
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

6. `동적 속성 지정` *(configurable_fields, configurable_alternatives)*

1) `런타임에 체인 내부 구성하기`

- `Chain` 호출시 → `다양한 옵션`을 `동적`으로 `설정`할 수 있는 방법
  - **a.** **`configurable_fields`**: 실행 가능한 객체의 **`특정 필드 구성 가능`**
  - **b.** **`configurable_alternatives`**: 런타임 중 설정할 수 있는 **`특정 실행 가능한 객체에 대한 대안을 나열`할 수 있음**

---

2) `configurable_fields`

- **`configurable_fields`** = 시스템의 설정 값 정의하는 필드

- **환경 설정**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                           # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')")
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

✦

- **동적 속성 지정**
  - ① **model_name**: `configurable_fields()` → `model_name`

- *참고: LangChain – gemini 공식 가이드*

```
from langchain.prompts import PromptTemplate
from langchain_core.runnables import ConfigurableField
from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv
import os

# LLM 초기화
# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 생성하기
base_model = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0,
)
```

- 기본 `LLM` 생성하기 (`base_model`) - `gemini-2.5.flash-lite`

```
E0000 00:00:1759990567.375083 1914336 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
base_model.invoke("대한민국의 수도는 어디야?").__dict__
```

- 속성 확인하기 (`0.9s`)

```
{'content': '대한민국의 수도는 **서울**입니다.',
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--05184c02-ed14-4d35-a85d-d113fd25dd39-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 9,
'output_tokens': 10,
'total_tokens': 19,
'input_token_details': {'cache_read': 0}}}
```

```
base_model_output = base_model.invoke("대한민국의 수도는 어디야?").__dict__
print(f"기본 모델: {base_model_output.get('response_metadata', {}).get('model_name')}")
```

- 기본 모델: gemini-2.5-flash-lite (`0.5s`)

```
# ConfigurableField 적용하기
# .configurable_fields()는 새로운 Runnable 객체를 반환함

configurable_model = base_model.configurable_fields(
    model= ConfigurableField(
        id="model_name",                                # model_name의 id 설정
        name="Version of gemini",                       # model_name의 이름 설정
        # model_name의 설명 설정하기
        description="Official model name of gemini. ex) gemini-2.5-flash-lite, 2.5-flash, 1.5-flash",
    )
)
```

```
```

```
# --- ConfigurableField를 이용한 동적 호출 ---

print("\n--- 동적 모델 호출 시작 ---")
# 기본 모델 (gemini-2.5-flash-lite)로 호출 (Config 설정 안 함)
configurable_model.invoke("대한민국의 수도는 어디야?").__dict__
```

- --- 동적 모델 호출 시작 --- (`0.6s`)

```
{'content': '대한민국의 수도는 **서울**입니다.',
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--269c6a20-e442-40a0-afdc-a5a83060d624-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 9,
'output_tokens': 10,
'total_tokens': 19,
'input_token_details': {'cache_read': 0}}}
```

- **`model.invoke()`** 호출 시 → **`config={"configurable": {"키": "값"}}`** 형식으로 **동적 지정 가능**

```
output_flash_lite = configurable_model.invoke("대한민국의 수도는 어디야?").response_metadata
print(f"기본 호출 모델: {output_flash_lite.get('model_name')}")
```

- 기본 호출 모델: gemini-2.5-flash-lite (`0.6s`)

```
# config를 사용하여 gemini-2.5-flash로 바꿔서 호출해보기
configurable_model.invoke(
    "대한민국의 수도는 어디야?",
    config={"configurable": {"model_name": "gemini-2.5-flash"}},
)
```

- `config`를 사용해 모델 바꿔서 호출 (`1.1s`)

```
E0000 00:00:1759993034.490020 1914336 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
AIMessage(content='대한민국의 수도는 **서울**입니다.', additional_kwargs={}, response_metadata={'prompt_feedback': {'blo
```

```
output_flash2 = configurable_model.invoke(
    "대한민국의 수도는 어디야?",
    config={"configurable": {"model_name": "gemini-2.5-flash"}},
).response_metadata

print(f"Configured 호출 모델: {output_flash2.get('model_name')}")
print("\n✅ ConfigurableField를 사용하여 모델이 성공적으로 변경되었습니다.")
```

- Configured 호출 모델: gemini-2.5-flash (`1.6s`)
- ✅ ConfigurableField를 사용하여 모델이 성공적으로 변경되었습니다.

```
# config를 사용하여 gemini-2.5-pro로 바꿔서 호출해보기

output_pro = configurable_model.invoke(
    "대한민국의 수도는 어디야?",
    config={"configurable": {"model_name": "gemini-2.5-pro"}},
).response_metadata
```

```
print(f"Configured 호출 모델 (Pro):   {output_pro.get('model_name')}")
print("\n✅ ConfigurableField를 사용하여 모델이 3번째 변경을 성공했습니다.")
```

- Configured 호출 모델 (Pro): gemini-2.5-pro (`13.6s`)
- ✅ ConfigurableField를 사용하여 모델이 3번째 변경을 성공했습니다.

*

```
E0000 00:00:1759993379.868857 1914336 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untru
```

---

*

- ② **`model`** 객체의 **`with_config()`** → `configurable` 매개변수 설정 가능
  - *동작 방식 동일*

```
# with_config():  gemini-2.5-flash → gemini-2.5-flash-lite로 바꿔서 호출해보기

configurable_model.with_config(
    configurable={"model_name":"gemini-2.5-flash-lite"}).invoke(
        "대한민국의 수도는 어디야?"
        ).__dict__
```

- `with_config()` 로 모델 바꾸기: * `gemini-2.5-flash` → `gemini-2.5-flash-lite` (`1.2s`)

```
{'content': '대한민국의 수도는 **서울**입니다.',
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--b451072a-ae31-4c37-bb5b-16539ea92a93-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 9,
'output_tokens': 10,
'total_tokens': 19,
'input_token_details': {'cache_read': 0}}}
```

```
output_flash_lite2 = configurable_model.with_config(
    configurable={"model_name":"gemini-2.5-flash-lite"}).invoke(
        "대한민국의 수도는 어디야?"
        ).response_metadata

print(f"Configured 호출 모델: {output_flash_lite2.get('model_name')}")
print(f"\n✅ with_config()를 사용하여 {output_flash_lite2.get('model_name')}모델로 성공적으로 변경되었습니다.")
```

- Configured 호출 모델: gemini-2.5-flash-lite (`0.7s`)
- ✅ with_config()를 사용하여 gemini-2.5-flash-lite모델로 성공적으로 변경되었습니다.

*

```
E0000 00:00:1759994182.623158 1914336 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untru
```

---

*

- ③ **`chain`** 의 일부로도 함수 사용 가능

```
# 템플릿에서 프롬프트 템플릿 생성하기
prompt = PromptTemplate.from_template("{x} 보다 큰 위의 난수를 선택합니다.")

# 프롬프트와 모델을 연결하여 체인 생성하기
chain = (
    prompt | configurable_model      # 프롬프트의 출력이 모델의 입력으로 전달됨
)
```

```
# 체인 호출 → 입력 변수 "x"에 0 전달하기

chain.invoke({"x": 0}).__dict__
```

- 체인의 일부로 사용해보기 (`gemini-2.5-flash-lite`) - (`1.8s`)

```
{'content': '네, 0보다 큰 임의의 난수를 선택해 드리겠습니다.\n\n어떤 범위의 난수를 원하시나요? 예를 들어,\n\n*   **특정 범위 내의 난수:**
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--e917dee9-33b0-4576-81bb-427d23fbdb59-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 10,
'output_tokens': 200,
'total_tokens': 210,
'input_token_details': {'cache_read': 0}}}
```

```
chain_response1 = chain.invoke({"x": 0}).response_metadata
print(f"Configured 호출 모델: {chain_response1.get('model_name')}")
print(f"\n✅ with_config()를 사용하여 {chain_response1.get('model_name')}모델로 성공적으로 변경되었습니다.")
```

```
# 체인 호출 시 설정 지정 → 체인 호출 가능

chain.with_config(
    configurable={"model_name": "gemini-2.5-flash"}
    ).invoke({"x": 0}).__dict__
```

- `chain` 호출에서 **`with_config`** 설정으로 모델 변경해보기 (*`gemini-2.5-flash-lite`* → *`gemini-2.5-flash`*) - (`9.7s`)

```
{'content': '네, 0보다 큰 난수를 하나 선택해 드릴게요.\n\n예를 들어, **5.731** 입니다.\n\n저는 진정한 의미의 난수 생성기가 아니지만, 요청
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--d9d8d41c-c34f-475a-a4cb-8c0c4dfcae63-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 10,
'output_tokens': 1253,
'total_tokens': 1263,
'input_token_details': {'cache_read': 0},
'output_token_details': {'reasoning': 1170}}}
```

```
chain_response2 = chain.with_config(
    configurable={"model_name": "gemini-2.5-flash"}
    ).invoke({"x": 0}).response_metadata
```

```
print(f"Configured 호출 모델: {chain_response2.get('model_name')}")
print(f"\n✅ with_config()를 사용하여 {chain_response2.get('model_name')}모델로 성공적으로 변경되었습니다.")
```

- Configured 호출 모델: gemini-2.5-flash (`8.5s`)
- ✅ with_config()를 사용하여 gemini-2.5-flash모델로 성공적으로 변경되었습니다.

---

3) `HubRunnable` : `LangChain Hub의 설정 변경`

- **`HubRunnable`** → `Hub`에 등록된 프롬프트의 전환을 용이하게 해줌

```
from langchain.runnables.hub import HubRunnable

prompt = HubRunnable("teddynote/rag-prompt-korean").configurable_fields(
    owner_repo_commit=ConfigurableField(                # 소유자 저장소 커밋을 설정하는 ConfigurableField
        id="hub_commit",                                # 필드의 ID
        name="Hub Commit",                              # 필드의 이름
        description="Korean RAG prompt by teddynote",   # 필드에 대한 설명
    )
)
prompt
```

- 교재에서 사용된 프롬프트 - *`Hub`에 등록된 저자의 프롬프트*

```
RunnableConfigurableFields(default=HubRunnable(bound=ChatPromptTemplate(input_variables=['context', 'question'],
```

- **`prompt.invoke()`** 호출
  - 별도의 *`with_config`* 설정 불필요
  - 처음 설정한 **`"rlm/rag-prompt"`** `hub`에 등록된 프롬프트를 `pull`하여 가져옴

```
# prompt 객체의 invoke 메서드 호출 → "question"과 "context" 매개변수 전달하기

prompt.invoke({"question": "Hello", "context": "World"}).messages
```

- `prompt.invoke()`로 **`question`**, **`context`** 전달하기

```
[SystemMessage(content="당신은 질문-답변(Question-Answering)을 수행하는 친절한 AI 어시스턴트입니다. 당신의 임무는 주어진 문맥(contex
HumanMessage(content='#Question: \nHello \n\n#Context: \nWorld \n\n#Answer:', additional_kwargs={}, response_meta
```

```
# hub_commit을 teddynote/simple-summary-korean으로 설정함

prompt.with_config(
    configurable={"hub_commit": "teddynote/simple-summary-korean"}
).invoke({"context": "Hello"})
```

- **`with_config()`** 에서 **`configurable`** **매개변수 속** **`hub_commit`** **을** **`"teddynote/simple-summary-korean"`** **로 설정**
  - `content` = 한국어 요약으로 생성됨

```
ChatPromptValue(messages=[HumanMessage(content='주어진 내용을 바탕으로 다음 문장을 요약하세요. 답변은 반드시 한글로 작성하세요\
```

---

4) `Configurable Alternatives` : *Runnable 객체 자체의 대안 설정*

- 런타임에 설정할 수 있는 `Runnable`에 대안 대안 구성하기

- **① 구성 가능한 대안들**
  - *ChatAnthropic* 구성 가능한 언어 모델 = 다양한 작업, 컨텍스트에 적용할 수 있는 유연성 제공
    - 동적으로 설정 (Config) 값을 변경하기 위해 **모델에 설정하는 파라미터 = ConfigurableField 객체로 설정**
      - **model**: 사용할 기본 언어 모델
      - **temperature**:
        - **0~1 사이의 값 / 샘플링의 무작위성 제어**
        - 값이 낮을수록 더 결정적이고 반복적인 출력이 생성됨
        - 값이 높을수록 더 다양하고 창의적인 출력이 생성됨

- **② LLM 객체의 대안** *(alternatives)* **설정 방법**
  - 참고: LLM 모델 사용 위해서는 각 회사의 API KEY 발급받아 설정해야 함
  - *.env* 파일에 설정 → 환경변수로 설정하기
    - *GROQ 모델 사용해보기*
    - *참고: LangChain 공식 도큐먼트 - GROQ*

```
from langchain.prompts import PromptTemplate
from langchain_core.runnables import ConfigurableField
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_groq import ChatGroq                    # Groq 모델 사용을 위해 변경
from dotenv import load_dotenv
import os
import warnings

# 경고 메시지 무시
warnings.filterwarnings("ignore")

# --- LLM 초기화 및 기본 테스트 ---

# API 키 확인 (GROQ_API_KEY로 변경)
if not os.getenv("GROQ_API_KEY"):
    # 환경 변수에서 로드하거나 사용자에게 입력 요청
    api_key = os.getenv("GROQ_API_KEY")

    if not api_key:
        print("GROQ_API_KEY 환경 변수를 설정해주세요.")
```

```
# ChatGroq 사용

llm = ChatGroq(
    model="llama-3.1-8b-instant",
    temperature=0).configurable_alternatives(
        # ConfigurableField에 id 부여
        ConfigurableField(id="llm"),        # 최종 실행 가능한 객체를 구성할 때, 이 id를 사용하여 이 필드를 구성할 수 있음
        default_key="groq",                 # 기본 키 설정 → 초기화된 기본 LLM (groq)이 사용됨

        # 새 옵션 추가 `ChatGoogleGenerativeAI()`와 동일
        gemini=ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite"),   # gemini 이름, `2.5-flash-lite 모델`
        gemini2=ChatGoogleGenerativeAI(model="gemini-2.5-flash"),       # gemini2 이름, `2.5-flash 모델`
        # ,...                                                          # 여기에 더 많은 구성 옵션을 추가할 수 있음
)

prompt = PromptTemplate.from_template("{topic} 에 대해 간단히 설명해주세요.")

chain = prompt | llm
```

- **chain.invoke()** 메서드 → **기본 LLM 활용한 체인 호출해보기**

```
chain.invoke({"topic": "추석"}).__dict__
```

- 기본 LLM 호출해보기 (Groq) - (0.7s)

```
{'content': '추석은 한국의 대표적인 명절 중 하나입니다. 추석은 한 해의 마지막 달인 음력 9월 9일을 기념하는 명절로, 가족과 함께 모여 음식을 먹고
'additional_kwargs': {},
'response_metadata': {'token_usage': {'completion_tokens': 252,
'prompt_tokens': 46,
'total_tokens': 298,
'completion_time': 0.449131018,
'prompt_time': 0.002133234,
'queue_time': 0.045223673,
```

```
'total_time': 0.451264252},
'model_name': 'llama-3.1-8b-instant',
'system_fingerprint': 'fp_7b3cfae3af',
'service_tier': 'on_demand',
'finish_reason': 'stop',
'logprobs': None},
'type': 'ai',
'name': None,
'id': 'run--b9283296-3f89-438a-b938-36c0c26c6797-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 46,
'output_tokens': 252,
'total_tokens': 298}}
```

```
# Groq을 기본으로 호출

response1 = chain.invoke({"topic": "추석"}).response_metadata
```

- **response1** = `chain.invoke({"topic": "추석"}).response_metadata` - (`0.8s`)

```
# <class 'dict'>

{'token_usage': {'completion_tokens': 252,
'prompt_tokens': 46,
'total_tokens': 298,
'completion_time': 0.514032268,
'prompt_time': 0.002759986,
'queue_time': 0.045235344,
'total_time': 0.516792254},
'model_name': 'llama-3.1-8b-instant',
'system_fingerprint': 'fp_7b3cfae3af',
'service_tier': 'on_demand',
'finish_reason': 'stop',
'logprobs': None}
```

```
response2 = chain.invoke({"topic":"추석"}).content
```

```
print(type(response2))
```

```
print(response2)
```

- **response2** = `chain.invoke({"topic":"추석"}).content` - (`0.8s`)
- `response2` : type <class 'str'>

```
추석은 한국의 대표적인 명절 중 하나입니다. 추석은 한 해의 마지막 달인 음력 9월 9일을 기념하는 명절로, 가족과 함께 모여 음식을 먹고 즐기며 추억을 만

추석의 주요 행사로는 다음과 같습니다.

1. 가족 모임: 가족과 함께 모여 음식을 먹고 즐기며 추억을 만들기 위해 많은 사람들이 집으로 돌아갑니다.
2. 추석 음식: 추석에는 다양한 음식이 있습니다. 대표적인 음식으로는 단무지, 김치, 부침개, 떡국, 송편 등이 있습니다.
3. 추석 선물: 추석에는 선물을 주고받는 풍습이 있습니다. 일반적으로는 부모님에게 선물을 주는 것이 관행입니다.
4. 추석 여행: 추석에는 많은 사람들이 여행을 떠나 가족과 함께 모여 즐기기 위해 집으로 돌아갑니다.

추석은 한국의 대표적인 명절 중 하나로, 가족과 함께 모여 즐기며 추억을 만들기 위해 많은 사람들이 집으로 돌아갑니다.
```

```
print(f"✅ {response1.get('model_name')} 모델로 답변을 생성하였습니다.")
print(f"\n기본 호출로 생성한 내용: {response2}")
```

- 기본 `LLM` 호출 결과 정리

```
✅ llama-3.1-8b-instant 모델로 답변을 생성하였습니다.

기본 호출로 생성한 내용: 추석은 한국의 대표적인 명절 중 하나입니다. 추석은 한 해의 마지막 달인 음력 9월 9일을 기념하는 명절로, 가족과 함께 모여 음

추석의 주요 행사로는 다음과 같습니다.

1. 가족 모임: 가족과 함께 모여 음식을 먹고 즐기며 추억을 만들기 위해 많은 사람들이 집으로 돌아갑니다.
2. 추석 음식: 추석에는 다양한 음식이 있습니다. 대표적인 음식으로는 단무지, 김치, 부침개, 떡국, 송편 등이 있습니다.
3. 추석 선물: 추석에는 선물을 주고받는 풍습이 있습니다. 일반적으로는 부모님에게 선물을 주는 것이 관행입니다.
4. 추석 여행: 추석에는 많은 사람들이 여행을 떠나 가족과 함께 모여 즐기기 위해 집으로 돌아갑니다.

추석은 한국의 대표적인 명절 중 하나로, 가족과 함께 모여 즐기며 추억을 만들기 위해 많은 사람들이 집으로 돌아갑니다.
```

- **`chain.with_config(configurable={"llm": "모델"})` → `LLM` 바꾸기**

```
# 체인 설정 변경 → 호출해보기

chain.with_config(configurable={"llm": "gemini"}).invoke({"topic": "추석"}).__dict__
```

- **`chain.with_config(configurable={"llm":"모델"})` → `LLM` 바꾸기** - (`2.9s`)

```
{'content': '추석은 한국의 가장 큰 명절 중 하나로, 음력 8월 15일입니다. 가을의 풍요로움을 감사하고 조상님께 차례를 지내며 가족들이 모여 송편을
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--37071df1-926e-4c7d-87aa-d9d23b7eacac-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 10,
'output_tokens': 319,
'total_tokens': 329,
'input_token_details': {'cache_read': 0}}}
```

```
response_gemini=chain.with_config(configurable={"llm": "gemini"}).invoke({"topic": "추석"}).response_metadata
```

```
print(response_gemini)
```

- `chain.with_config(configurable={"llm": "gemini"}).invoke({"topic": "추석"}).response_metadata` - (`2.4s`)

```
# <class 'dict'>

{'prompt_feedback': {'block_reason': 0, 'safety_ratings': []}, 'finish_reason': 'STOP', 'model_name': 'gemini-2.5
```

```
print(f"✅ {response_gemini.get('model_name')} 모델로 답변을 생성하였습니다.")
```

- ✅ gemini-2.5-flash-lite 모델로 답변을 생성하였습니다.

```
response_gemini2=chain.with_config(configurable={"llm": "gemini"}).invoke({"topic": "추석"}).content
```

```
print(response_gemini2)
```

- `chain.with_config(configurable={"llm": "gemini"}).invoke({"topic": "추석"}).content` - (`2.6s`)

```
## 추석: 풍요로운 한가위, 감사와 나눔의 명절

추석은 우리나라의 가장 큰 명절 중 하나로, 음력 8월 15일입니다. 가을의 풍요로움을 만끽하며 조상님께 감사드리고 가족, 친지들과 함께 즐거운 시간을 보내

**추석의 의미:**

* **가을 추수 감사:** 풍성한 가을걷이를 감사하며 조상님께 햇곡식과 햇과일을 올리는 제사를 지냅니다.
* **보름달:** 가장 크고 밝은 보름달을 보며 소원을 빌고 풍요를 기원합니다.
* **가족 공동체:** 오랜만에 만나는 가족, 친지들과 함께 음식을 나누고 즐거운 시간을 보내며 유대감을 다집니다.

**추석의 풍습:**

* **차례:** 조상님께 감사하는 마음으로 음식을 올리는 제사입니다.
* **성묘:** 조상님의 묘를 찾아 정성껏 돌보고 예를 올립니다.
* **송편:** 햅쌀로 빚은 반달 모양의 떡으로, 추석의 대표적인 음식입니다.
* **강강술래:** 여성들이 손을 잡고 둥글게 돌며 노래하고 춤추는 전통 놀이입니다.
* **씨름, 널뛰기 등 민속놀이:** 온 가족이 함께 즐기는 다양한 전통 놀이를 합니다.

**추석은 단순히 쉬는 날이 아니라, 우리 고유의 전통과 문화를 되새기고, 가족의 소중함을 느끼며, 감사와 나눔의 정신을 실천하는 의미 깊은 날입니다.**

간단히 말하자면, 추석은 **가을의 풍요로움을 감사하며 가족과 함께 즐기는 한국의 가장 큰 명절**이라고 할 수 있습니다.
```

- **`체인 설정 변경` → 사용할 `언어 모델`을 다시 변경하기**

```
# 체인 설정 변경 → 또 다른 모델로 호출해보기 (`gemini2` = `gemini-2.5-flash`)

chain.with_config(configurable={"llm": "gemini2"}).invoke(
    {"topic": "한글날"}
).__dict__
```

- **`chain.with_config(configurable={"llm": "gemini2"}).invoke()`** - (`8.3s`)

```
{'content': "한글날은 매년 **10월 9일**로, **한글의 창제와 반포를 기념하고 한글의 우수성을 기리는 대한민국의 공휴일**입니다.\n\n간단히 설명하
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--f6662afb-855f-4cd8-bb97-f5166a4b1e59-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 11,
'output_tokens': 1147,
'total_tokens': 1158,
'input_token_details': {'cache_read': 0},
'output_token_details': {'reasoning': 896}}}
```

```
chain.with_config(configurable={"llm": "gemini2"}).invoke(
    {"topic": "한글날"}
).response_metadata
```

- **`chain.with_config(configurable={"llm": "gemini2"}).invoke()`** 의 `response_metadata` - (`11.6s`)

```
# <class 'dict'>

{'prompt_feedback': {'block_reason': 0, 'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash',
'safety_ratings': []}
```

```
response_flash = chain.with_config(configurable={"llm": "gemini2"}).invoke(
    {"topic": "한글날"}
).response_metadata

print(f"✅ {response_flash.get('model_name')} 모델로 답변을 생성하였습니다.")
```

- ✅ gemini-2.5-flash 모델로 답변을 생성하였습니다. (`12.0s`)

```
response_flash2=chain.with_config(configurable={"llm": "gemini2"}).invoke(
    {"topic": "한글날"}
).content

print(response_flash2)
```

- **`gemini2`** = *gemini-2.5-flash* 로 생성한 답변 - (`9.7s`)

```
한글날은 **우리나라의 고유한 글자인 '한글'의 창제와 반포를 기념하고, 그 우수성을 기리기 위한 날**입니다.

*   **언제:** 매년 **10월 9일**
*   **무엇을 기념하는가:** 조선 시대 **세종대왕**께서 백성들이 어려운 한자 대신 **쉽게 배우고 쓸 수 있는 글자를 만들고자 하여 한글을 창제하고 서
*   **의미:** 한글은 배우기 쉽고 과학적인 글자로 평가받으며, 한글날은 세종대왕의 위대한 업적을 기리고 우리가 사용하는 한글의 소중함과 아름다움을 다

쉽게 말해, **세종대왕님이 우리 백성들이 글을 쉽게 읽고 쓰도록 과학적인 글자 '한글'을 만드신 것을 기념하는 날**이라고 할 수 있습니다.
```

- **③ 프롬프트의 대안 설정 방법**
  - *프롬프트 ≒ 이전의 `LLM` 대안 설정 방법과 유사한 작업 수행 가능*

```
# ChatGroq 사용 → 초기화, temperature = 0으로 설정

llm = ChatGroq(
    model="llama-3.1-8b-instant",
    temperature=0)

# 프롬프트에 configurable_altervatives( ConfigurableField(id="prompt") ) 설정하기
prompt = PromptTemplate.from_template(
    "{country} 의 수도는 어디야?"                                  # 기본 프롬프트 템플릿
).configurable_alternatives(
    ConfigurableField(id="prompt"),                              # 필드에 id 부여하기
    default_key="capital",                                       # 기본 키 설정하기

    # 새로운 옵션 추가하기
    area=PromptTemplate.from_template("{country} 의 면적은 얼마야?"),               # 'area'
    population=PromptTemplate.from_template("{country} 의 인구는 얼마야?"),         # 'population'
    eng=PromptTemplate.from_template("{input} 을 영어로 번역해주세요."),             # 'eng'
    # ,...                                                       # 더 많은 구성 옵션 추가 가능
)

# 프롬프트와 언어 모델 연결 → 체인 생성하기
chain = prompt | llm
```

- 기본 프롬프트 입력됨
  - *아무런 설정 변경이 없을 경우*

```
# query_1
# config 변경 없이 체인 호출하기
chain.invoke({"country": "대한민국"})
```

- 기본 프롬프트로 생성된 답변 - (`0.4s`)

```
AIMessage(content='대한민국의 수도는 서울입니다.', additional_kwargs={}, response_metadata={'token_usage': {'completion_
```

- **`with_config()`** → 다른 프롬프트 호출하기

```
# query_2
# with_config() → 체인 설정 변경 → 호출해보기
chain.with_config(configurable={"prompt": "area"}).invoke({"country": "대한민국"})
```

- **`with_config()`** *(지역)* - (`0.5s`)

```
AIMessage(content='대한민국의 면적은 약 100,363 km²입니다.', additional_kwargs={}, response_metadata={'token_usage': {'c
```

```
# query_3
# with_config() → 체인 설정 변경 → 호출해보기
chain.with_config(configurable={"prompt": "population"}).invoke({"country": "대한민국"})
```

- **`with_config()`** *(인구)* - (`0.6s`)

```
AIMessage(content='2022년 12월 31일 기준으로, 대한민국의 인구는 약 51,811,177 명입니다.', additional_kwargs={}, response_met
```

- **`eng`** 프롬프트 사용 → 번역 요청해보기
- 입력 변수 = **`input`**

```
# query_4
# with_config() → 체인 설정 변경 → 호출해보기
chain.with_config(configurable={"prompt": "eng"}).invoke({"input": "사과는 맛있어!"})
```

- **`with_config()`** *(input으로 입력값을 새로 넣기)* - (`0.3s`)

```
AIMessage(content='사과는 맛있어! 를 영어로 번역하면 "Apples are delicious!" 이라고 합니다.', additional_kwargs={}, response
```

- **④ 프롬프트 & LLM 모두 변경하기**

```
# LLM config 설정 및 초기화

llm = ChatGroq(
    model="llama-3.1-8b-instant",
    temperature=0).configurable_alternatives(
        # ConfigurableField에 id 부여
        ConfigurableField(id="llm"),        # 최종 실행 가능한 객체를 구성할 때, 이 id를 사용하여 이 필드를 구성할 수 있음
        default_key="groq",                 # 기본 키 설정 → 초기화된 기본 LLM (groq)이 사용됨

        # 새 옵션 추가 `ChatGoogleGenerativeAI()`와 동일
        gemini=ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite"),   # gemini 이름, `2.5-flash-lite 모델`
        gemini2=ChatGoogleGenerativeAI(model="gemini-2.5-flash"),       # gemini2 이름, `2.5-flash 모델`
        # ,...                                                          # 여기에 더 많은 구성 옵션을 추가할 수 있음
)


# 프롬프트에 configurable_altervatives( ConfigurableField(id="prompt") ) 설정하기
prompt = PromptTemplate.from_template(
    "{company} 에 대해서 20자 이내로 설명해 줘."                    # 기본 프롬프트 템플릿
).configurable_alternatives(
    ConfigurableField(id="prompt"),                                 # 필드에 id 부여하기
    default_key="capital",                                          # 기본 키 설정하기
```

```
    # 새로운 옵션 추가하기
    founder=PromptTemplate.from_template("{company} 의 창립자는 누구인가요?"),           # 'founder'
    competitor=PromptTemplate.from_template("{company} 의 경쟁사는 누구인가요?"),        # 'competitor'
    eng=PromptTemplate.from_template("{input} 을 영어로 번역해주세요."),                 # 'eng'
    # ,...                                                                  # 더 많은 구성 옵션 추가 가능
)

# 프롬프트와 언어 모델 연결 → 체인 생성하기
chain = prompt | llm
```

```
# new_query_1
# with_config로 설정 값 지정 → 구성
chain.with_config(configurable={
    "prompt": "founder",
    "llm": "gemini"}).invoke(              # gemini 지정
    {"company": "애플"}                    # 사용자가 제공한 회사에 대한 처리 요청하기
).__dict__
```

- ***gemini*** / ***사용자가 입력한 company에 대한 답변 생성*** - (1.1s)

```
{'content': '애플의 창립자는 **스티브 잡스(Steve Jobs), 스티브 워즈니악(Steve Wozniak), 로널드 웨인(Ronald Wayne)** 세 명입니다.\
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--3c45637d-a6a8-4953-8c89-6830f2ae5cc1-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 11,
'output_tokens': 62,
'total_tokens': 73,
'input_token_details': {'cache_read': 0}}}
```

```
# new_query_2
# 하나만 구성하려는 경우
chain.with_config(configurable={"llm": "groq"}).invoke(          # 기본 llm 사용
    {"company": "애플"}
).__dict__
```

- ***groq*** / ***하나만 구성하여 답변 생성*** - (0.3s)

```
{'content': '애플은 미국의 전자 제품 회사로, 스마트폰, 컴퓨터, 음악 플레이어 등 다양한 제품을 생산합니다.',
'additional_kwargs': {},
'response_metadata': {'token_usage': {'completion_tokens': 31,
'prompt_tokens': 52,
'total_tokens': 83,
'completion_time': 0.05269561,
'prompt_time': 0.002395054,
'queue_time': 0.045412766,
'total_time': 0.055090664},
'model_name': 'llama-3.1-8b-instant',
'system_fingerprint': 'fp_e32974efee',
'service_tier': 'on_demand',
'finish_reason': 'stop',
'logprobs': None},
'type': 'ai',
'name': None,
'id': 'run--4f8bc6f7-84b1-4659-b9be-5849f41ffae6-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 52,
'output_tokens': 31,
```

```
'total_tokens': 83}}
```

```
# new_query_3
# 하나만 구성하려는 경우
chain.with_config(configurable={"prompt": "competitor"}).invoke(
    {"company": "애플"}
).__dict__
```

- ***groq*** / ***하나만 구성하여 답변 생성*** - (1.1s)
  - *llm을 따로 지정하지 않았으므로 기본 llm = Groq으로 지정되어 사용되었음*
  - *metadata 속 id가 Groq과 같음을 확인할 수 있음*

```
{'content': '애플은 다양한 제품과 서비스를 제공하는 세계적인 기술 기업입니다. 따라서 애플의 경쟁사는 여러 분야에 걸쳐 있습니다. 여기 몇 가
'additional_kwargs': {},
'response_metadata': {'token_usage': {'completion_tokens': 391,
'prompt_tokens': 47,
'total_tokens': 438,
'completion_time': 0.66336699,
'prompt_time': 0.006140743,
'queue_time': 0.049014037,
'total_time': 0.669507733},
'model_name': 'llama-3.1-8b-instant',
'system_fingerprint': 'fp_7b3cfae3af',
'service_tier': 'on_demand',
'finish_reason': 'stop',
'logprobs': None},
'type': 'ai',
'name': None,
'id': 'run--efa9bdd8-7901-457e-8de9-2b2e92f25d2e-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 47,
'output_tokens': 391,
'total_tokens': 438}}
```

```
# new_query_4
# 하나만 구성하려는 경우
chain.invoke({"company": "애플"}).__dict__
```

- ***하나만 구성하여 답변 생성*** - (0.4s)

```
{'content': '애플은 미국의 전자 제품 회사로, 스마트폰, 컴퓨터, 음악 플레이어 등 다양한 제품을 생산합니다.',
'additional_kwargs': {},
'response_metadata': {'token_usage': {'completion_tokens': 31,
'prompt_tokens': 52,
'total_tokens': 83,
'completion_time': 0.051515506,
'prompt_time': 0.002432587,
'queue_time': 0.047708393,
'total_time': 0.053948093},
'model_name': 'llama-3.1-8b-instant',
'system_fingerprint': 'fp_ab04adca7d',
'service_tier': 'on_demand',
'finish_reason': 'stop',
'logprobs': None},
'type': 'ai',
'name': None,
'id': 'run--21483bb0-74d4-4e5a-a574-38b4134bb481-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 52,
'output_tokens': 31,
```

```
'total_tokens': 83}}
```

---

5) `설정 저장`

- 구성된 체인 = **`별도의 객체로 쉽게 저장 가능`**
  - *예시:* **`특정 작업`을 위해 `사용자 정의`된 `체인` 구성 → `재사용 가능한 객체`로 저장** → 향후 유사한 작업에서 손쉽게 활용 가능

- **`① with_config로 설정을 변경하여 생성한 체인 = 별도의 변수에 저장하기`**
  - `llm` = `gemini` = `gemini-2.5-flash-lite`
  - `prompt` = `"{company} 에 대해서 20자 이내로 설명해줘."`, {`"prompt"`:`"competitor"`}

```
# with_config로 설정을 변경하여 생성한 체인 = 별도의 변수에 저장하기

gemini_competitor_chain = chain.with_config(
    configurable={"llm": "gemini", "prompt": "competitor"}
)
```

```
# 체인 호출해보기

gemini_competitor_chain.invoke({"company": "애플"}).__dict__
```

- **`gemini_competitor_chain.invoke()`** = *gemini-2.5-flash-lite* 로 생성한 답변 - (`4.3s`)

```
{'content': '애플의 경쟁사는 매우 다양하며, 어떤 제품 또는 서비스 분야를 기준으로 보느냐에 따라 달라집니다. 크게 다음과 같이 분류할 수 있습니다.
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash-lite',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--cefd0b41-b56e-4633-b3fd-b45207fcc404-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 10,
'output_tokens': 828,
'total_tokens': 838,
'input_token_details': {'cache_read': 0}}}
```

```
response_com1=gemini_competitor_chain.invoke({"company": "애플"}).response_metadata
print(f"✅ {response_com1.get('model_name')} 모델로 답변을 생성하였습니다.")
```

- ✅ gemini-2.5-flash-lite 모델로 답변을 생성하였습니다. (`4.1s`)

```
response_com2=gemini_competitor_chain.invoke({"company": "애플"}).content
print(response_com2)
print(f"\n ⭕ {response_com1.get('model_name')} 모델로 체인 저장 완료 및 답변을 완료하였습니다.")
```

- 답변 완료 (`3.9s`)

```
애플은 매우 다양한 분야에서 경쟁사를 가지고 있으며, 그 경쟁 상대들은 특정 제품이나 서비스에 따라 달라집니다. 주요 경쟁사들을 몇 가지 범주로 나누어 설

**1. 스마트폰 및 태블릿:**

*   **삼성전자:** 안드로이드 스마트폰 시장의 가장 강력한 경쟁자이며, 갤럭시 시리즈는 아이폰의 직접적인 대항마입니다. 태블릿 시장에서도 갤럭시 탭으로
*   **구글:** 픽셀 스마트폰을 통해 하드웨어와 소프트웨어 통합 경험을 제공하며, 안드로이드 운영체제 자체로도 애플에 큰 영향을 미칩니다.
*   **기타 안드로이드 제조사:** 샤오미, 오포, 비보, 원플러스 등 다양한 안드로이드 스마트폰 제조사들이 각자의 영역에서 경쟁하고 있습니다.

**2. 컴퓨터 (노트북 및 데스크톱):**
```

```
*   **마이크로소프트:** 서피스 라인업을 통해 하드웨어 시장에 진출했으며, 윈도우 운영체제는 맥OS의 가장 큰 경쟁자입니다.
*   **HP, 델, 레노버:** 이들은 윈도우 기반 노트북 및 데스크톱 시장에서 애플의 맥북 및 아이맥과 경쟁합니다.
*   **기타:** 닌텐도 스위치와 같은 게임 콘솔도 엔터테인먼트 소비라는 측면에서 넓은 의미의 경쟁자로 볼 수 있습니다.

**3. 웨어러블 기기 (스마트워치, 이어폰):**

*   **삼성전자:** 갤럭시 워치와 갤럭시 버즈 시리즈로 애플 워치와 에어팟에 직접적으로 경쟁합니다.
*   **구글:** 핏빗 인수 이후 스마트워치 시장에서도 경쟁력을 강화하고 있습니다.
*   **소니, 보스, 젠하이저:** 고급 오디오 시장에서 무선 이어폰 및 헤드폰으로 에어팟 프로 및 에어팟 맥스와 경쟁합니다.
*   **기타:** 샤오미, 화웨이 등 다양한 제조사들이 가성비 좋은 웨어러블 기기로 경쟁합니다.

**4. 스트리밍 서비스 (음악, 동영상):**

*   **음악 스트리밍:**
    *   **스포티파이:** 애플 뮤직의 가장 강력한 경쟁자입니다.
    *   **유튜브 뮤직:** 구글이 제공하는 음악 스트리밍 서비스입니다.
    *   **아마존 뮤직:** 아마존의 음악 스트리밍 서비스입니다.
*   **동영상 스트리밍:**
    *   **넷플릭스:** 애플 TV+의 가장 큰 경쟁자입니다.
    *   **디즈니+:** 공격적인 콘텐츠 투자로 빠르게 성장하고 있습니다.
    *   **아마존 프라임 비디오:** 아마존의 동영상 스트리밍 서비스입니다.
    *   **HBO Max, Hulu, Peacock 등:** 다양한 유료 스트리밍 서비스들이 경쟁합니다.

**5. 클라우드 서비스:**

*   **구글 드라이브:** 아이클라우드의 주요 경쟁자입니다.
*   **마이크로소프트 원드라이브:** 역시 아이클라우드와 경쟁합니다.
*   **아마존 드라이브:** 아마존의 클라우드 스토리지 서비스입니다.

**6. 결제 서비스:**

*   **구글 페이:** 애플 페이의 주요 경쟁자입니다.
*   **삼성 페이:** 삼성 스마트폰 사용자들에게 중요한 결제 수단입니다.
*   **PayPal, Square 등:** 다양한 핀테크 기업들이 디지털 결제 시장에서 경쟁합니다.

**7. 앱 생태계:**

*   **구글 플레이 스토어:** 애플 앱스토어의 가장 큰 경쟁자입니다. 안드로이드 기기 사용자들에게 다양한 앱을 제공합니다.

**8. 기술 혁신 및 미래 시장:**

*   **메타 (구 페이스북):** VR/AR, 메타버스 분야에서 애플과 미래 경쟁을 예고하고 있습니다.
*   **아마존:** AI, 스마트홈, 클라우드 등 다양한 분야에서 애플과 경쟁합니다.

애플은 강력한 브랜드 충성도와 자체 생태계를 기반으로 경쟁하고 있지만, 위에서 언급된 기업들은 각자의 강점과 시장 점유율을 바탕으로 애플에게 끊임없이 도
```

⭕ `gemini-2.5-flash-lite` 모델로 체인 저장 완료 및 답변을 완료하였습니다.

- **② 다른 LLM, 프롬프트를 변수로 저장해보기**
  - `llm` = `default` = `groq`
  - `prompt` = `"{company} 에 대해서 20자 이내로 설명해줘."`, {`"prompt"`:`"competitor"`}

```
# with_config로 설정을 변경하여 생성한 체인 = 별도의 변수에 저장하기

groq_competitor_chain = chain.with_config(
    configurable={"llm": "groq", "prompt": "competitor"}
)
```

```
# 체인 호출해보기_2

groq_competitor_chain.invoke({"company": "애플"}).__dict__
```

- **`groq_competitor_chain.invoke()`** = *groq* 으로 생성한 답변 - (`1.1s`)

```
{'content': '애플은 다양한 제품과 서비스를 제공하는 세계적인 기술 기업입니다. 따라서 애플의 경쟁사는 여러 가지 분야에서 다양한 회사들이 있습니다.
```

```
'additional_kwargs': {},
'response_metadata': {'token_usage': {'completion_tokens': 400,
'prompt_tokens': 47,
'total_tokens': 447,
'completion_time': 0.712562395,
'prompt_time': 0.002615139,
'queue_time': 0.046669167,
'total_time': 0.715177534},
'model_name': 'llama-3.1-8b-instant',
'system_fingerprint': 'fp_33e8adf159',
'service_tier': 'on_demand',
'finish_reason': 'stop',
'logprobs': None},
'type': 'ai',
'name': None,
'id': 'run--cebafab2-9cf0-4b54-aa9c-6ee84540189d-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 47,
'output_tokens': 400,
'total_tokens': 447}}
```

```
response_com3=groq_competitor_chain.invoke({"company": "애플"}).response_metadata
print(f"✅ {response_com3.get('model_name')} 모델로 답변을 생성하였습니다.")
```

- ✅ llama-3.1-8b-instant 모델로 답변을 생성하였습니다. (`0.9s`)

```
response_com4=groq_competitor_chain.invoke({"company": "애플"}).content
print(response_com4)
print(f"\n ⭕ {response_com3.get('model_name')} 모델로 체인 저장 완료 및 답변을 완료하였습니다.")
```

- 답변 완료 (`1.0s`)

```
애플은 다양한 제품과 서비스를 제공하는 세계적인 기술 기업입니다. 따라서 애플의 경쟁사는 여러 가지 분야에서 다양한 회사들이 있습니다. 여기 몇 가지 예

1. **스마트폰**: 애플의 경쟁사로는 삼성전자, 구글, 오폰, 하이닉스 등이 있습니다. 삼성전자와 구글은 애플의 아이폰과 안드로이드 운영 체제를 제공하는
2. **컴퓨터**: 애플의 맥북과 아이맥은 PC 시장에서 경쟁하고 있습니다. 주요 경쟁사는 마이크로소프트, 델, HP, 레노버 등이 있습니다.
3. **태블릿**: 애플의 아이패드는 태블릿 시장에서 경쟁하고 있습니다. 주요 경쟁사는 삼성전자, 구글, 아마존, 마이크로소프트 등이 있습니다.
4. **음악 스트리밍**: 애플의 음악 스트리밍 서비스인 애플 뮤직은 스포티파이, 아마존 뮤직, 구글 플레이 뮤직 등과 경쟁하고 있습니다.
5. **비디오 스트리밍**: 애플의 비디오 스트리밍 서비스인 애플 TV+는 넷플릭스, 아마존 프라임 비디오, 디즈니+ 등과 경쟁하고 있습니다.

이러한 경쟁사들은 애플의 제품과 서비스를 제공하는 다양한 분야에서 경쟁하고 있습니다. 애플은 이러한 경쟁사들과 끊임없이 경쟁하고 있으며, 새로운 제품과

⭕ llama-3.1-8b-instant 모델로 체인 저장 완료 및 답변을 완료하였습니다.
```

- **③ 다른 LLM, 프롬프트를 변수로 저장해보기**
  - `llm` = `gemini2` = `gemini-2.5-flash`
  - `prompt` = {`"prompt"`:`"eng"`}

```
# with_config로 설정을 변경하여 생성한 체인 = 별도의 변수에 저장하기

gemini2_competitor_chain = chain.with_config(
    configurable={"llm": "gemini2", "prompt": "eng"}
)
```

```
# 체인 호출해보기

gemini2_competitor_chain.invoke({"input": "다 좋은 것은 좋다"}).__dict__
```

- **`gemini2_competitor_chain.invoke()`** = *gemini-2.5-flash* 로 생성한 답변 - (`13.3s`)

```
{'content': '"다 좋은 것은 좋다"는 상황과 뉘앙스에 따라 여러 가지 영어 표현으로 번역될 수 있습니다. 이 문장은 말 그대로 "모든 좋은 것은 좋다"는
'additional_kwargs': {},
'response_metadata': {'prompt_feedback': {'block_reason': 0,
'safety_ratings': []},
'finish_reason': 'STOP',
'model_name': 'gemini-2.5-flash',
'safety_ratings': []},
'type': 'ai',
'name': None,
'id': 'run--ea281ecc-e04f-4acb-b692-ab686a402392-0',
'example': False,
'tool_calls': [],
'invalid_tool_calls': [],
'usage_metadata': {'input_tokens': 12,
'output_tokens': 1610,
'total_tokens': 1622,
'input_token_details': {'cache_read': 0},
'output_token_details': {'reasoning': 1052}}}
```

```
response_com5=gemini2_competitor_chain.invoke({"input": "다 좋은 것은 좋다"}).response_metadata
print(f"✅ {response_com5.get('model_name')} 모델로 답변을 생성하였습니다.")
```

- ✅ gemini-2.5-flash 모델로 답변을 생성하였습니다. (`11.3s`)

```
response_com6=gemini2_competitor_chain.invoke({"input": "다 좋은 것은 좋다"}).content
print(response_com6)
print(f"\n ⭕ {response_com5.get('model_name')} 모델로 체인 저장 완료 및 답변을 완료하였습니다.")
```

- 답변 완료 (`11.7s`)

```
"다 좋은 것은 좋다"는 상황에 따라 여러 가지 뉘앙스로 번역될 수 있습니다. 가장 일반적이고 자연스러운 표현들은 다음과 같습니다:

1.  **What's good is good.**
    *   가장 간결하고 자연스러운 표현입니다. "좋은 것은 좋다고 인정한다"는 의미를 담습니다.

2.  **If it's good, it's good.**
    *   조건을 명확히 하여 "좋은 것이라면 좋은 것이다"라는 의미를 전달합니다. 어떤 것을 굳이 복잡하게 생각할 필요 없이 좋게 받아들일 때 사용하기

3.  **Good is good.**
    *   더 짧고 단호한 느낌을 줍니다. 좋은 것은 그 자체로 좋다는 것을 강조합니다.

**어떤 상황에서 쓰이는지:**
이 표현은 특별히 복잡하게 생각할 필요 없이 좋은 것은 좋다고 인정하거나, 좋은 일에 대해 굳이 반대할 이유가 없다는 뜻을 나타낼 때 사용됩니다.

**예시:**
*   A: 이 제안, 너무 간단해서 뭔가 놓친 게 있을까봐 걱정돼. (I'm worried this proposal is too simple, maybe we're missing som
*   B: **다 좋은 것은 좋다.** 그냥 좋게 받아들이자. (**If it's good, it's good.** Let's just accept it as good.)

가장 추천하는 번역은 **"What's good is good."** 또는 **"If it's good, it's good."** 입니다.

⭕ gemini-2.5-flash 모델로 체인 저장 완료 및 답변을 완료하였습니다.
```

- next: ***07. @chain 데코레이터로 Runnable 구성***
