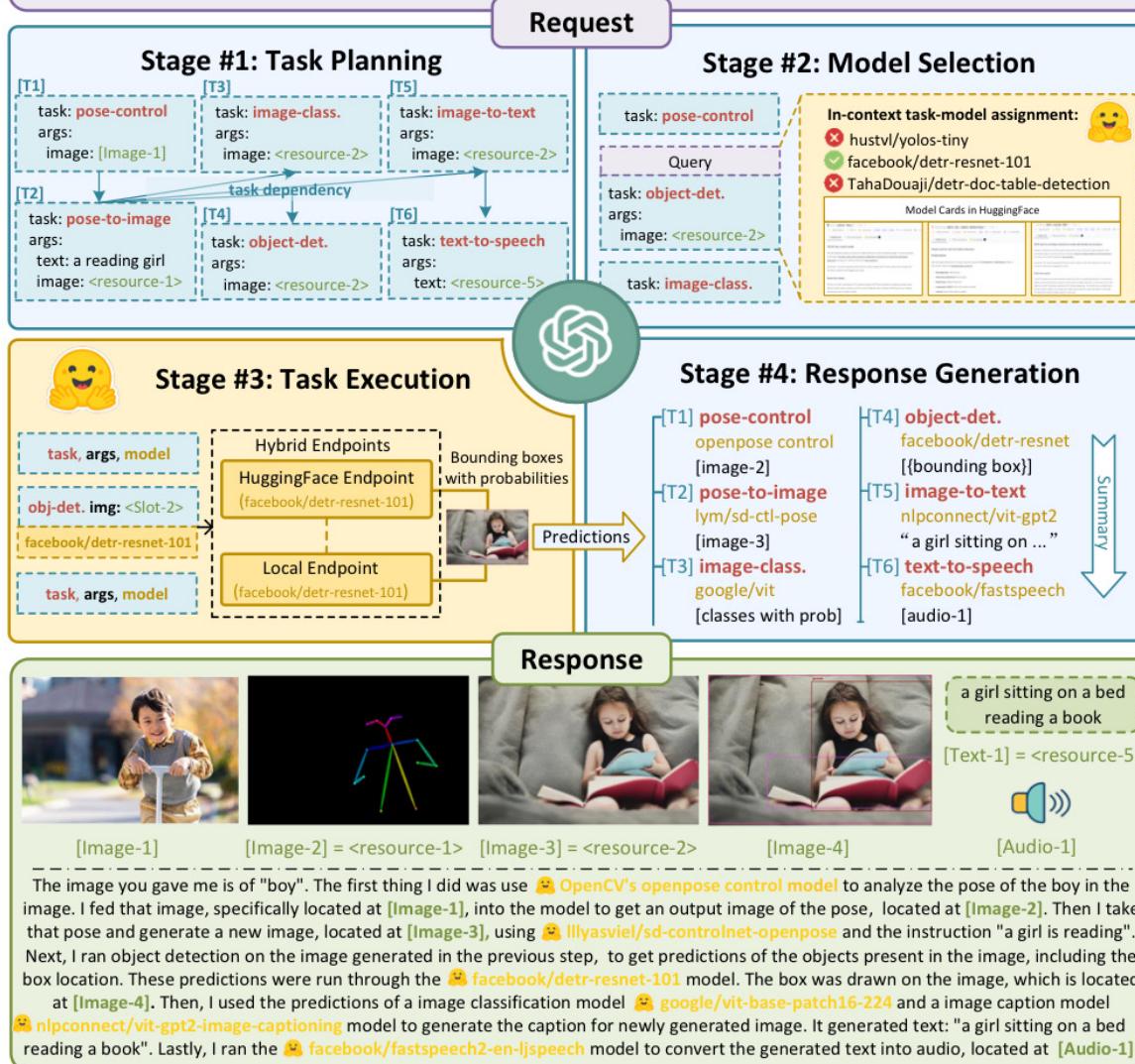
Response Generation	Prompt
	#4 Response Generation Stage - With the input and the inference results, the AI assistant needs to describe the process and results. The previous stages can be formed as - User Input: {{ User Input }}, Task Planning: {{ Tasks }}, Model Selection: {{ Model Assignment }}, Task Execution: {{ Predictions }}. You must first answer the user's request in a straightforward manner. Then describe the task process and show your analysis and model inference results to the user in the first person. If inference results contain a file path, must tell the user the complete file path. If there is nothing in the results, please tell me you can't make it.

02 | HuggingGPT



19기 NLP 심화세션
TOBIG'S 19기 권유진

please generate an image where a girl is reading a book, and her pose is the same as the boy in the image example.jpg. Then please describe the new image with your voice.



03 | Experiments



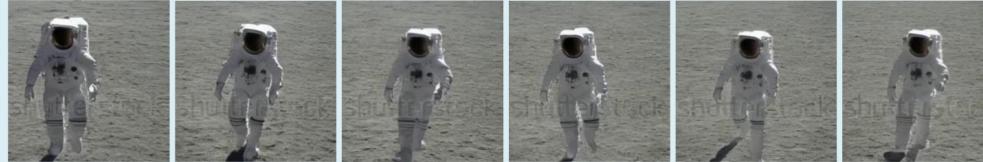
19기 NLP 심화세션

TOBIG'S 19기 권유진

- LLM으로 gpt-3.5-turbo, text-davinci-003 사용
- LLM output이 더욱 안정적으로 되도록 decoding temperature을 0으로 설정
 - decoding temperature: softmax의 확률 분포를 조정 (값이 클수록 분산이 커짐 = 다양한 단어가 출력)
- LLM output이 format에 맞게 나오도록 logit_bias를 0.2로 설정
 - Logit bias가 클수록, 특정 class, token이 자주 등장

Query: generate a video with the title "An astronaut is walking in space" and dub it.

Response: Sure. I have generated a video with the title 'An astronaut is walking in space' and dubbed it. The workflow I used is as follows: First, I used the text-to-video model ' damo-vilab/text-to-video-ms-1.7b' to generate a video from the text. The inference result is the generated video ' /videos/5696.mp4'. Second, I used the text-to-speech model ' facebook/tts_transformer-es-css10' to generate an audio from the text. The inference result is the generated audio ' /audios/3f9e.flac'. Finally, I combined the video and audio to generate the dubbed video. I hope this answers your question. Is there anything else I can help you with?



Task Planning:

```
1: {"task": "text-to-video", "id": 0, "dep": [-1], "args": {"text": "An astronaut is walking in space" }}
```

```
2: {"task": "text-to-speech", "id": 1, "dep": [-1], "args": {"text": "An astronaut is walking in space" }}
```

Execution Results:

```
1. {"generated video": "/videos/5696.mp4"}      2. {"generated audio": "/audios/3f9e.flac"}
```

03 | Experiments



19기 NLP 심화세션

TOBIG'S 19기 권유진

Query: based on the pose image of example1.jpg and the description of example2.jpg, please generate a new one.



Response: Based on the pose image of [example1.jpg](#) and the description of [example2.jpg](#), I have generated a new image [s5f8.jpg](#) for you. The process of generating this image involved using the [openpose-control](#) model to extract the pose information from the basketball image, and then using the [nlpconnect/vit-gpt2-image-captioning](#) to generate a description of the boy in the image. Finally, I used the [Illyasviel/sd-controlnet-openpose](#) model to combine the pose information and the description to generate the final image. The complete path of the generated image is [s5f8.jpg](#). If you have any other questions, please let me know.

Task Planning:

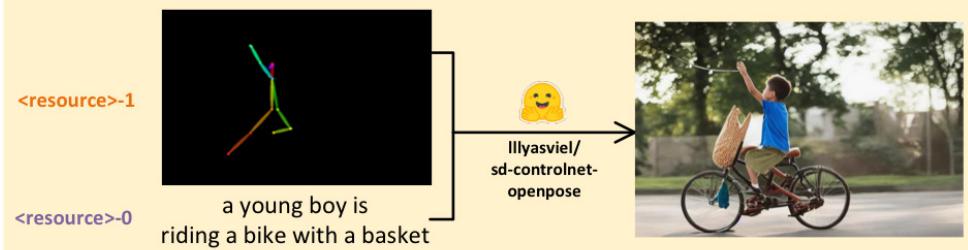
```
1: {"task": "openpose-control", "id": 0, "dep": [-1], "args": {"image": "example1.jpg"}}

2. {"task": "image-to-text", "id": 1, "dep": [-1], "args": {"image": "example2.jpg"}}

3. {"task": "openpose-text-to-image", "id": 2, "dep": [0, 1], "args": {"text": "<resource>-0", "image": "<resource>-1"}}
```



Execution Results:



Query: Describe this image in as much detail as possible. [example.jpg](#)

Response: The image you provided appears to show a family of four dogs playing in the grass. For this task, I ran multiple CV tasks, including image-to-text, image-classification, object-detection, visual-question-answering, and document-question-answering. I chose specific models for each task that were best suited to addressing your query, based on their training and performance characteristics.

For image-to-text, I chose the [nlpconnect/vit-gpt2-image-captioning](#) model. This model generates an image caption with the given image. I applied this model to the input image and the developed output was: a family of four dogs are playing in the grass. For image-classification, I selected the [google/vit-base-patch16-224](#) model. This model is trained on natural images dataset and it can predict the label of the image output. I applied this model to the image and get the results showing the risk of each label. It shows the highest risk at "Rhodesian ridgeback" with a score of 93.8%.

For object-detection, I used [facebook/detr-resnet-101](#) model. This model is trained on COCO panoptic segmentation for object detection. I used this model to generate an image file with the predicted box. The generated image with predicted box is at: [/images/sd48.jpg](#)

Regarding your request for image segmentation, I have completed the task using a pre-trained model called [facebook/detr-resnet-50-panoptic](#). The segmentation mask shows the different regions of the image that belong to different classes. In this case, the model has identified two main classes in the image, which are "dog", "grass" and "tree". The generated image with the segmentation mask can be found at: [/images/t4s1.jpg](#)

For visual question answering, I used [dandelin/vilt-b32-finetuned-vqa](#) model. This model is a Vision-and-Language Transformer (ViLT) model, which is fine-tuned on VQAv2. It is suitable for visual-question-answering tasks, and is likely to provide accurate results for the given image. For the question "what are in the image?" It reply with a prediction "dogs" and the confidence is 84.9%.



Task Planning:

- {"id": "0", "dep": [-1], "task": "image-to-text", "args": {"image": "examples.jpg"}},
- {"id": "1", "dep": [-1], "task": "image-classification", "args": {"image": "examples.jpg"}},
- {"id": "2", "dep": [-1], "task": "object-detection", "args": {"image": "examples.jpg"}},
- {"id": "3", "dep": [-1], "task": "image-segmentation", "args": {"image": "examples.jpg"}},
- {"id": "4", "dep": [-1], "task": "visual-question-answering", "args": {"text": "describe this image in detail", "image": "examples.jpg"}},

Execution Results:

- "generated_text": "a family of four dogs are playing in the grass"
- {"score": 0.938717246055603, "label": "Rhodesian ridgeback"}, {"score": 0.01652258075773716, "label": "boxer"}, {"score": 0.006381669547408819, "label": "Great Dane"}, {"score": 0.006234415341168642, "label": "vizsla, Hungarian pointer"}, {"score": 0.005944834090769291, "label": "bul mastiff"}
- ["generated image with predicted box": "/images/79f2.jpg", "predicted": [{"box": {"xmax": 463, "xmin": 373, "ymax": 267, "ymin": 199}, "label": "bus", "score": 0.9981155395507812}, {"box": {"xmax": 292, "xmin": 124, "ymax": 224, "ymin": 175}, "label": "airplane", "score": 0.9983609318733215}, {"box": {"xmax": 554, "xmin": 487, "ymax": 277, "ymin": 226}, "label": "person", "score": 0.99108362197875981}], "score": 0.9981155395507812}
- ["generated image with segmentation": "/images/3fbfa.jpg", "predicted": [{"score": 0.989, "label": "grass"}, {"score": 0.999, "label": "dog"}, {"score": 0.999, "label": "tree"}, {"score": 0.999, "label": "dog"}]
- ["answer": "dogs", "score": 0.8488452434539795}, {"answer": "dog", "score": 0.04168461635708809]



04 | Limitations



19기 NLP 심화세션

TOBIG'S 19기 권유진

- Bottleneck of Efficiency
 - Task planning, model selection, response generation 과정에서 LLM과 최소 1번의 interaction 필요
 - Latency 증가 및 UX 감소 초래
- Context Length
 - LLM에 최대 토큰 수가 제한되어 있어 최대 길이에 한계 존재
- 시스템 안정성
 - LLM 추론 과정에서 rebellion 발생
 - 때때로 instruction을 준수하지 못해 예외 출력 발생
 - Hugging Face Inference Endpoint에 호스팅된 expert model을 통제 불가능
 - 네트워크 지연, 서비스 상태의 영향을 많이 받음

05 | Code



19기 NLP 심화세션

TOBIG'S 19기 권유진

```

1001 def cli():
1002     messages = []
1003     print("Welcome to Jarvis! A collaborative system that consists of an LLM as the controller and numerous expert models")
1004     while True:
1005         message = input("[ User ]: ")
1006         if message == "exit":
1007             break
1008         messages.append({"role": "user", "content": message})
1009         answer = chat_huggingface(messages, API_KEY, API_TYPE, API_ENDPOINT, return_planning=False, return_results=False)
1010         print("[ Jarvis ]: ", answer["message"])
1011         messages.append({"role": "assistant", "content": answer["message"]})

def chat_huggingface(messages, api_key, api_type, api_endpoint, return_planning = False, return_results = False):
    start = time.time()
    context = messages[:-1]
    input = messages[-1]["content"]
    logger.info("*"*80)
    logger.info(f"input: {input}")

    task_str = parse_task(context, input, api_key, api_type, api_endpoint)

    if "error" in task_str:
        record_case(success=False, **{"input": input, "task": task_str, "reason": f"task parsing error: {task_str['error']['message']}"}, "op": "return {"message": task_str["error"]["message"]}")

    task_str = task_str.strip()
    logger.info(task_str)

    try:
        tasks = json.loads(task_str)
    except Exception as e:
        logger.debug(e)
        response = chitchat(messages, api_key, api_type, api_endpoint)
        record_case(success=False, **{"input": input, "task": task_str, "reason": "task parsing fail", "op": "chitchat"})
        return {"message": response}

    if task_str == "[]": # using LLM response for empty task
        record_case(success=False, **{"input": input, "task": [], "reason": "task parsing fail: empty", "op": "chitchat"})
        response = chitchat(messages, api_key, api_type, api_endpoint)
        return {"message": response}

    if len(tasks) == 1 and tasks[0]["task"] in ["summarization", "translation", "conversational", "text-generation", "text2text-generation"]:
        record_case(success=True, **{"input": input, "task": tasks, "reason": "chitchat tasks", "op": "chitchat"})
        response = chitchat(messages, api_key, api_type, api_endpoint)
        return {"message": response}

    tasks = unfold(tasks)
    tasks = fix_dep(tasks)
    logger.debug(tasks)

    if return_planning:
        return tasks

```

```

def parse_task(context, input, api_key, api_type, api_endpoint):
    demos_or_presteps = parse_task_demos_or_presteps
    messages = json.loads(demos_or_presteps)
    messages.insert(0, {"role": "system", "content": parse_task_tprompt})

    # cut chat logs
    start = 0
    while start <= len(context):
        history = context[start:]
        prompt = replace_slot(parse_task_prompt, {
            "input": input,
            "context": history
        })
        messages.append({"role": "user", "content": prompt})
        history_text = "<im_end>\nuser->\n".join([m["content"] for m in messages])
        num = count_tokens(LLM_encoding, history_text)
        if get_max_context_length(LLM) - num > 800:
            break
        messages.pop()
        start += 2

    logger.debug(messages)
    data = {
        "model": LLM,
        "messages": messages,
        "temperature": 0,
        "logit_bias": {item: config["logit_bias"]["parse_task"] for item in task_parsing_highlight_ids},
        "api_key": api_key,
        "api_type": api_type,
        "api_endpoint": api_endpoint
    }
    return send_request(data)

```

```

def unfold(tasks):
    flag_unfold_task = False
    try:
        for task in tasks:
            for key, value in task["args"].items():
                if "<GENERATED>" in value:
                    generated_items = value.split(",")
                    if len(generated_items) > 1:
                        flag_unfold_task = True
                        for item in generated_items:
                            new_task = copy.deepcopy(task)
                            dep_task_id = int(item.split("-")[1])
                            new_task["dep"] = [dep_task_id]
                            new_task["args"][key] = item
                            tasks.append(new_task)
                            tasks.remove(task)
    except Exception as e:
        print(e)
        traceback.print_exc()
        logger.debug("unfold task failed.")

    if flag_unfold_task:
        logger.debug(f"unfold tasks: {tasks}")

```

```

def fix_dep(tasks):
    for task in tasks:
        args = task["args"]
        task["dep"] = []
        for k, v in args.items():
            if "<GENERATED>" in v:
                dep_task_id = int(v.split("-")[1])
                if dep_task_id not in task["dep"]:
                    task["dep"].append(dep_task_id)
    if len(task["dep"]) == 0:
        task["dep"] = [-1]
    return tasks

```

05 | Code



19기 NLP 심화세션

TOBIG'S 19기 권유진

```
def parse_task(context, input, api_key, api_type, api_endpoint):
    demos_or_presteps = parse_task_demos_or_presteps
    messages = json.loads(demos_or_presteps)
    messages.insert(0, {"role": "system", "content": parse_task_tprompt})

    # cut chat logs
    start = 0
    while start <= len(context):
        history = context[start:]
        prompt = replace_slot(parse_task_prompt, {
            "input": input,
            "context": history
        })
        messages.append({"role": "user", "content": prompt})
        history_text = "<im_end>\nuser<im_start>".join([m["content"] for m in messages])
        num = count_tokens(LLM_encoding, history_text)
        if get_max_context_length(LLM) - num > 800:
            break
        messages.pop()
        start += 2

    logger.debug(messages)
    data = {
        "model": LLM,
        "messages": messages,
        "temperature": 0,
        "logit_bias": {item: config["logit_bias"]["parse_task"] for item in task_parsing_highlight_ids},
        "api_key": api_key,
        "api_type": api_type,
        "api_endpoint": api_endpoint
    }
    return send_request(data)
```

parse_task_tprompt

#1 Task Planning Stage: The AI assistant can parse user input to several tasks: [{"task": task, "id": task_id, "dep": dependency_task_id, "args": {"text": text or <GENERATED>-dep_id, "image": image_url or <GENERATED>-dep_id, "audio": audio_url or <GENERATED>-dep_id}}]. The special tag "<GENERATED>-dep_id" refer to the one generated text/image/audio in the dependency task (Please consider whether the dependency task generates resources of this type.) and "dep_id" must be in "dep" list. The "dep" field denotes the ids of the previous prerequisite tasks which generate a new resource that the current task relies on. The "args" field must in ["text", "image", "audio"], nothing else. The task MUST be selected from the following options: "token-classification", "text2text-generation", "summarization", "translation", "question-answering", "conversational", "text-generation", "sentence-similarity", "tabular-classification", "object-detection", "image-classification", "image-to-image", "image-to-text", "text-to-image", "text-to-video", "visual-question-answering", "document-question-answering", "image-segmentation", "depth-estimation", "text-to-speech", "automatic-speech-recognition", "audio-to-audio", "audio-classification", "canny-control", "hed-control", "mlsd-control", "normal-control", "openpose-control", "canny-text-to-image", "depth-text-to-image", "hed-text-to-image", "mlsd-text-to-image", "normal-text-to-image", "openpose-text-to-image", "seg-text-to-image". There may be multiple tasks of the same type. Think step by step about all the tasks needed to resolve the user's request. Parse out as few tasks as possible while ensuring that the user request can be resolved. Pay attention to the dependencies and order among tasks. If the user input can't be parsed, you need to reply empty JSON [].

	Prompt
Task Planning	#1 Task Planning Stage - The AI assistant can parse user input to several tasks: [{"task": task, "id": task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": URL, "video": URL}}]. The "dep" field denotes the id of the previous task which generates a new resource that the current task relies on. A special tag "<resource>-task_id" refers to the generated text image, audio and video in the dependency task with id as task_id. The task MUST be selected from the following options: {{ Available Task List }}. There is a logical relationship between tasks, please note their order. If the user input can't be parsed, you need to reply empty JSON. Here are several cases for your reference: {{ Demonstrations }}. The chat history is recorded as {{ Chat Logs }}. From the chat logs, you can find the path of the user-mentioned resources for your task planning.
	Demonstrations
	Look at /exp1.jpg, Can you tell me how many objects in the picture?

In /exp2.jpg, what's the animal and what's it doing?

Given an image /exp3.jpg, first generate a hed image, then based on the hed image and a prompt: a girl is reading a book, you need to reply with a new image.

```
[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/exp1.jpg" }}, {"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "/exp1.jpg" }}, [{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/exp2.jpg" }}, {"task": "image-classification", "id": 1, "dep": [-1], "args": {"image": "/exp2.jpg" }}, {"task": "object-detection", "id": 2, "dep": [-1], "args": {"image": "/exp2.jpg" }}, {"task": "visual-question-answering", "id": 3, "dep": [-1], "args": {"text": "What's the animal doing?", "image": "/exp2.jpg" }}}]
```

```
[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/examples/boy.jpg" }}, {"task": "openpose-control", "id": 1, "dep": [-1], "args": {"image": "/examples/boy.jpg" }}, {"task": "openpose-text-to-image", "id": 2, "dep": [1], "args": {"text": "a girl is reading a book", "image": "<resource>-1" }}]
```

parse_task_prompt

The chat log [{{context}}] may contain the resources I mentioned. Now I input { {{input}} }. Pay attention to the input and output types of tasks and the dependencies between tasks.

05 | Code



19기 NLP 심화세션

TOBIG'S 19기 권유진

```
results = {}
threads = []
tasks = tasks[:]
d = dict()
retry = 0
while True:
    num_thread = len(threads)
    for task in tasks:
        # logger.debug(f"d.keys(): {d.keys()}, dep: {dep}")
        for dep_id in task["dep"]:
            if dep_id >= task["id"]:
                task["dep"] = [-1]
                break
        dep = task["dep"]
        if dep[0] == -1 or len(list(set(dep).intersection(d.keys()))) == len(dep):
            tasks.remove(task)
            thread = threading.Thread(target=run_task, args=(input, task, d, api_key, api_type, api_endpoint))
            thread.start()
            threads.append(thread)
    if num_thread == len(threads):
        time.sleep(0.5)
        retry += 1
    if retry > 160:
        logger.debug("User has waited too long, Loop break.")
        break
    if len(tasks) == 0:
        break
    for thread in threads:
        thread.join()

    results = d.copy()
    logger.debug(results)
    if return_results:
        return results

    response = response_results(input, results, api_key, api_type, api_endpoint).strip()

    end = time.time()
    during = end - start

    answer = {"message": response}
    record_case(success=True, **{"input": input, "task": task_str, "results": results, "response": response, "during": during, "op": "response"})
    logger.info(f"response: {response}")
return answer
```

Model Selection → Model Execution

05 | Code



19기 NLP 심화세션

TOBIG'S 19기 권유진

```
def choose_model(input, task, metas, api_key, api_type, api_endpoint):
    prompt = replace_slot(choose_model_prompt, {
        "input": input,
        "task": task,
        "metas": metas,
    })
    demos_or_presteps = replace_slot(choose_model_demos_or_presteps, {
        "input": input,
        "task": task,
        "metas": metas
    })
    messages = json.loads(demos_or_presteps)
    messages.insert(0, {"role": "system", "content": choose_model_tprompt})
    messages.append({"role": "user", "content": prompt})
    logger.debug(messages)
    data = {
        "model": LLM,
        "messages": messages,
        "temperature": 0,
        "logit_bias": {item: config["logit_bias"]["choose_model"] for item in choose_model_highlight_ids}, # 5
        "api_key": api_key,
        "api_type": api_type,
        "api_endpoint": api_endpoint
    }
    return send_request(data)
```

parse_task_tprompt

#2 Model Selection Stage: Given the user request and the parsed task s, the AI assistant helps the user to select a suitable model from a list of models to process the user request. The assistant should focus more on the description of the model and find the model that has the most potential to solve requests and tasks. Also, prefer models with local inference endpoints for speed and stability.

Model Selection	Prompt
	#2 Model Selection Stage - Given the user request and the call command, the AI assistant helps the user to select a suitable model from a list of models to process the user request. The AI assistant merely outputs the model id of the most appropriate model. The output must be in a strict JSON format: {"id": "id", "reason": "your detail reason for the choice"}. We have a list of models for you to choose from {{ Candidate Models }}. Please select one model from the list.
Candidate Models	
	{"model_id": model id #1, "metadata": meta-infos #1, "description": description of model #1} {"model_id": model id #2, "metadata": meta-infos #2, "description": description of model #2} [Select top-K candidate models with the same task type as the current task] {"model_id": model id #K, "metadata": meta-infos #K, "description": description of model #K}

parse_task_prompt

Please choose the most suitable model from {{metas}} for the task {{task}}. The output must be in a strict JSON format: {"id": "id", "reason": "your detail reasons for the choice"}.

05 | Code



19기 NLP 심화세션

TOBIG'S 19기 권유진

```
def get_available_models(candidates, topk=5):
    all_available_models = {"local": [], "huggingface": []}
    threads = []
    result_queue = Queue()

    for candidate in candidates:
        model_id = candidate["id"]

        if inference_mode != "local":
            huggingfaceStatusUrl = f"https://api-inference.huggingface.co/status/{model_id}"
            thread = threading.Thread(target=get_model_status, args=(model_id, huggingfaceStatusUrl, HUGGINGFACE_HEADERS, result_queue))
            threads.append(thread)
            thread.start()

        if inference_mode != "huggingface" and config["local_deployment"] != "minimal":
            localStatusUrl = f"{Model_Server}/status/{model_id}"
            thread = threading.Thread(target=get_model_status, args=(model_id, localStatusUrl, {}, result_queue))
            threads.append(thread)
            thread.start()

    result_count = len(threads)
    while result_count:
        model_id, status, endpoint_type = result_queue.get()
        if status and model_id not in all_available_models:
            all_available_models[endpoint_type].append(model_id)
        if len(all_available_models["local"]) + len(all_available_models["huggingface"]) >= topk:
            break
        result_count -= 1

    for thread in threads:
        thread.join()

    return all_available_models
```

Response Generation

Prompt
#4 Response Generation Stage - With the input and the inference results, the AI assistant needs to describe the process and results. The previous stages can be formed as - User Input: {{ User Input }} Task Planning: {{ Tasks }}, Model Selection: {{ Model Assignment }}, Task Execution: {{ Predictions }}. You must first answer the user's request in a straightforward manner. Then describe the task process and show your analysis and model inference results to the user in the first person. If inference results contain a file path, must tell the user the complete file path. If there is nothing in the results, please tell me you can't make it.

```
def response_results(input, results, api_key, api_type, api_endpoint):
    results = [v for k, v in sorted(results.items(), key=lambda item: item[0])]
    prompt = replace_slot(response_results_prompt, {
        "input": input
    })
    demos_or_presteps = replace_slot(response_results_demos_or_presteps, {
        "input": input,
        "processes": results
    })
    messages = json.loads(demos_or_presteps)
    messages.insert(0, {"role": "system", "content": response_results_tprompt})
    messages.append({"role": "user", "content": prompt})
    logger.debug(messages)
    data = {
        "model": LLM,
        "messages": messages,
        "temperature": 0,
        "api_key": api_key,
        "api_type": api_type,
        "api_endpoint": api_endpoint
    }
    return send_request(data)
```

parse_task_prompt

Yes. Please first think carefully and directly answer my request based on the inference results. Some of the inferences may not always turn out to be correct and require you to make careful consideration in making decisions. Then please detail your workflow including the used models and inference results for my request in your friendly tone. Please filter out information that is not relevant to my request. Tell me the complete path or urls of files in inference results. If there is nothing in the results, please tell me you can't make it.

parse_task_tprompt

#4 Response Generation Stage: With the task execution logs, the AI assistant needs to describe the process and inference results.

Reference



19기 정규세션

TOBIG'S 18기 국주현

Shen, Yongliang, et al. "Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface." *arXiv preprint arXiv:2303.17580* (2023).



감사합니다

TOBIG's