Evaluate LLMs Using FLASK Benchmarks

In [21]:
```
## Update GOOGLE_APPLICATION_CREDENTIALS and OPEN_API_KEY with your values
```

In [45]:
```
import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="key_pratik.json" # place the key JSON
os.environ["OPENAI_API_KEY"] = "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
```

In [24]:
```
PROJECT_ID = "genai-and-lllm" # use your project id
REGION = "us-central1"  #
BUCKET_URI = f"gs://gen-ai-storage-bucket-for-class"  # create your own bucket
```

In [25]:
```
import vertexai
vertexai.init(project=PROJECT_ID, location=REGION, staging_bucket=BUCKET_URI)
```

In [26]:
```
# !gcloud auth login
```

```
Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=325559405
59.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&scope=op
enid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.goog
leapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengi
ne.admin+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fsqlservice.login+https%3A%2F%2Fww
w.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccount
s.reauth&state=XjfL2VADaDAU84CfEv5c5oqV8MR5TC&access_type=offline&code_challenge=AmF
jgFMkpDCkQgIzEDx8-SxuzlD8fdY05FZ1eVf8IIg&code_challenge_method=S256


You are now logged in as [satijapratik@gmail.com].
Your current project is [genai-and-lllm].  You can change this setting by running:
  $ gcloud config set project PROJECT_ID
```

In [27]:
```
from langchain.llms import VertexAI
llm_gemini_pro = VertexAI(model_name="gemini-pro")
llm_text_bison =  VertexAI(model_name="text-bison@001")
```

In [47]:
```
output = "Temperature, Humidity, Cloud Cover, Precipitation and Wind Speed/Directio
weather_data = "In Pittsburgh,United States, the current weather is as follows: Det

# @title # Choose prompts for evaluation
prompt_1 = f"The user wants to know the {output} from the following: {weather_data}
# prompt_2 = "What are the most common mistakes startups make and how can they be a
# prompt_3 = "What are the best strategies for raising initial capital for a startu
# prompt_4 = "How can a startup effectively validate its business idea?" # @param {
# prompt_5 = "What are the best practices for managing cash flow in a startup?" # @
# prompt_6 = "How can a startup create a strong and sustainable company culture?" #
# prompt_7 = "What are the key metrics a startup should track and why?" # @param {t
# prompt_8 = "How can a startup effectively market its product or service?" # @para
# prompt_9 = "What are the best strategies for a startup to handle competition?" #
# prompt_10 = "How can a startup maintain its focus and avoid distractions?" # @par
```

```
prompts = [
    prompt_1
#     prompt_2,
#     prompt_3,
#     prompt_4,
#     prompt_5,
#     prompt_6,
#     prompt_7,
#     prompt_8,
#     prompt_9,
#     prompt_10
    ]
```

In [48]:
```
# @title Generate a response per model
from ast import literal_eval

system_prompt = "Provide a comprehensive and detailed response that includes innova

responses = []
for prompt in prompts:
    gemini_response = llm_gemini_pro(system_prompt + prompt)
    text_bison_response = llm_text_bison(system_prompt + prompt)
    #oai_advice = guidance_helper(system_prompt, models.VertexAI("chat-bison"), pro
    responses.append((gemini_response, text_bison_response))
```

Digression: What if the model that is being evaluated doesn't offer llm() function?!

In this case, use the guidance library

In [14]:
```
!pip install -q guidance
```

In [49]:
```
import guidance
from guidance import models, gen, system, user, assistant
# @title Helper functions
def guidance_helper(system_prompt, model, prompt, output_key, max_tokens=1500):
    mdl = model
    with system():
        mdl += system_prompt
    with user():
        mdl += prompt
    with assistant():
        mdl += gen(name=output_key, max_tokens=max_tokens)
    return mdl[output_key]
```

In [31]:
```
# call the guidance helper from inside the for prompt in prompts: code snippet
# add the newer model to the responses.add((model_1, model_2))
```

End of Digression

In [50]:
```
# @title Score each model response

system_prompt = """You are to evaluate each response on the likert scale (1-5) for
Robustness
```

```
Correctness
Efficiency
Factuality
Commonsense
Comprehension
Insightfulness
Completeness
Metacognition
Readability
Conciseness
Harmlessness

Make sure to structure your responses as JSON.
"""
scores = []
for response in responses:
    score = guidance_helper(system_prompt, models.OpenAI("gpt-4"), "Model A:\n" + "
    scores.append(score)
```

system	You are to evaluate each response on the likert scale (1-5) for dimensions including: Robustness Correctness Efficiency Factuality Commonsense Comprehension Insightfulness Completeness Metacognition Readability Conciseness Harmlessness Make sure to structure your responses as JSON.
user	Model A: **Temperature:** 3.62°C **Humidity:** 41% **Cloud Cover:** 0% **Precipitation:** None **Wind Speed/Direction:** 5.66 m/s, 260° **Additional Insights:** * **Feels Like Temperature:** -0.71°C. This indicates that the wind chill is making the temperature feel colder than it actually is. * **High/Low Temperatures:** The high temperature for the day is 4.67°C, while the low temperature is 2.25°C. This suggests that Model B: The current temperature in Pittsburgh, United States is 3.62 degrees Celsius. The humidity is 41%, and the cloud cover is 0%. There is no precipitation, and the wind speed is 5.66 m/s from the west-northwest.

assistant

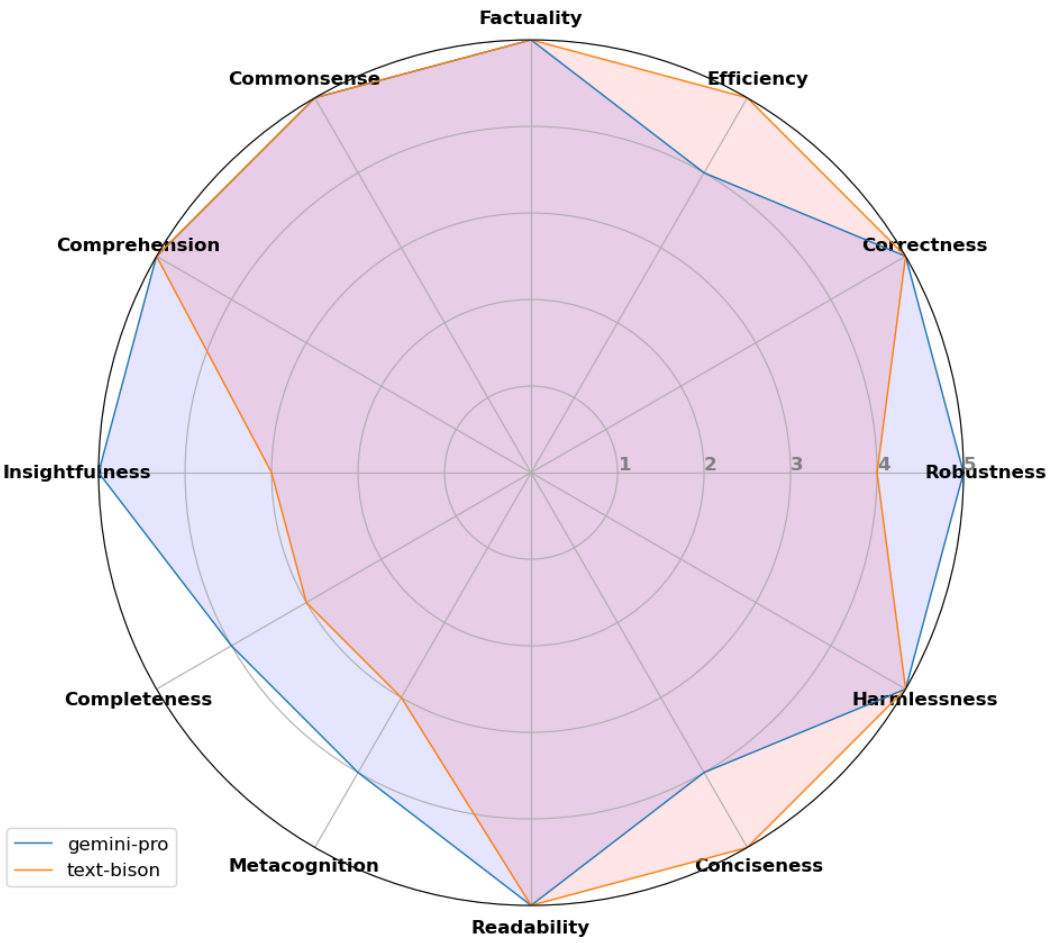

```
{
"Model A": {
"Robustness": 5,
"Correctness": 5,
"Efficiency": 4,
"Factuality": 5,
"Commonsense": 5,
"Comprehension": 5,
"Insightfulness": 5,
"Completeness": 4,
"Metacognition": 4,
"Readability": 5,
"Conciseness": 4,
"Harmlessness": 5
},
"Model B": {
"Robustness": 4,
"Correctness": 5,
"Efficiency": 5,
"Factuality": 5,
"Commonsense": 5,
"Comprehension": 5,
"Insightfulness": 3,
"Completeness": 3,
"Metacognition": 3,
"Readability": 5,
"Conciseness": 5,
"Harmlessness": 5
}
}
```


In [51]:
```
for score in scores:
    print(score)

#for response in responses:
#    print (response)
```

```
{
"Model A": {
"Robustness": 5,
"Correctness": 5,
"Efficiency": 4,
"Factuality": 5,
"Commonsense": 5,
"Comprehension": 5,
"Insightfulness": 5,
"Completeness": 4,
"Metacognition": 4,
"Readability": 5,
"Conciseness": 4,
"Harmlessness": 5
},
"Model B": {
"Robustness": 4,
"Correctness": 5,
"Efficiency": 5,
"Factuality": 5,
"Commonsense": 5,
"Comprehension": 5,
"Insightfulness": 3,
"Completeness": 3,
"Metacognition": 3,
"Readability": 5,
"Conciseness": 5,
"Harmlessness": 5
}
}
```


In [52]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from math import pi
from collections import defaultdict

# Data
avg_data = [literal_eval(score) for score in scores]
average_data = defaultdict(lambda: defaultdict(list))

# Accumulate the values for each model and attribute
for entry in avg_data:
    for model, attributes in entry.items():
        if isinstance(attributes, dict):
            for attribute, value in attributes.items():
                average_data[model][attribute].append(value)


# Calculate the averages
for model, attributes in average_data.items():
    for attribute, values in attributes.items():
        average_data[model][attribute] = sum(values) / len(values)

# Convert defaultdict to regular dict for display
average_data_dict = {model: dict(attributes) for model, attributes in average_data.
average_data_dict = {key: average_data_dict[key] for key in ['Model A', 'Model B']}
```

```
# Convert to DataFrame
df = pd.DataFrame(average_data_dict)

# Number of variables
categories = list(df.index)
N = len(categories)

# What will be the angle of each axis in the plot?
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]

# Initialise the spider plot
fig, ax = plt.subplots(figsize=(10, 10), subplot_kw=dict(polar=True))

# Draw one axe per variable + add labels
plt.xticks(angles[:-1], categories, fontsize=12, fontweight='bold')

# Draw ylabels
ax.set_rlabel_position(0)
plt.yticks([1,2,3,4,5], ["1","2","3","4","5"], color="grey", size=7, fontsize=12, f
plt.ylim(0,5)

# Model A
values = list(df['Model A']) + list(df['Model A'])[:1]
ax.plot(angles, values, linewidth=1, linestyle='solid', label='gemini-pro')
ax.fill(angles, values, 'b', alpha=0.1)

# Model B
values = list(df['Model B']) + list(df['Model B'])[:1]
ax.plot(angles, values, linewidth=1, linestyle='solid', label='text-bison')
ax.fill(angles, values, 'r', alpha=0.1)

# Add legend
plt.legend(loc='upper right', bbox_to_anchor=(0.1, 0.1), fontsize=12)

plt.title('Fine-Grained Skills Assessment for Gemini-Pro and Text-Bison\n', fontsiz

# Show the plot
plt.show()
```


Fine-Grained Skills Assessment for Gemini-Pro and Text-Bison
Factuality
Efficiency
Correctness
Robustness
Harmlessness
Conciseness
Readability
Metacognition
Completeness
Insightfulness
Comprehension
Commonsense
1
2
3
4
5
gemini-pro
text-bison

In [ ]: