Search ... ⌘ K

/ Examples / Document Grounding

Document Grounding

The Document Grounding is a module in the Orchestration Service.

The Document Grounding implements the Retrieval Augmented Generation (RAG) approach. It leverages the SAP Hana Vector Engine to retrieve info from relevant documents i.e., the "context" and uses them to generate more accurate responses.

Prerequisites

A vector knowledge base is set up according to the Prerequisites for the Grounding Module

The vector knowledge base can be created from

- a collection of documents in a sharepoint folder, or
- feeding text (chunks) directly via Vector API.

Another option is to use a web site which provides elastic search capabilities. At the moment, only the SAP Help website is supported.

Configuration of the Grounding Module

Provide the Orchestration Service URL and create a client for the Orchestration Service.

```
from gen_ai_hub.orchestration.service import OrchestrationService
from gen_ai_hub.orchestration.models.config import OrchestrationConfig
from gen_ai_hub.orchestration.models.document_grounding import (Groundin
                                                                Groundin
from gen_ai_hub.orchestration.models.llm import LLM
orchestration_service_url = "https://api.ai.<*** cluster-name ***>.aws.m
```

```
llm = LLM(
    name="gpt-4o-mini",
    parameters={
        'temperature': 0.0,
    }
)
```

Create the configuration

1. Define the prompts

2. Define the Grounding Module configuration

```
from gen_ai_hub.orchestration.models.message import SystemMessage, UserM
from gen_ai_hub.orchestration.models.template import Template, TemplateV

prompt = Template(messages=[
        SystemMessage("You are an expert on SAP Product features."),
        UserMessage("""Context: {{ ?grounding_response }}
                       Question: What are the features of {{ ?product }}
                    """),
    ])
```

Grounding configuration for searching SAP Help via elastic search

```
filters = [DocumentGroundingFilter(id="SAPHelp", data_repository_type="h

grounding_config = GroundingModule(type=GroundingType.DOCUMENT_GROUNDING
                                   config=DocumentGrounding(input_params
                                                            output_param
                                                            filters=filt
                                                            )
                                   )

config = OrchestrationConfig(template= prompt, llm=llm, grounding=ground

response = orchestration_service.run(config=config,
                                     template_values=[TemplateValue("pro

print(response.orchestration_result.choices[0].message.content)
```

Assume the documentation for custom product extension is vectorized and stored in a data repository.

```
filters = [DocumentGroundingFilter(id="vector",
                                   data_repositories=["<*** id ***>"],
                                   search_config=GroundingFilterSearch(m
                                   data_repository_type=DataRepositoryTy
                                   )]

grounding_config = GroundingModule(
            type=GroundingType.DOCUMENT_GROUNDING_SERVICE.value,
            config=DocumentGrounding(input_params=["product"], output_pa
                   )

config = OrchestrationConfig(template=prompt, llm=llm, grounding=groundi

response = orchestration_service.run(config=config,
                                     template_values=[TemplateValue("pro

print(response.orchestration_result.choices[0].message.content)
```

One can also show the retrieved context from the grounding module, which is added to the prompt for improving the response.

```
print(response.module_results.grounding.data['grounding_result'])
```

Data Masking of the retrieved context

The retrieved context can be masked in the same way as in the Orchestration Service to avoid passing sensitive information to the LLM.

```
from gen_ai_hub.orchestration.models.sap_data_privacy_integration import
from gen_ai_hub.orchestration.models.data_masking import DataMasking

data_masking = DataMasking(
    providers=[
        SAPDataPrivacyIntegration(
            method=MaskingMethod.ANONYMIZATION,
            entities=[ProfileEntity.SAP_IDS_INTERNAL],
            mask_grounding_input=True
        )
    ]
```

```
masking_config = OrchestrationConfig(template=prompt, llm=llm, grounding
response = orchestration_service.run(config=masking_config,
                                      template_values=[TemplateValue("pro

print(response.orchestration_result.choices[0].message.content)

print(response.module_results.grounding.data['grounding_result'])
```