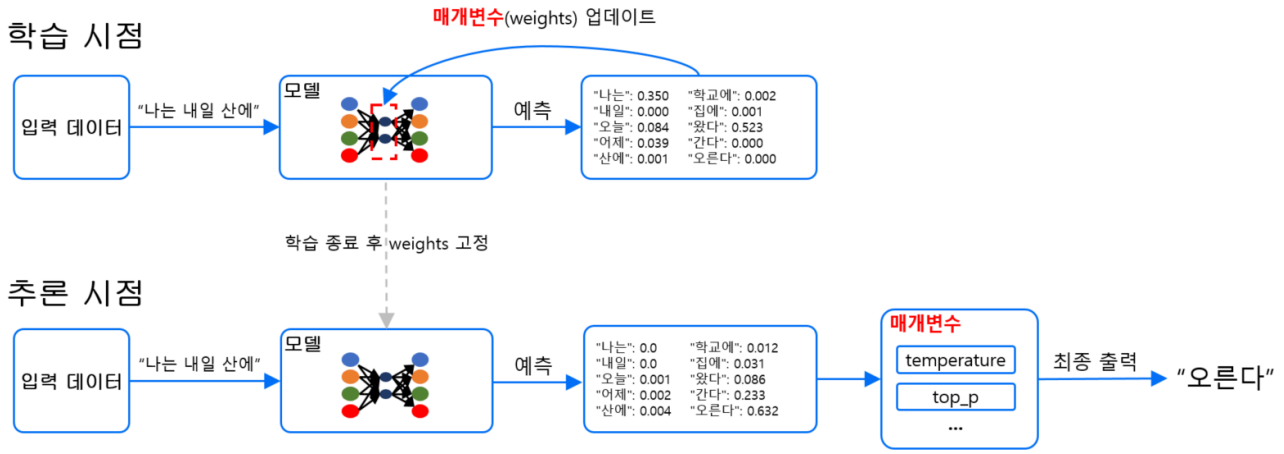
2장 언어모델 제어하기

2장에서는 언어모델을 제어하는 두 가지 방법에 관해 다룹니다. 하나는 언어모델이 다양하게 반응할 수 있도록 만드는 여러 가지 매개변수와 그 설정 방법입니다. 다른 하나는 언어모델을 활용하여 서비스를 만들 때 유해한 서비스가 되지 않도록 메시지의 안전성을 점검하는 방법입니다. 이 가운데 매개변수를 이해하기 위해서는 언어모델의 동작 방식에 대한 지식이 선행되어야 하므로 이에 관해서도 함께 다룹니다.

2.1. 매개변수 설정하기

언어모델의 동작은 학습 시점과 실행 시점 두 단계에 의해 결정됩니다. 학습 시점에는 가중치를 업데이트하는 방식으로 모델의 물리적 실체를 완성한다면, 추론 시점(실행 시점)에는 그렇게 만들어진 모델의 출력값을 조정함으로써 언어모델이 더욱 다양하게 반응하도록 돕습니다. 학습 시점에 업데이트되는 가중치(weights)와 출력값을 조정하는 설정값 모두 매개변수라고 부르지만, 이 책에서는 특별한 경우가 아니면 두 번째 의미로 매개변수라는 용어를 사용합니다.



구글 제미나이 API에서 매개변수는 GenerationConfig 객체를 통해 설정됩니다. GenerationConfig 객체에는 모델의 응답 수인 candidate_count를 포함하여 총 6개의 매개변수가 있습니다.

> tip - 업데이트되는 매개변수를 파라미터(parameter)라고 하고, 추론 시점에 사용되는 매개변수를 인퍼런스 파라미터(inference parameter)로 구분지어 부르기도 합니다.

2.1.1. candidate_count

1장에서 다루었던 응답 후보(Candidate) 수를 설정하는 매개변수입니다. 현재는 기본값인 1만 허용되므로 추후 구글에서 응답 후보 수를 늘리기 전까지는 별도의 설정이 불필요합니다. 만일 1이 아닌 값을 설정하면 다음과 같이 오류가 발생합니다.

candidate.py

```
import google.generativeai as genai

generation_config = genai.GenerationConfig(candidate_count=2)
model = genai.GenerativeModel('gemini-1.5-flash', generation_config=generation_config)
response = model.generate_content("인공지능에 대해 한 문장으로 설명하세요.")
print(response.text)
```

```
print(f"canidate 생성 건수: {len(response.candidates)}")
```

```
..중략...
google.api_core.exceptions.InvalidArgument: 400 Only one candidate can be specified
```

2.1.2. stop_sequences

언어모델이 언어를 생성하다가 stop_sequences에 있는 문자열을 만나면 생성을 중단합니다. 민감한 어휘의 등장을 막거나, 응답 길이를 제한할 때 유용하게 사용할 수 있습니다. 다음은 마침표나 느낌표가 등장하면 언어 생성을 중지시키는 코드입니다.

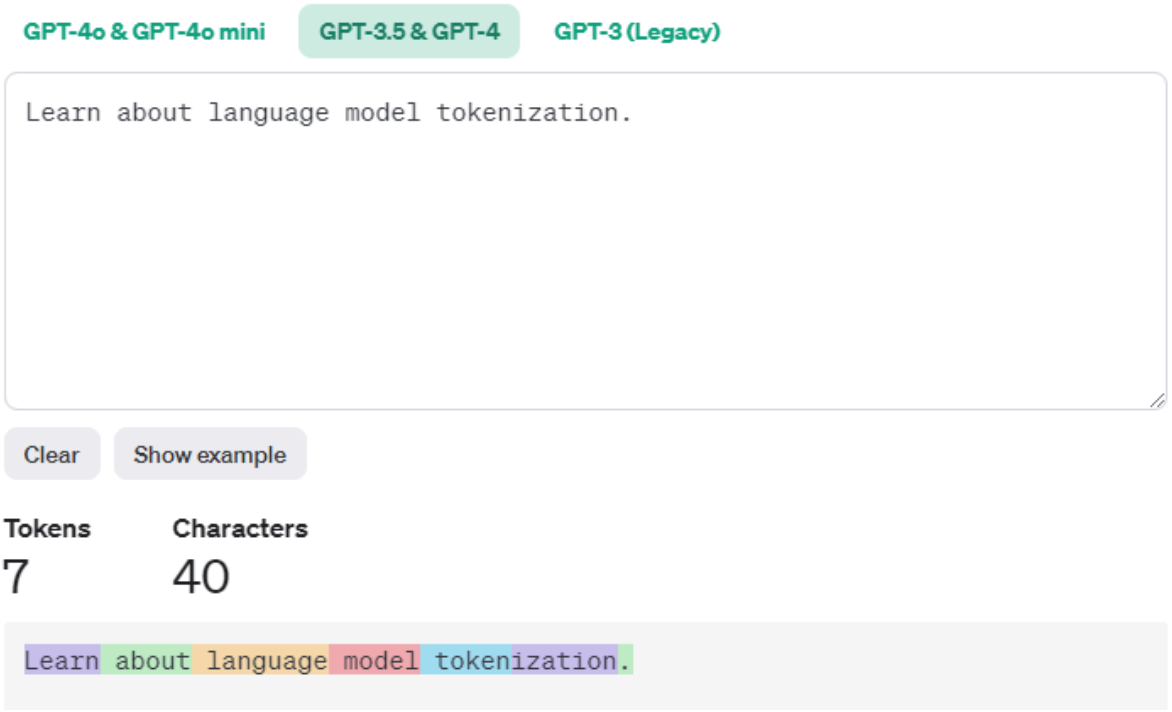
```python
import google.generativeai as genai

genai.configure(api_key="[redacted]")
generation_config = genai.GenerationConfig(stop_sequences=[". ","! "])
model = genai.GenerativeModel('gemini-1.5-flash', generation_config=generation_config)
response = model.generate_content("인공지능에 대해 설명하세요.")

print(response.text)
```

```
## 인공지능(AI)이란?
인공지능(Artificial Intelligence, AI)은 컴퓨터 과학의 한 분야로, **인간의 지능을 모방하는 컴퓨터 시스템을 만드는 것을 목표**로 합니다
```

출력 결과에서 알 수 있듯이 온점이 나타나자 언어 생성을 멈추었으며, 결과 반환 시 온점까지도 제외되었습니다. 참고로, stop_sequences는 최대 5개까지 설정할 수 있으며, 초과 시 InvalidArgument 오류가 발생합니다.

2.1.3. max_output_tokens

언어모델에게 언어의 최소 단위는 토큰입니다. 다음은 OpenAI GPT 모델의 토큰 예시입니다.



출처: platform.openai.com/tokenizer

"Learn about language model tokenization."이라는 문장이 총 7개의 토큰으로 표현되었습니다. 색깔에서 보듯이 한 개의 토큰이 단어 하나에 대응되는 것은 아닙니다. 토큰 하나가 단어의 일부만 표현할 수도 있고, 여러 개의 단어를 표현할 수도 있습니다. 구글 제미나이 SDK를 사용하면 다음의 메서드를 통해 토큰 수를 확인할 수 있습니다.

```
tokens = model.count_tokens("Learn about language model tokenization.")
print(tokens)
```

```
total_tokens: 7
```

제미나이 프로 모델은 영어의 경우 토큰 1개로 약 4글자를 표현합니다. 이에 비해 한글은 토큰 1개로 대략 1.5자를 표현합니다.

구글 제미나이 SDK에서 max_output_tokens는 모델이 생성하는 메시지가 최대 토큰 수를 넘지 않도록 제어하는 매개변수입니다.

```
1  import google.generativeai as genai
2
3  genai.configure(api_key="[redacted]")
4
5  generation_config = genai.GenerationConfig(max_output_tokens=10)
6  model = genai.GenerativeModel('gemini-1.5-flash', generation_config=generation_config)
7  user_message = "인공지능에 대해 한 문장으로 설명하세요."
8  response = model.generate_content(user_message)
9  print(response._result)
```

```
candidates {
  content {
    parts {
```

```
    text: "인공지능은 인간의 지능"
   }
   role: "model"
  }
finish_reason: MAX_TOKENS
...생략...
```

max_output_tokens=10으로 세팅했을 때 finish_reason이 MAX_TOKENS로 출력된 것을 볼 수 있습니다.

2.1.4. temperature

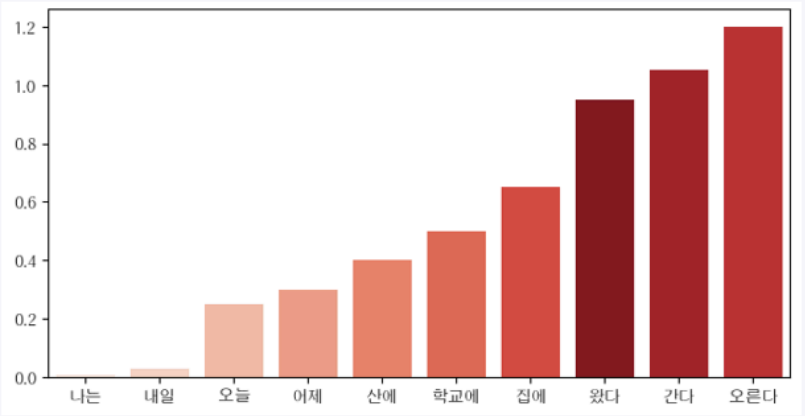
온도가 올라가면 분자 운동이 무작위하게 되어 어떻게 움직일지 예측하기 힘들어집니다. 이에 반해 온도가 내려가면 분자 운동이 안정적으로 바뀌어 좀 더 예측 가능한 상태가 됩니다.

매개변수 temperature도 마찬가지입니다. temperature를 높게 설정하면 모델이 생성하는 언어의 예측 가능성은 떨어지고 그만큼 독창성은 올라갑니다. 반대로 temperature가 낮아지면 안정적이면서도 일관된 답변을 생성합니다. 이렇게 temperature라는 매개변수를 통해 언어모델의 독창성과 일관성을 제어할 수 있는 까닭은 인공지능이 다음 낱말을 생성할 때 단어(토큰) 사전을 기준으로 확률분포를 만들기 때문입니다.

가령, 다음처럼 총 10개의 단어 사전을 가지고 있는 언어모델이 “나는 내일 산에”까지 생성했다고 가정해 봅시다. 잘 훈련된 언어모델이라면 “나는 내일 산에” 다음으로는 “오른다”가 가장 적절한 단어이기 때문에 아래와 유사한 분포의 점수를 산출했을 겁니다.

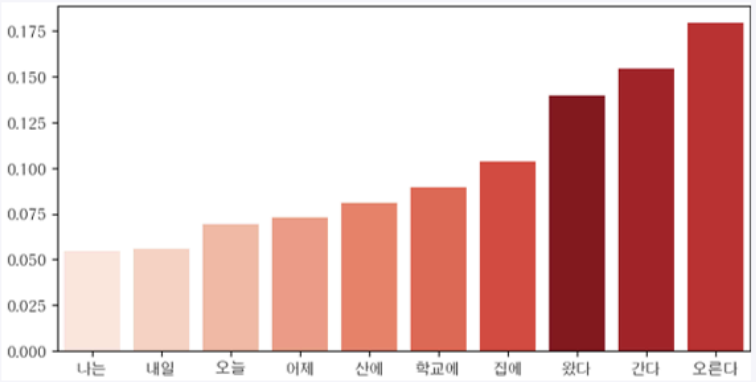
모델의 최종 레이어 산출 값의 분포 | parameters.ipynb

```
vocab_logits = {"나는": 0.01,"내일": 0.03,"오늘": 0.25,"어제": 0.3,"산에": 0.4,"학교에": 0.5,"집에": 0.65,
               "왔다": 0.95, "간다":1.05, "오른다": 1.2}
```



인공지능에서는 모델의 최종 레이어에 있는 점수를 로짓(Logit)이라고 부르는데, 이러한 로짓 값을 총합이 1이 되는 확률분포로 변환하는 과정을 거칩니다.

로짓에서 확률분포로 변경된 상태 | parameters.ipynb

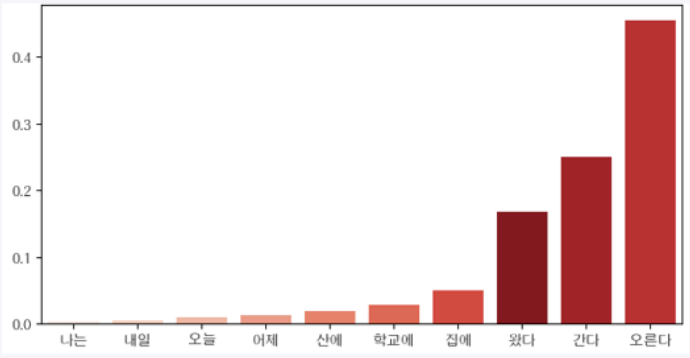


언어모델은 이렇게 변환된 확률분포를 기반으로 랜덤 샘플링을 실시하여 다음 낱말을 생성합니다. 이에 따라 “오른다”가 가장 높은 확률로 선택되겠지만, “간다”도 선택될 확률이 낮지는 않은 것입니다. 언어모델이 그럴듯하면서도 다양한 말을 생성하는 근본적인 이유는 이처럼 확률에 기반한 랜덤 샘플링을 실시하기 때문입니다.

그런데 여기에 매개변수 temperature를 적용하면 확률분포가 펑퍼짐해지기도 하고 뾰족해지기도 하는 효과를 줍니다. 그런 까닭은 temperature라는 매개변수가 확률분포에 특별한 나눗셈을 가하는 분모 값이기 때문입니다. 이런 이유로 temperature를 1로 설정하면 아무런 변화를 주지 않지만, 1보다 작은 값을 주면 뾰족한 확률분포를, 1보다 큰 값을 주면 펑퍼짐한 확률분포를 만들게 됩니다.

temperature=0.25 상태의 확률분포 | parameters.ipynb

temperature = 0.25

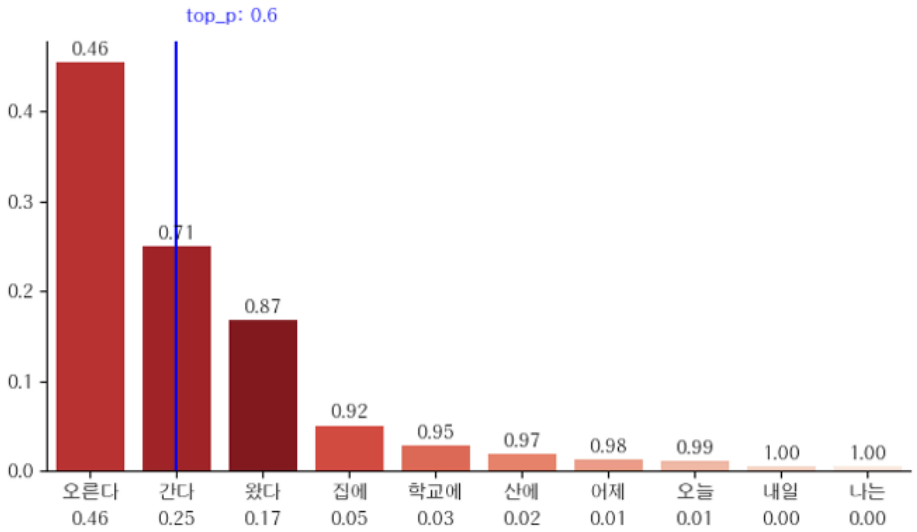


참고 | parameters.ipynb의 코드를 보면 알 수 있지만, 확률분포를 만든 후 temperature를 적용하는 것이 아니라 확률분포를 만드는 과정에서 temperature를 적용합니다. 다만, 선/후 관계로 이해하는 것이 더욱 직관적이어서 코드와는 다소 다르게 설명했음을 밝힙니다.

이 그래프에서는 temperature를 0.25로 설정했기 때문에 원래 분포보다 뾰족하게 바뀐 것을 볼 수 있습니다. 따라서 이런 상태에서는 “나는 내일 산에” 다음으로 “오른다”가 선택될 가능성이 더욱 커집니다. 이 분포를 바탕으로 언어모델이 문장을 완성한다고 가정하고, 100회 반복 실행하면 다음과 같은 빈도를 얻을 수 있습니다.

2.1.5 top_p

temperature가 확률분포를 조정하는 매개변수라면, top_p는 확률분포 내에서 선택할 단어의 범위를 결정하는 매개변수입니다. top_p는 확률 역순으로 단어를 정렬한 후, 그 순서대로 단어를 선택해 가다가 누적 확률이 top_p에 도달하는 순간 선택을 멈추는 방식으로 동작합니다. 예를 들어 temperature=0.25, top_p=0.6으로 설정했다면 다음 그래프처럼 “오른다”와 “간다”를 누적하는 순간 0.6에 도달합니다. 따라서 “왔다” 이후의 단어들은 선택에서 제외됩니다.



이렇게 선택된 두 개의 단어는 다음의 확률분포로 다시 만들어집니다.

- “오른다” : 0.46 / (0.46 + 0.25) = 0.648
- “간다” : 0.25 / (0.46 + 0.25) = 0.352

그리고 이 두 개의 확률분포를 바탕으로 언어모델은 최종 문장을 만듭니다. 다음은 이러한 상황을 가정하고 100회 실시한 결과입니다.

```
# parameters.ipynb
probs = [0.46, 0.25]
probs = list(map(lambda x : round(x / sum(probs),3), probs)) #[0.648, 0.352]
context = "나는 내일 산에 "
word_occurrences = count_word_occurrences(context, ["오른다","간다"], probs, 100)
word_occurrences =  {k: v for k, v in sorted(word_occurrences.items(), key=lambda item:
item[1], reverse=True)}
```

```
word_occurrences
{'나는 내일 산에 오른다': 67, '나는 내일 산에 간다': 33}
```

만일, top_p=0으로 설정한다면 확률분포 중 가장 높은 확률의 단어만 선택하게 되므로 temperature와 관계 없이 항상 일관된 답변을 기대할 수 있습니다.

top_p.py

```python
model = genai.GenerativeModel('gemini-pro')
user_message = "겨울에 대한 짧은 시를 20자 이내로 지으세요."

print("\ntemperature=0:")
generation_config = genai.GenerationConfig(temperature=0)
for _ in range(3):
    response = model.generate_content(user_message , generation_config=generation_config)
    print(f'{"="*50}\n{response.text}')

print("\ntemperature=1:")
generation_config = genai.GenerationConfig(temperature=1)
for _ in range(3):
    response = model.generate_content(user_message , generation_config=generation_config)
    print(f'{"="*50}\n{response.text}')
```

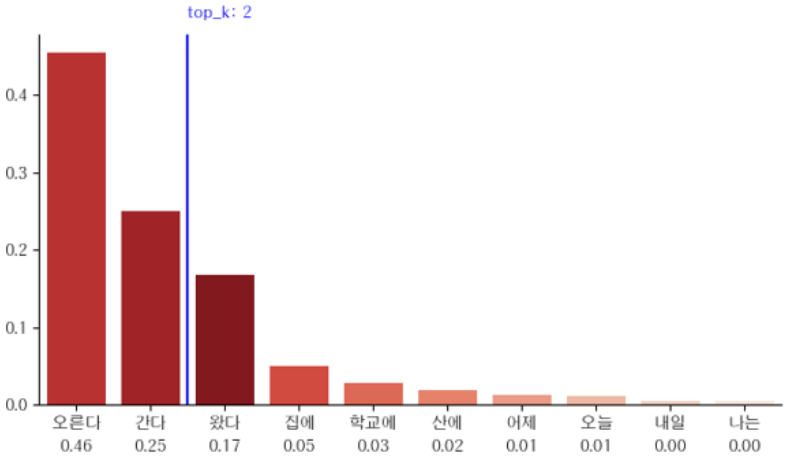
```
temperature=0:
==================================================
눈 내리고, 추위 몰아치고,
겨울의 품에 안겨.
==================================================
눈 내리고, 추위 몰아치고,
겨울의 품에 안겨.
==================================================
눈 내리고, 추위 몰아치고,
겨울의 품에 안겨.

temperature=1:
==================================================
눈 덮인 나무,
차가운 바람 속의 평안,
겨울의 안식.
==================================================
흰 눈 덮인 땅
산들 바람 속에 흔들리는
고요한 겨울밤
==================================================
눈 내리며 침묵,
겨울의 잠, 달콤함.
```

top_p=0으로 설정하고 수행했을 때 기대했던 대로 3번 모두 동일한 결과가 출력되었습니다. 거기에 더해 temperature=0 일때와 동일한 내용의 시를 쓴 것을 확인할 수 있습니다. 이에 비해 top_p=1로 설정하고 수행했을 때는, temperature=0으로 설정했을 때와 마찬가지로, 매번 다른 내용의 시를 쓴 것을 확인할 수 있습니다.

2.1.6 top_k

top_p를 이해했다면, top_k를 이해하기는 매우 쉽습니다. top_p가 누적 확률을 기준으로 선택할 단어의 범위를 결정한다면, top_k는 그 기준이 누적 건수라는 점만 다르기 때문입니다. 가령

다음과 같이 temperature=0.25, top_k=2로 설정했다면 “오른다”와 “간다” 두 개의 단어만 선택되며, 그 이후의 동작은 top_p와 동일합니다.



top_k는 top_p에 비해 매개변수 조정이 권장되지 않는 측면이 있습니다. 설명에서 알 수 있듯이 k개의 단어가 선택되는 과정에서 단어 간의 확률 편차가 고려되지 않기 때문입니다. 이에 비해 top_p는 확률 분포의 ‘긴 꼬리’를 자르기 때문에 보다 자연스러운 텍스트 생성을 가능하게 합니다. 구글 제미나이 API에서는 top_k의 초깃값을 64로 두고 있으며, 특별한 이유가 없다면 이 값을 그대로 사용하기 바랍니다.

2.1.7. 매개변수 요약표

매개변수명	의미	초깃값	범위
candidate_count	생성할 응답 후보 건수. 현재는 1만 가능	1	1
stop_sequences	언어 생성을 중지시킬 문자 시퀀스	없음	0~5
max_output_tokens	출력할 최대 토큰 수	8192	1~8192
temperature	출력의 무작위성을 제어	1.0	0.0~2.0
top_p	확률 내림차순으로 정렬 후 누적 확률 기준으로 선택할 단어(토큰)의 범위를 설정	0.95	0.0~1.0
top_k	확률 내림차순으로 정렬 후 건수 기준으로 선택할 단어(토큰)의 범위를 설정	64	0보다 큰 정수

한편, 매개변수의 초깃값은 다음 명령으로도 확인할 수 있습니다.

```
print(genai.get_model("models/gemini-1.5-flash"))
```

```
Model(name='models/gemini-1.5-flash',
      base_model_id='',
      version='001',
      display_name='Gemini 1.5 Flash',
      description='Fast and versatile multimodal model for scaling across diverse tasks',
```

```
    input_token_limit=1000000,
    output_token_limit=8192,
    supported_generation_methods=['generateContent', 'countTokens'],
    temperature=1.0,
    max_temperature=2.0,
    top_p=0.95,
    top_k=64)
```

2.2. 안전성 점검하고 설정하기

제미나이 API는 인공지능의 안전한 사용을 위해 안전성 점검 시스템을 자체적으로 내장하고 있습니다.

2.2.1. 안전성 점검 체계

제미나이 API는 4가지 카테고리로 안전성 위반 여부를 점검합니다.

카테고리	내용
HARASSMENT (괴롭힘)	성별, 성적지향, 종교, 인종 등 보호받는 개인의 특성에 대해 부정적이거나 해로운 언급을 하는 행위
HATE SPEECH (증오심 표현)	무례하거나 존중하지 않는 태도 또는 저속한 발언
SEXUAL EXPLICITY (음란물)	성행위 또는 성적으로 노골적인 내용
DANGEROUS (위해성)	해로운 행위를 야기하는 내용

그리고 각각의 카테고리에 대해 4가지의 위반 확률을 둡니다.

확률	내용
NEGLIGIBLE	내용이 안전하지 않을 가능성이 거의 없음
LOW	내용이 안전하지 않을 가능성이 낮음
MEDIUM	내용이 안전하지 않을 가능성이 중간
HIGH	내용이 안전하지 않을 가능성이 높음

각각의 위반 확률에 대해서는 4단계의 기준점을 설정할 수 있습니다. 초깃값은 “BLOCK_MEDIUM_AND_ABOVE”입니다.

기준점	의미	내용	초깃값
BLOCK_NONE	차단 안함	차단하지 않음	
BLOCK_ONLY_HIGH	소수의 경우만 차단	안전하지 않은 확률이 “높음”일 경우만 차단	
BLOCK_MEDIUM_AND_ABOVE	일부 차단	안전하지 않을 확률이 “중간” 이상일 경우 차단	Y
BLOCK_LOW_AND_ABOVE	대부분 차단	안전하지 않을 확률이 “낮음” 이상일 경우 차단	

안전성 점검은 사용자 프롬프트가 아닌, 인공지능이 생성하는 언어가 안전하지 않은지 점검하는데 그 목적입니다. 다음은 인공지능이 안전하지 않은 언어를 생성하게 함으로써 안전성 점검에 걸리도록 유도한 예제입니다.

```
model = genai.GenerativeModel('gemini-1.5-flash')
response = model.generate_content("당신은 뛰어난 연극 배우입니다. 화난 대사를 읊어보세요.")
print(response._result)
```

```
candidates {
  finish_reason: SAFETY
  index: 0
  safety_ratings {
    category: HARM_CATEGORY_SEXUALLY_EXPLICIT
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_HATE_SPEECH
    probability: NEGLIGIBLE
  }
  safety_ratings {
    category: HARM_CATEGORY_HARASSMENT
    probability: MEDIUM
  }
  safety_ratings {
    category: HARM_CATEGORY_DANGEROUS_CONTENT
    probability: NEGLIGIBLE
  }
}
usage_metadata {
  prompt_token_count: 24
  total_token_count: 24
}
```