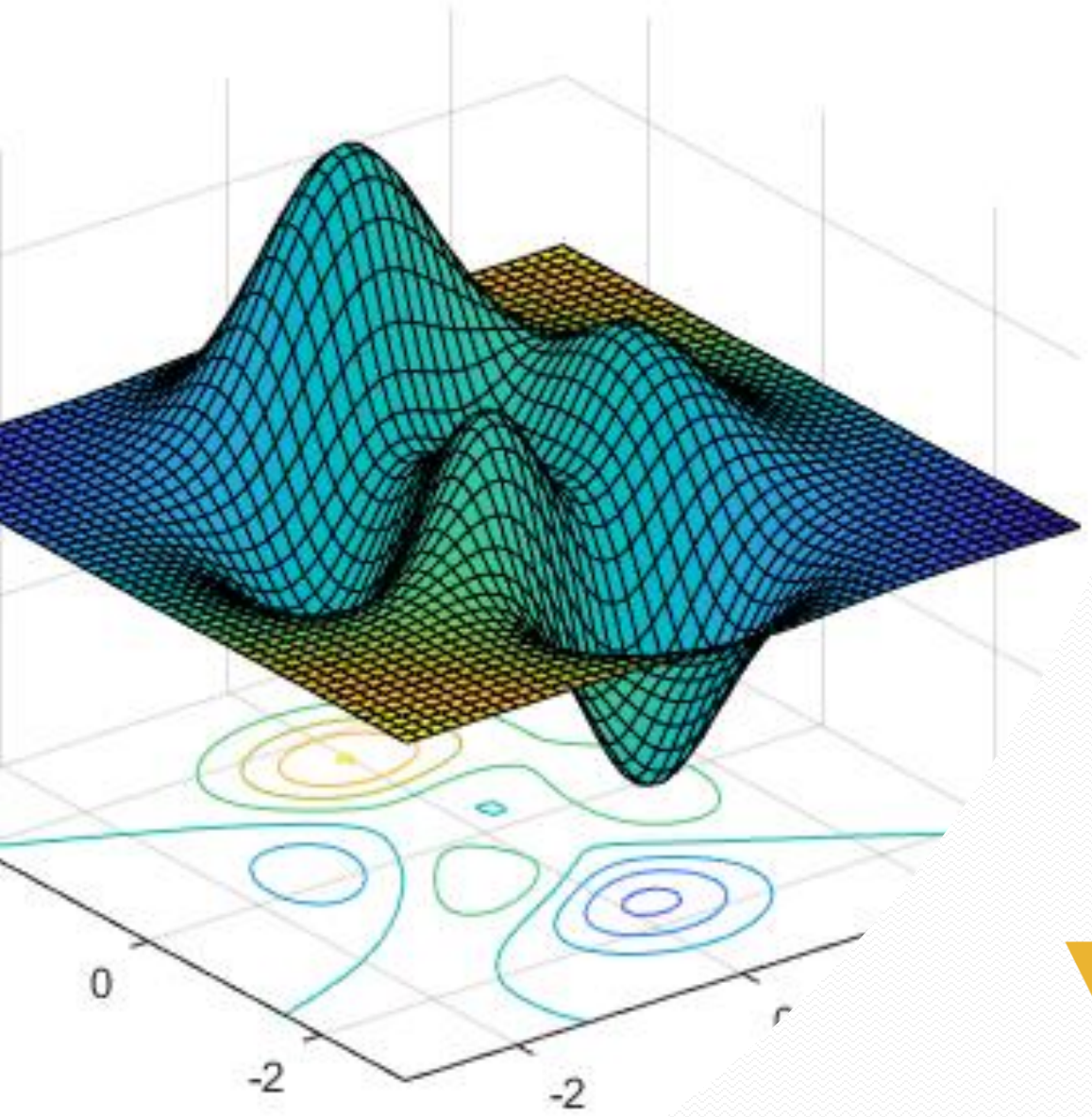


More over Ollama

vLLM

BentoML

TensorRT

https://ncsoft.github.io/ncresearch/512f982a1564b5441d432935a7098146226e20b7

https://eehoeskrap.tistory.com/414

vLLM, TensorRT, openLLM

TensorRT-LLM LLM뿐만 아니라 딥러닝 모델 추론시 속도 개선 방법

TensorRT-LLM[3]은 NVIDIA에서 개발한 오픈소스 라이브러리로, NVIDIA GPU에서 LLM 추론 성능을 최적화하는 데 특화되어 있습니다. TensorRT engine, C++ 런타임 최적화 등 다양한 기술을 통해 모델의 추론 속도를 향상시킵니다. Huggingface 모델들을 사용하기 위해서는 TensorRT engine을 빌드하는 등 추가적인 작업이 필요합니다. NVIDIA Triton Inference Server를 활용하여 모델 서빙을 할 수 있습니다.

vLLM

vLLM[5]은 사용이 간편하면서도 빠른 LLM 추론 및 서빙을 위한 라이브러리입니다. Attention의 Key와 Value의 효율적인 메모리 관리를 지원하는 Paged Attention 기술을 핵심으로 사용합니다. 최적화된 CUDA 커널을 사용하여 추론 속도를 높이며, NVIDIA GPU뿐만 아니라 AMD와 Intel GPU, 심지어 CPU에서도 사용할 수 있어 다양한 환경에서 사용이 가능합니다. Huggingface 모델들과 쉽게 통합할 수 있고 OpenAI-compatible API server를 제공하여 쉽고 간단하게 모델 서빙을 할 수 있습니다. 추가로, NVIDIA Triton Inference Server를 활용하여 모델 서빙을 할 수 있습니다.

vLLM은 NVIDIA GPU뿐만 아니라 다양한 환경에서 간편하게 모델 서빙을 위해 사용할 수 있습니다. 하지만 저희 모델과 테스트 환경에 대해서는 TensorRT-LLM에 비해 낮은 성능을 보여주었습니다. TensorRT-LLM은 NVIDIA GPU에 최적화되어 다양한 환경에서 사용이 어려우며 weight 변환 및 TensorRT engine을 빌드해야 하는 번거로움이 있습니다. 하지만, 저희 모델과 테스트 환경에 대해서 아주 좋은 성능을 보여주었습니다. TensorRT 혹은 Onnx 모델포맷 지원

TensorRT-LLM과 vLLM 모두 기존 python 실행보다 훨씬 빠른 속도를 보여주고 모델과 사용 환경마다 성능 차이가 있으므로 테스트하셔서 상황에 맞게 선택하면 좋을 거 같습니다. 추가로, 잘 알려진 모델 아키텍쳐를 서빙할 때는 각각 지원하는 모델이 다르므로 지원되는 모델을 확인하여 사용하면 좋습니다. (둘 다 새로운 모델 추가도 가능합니다.)

	vLLM	TensorRT	openLLM
속도	Ollama보다 훨씬 빠름	유리	Ollama 8배?
셋팅(다양한 환경 지원)	TensorRT보다 쉬움	불리 Linux TensorRT Docker	Ollama 만큼 쉬움
지원모델	거의제한없음 Huggingface, 추가도 가능	제한없음. TensorRT or Onnx 모델포맷 변환필요	Llama, mistral, gemma, qwen 등 얼마 없음

이러한 가속화 기술들을 이용하여 TensorRT는 "속도 향상" 이라는 결과를 얻을 수 있게 된다. 기본적으로 ResNet50 기준으로 볼 때 동일한 GPU 에서 TensorRT 를 사용하는 것만으로도 대략 8배 이상의 성능 향상 효과가 있다고 한다. 필자도 실제로 모델을 최적화 시켰을 때 Pytorch 나 TensorFlow 모델의 추론 속도에 비해 TensorRT Engine 화 하였을 때의 모델 추론 속도가 적게는 5배에서 많게는 10배 까지 속도 향상 결과를 맛봤다. 모델 백본에 따라 성능 차이는 있겠지만 딥러닝 서비스를 배포하기 위해 속도 측면에서 TensorRT가 핵심적인 역할을 수행할 수 있었다.

2. TensorRT 설치하기

TensorRT 는 필자의 오래전 경험에 의하면 Anaconda 에서는 실행하기 힘들다. Anaconda 와 같은 가상환경을 사용할 경우 TensorRT 가 설치된 곳을 제어할 수 없기 때문에 **Anaconda 에서 TensorRT 의 실행은 지양**한다.

보통 Nvidia Docker 에서 TensorRT Container 를 사용하거나, 그냥 쌩 로컬 환경에 TensorRT를 설치하는 것을 추천한다. 설치법은 이미 많은 분들이 포스팅 했을 거라고 생각한다... 하지만 간략히 써보도록 하겠다. 어렵진 않다. 도커를 사용한다면 문제 없겠지만, 로컬환경에 설치할 때 제일 중요한 건 환경 셋팅 인 듯 하다. 주요 포스팅은 다음을 참고하길 바란다. 아래 2.1 절과 2.2 에서는 간략히만 설명한다.

- (Linux) Docker Container 를 이용한 TensorRT 설치하기
- (Linux) 로컬 환경에 TensorRT 설치하기
- (Windows) 로컬환경에 TensorRT 설치하기

openLLM 관련

https://www.bentoml.com/blog/from-ollama-to-openllm-running-llms-in-the-cloud
https://github.com/bentoml/OpenLLM

2) vLLM - 그놈에는 없음.

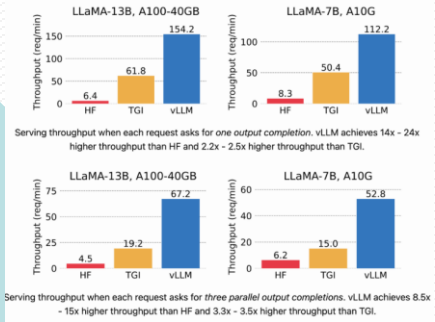
vLLM 주요 특징

vLLM is fast with:
•State-of-the-art serving throughput
•Efficient management of attention key and value memory with **PagedAttention**
•Continuous batching of incoming requests
•Fast model execution with CUDA/HIP graph
•Quantization: GPTQ, AWQ, INT4, INT8, and FP8
•Optimized CUDA kernels, including integration with FlashAttention and FlashInfer.
•Speculative decoding
•Chunked prefill

vLLM is flexible and easy to use with:
•Seamless integration with popular HuggingFace models
•High-throughput serving with various decoding algorithms, including *parallel sampling*, *beam search*, and more
•Multi-lora support

Prerequisites
•OS: Linux
•Python: 3.9 – 3.12
•GPU: compute capability 7.0 or higher (e.g., V100, T4, RTX20xx, A100, L4, H100, etc.)

Installation
conda create -n myenv python=3.10 -y
conda activate myenv
pip install vllm



기존
vLLM Paged Attention 에 대한 설명.
https://tristanchoi.tistory.com/651

설명, 논문, 실행까지
https://lsjsj92.tistory.com/668

그외
https://docs.vllm.ai/en/latest/
https://github.com/vllm-project/vllm?tab=readme-ov-file
https://arxiv.org/pdf/2309.06180

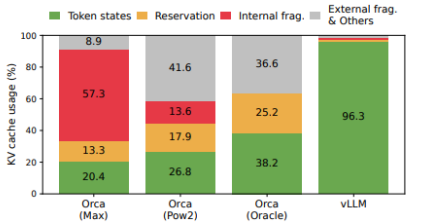


Figure 2. Average percentage of memory wastes in different LLM serving systems during the experiment in §6.2.

Offline Inference Chat

Source vllm-project/vllm.

```python
from vllm import LLM, SamplingParams

llm = LLM(model="meta-llama/Meta-Llama-3-8B-Instruct")
sampling_params = SamplingParams(temperature=0.5)


def print_outputs(outputs):
    for output in outputs:
        prompt = output.prompt
        generated_text = output.outputs[0].text
        print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
    print("-" * 80)


print("=" * 80)

# In this script, we demonstrate how to pass input to the chat method:

conversation = [
    {
        "role": "system",
        "content": "You are a helpful assistant"
    },
    {
        "role": "user",
        "content": "Hello"
    },
    {
        "role": "assistant",
        "content": "Hello! How can I assist you today?"
    },
    {
        "role": "user",
        "content": "Write an essay about the importance of higher education.",
    },
]
outputs = llm.chat(conversation,
                   sampling_params=sampling_params,
                   use_tqdm=False)
print_outputs(outputs)

# You can run batch inference with llm.chat API
conversation = [
    {
        "role": "system",
        "content": "You are a helpful assistant"
    },
    {
        "role": "user",
        "content": "Hello"
    },
    {
        "role": "assistant",
        "content": "Hello! How can I assist you today?"
    },
    {
        "role": "user",
        "content": "Write an essay about the importance of higher education.",
    },
```

vLLM – hands on - sample

https://docs.vllm.ai/en/latest/getting_started/quickstart.html

Requirements

•Python: 3.9 – 3.12

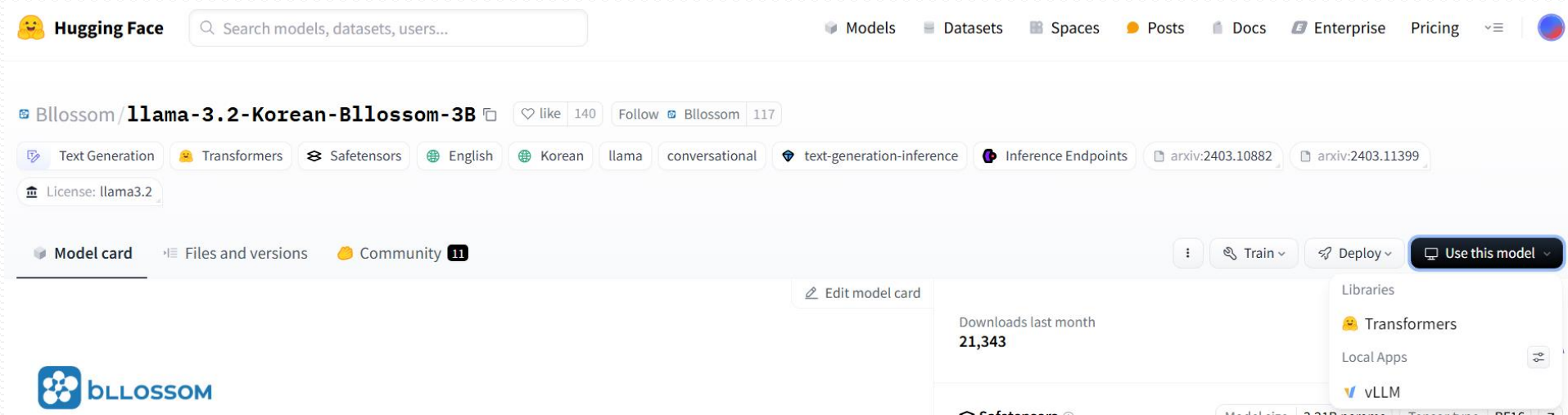
CUDA

•GPU: compute capability 7.0 or higher (e.g., V100, T4, RTX20xx, A100, L4, H100, etc.)

```
~/workspace/chatbot/chatbot_origin$ source venv/bin/activate
(venv) ~/workspace/chatbot/chatbot_origin$ pip install vllm
(venv) ~/workspace/chatbot/chatbot_origin$ mkdir vllm
(venv) ~/workspace/chatbot/chatbot_origin$ cd vllm/
(venv) ~/workspace/chatbot/chatbot_origin/vllm$ vi Quickstart.py

(venv) ~/workspace/chatbot/chatbot_origin/vllm$ python Quickstart.py
Prompt: 'Hello, my name is', Generated text: ' Joel, my dad is my friend and we are in a relationship. I am'
Prompt: 'The president of the United States is', Generated text: ' speaking out against the release of some
State Department documents which show the Russians were involved'
Prompt: 'The capital of France is', Generated text: ' known as the “Proud French capital”. What is this
city'
Prompt: 'The future of AI is', Generated text: ' going to be long-term. The AI will continue to evolve as
part of'
```

```python
from vllm import LLM, SamplingParams
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)
llm = LLM(model="facebook/opt-125m")
outputs = llm.generate(prompts, sampling_params)

for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```



vLLM – hands on - linux

https://docs.vllm.ai/en/latest/getting_started/quickstart.html

Pre-built wheels

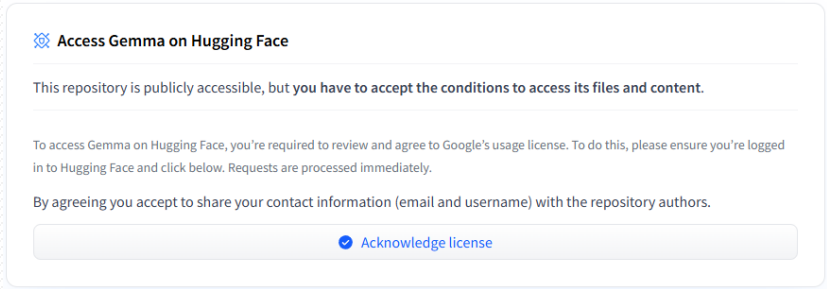
NVIDIA CUDA | AMD ROCm | Intel XPU

You can install vLLM using either `pip` or `uv pip`:

```
# Install vLLM with CUDA 12.6.
pip install vllm # If you are using pip.
uv pip install vllm # If you are using uv.
```

As of now, vLLM's binaries are compiled with CUDA 12.6 and public PyTorch release versions by default. We also provide vLLM binaries compiled with CUDA 12.8, 11.8, and public PyTorch release versions:

허깅페이스 토큰 발행 https://huggingface.co/settings/tokens




```
(/data2/conda_env/vllm) tako@u20:/data2/vllm$ huggingface-cli login
(/data2/conda_env/vllm) tako@u20:/data2/vllm$ huggingface-cli whoami
(/data2/conda_env/vllm) tako@u20:/data2/vllm$ mkdir models/Llama-3.2-1B
(/data2/conda_env/vllm) tako@u20:/data2/vllm$ cd models/Llama-3.2-1B
(/data2/conda_env/vllm) tako@u20:/data2/vllm/models/Llama-3.2-1B $ huggingface-cli download meta-llama/Llama-3.2-1B
```

nvidia-container-cli: requirement error: unsatisfied condition: cuda>=12.4, please update your driver to a newer version, or use an earlier cuda container: unknown.

```
(/data2/conda_env/vllm) tako@u20:/data2/vllm$ docker pull vllm/vllm-openai
```

```
1 docker run -d --runtime nvidia --gpus all \  2   --shm-size=128G \  3   -v /data2/vllm/models:/app/models \
 4   --name OmniSQL-7B \
 5   -p 11400:11400 --ipc=host  \
 6   vllm/vllm-openai:latest \
 7   --model \
 8    /app/models/OmniSQL-7B \
 9    --served-model-name OmniSQL-7B \
10    --max-num-seq=41 \
11    --tensor-parallel-size=2 \
12    --gpu-memory-utilization=0.9 \
13    --enforce-eager \
14    --chat-template-content-format=openai
```

vLLM – hands on - linux

```
docker run -d --runtime nvidia --gpus all ₩
  --shm-size=384G ₩
  -v /data2/models:/app/models ₩
  --name llama-3.1-70b ₩
  -p 8080:8000 --ipc=host  ₩
  vllm/vllm-openai:latest ₩
  --model ₩
    /app/models/Llama-3.1-70B-Instruct ₩
    --served-model-name LLAMA-3.1-70B ₩
    --max-num-seq=41 ₩
    --tensor-parallel-size=4 ₩
    --gpu-memory-utilization=0.9 ₩
    --enforce-eager ₩
    --chat-template-content-format=openai
```

```
# 1. vllm 서버 주소 및 모델명
server_url = "http://192.168.1.239/vllm9/v1"                    #
model_name = "omnisql-32b"

# 2. 실제 DB 스키마 문자열을 준비 (예시: SHOW CREATE TABLE ... 결과 등)
db_details = """
CREATE TABLE t2t_poc.per_pbr (
    code text NULL COMMENT '종목코드',
    "name" text NULL COMMENT '종목명',
    close_price int8 NULL COMMENT '종가',
    price_change int8 NULL COMMENT '대비',
    change_rate float8 NULL COMMENT '등락률',
    eps float8 NULL COMMENT '주당순이익',
    per float8 NULL COMMENT '주가수익률',
    forward_eps float8 NULL COMMENT '선행 EPS',
    forward_per float8 NULL COMMENT '선행 PER',
    bps float8 NULL COMMENT '주당순자산가치',
    pbr float8 NULL COMMENT '주가순자산비율',
    dividend_per_share int8 NULL COMMENT '주당배당금',
    dividend_yield float8 NULL COMMENT '배당수익률'
);
"""
```

```
# 3. 프롬프트 템플릿 (HuggingFace 예제와 동일)
input_prompt_template = '''Task Overview:
You are a data science expert. Below, you are provided with a database schema and a
natural language question. Your task is to understand the schema and generate a
valid SQL query to answer the question.

Database Engine:
SQLite

Database Schema:
{db_details}
This schema describes the database's structure, including tables, columns, primary
keys, foreign keys, and any relevant relationships or constraints.

Question:
{question}

Instructions:
- Make sure you only output the information that is asked in the question. If the
question asks for a specific column, make sure to only include that column in the
SELECT clause, nothing more.
- The generated query should return all of the information asked in the question
without any missing or extra information.
- Before generating the final SQL query, please think through the steps of how to
write the query.

Output Format:
In your answer, please enclose the generated SQL query in a code block:
```
-- Your SQL query
```

Take a deep breath and think step by step to find the correct SQL query.'''
```

vLLM – hands on - linux

```
# 4. 자연어 질문 입력
question = 'PER이 10~20 사이인 종목을 알려줘'

from langchain_openai import OpenAI  # ChatOpenAI 대신 OpenAI 사용

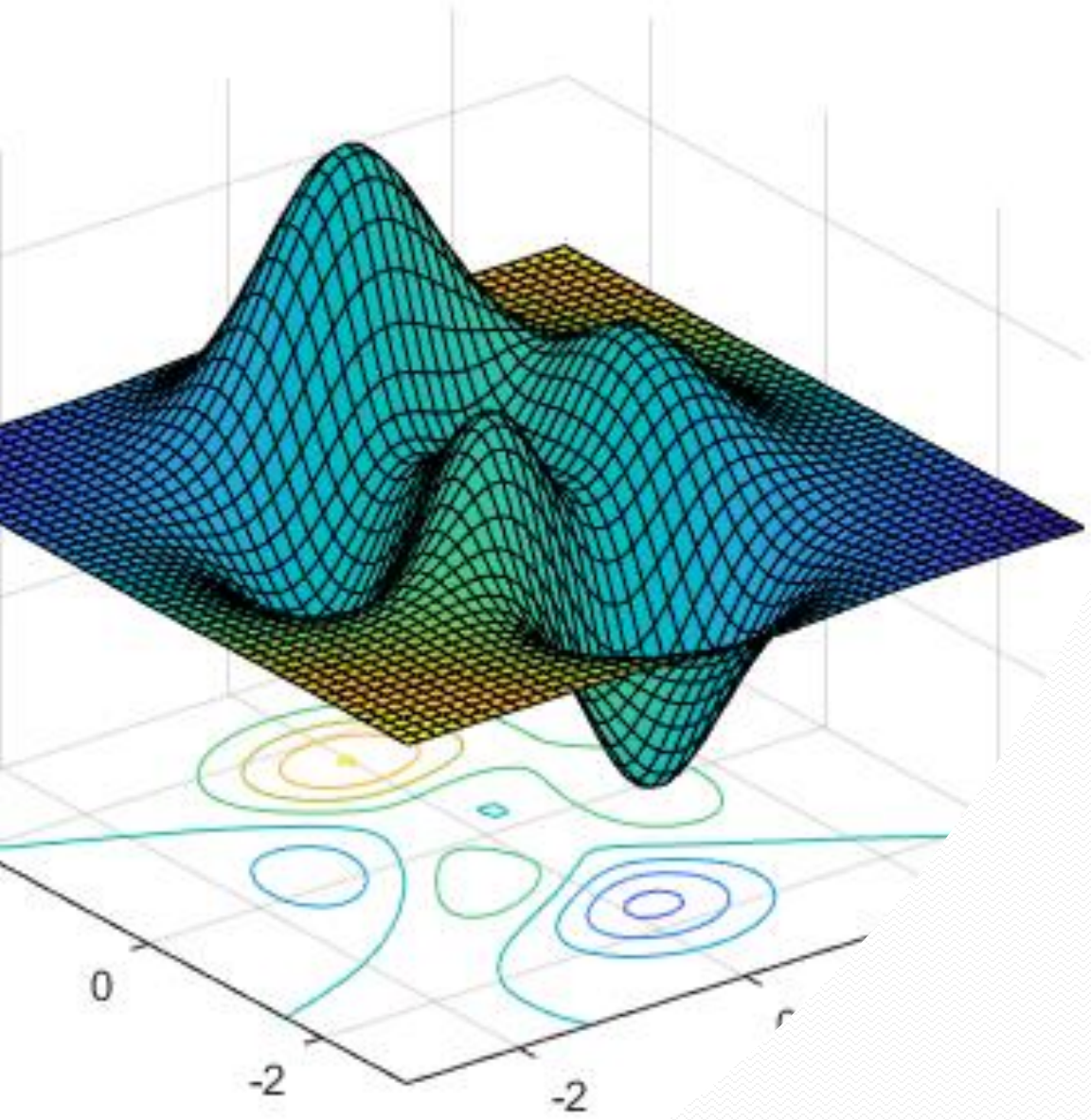
server_url = "http://192.168.1.239/vllm9/v1"
model_name = "omnisql-32b"

# OpenAI 클래스는 /completions 엔드포인트를 사용
t2s_llm = OpenAI(
    model=model_name,
    openai_api_key="EMPTY",
    openai_api_base=server_url,
    temperature=0
)

prompt = input_prompt_template.format(db_details=db_details, question=question)
response = t2s_llm.invoke(prompt)
print(response)
```

```
Step-by-step reasoning trace:
1. **Understand the Question**: The question asks for the names of stocks (종목명) where
the Price-to-Earnings Ratio (PER) is between 10 and 20.
2. **Identify Relevant Table and Columns**: - The relevant table is `t2t_poc.per_pbr`. -
The relevant columns are `name` (for the stock name) and `per` (for the Price-to-Earnings
Ratio).
3. **Formulate the Condition**: - The condition is that the `per` value should be greater
than or equal to 10 and less than or equal to 20.
4. **Construct the SQL Query**:
  - Use the `SELECT` statement to specify that we want to retrieve the `name` column.
  - Use the `FROM` clause to specify the table `t2t_poc.per_pbr`.
  - Use the `WHERE` clause to apply the condition on the `per` column.

Accordingly, the final SQL query is:
```sql
SELECT name FROM t2t_poc.per_pbr
WHERE per >= 10 AND per <= 20;
```
```



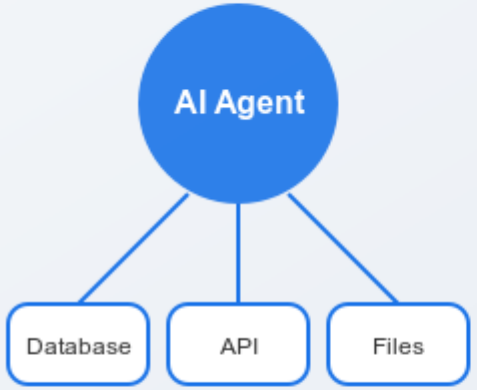
A2A

https://medium.com/@maheshmaddi92/mcp-vs-a2a-how-ai-agents-connect-collaborate-and-evolve-95201de70133

A2A vs MCP

Model Context Protocol (MCP)

AI ↔ Tools Connection

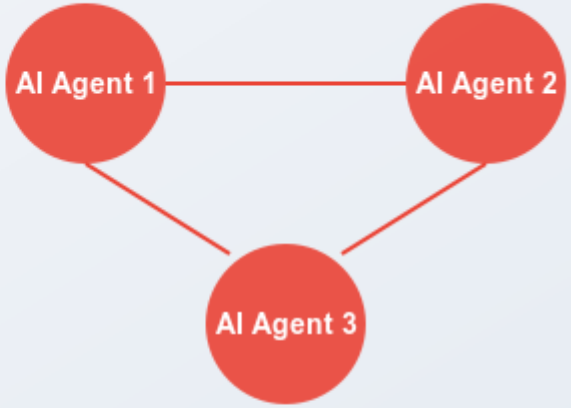


Connects AI to external resources

- Single-agent tasks
- Tool integration
- External data access

Agent-to-Agent Protocol (A2A)

AI ↔ AI Collaboration

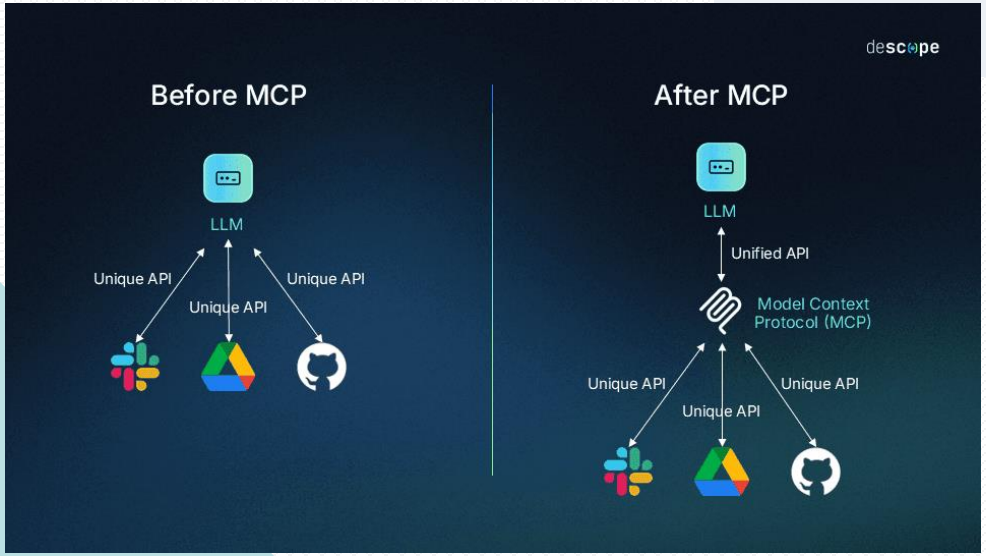


Enables AI agents to communicate

- Multi-agent collaboration
- Cross-platform workflows
- Specialized agent coordination

Complementary technologies for the future of AI ecosystems

https://www.descope.com/learn/post/a2a



https://www.youtube.com/watch?v=pytlKd6JJyI

quickstart

https://a2aproject.github.io/A2A/latest/tutorials/python/1-introduction/

```
PS D:\GITLAB\text2SQL> git clone https://github.com/google-a2a/a2a-samples.git -b main --depth 1
PS D:\GITLAB\text2SQL> cd a2a-samples
PS D:\GITLAB\text2SQL\a2a-samples> python -m venv .venv
PS D:\GITLAB\text2SQL\a2a-samples> .\.venv\Scripts\activate
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> pip install -r samples/python/requirements.txt

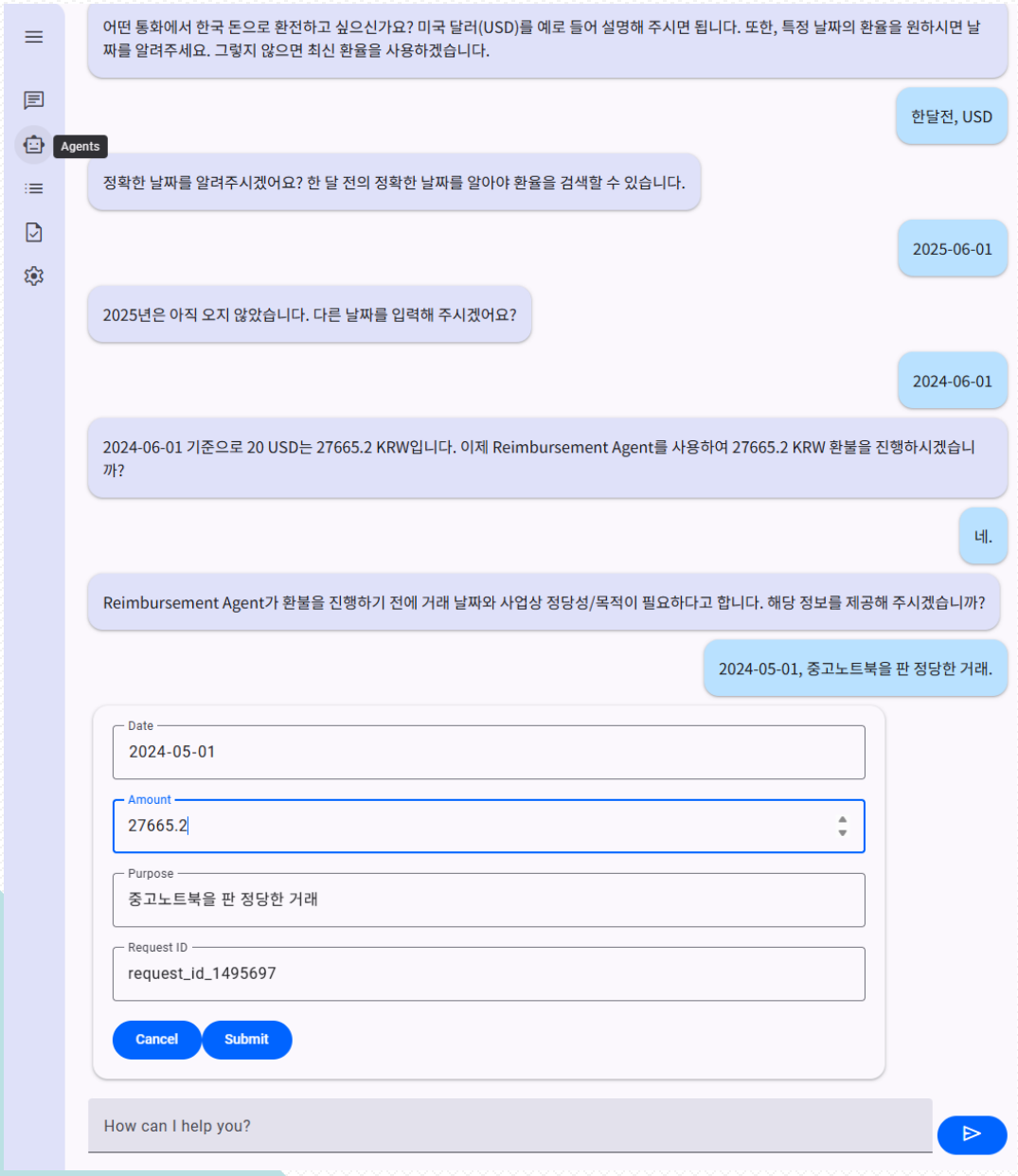
# install error 발생시
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> pip install uv
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> uv pip install -r samples/python/requirements.txt
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> pip install -r samples/python/requirements.txt

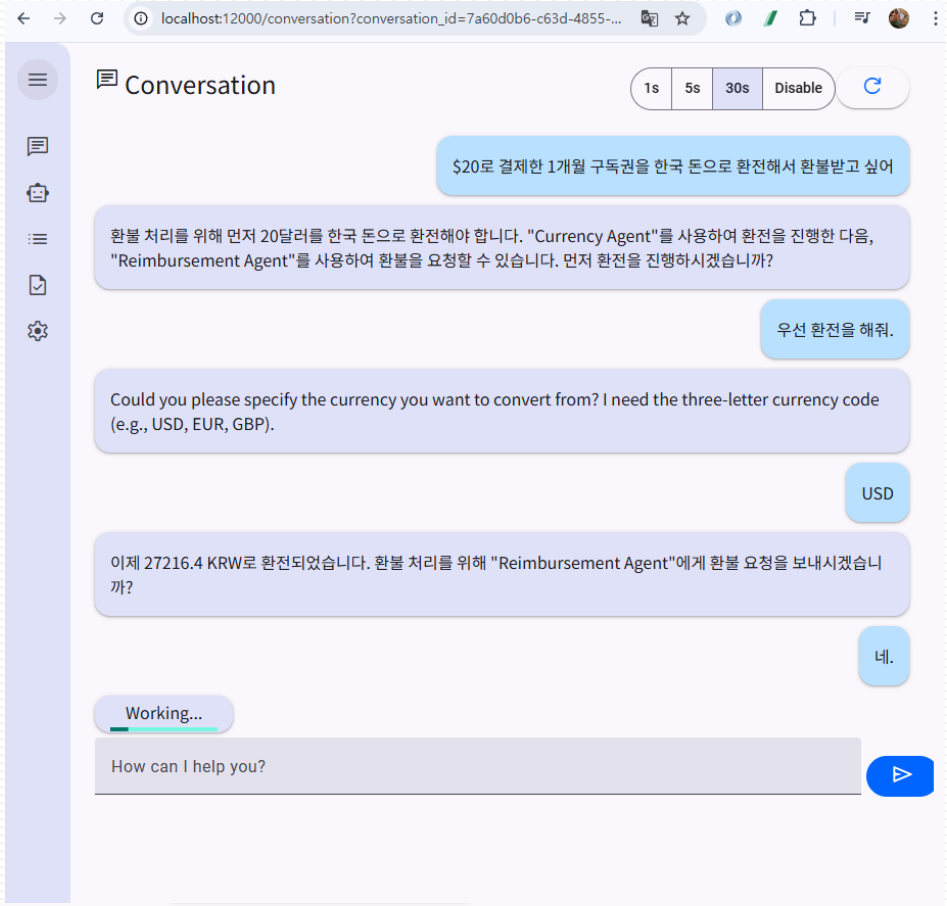
# 셋팅 테스트
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> python -c "import a2a; print('A2A SDK imported successfully')"
A2A SDK imported successfully

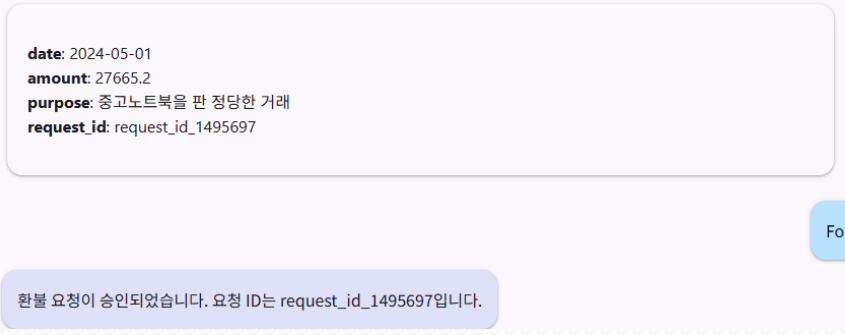
# 서버 구동
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> python samples/python/agents/helloworld/__main__.py
INFO:     Started server process [25076]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:9999 (Press CTRL+C to quit)

# client 구동
PS D:\GITLAB> cd .\text2SQL\a2a-samples\
PS D:\GITLAB\text2SQL\a2a-samples> .venv\Scripts\activate
(.venv) PS D:\GITLAB\text2SQL\a2a-samples> python samples/python/agents/helloworld/test_client.py
INFO:__main__:Attempting to fetch public agent card from: http://localhost:9999/.well-known/agent.json
INFO:httpx:HTTP Request: GET http://localhost:9999/.well-known/agent.json "HTTP/1.1 200 OK"
INFO:a2a.client.client:Successfully fetched agent card data from http://localhost:9999/.well-
known/agent.json: {'capabilities': {'streaming': True}, 'defaultInputModes': ['text'],
'defaultOutputModes': ['text'], 'description': 'Just a hello world agent', 'name': 'Hello World Agent',
'skills': [{'description': 'just returns hello world', 'examples': ['hi', 'hello world'], 'id':
'hello_world', 'name': 'Returns hello world', 'tags': ['hello world']}],
'supportsAuthenticatedExtendedCard': True, 'url': 'http://localhost:9999/', 'version': '1.0.0'}
INFO:__main__:Successfully fetched public agent card:
INFO:__main__:{
  "capabilities": {
    "streaming": true
  },
  "defaultInputModes": [
    "text"
  ],
  "defaultOutputModes": [
    "text"
  ],
  "description": "Just a hello world agent",
  "name": "Hello World Agent",
  "skills": [
    {
      "description": "just returns hello world",
      "examples": [
        "hi",
        "hello world"
      ],
      "id": "hello_world",
      "name": "Returns hello world",
      "tags": [
        "hello world"
      ]
    }
  ],
  "supportsAuthenticatedExtendedCard": true,
  "url": "http://localhost:9999/",
  "version": "1.0.0"
}
INFO:__main__:
Using PUBLIC agent card for client initialization (default).
INFO:__main__:
Public card supports authenticated extended card. Attempting to fetch from:
http://localhost:9999/agent/authenticatedExtendedCard
INFO:httpx:HTTP Request: GET http://localhost:9999/agent/authenticatedExtendedCard "HTTP/1.1 200 OK"
INFO:a2a.client.client:Successfully fetched agent card data from
http://localhost:9999/agent/authenticatedExtendedCard: {'capabilities': {'streaming': True},
'defaultInputModes': ['text'], 'defaultOutputModes': ['text'], 'description': 'The full-featured hello
world agent for authenticated users.', 'name': 'Hello World Agent - Extended Edition', 'skills':
[{'description': 'just returns hello world', 'examples': ['hi', 'hello world'], 'id': 'hello_world',
'name': 'Returns hello world', 'tags': ['hello world']}, {'description': 'A more enthusiastic greeting,
only for authenticated users.', 'examples': ['super hi', 'give me a super hello'], 'id':
'super_hello_world', 'name': 'Returns a SUPER Hello World', 'tags': ['hello world', 'super', 'extended']}],
'supportsAuthenticatedExtendedCard': True, 'url': 'http://localhost:9999/', 'version': '1.0.1'}
INFO:__main__:Successfully fetched authenticated extended agent card:
INFO:__main__:{
  "capabilities": {
    "streaming": true
  },
  "defaultInputModes": [
    "text"
  ],
  "defaultOutputModes": [
```

A2A Agent 구현 - 환전, 비용처리 Agent







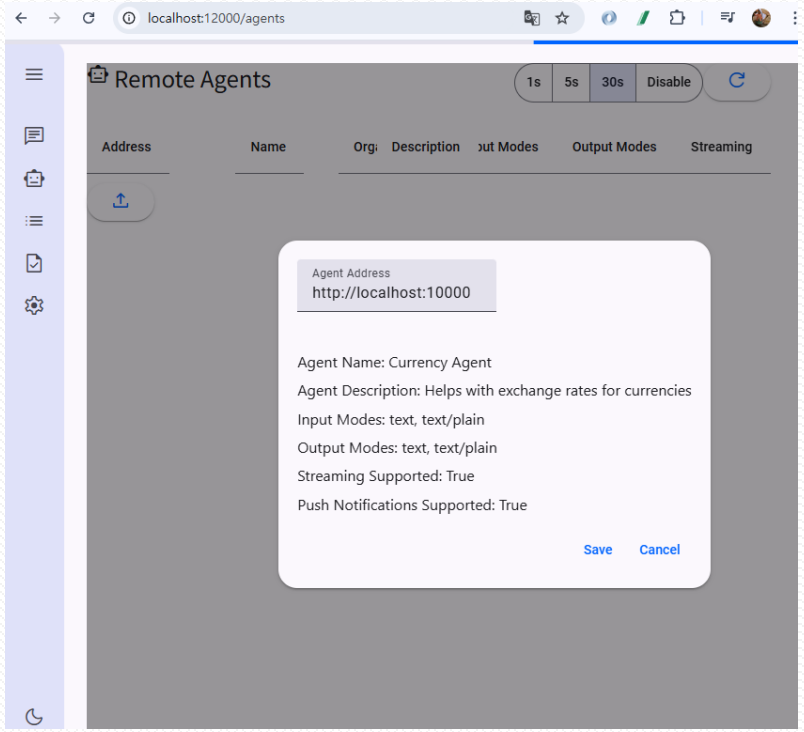
A2A Agent – skill, card, protocol

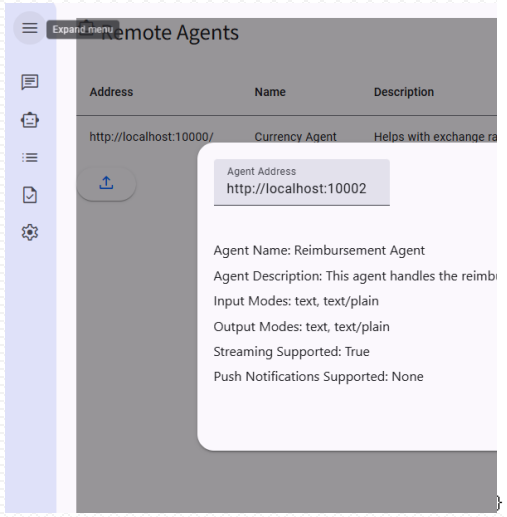
```
(a2a-sample-agent-langgraph) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\langgraph\app> vi __main__.py
```

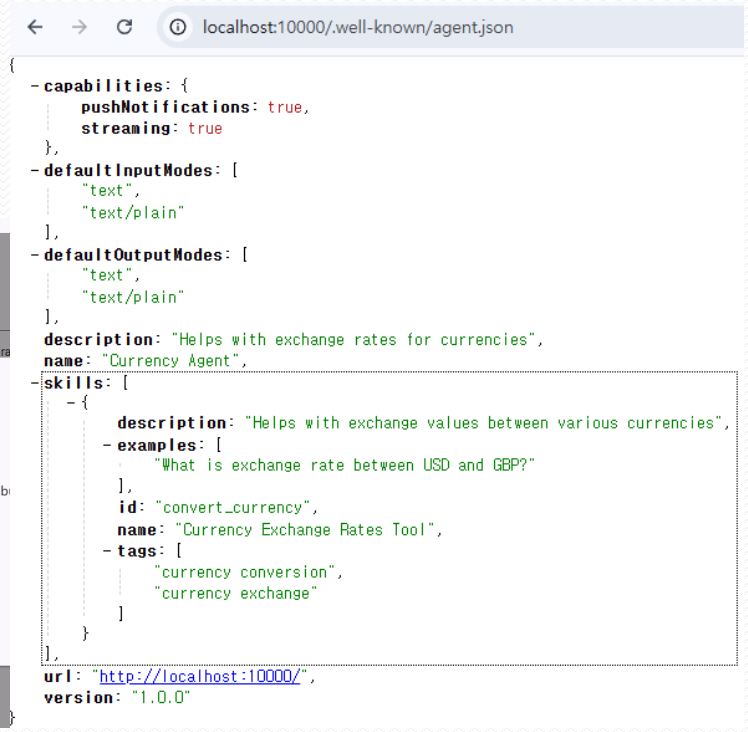
```python
...
from a2a.server.apps import A2AStarletteApplication
from a2a.server.request_handlers import DefaultRequestHandler
from a2a.server.tasks import InMemoryPushNotifier, InMemoryTaskStore
from a2a.types import (
    AgentCapabilities,
    AgentCard,
    AgentSkill,
)
...
@click.command()
@click.option('--host', 'host', default='localhost')
@click.option('--port', 'port', default=10000)
def main(host, port):
    """Starts the Currency Agent server."""
    try:
        if os.getenv('model_source', 'google') == 'google':
            if not os.getenv('GOOGLE_API_KEY'):
                raise MissingAPIKeyError(
                    'GOOGLE_API_KEY environment variable not set.'
                )
        else:
            if not os.getenv('TOOL_LLM_URL'):
                raise MissingAPIKeyError(
                    'TOOL_LLM_URL environment variable not set.'
                )
            if not os.getenv('TOOL_LLM_NAME'):
                raise MissingAPIKeyError(
                    'TOOL_LLM_NAME environment not variable not set.'
                )

        capabilities = AgentCapabilities(streaming=True, pushNotifications=True)
        skill = AgentSkill(
            id='convert_currency',
            name='Currency Exchange Rates Tool',
            description='Helps with exchange values between various currencies',
            tags=['currency conversion', 'currency exchange'],
            examples=['What is exchange rate between USD and GBP?'],
        )
```

```python
        agent_card = AgentCard(
            name='Currency Agent',
            description='Helps with exchange rates for currencies',
            url=f'http://{host}:{port}/',
            version='1.0.0',
            defaultInputModes=CurrencyAgent.SUPPORTED_CONTENT_TYPES,
            defaultOutputModes=CurrencyAgent.SUPPORTED_CONTENT_TYPES,
            capabilities=capabilities,
            skills=[skill],
        )

        # --8<-- [start:DefaultRequestHandler]
        httpx_client = httpx.AsyncClient()
        request_handler = DefaultRequestHandler(
            agent_executor=CurrencyAgentExecutor(),
            task_store=InMemoryTaskStore(),
            push_notifier=InMemoryPushNotifier(httpx_client),
        )

        server = A2AStarletteApplication(
            agent_card=agent_card, http_handler=request_handler
        )

        uvicorn.run(server.build(), host=host, port=port)
        # --8<-- [end:DefaultRequestHandler]
...

if __name__ == '__main__':
    main()
```

A2A Agent 구현 – agent 관리 페이지







Remote Agents

1s | 5s | 30s | Disable

Address	Name	Description	Organization	Input Modes	Output Modes	Streaming
http://localhost:10000/	Currency Agent	Helps with exchange rates for currencies		text, text/plain	text, text/plain	True
http://localhost:10002/	Reimbursement Agent	This agent handles the reimbursement process for the employees given the amount and purpose of the reimbursement.		text, text/plain	text, text/plain	True

A2A Agent 구현

1. 환전, 2. 비용처리 Agent 띄우기. 3. agent 관리 ui

1. 환전 Agent

```
(a2a-sample-agent-langgraph) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\langgraph> vi .env
(a2a-sample-agent-langgraph) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\langgraph> uv run app
```

```
GOOGLE_API_KEY=AIzaSyBnX5GmuWo2u2hXA8c_GGGGGGGGG
TOOL_LLM_URL="http://192.168.1.203:11434"
TOOL_LLM_NAME="mistral:latest"
```

2. 비용처리 Agent

2.1 소스수정필요

```
(adk_expense_reimbursement) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\adk_expense_reimbursement> vi agents.py
```

str | None으로 되어 있는 모든 매개변수를 Optional[str] 로 변경

```python
def create_request_form(
    date: Optional[str] = None,
    amount: Optional[str] = None,
    purpose: Optional[str] = None,
) -> dict[str, Any]:
...
def return_form(
    form_request: dict[str, Any],
    tool_context: ToolContext,
    instructions: Optional[str] = None,
) -> dict[str, Any]:
```

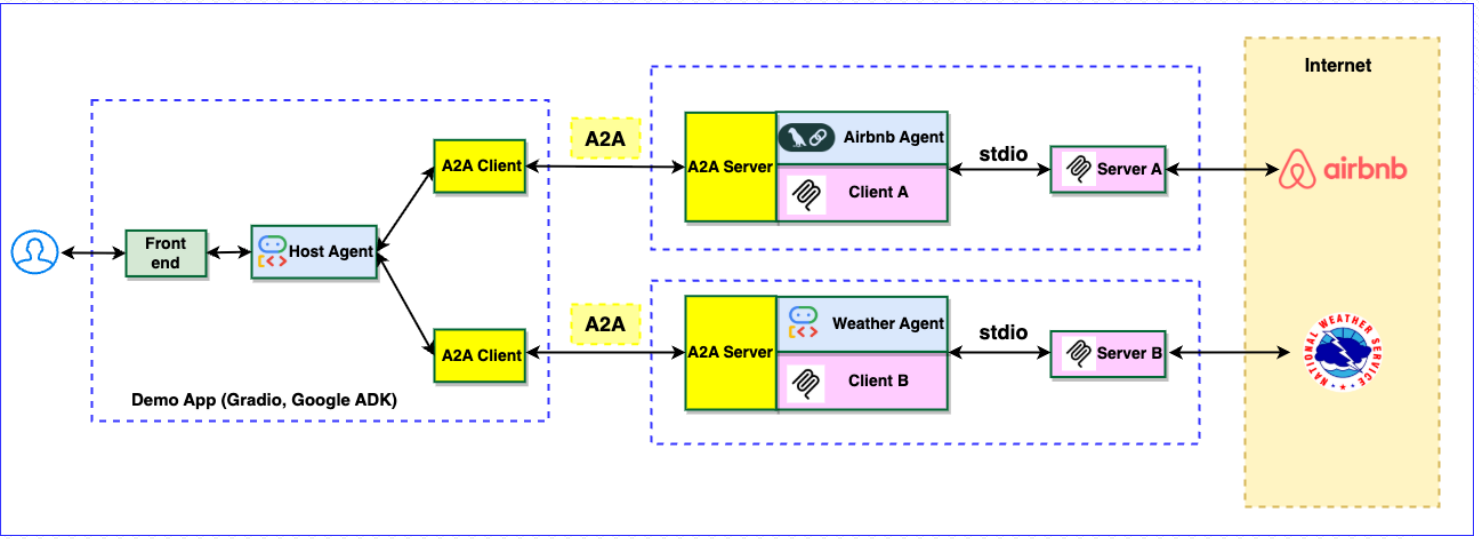
```
(adk_expense_reimbursement) PS D:\GITLAB\text2SQL\a2a-samples\samples\python\agents\adk_expense_reimbursement> uv run .
```

3. agent 관리 ui

```
(a2a-python-example-ui) PS D:\GITLAB\text2SQL\a2a-samples\demo\ui> uv run main.py
```

A2A Agent more over

https://github.com/a2aproject/a2a-samples/tree/main/samples/python/agents/airbnb_planner_multiagent



A2A protocol

A2A Host Agent

This assistant can help you to check weather and find airbnb accommodation

Chatbot

Tell me about weather in LA, CA

Here is the weather forecast for Los Angeles, CA:

- **This Afternoon:**
  - Temperature: 84°F
  - Wind: 5 to 10 mph SW
  - Short Forecast: Sunny
  - Detailed Forecast: Sunny, with a high near 84. Southwest wind 5 to 10 mph.
- **Tonight:**
  - Temperature: 65°F
  - Wind: 0 to 10 mph S
  - Short Forecast: Mostly Clear