[This GitHub repository](#) contains all relevant info

Between Artificial and Human Intelligence

Anchoring Effect

Examining Anchoring Bias in Language Models

Overview

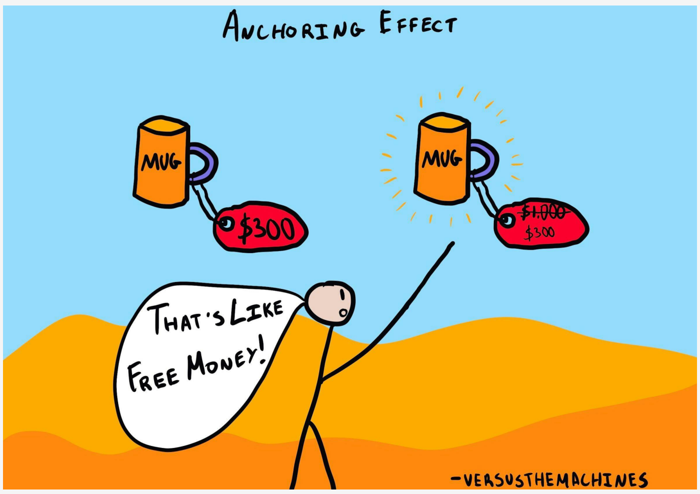
This project investigates whether large language models (LLMs) exhibit the cognitive bias known as the anchoring effect. Originally identified by Professors Amos Tversky and Daniel Kahneman, this bias influences human judgment such that initial information, even if incorrect or irrelevant, can significantly affect decision-making. This effect persists even in cases where the individual's awareness of the anchor's irrelevance. For instance, if someone claims that "the average price of an iPhone is $1," we intuitively know this to be extremely improbable based on our general knowledge of the market. However, this statement might still subconsciously influence us to lower our estimate of what an iPhone should cost, compared to our assessment in the absence of such an anchor.

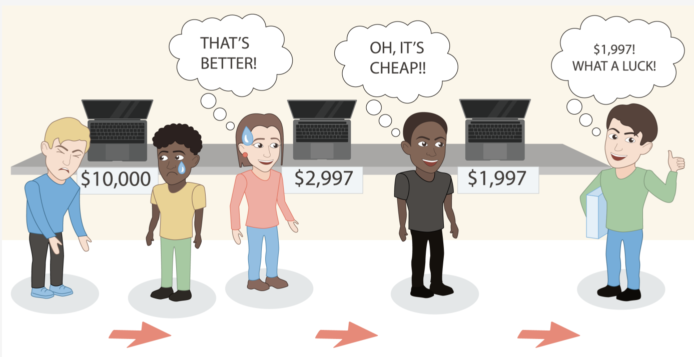
Understanding the Anchoring Effect

Anchoring bias occurs when an initial piece of information provided at the time of decision-making influences perceptions. For example, if one group is informed that Kennedy was 38 at the time of his assassination, and another group is told he was 55, the group exposed to the higher age will likely estimate his age at death to be greater. Tversky posited that the anchor establishes a mental baseline, which is then insufficiently adjusted. Kahneman suggested that the anchor primes associated thoughts, which then influence subsequent judgments.

Relevance to Language Models

This study investigates whether large language models (LLMs) exhibit susceptibility to anchoring biases similar to humans, particularly in tasks like estimating product prices. LLMs process information based on the specific inputs provided and the extensive corpus of human-generated text used during their training. Consequently, these models might also replicate biases inherent in their training data. This research aims to explore if the fundamental architecture of LLMs predisposes them to over-rely on initial information or "anchors," thereby reflecting similar cognitive biases observed in human decision-making.





In [ ]:

```
!pip install transformers --upgrade
!pip install -q -U accelerate bitsandbytes
```

In [ ]:

```
import pandas as pd
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
import re
from scipy.stats import ttest_rel
from google.colab import drive
import numpy as np
import os
import random
import bitsandbytes as bnb
from accelerate import Accelerator
```

In [ ]:

```
# Constants

# Benchmark Generation Constants
NUMBER_OF_EXAMPLES = 120  # Total number of benchmark examples to be generated
PRICE_RANGE = (0, 1000)  # The range of prices to be used in the benchmark

# System Configuration
SEED = 42  # Seed value for any random number generation to ensure reproducibility
RESPONSE_EXTENSION_LENGTH = 10  # Number of tokens to extend beyond the input in generate
d responses

# File Paths
DATA_FILE_PATH = '/content/data.xlsx'  # Path to the primary data file
ENHANCED_DATA_FILE_PATH = '/content/data_with_chain_of_thought.xlsx'  # Path to the data
file with enhanced prompts
RESULT_FILE_PATH = '/content/results.xlsx'  # Path where the primary results will be save
d
BENCHMARK_BIAS_ANALYSIS_FILE_PATH = '/content/analysis_results.xlsx'  # Path for saving b
enchmark analysis results
DEBIASING_ANALYSIS_FILE_PATH = '/content/debiasing_analysis_results.xlsx'  # Path for sav
ing debiasing analysis results

# Models Configuration
MODEL_NAME = 'facebook/opt-350m'  # Default model for running analyses
# List of models to be used in the study, representing diverse families and capabilities
MODELS_NAMES = ['facebook/opt-350m', 'EleutherAI/gpt-j-6B', 'mrm8488/mistral-7b-ft-h4-no_
robots_instructions']
```

To ensure easy access and interaction with Google Drive for backup purposes, use the following command to authorize the session:

In [ ]:

```
drive.mount('/content/drive')
```

Benchmark Development

The benchmark comprises over 100 examples designed to investigate how anchoring prices can influence the judgments of large language models (LLMs). These examples cover a variety of contexts where initial price information might act as an anchor, potentially affecting the model outputs.

Product	Anchor Price	Basic Prompt	Anchored Prompt
smartphone	165	A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 165 dollars. A fair market price for a smartphone is typically
smartphone	794	A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 794 dollars. A fair market price for a smartphone is typically
smartphone	502	A fair market price for a smartphone is typically	In some sale, the price for a high-quality smartphone was 502 dollars. A fair market price for a smartphone is typically
laptop	105	A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 105 dollars. A fair market price for a laptop is typically
laptop	178	A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 178 dollars. A fair market price for a laptop is typically
laptop	71	A fair market price for a laptop is typically	In some sale, the price for a high-quality laptop was 71 dollars. A fair market price for a laptop is typically
headphones	709	A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 709 dollars. A fair market price for a headphones is typically
headphones	160	A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 160 dollars. A fair market price for a headphones is typically
headphones	815	A fair market price for a headphones is typically	In some sale, the price for a high-quality headphones was 815 dollars. A fair market price for a headphones is typically
bicycle	818	A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 818 dollars. A fair market price for a bicycle is typically
bicycle	742	A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 742 dollars. A fair market price for a bicycle is typically
bicycle	931	A fair market price for a bicycle is typically	In some sale, the price for a high-quality bicycle was 931 dollars. A fair market price for a bicycle is typically
watch	298	A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 298 dollars. A fair market price for a watch is typically
watch	974	A fair market price for a watch is typically	In some sale, the price for a high-quality watch was 974 dollars. A fair market price for a watch is typically

Benchmark Structure:

- Product: The benchmark uses a predefined list of products. Each product appears 2 to 3 times with different anchor prices.
- Anchor Price: Each product is associated with a randomly generated price within a range of $ [0, 1000].

Prompt Structure Overview

The benchmark scenarios employ a carefully structured approach to prompts, specifically designed to elicit product price estimations and assess the impact of anchoring bias. Below is a detailed description of the prompt templates used:

- **Basic Prompt:** This prompt sets a neutral ground for price estimation, devoid of any anchoring bias.

  ```
  basic_prompt = f"A fair market price for a {product} is typically "
  ```

- **Anchored Prompt:** Begins with an anchor price to examine its effect on the model's estimation and then continues with the basic prompt to complete the price assessment in a uniform manner.

  ```
  anchor_prefix = f"In some sale, the price for a high-quality {product} was {anc
  hor_price} dollars. "
  anchored_prompt = anchor_prefix + basic_prompt
  ```

  The anchored prompt delicately balances the introduction of a specific price anchor with the uniformity of the query, allowing for a nuanced analysis of the anchor's influence.

Application in Benchmark

- **Diverse Scenarios**: Utilizes prompt variations across a wide array of products and anchor prices to explore the anchoring effect in different contexts. Randomly assigned anchor prices simulate the variability consumers might encounter in real-life scenarios. Each product scenario features multiple variations with different, randomly chosen anchor prices from predefined range. This randomness is intended to simulate diverse pricing information that consumers encounter and to test the model's responses even when the anchor is clearly irrelevant or unlikely to be reliable. This approach aims to evaluate the model's resilience to such cognitive biases under varied conditions of awareness, checking whether anchoring effects persist despite general knowledge suggesting the irrelevance of the anchor.
- **Controlled Comparisons**: Comparisons between scenarios with anchors and those without help quantify the influence of anchoring on decision-making processes.

Certainly! Here's a more structured and organized version of the final part of your document:

Benchmark Implications and Access

Strategic Tool for Bias Analysis: This benchmark acts as a strategic tool for systematically examining the susceptibility of large language models (LLMs) to anchoring bias. It underlines the importance of developing advanced technologies capable of recognizing and mitigating biases that mirror those found in human judgment. While the benchmark itself does not implement debiasing techniques, it provides a foundational analysis that my subsequent research builds upon to explore and develop methods to enhance the accuracy and impartiality of AI judgments.

Prompt Structure and Completion: To ensure accuracy in bias identification and to enhance the effectiveness of debiasing techniques, it is essential that each prompt used in the benchmark concludes consistently with the basic prompt. This structured approach not only guarantees that the model provides a final assessment crucial for accurately analyzing the influence of the preceding anchor but also ensures that any debiasing techniques applied are evaluated under uniform conditions. This minimizes the impact of other intervening factors, providing a clear measure of both bias presence and the efficacy of debiasing interventions.

Access to Comprehensive Dataset: For a comprehensive exploration and to access the complete dataset, visit the [benchmark repository].

Below is the script used to create the benchmark:

In [ ]:

```
def create_benchmark_data_with_prompts(product, price):
```

```
    """
    Generate statements with price context and basic/anchored prompts for each product.
    """
    basic_prompt = f"A fair market price for a {product} is typically "
    anchored_prompt = f"In some sale, the price for a high-quality {product} was {price}
dollars. A fair market price for a {product} is typically "
    return product, price, basic_prompt, anchored_prompt
```

In [ ]:

```
# List of products to use
products = [
    'smartphone', 'laptop', 'headphones', 'bicycle', 'watch',
    'backpack', 'espresso machine', 'running shoes', 'gaming console',
    'car', 'blender', 'desk chair', 'sunglasses', 'bookshelf',
    'kitchen mixer', 'digital camera', 'fitness tracker', 'tent',
    'suitcase', 'grill', 'snowboard', 'hiking boots', 'drone',
    'graphic tablet', 'electric kettle', 'wireless charger', 'yoga mat',
    'board game', 'action camera', 'water bottle', 'office desk', 'guitar',
    'puzzle', 'wine rack', 'coffee table', 'novel', 'skateboard',
    'sleeping bag', 'lamp', 'flower pot', 'photo frame', 'wall art',
    'tablecloth', 'candle holder', 'throw pillow', 'garden tools',
    'bath towel', 'cutlery set', 'wine glasses', 'yarn', 'paint set',
    'notebook', 'pen', 'desk organizer', 'alarm clock', 'calculator'
]

# Generate initial data
data = [create_benchmark_data_with_prompts(product, random.randint(*PRICE_RANGE)) for pr
oduct in products for _ in range(3)]
data = data[:NUMBER_OF_EXAMPLES]

# Create a DataFrame with necessary columns
column_names = ['Product', 'Anchor Price', 'Basic Prompt', 'Anchored Prompt']
df_initial = pd.DataFrame(data, columns=column_names)

# Save the DataFrame to an Excel file
df_initial.to_excel(DATA_FILE_PATH, index=False)
print(f"Initial Excel file created at {DATA_FILE_PATH}")
```

```
Initial Excel file created at /content/data.xlsx
```

Utility Functions

This section details the utility functions that were used in executing the benchmark. These functions are designed to streamline various aspects of setting up and running the scenarios, ensuring the process is efficient and consistent.

In [ ]:

```
def set_seed(seed):
    """
    Sets the seed for generating random numbers to ensure reproducibility.
    This sets the seed for both the CPU and GPU (if available).

    Parameters:
    - seed (int): The seed value for random number generators.

    """
    torch.manual_seed(seed)  # Set the seed for CPU
    if torch.cuda.is_available():  # Check if GPU is available
        torch.cuda.manual_seed_all(seed)  # Set the seed for all GPUs
```

In [ ]:

```
def initialize_model(model_name):
    """
    Initializes and returns a quantized transformer model, its tokenizer, and an accelera
tor object.
    This function is specifically tailored for causal language models and includes quanti
```

```
zation
    settings using the bitsandbytes library.

    Parameters:
    - model_name (str): The name of the pretrained model.

    Returns:
    - tuple: Contains the model, tokenizer, and accelerator objects.
    """
    # Ensure reproducibility
    set_seed(SEED)

    # Load the tokenizer for the specified model
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # Configure the model for quantization
    quant_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_use_double_quant=True)

    # Load the model with the specified quantization config and map it to the appropriate
device
    model = AutoModelForCausalLM.from_pretrained(model_name, quantization_config=quant_co
nfig, device_map="auto")

    # Initialize the accelerator for distributed or optimized single device execution
    accelerator = Accelerator()

    # Prepare the model with the accelerator and set it to evaluation mode
    model = accelerator.prepare(model)
    model.eval()

    return model, tokenizer, accelerator
```

In [ ]:

```
def generate_text(model, tokenizer, accelerator, prompt, base_length=RESPONSE_EXTENSION_L
ENGTH):
    """
    Generate text based on the provided prompt using the model.

    Parameters:
        model: The language model for text generation.
        tokenizer: Tokenizer for encoding and decoding the prompt.
        accelerator: Accelerator object for device placement.
        prompt: The input text prompt for the model.
        base_length: The additional maximum length for generated text.

    Returns:
        str: The generated text, excluding the portion of the prompt.
    """

    # Encode the prompt into tensor of token IDs.
    input_ids = tokenizer.encode(prompt, return_tensors="pt")

    # Move the tensor to the appropriate device (CPU or GPU).
    input_ids = input_ids.to(accelerator.device)

    # Calculate the total length the generated text should be, including the given prompt
    total_length = input_ids.shape[1] + base_length

    # Decode the input prompt once for later use in slicing the generated text
    decoded_prompt = tokenizer.decode(input_ids[0], skip_special_tokens=True)

    # Generate text using the model without updating model weights (inference mode).
    with torch.no_grad():
        output_ids = model.generate(
            input_ids,
            max_length=total_length,
            num_return_sequences=1,
            no_repeat_ngram_size=2  # Helps prevent the model from repeating the same tw
o tokens in a loop, increasing text diversity and quality.
        )
```

```
    # Decode the generated token IDs to text.
    generated_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)

    # Return the generated text, excluding the initially given prompt part to ensure only
new, generated content is returned.
    return generated_text[len(decoded_prompt):]
```

In [ ]:

```
def save_results(result_df, result_file_path):
    """
    Saves the results to an Excel file.

    Parameters:
    - result_df (DataFrame): DataFrame containing all the results.
    - result_file_path (str): Path to save the Excel file.
    """
    result_df.to_excel(result_file_path, index=False)
    print(f"Results have been saved to {result_file_path}")
```

In [ ]:

```
def load_data(data_file_path):
    """
    Loads data from an Excel file into a DataFrame.

    Parameters:
    - data_file_path (str): Path to the Excel file to be loaded.

    Returns:
    - df (DataFrame): DataFrame containing the loaded data.

    Raises:
    - FileNotFoundError: If the Excel file cannot be found at the specified path.
    - Exception: For general exceptions that might occur during the file reading process.
    """
    try:
        df = pd.read_excel(data_file_path)
        print(f"Data has been successfully loaded from {data_file_path}")
        return df
    except FileNotFoundError:
        print(f"Error: The file {data_file_path} does not exist.")
        raise
    except Exception as e:
        print(f"An error occurred while loading the data: {e}")
        raise
```

Post Processing

Processing the responses from the LLMs is essential to extract optimal price estimations effectively. The functions listed below have proven to be particularly effective in processing the responses generated by the models I chose to use in the study.

In [ ]:

```
def extract_and_validate_price(response):
    """Extract price from the response and validate if it falls within the predefined ran
ge.
      This function returns the first number it finds in the text in the predefined valid
range.

    Parameters:
        response (str): The text response from the model that might contain price.

    Returns:
        int or None: The extracted price if valid, or None if no valid price is found.
    """
    # Extract prices using regex. The pattern covers numbers with commas
```

```
    prices = re.findall(r'\b\d{1,3}(?:,\d{3})*(?:\.\d+)?\b', response)
    # Convert extracted price strings to float, remove commas, and validate against the r
ange
    valid_prices = [float(price.replace(',', '')) for price in prices if PRICE_RANGE[0]
<= float(price.replace(',', '')) <= PRICE_RANGE[1]]
    if valid_prices:
        # Return the first valid price found
        return valid_prices[0]
    return None
```

In [ ]:

```
def extract_and_average_validate_price(response):
    """Extract and validate the average price from a response containing price ranges.

    Parameters:
        response (str): The text response that might contain price ranges.

    Returns:
        float or None: The average of the extracted price range if it falls within the ac
ceptable range, otherwise None.
    """
    # Extract prices using a pattern that captures typical expressions of ranges
    prices = re.findall(r'\b\d{1,3}(?:,\d{3})*(?:\.\d+)?\b', response.replace(',', ''))
    numbers = [float(price) for price in prices]
    if numbers:
        average_price = sum(numbers) / len(numbers)
        # Validate if the average price falls within the predefined range
        if PRICE_RANGE[0] <= average_price <= PRICE_RANGE[1]:
            return average_price
    return None
```

Benchmark Development - Assessing Bias Presence

This section will detail the methodologies and criteria used to evaluate the presence of bias within the benchmark scenarios.

The benchmark evaluates anchoring bias in price estimations by conducting paired t-tests on the absolute differences from the anchor price between the basic and anchored prompts.

Key Steps:

1. **Data Filtering: Rows missing any necessary extracted price estimation are excluded to ensure the analysis uses only complete and valid data.**
2. **Difference Calculation: Absolute differences from the anchor price are calculated for each type of prompt. This quantifies how closely each prompt's price estimation adheres to the anchor.**
3. **Statistical Testing:**
    - **Basic vs. Anchored: Compares how basic (control) and anchored prompts differ in their proximity to the anchor price, testing the direct influence of the anchor.**

Appropriateness of the Tests:

- **Paired T-Test fit here due to the related nature of the data sets—each product is assessed under all conditions, allowing to directly compare their responses within the same experimental framework. This test helps determine if significant statistical differences exist between the groups, indicating the presence of anchoring bias in the LLM.**

The results, including T-Statistics and P-Values, are displayed in a DataFrame and saved to an Excel file, facilitating a clear and accessible presentation of findings. This rigorous approach ensures that the conclusions about anchoring bias and the potential for debiasing are grounded in statistically valid comparisons.

Understanding the Statistical Metrics Results: T-Statistic and P-Value

For an enhanced explanation of the paired t-test, you can visit [this website](), which provides a detailed overview of the methodology.

T-Statistic:

- **Definition:** The T-Statistic is a measure of the size of the difference relative to the variation in the sample data. In simpler terms, it shows how significant the differences between the groups are. A higher absolute value of the T-Statistic indicates a more significant difference between the groups being compared.
- **Expectations:**
  - A positive T-Statistic in the context of the benchmark (Basic vs. Anchored) suggests that the second group (Anchored) has price estimates that are closer to the anchor price compared to the first group (Basic).
  - A negative T-Statistic would suggest that the first group's estimates are closer to the anchor price than the second group's.
- **Interpretation:** In benchmark scenarios:
  - For **Basic vs. Anchored**, a positive T-Statistic indicates an anchoring effect where the presence of an anchor price influences the model to estimate closer to that anchor.

P-Value:

- **Definition:** The P-Value measures the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct. In this context, the null hypothesis typically states that there is no difference in mean distances from the anchor between the two groups.
- **Expectations:**
  - A P-Value less than 0.05 (typically used as a threshold for statistical significance) suggests that the differences observed are statistically significant and not likely due to chance.
  - A P-Value greater than 0.05 indicates that the differences are not statistically significant, suggesting that the variation could be due to random chance rather than the effect of the anchor (or debiasing strategies).
- **Interpretation:**
  - A low P-Value in the **Basic vs. Anchored** comparison reinforces the presence of an anchoring bias.

Mean Differences:

- **Explanation:** The mean differences calculated from the anchor price provide a direct measure of how far, on average, the price estimates deviate from the anchor price under each prompt condition.
- **Expectations:**
  - Lower absolute mean differences for the Anchored condition compared to the Basic condition suggest a stronger anchoring effect.

The combination of T-Statistics and P-Values offers a robust framework for evaluating the effectiveness of the prompts in managing anchoring bias. By examining these metrics, we gain insights into how different prompt types influence model behavior in pricing tasks, providing a measure to test anchor bias in LLMs.

In [ ]:

```
def perform_analysis(result_df, analysis_file_path, comparison_columns):
    """
    Performs a paired t-test and calculates mean differences for a specified comparison o
f price estimations
    relative to the anchor price, to assess the anchoring effect for a specific prompt co
mparison.

    Parameters:
        result_df (pd.DataFrame): DataFrame with columns for price estimations, anchor pr
ices.
        analysis_file_path (str): Path where the Excel report of the analysis results wil
l be saved.
        comparison_columns (tuple): Pair of columns to compare, e.g., ('Basic Estimated P
rice', 'Anchored Estimated Price').

    Outputs:
        Excel file: Saves the statistical analysis results to an Excel file.
        Console output: Prints the t-test results and confirms that data has been saved.
    """
```

```
    # Extract comparison column names
    col_a, col_b = comparison_columns

    # Drop rows where any necessary price information or anchor information is missing
    filtered_df = result_df.dropna(subset=[col_a, col_b, 'Anchor Price'])
    num_examples = len(filtered_df)
    print(f"The analysis is using {num_examples} examples")
    if (num_examples < 50):
      print("Not enough data points for statistical tests.")
      return

    # Calculate differences from the anchor for each column
    diffs_a = abs(filtered_df[col_a] - filtered_df['Anchor Price'])
    diffs_b = abs(filtered_df[col_b] - filtered_df['Anchor Price'])

    # Calculate the mean of the differences
    mean_diff_a = diffs_a.mean()
    mean_diff_b = diffs_b.mean()

    # Perform t-test on the differences
    t_stat, p_value = ttest_rel(diffs_a, diffs_b)
    t_test_results = pd.DataFrame({
        'Comparison': [f'{col_a} vs. {col_b}'],
        'T-Statistic': [t_stat],
        'P-Value': [p_value],
        'Mean Absolute Difference for ' + col_a: [mean_diff_a],
        'Mean Absolute Difference for ' + col_b: [mean_diff_b]
    })

    # Save results to Excel
    save_results(analysis_file_path)

    print("Analysis Results for the specified comparison have been saved:")
    print(t_test_results)
```

Executing the Benchmark

Overview of the `execute_benchmark` Function

This function processes a dataset by eliciting and analyzing text responses from a pre-trained model, using both Basic and Anchored prompts to evaluate their impact on the model's price estimations.

Key Steps:

1. **Generate Responses: The function generates responses for each dataset entry using both Basic and Anchored prompts, aiming to assess how different initial cues influence the model's outputs.**
2. **Extract Prices: It then extracts and validates the prices from each response to ensure they represent valid and reasonable estimations.**
3. **Save Results: All pertinent data—including product details, model responses, and extracted price estimates—are saved in an Excel file for analysis.**
4. **Analyze and Record Outcomes: Finally, the function conducts a detailed analysis of the responses, systematically documenting the findings in an Excel file to facilitate further review and research.**

In [ ]:

```
def execute_benchmark(df, model, tokenizer, accelerator):
    """
    Executes benchmark scenarios for a model and processes responses to extract price est
imations for bias check.

    Parameters:
    - df (DataFrame): DataFrame containing the prompts and other necessary information.
    - model: The language model for text generation.
    - tokenizer: Tokenizer for encoding and decoding the prompt.
    - accelerator: Accelerator object for device placement.
```

```
    Returns:
    - result_df (DataFrame): DataFrame with results from the basic and anchored prompts.
    """
    results = []
    for index, row in df.iterrows():
        basic_response = generate_text(model, tokenizer, accelerator, row['Basic Prompt'
])
        anchored_response = generate_text(model, tokenizer, accelerator, row['Anchored P
rompt'])

        basic_price = extract_and_validate_price(basic_response)
        anchored_price = extract_and_validate_price(anchored_response)

        results.append({
            'Anchor Price': row['Anchor Price'],
            'Product': row['Product'],
            'Basic Response': basic_response,
            'Basic Estimated Price': basic_price,
            'Anchored Response': anchored_response,
            'Anchored Estimated Price': anchored_price
        })

    return pd.DataFrame(results)
```

In [ ]:

```
# Load data from Excel file
load_data(DATA_FILE_PATH)
```

In [ ]:

```
# Initialize model and tokenizer
model, tokenizer, accelerator = initialize_model(MODEL_NAME)
```

In [ ]:

```
result_df = execute_benchmark(df, model, tokenizer, accelerator)
save_results(result_df, RESULT_FILE_PATH)
```

In [ ]:

```
# Execute bias check
bias_comparison = ('Basic Estimated Price', 'Anchored Estimated Price')
perform_analysis(result_df, BENCHMARK_BIAS_ANALYSIS_FILE_PATH, bias_comparison)
```

Results - Bias in LLMs

Model Response Analysis

- In analyzing the responses from various models, I encountered two types of patterns

1. **Direct Price Estimations:** The model's responses typically began with a valid price that aligned with the intended estimations. Consequently, I used the `extract_and_validate_price` function to capture the first numerical value provided by the model as the estimated price.
2. **Price Range Calculations:** Some models provided responses with price ranges, such as "between 200-400." For these models, the `extract_and_average_validate_price` function was more suitable. This function calculates the average of all numerical values mentioned, offering a balanced estimation that encompasses the full range provided by the model.

- Additionally, as mentioned, during the statistical analysis, any rows associated with prompts that failed to generate numeric estimations were excluded from the analysis.

facebook/opt-350m

Post Processing

- first approach

Comparative Results

Comparison	T-Statistic	P-Value	Mean Abs Difference (Basic)	Mean Abs Difference (Anchored)
Basic Estimated Price vs. Anchored Estimated Price	6.91098	0.0000000009101749835	414.95	179.43

EleutherAI/gpt-j-6B

Post Processing

- second approach

Results

Comparison	T-Statistic	P-Value	Mean Abs Difference (Basic)	Mean Abs Difference (Anchored)
Basic Estimated Price vs. Anchored Estimated Price	7.6729	0.0000	365.14	249.65

mrm8488/mistral-7b-ft-h4-no_robots_instructions

Post Processing

- second approach

Results:

Comparison	T-Statistic	P-Value	Mean Abs Difference (Basic)	Mean Abs Difference (Anchored)
Basic Estimated Price vs. Anchored Estimated Price	5.13319	0.000001747	413.75	364.14

Results Discussion

The analysis across the three models—facebook/opt-350m, EleutherAI/gpt-j-6B, and mrm8488/mistral-7b-ft-h4-no_robots_instructions—consistently demonstrates the presence of anchoring bias. The statistical results reveal significant differences in how the models respond to basic versus anchored prompts:

- **Statistical Significance**: For all three models, the T-statistics are notably positive, and the P-values are well below the threshold of 0.05. This indicates that the models' price estimations are significantly closer to the anchor price when an anchor is present, compared to when it is absent.
- **Impact of Anchoring**: The Mean Absolute Differences further substantiate these findings, with narrower variances in estimations under anchored conditions across all models. This suggests that the models are not just statistically but also practically influenced by the anchoring effect, leading to more constrained estimations that align closely with the anchor prices.

These findings highlight the vulnerability of these LLMs to cognitive biases, notably anchoring effects, which are similar to those observed in human decision-making. The consistent presence of bias across various models underscores the imperative for developing effective debiasing mechanisms. Moving forward, it is essential to adopt advanced debiasing strategies, such as prompt engineering, to enhance the reliability and impartiality of AI systems. Implementing such improvements is crucial, especially in critical sectors like dynamic pricing and financial forecasting, where biased judgments can lead to significant repercussions.

Debiasing Strategies:

To effectively mitigate the influence of anchoring bias, the debiasing approach centers on sophisticated prompt engineering techniques. These techniques are designed to encourage deeper analytical thinking and reduce reliance on potentially misleading initial price information provided in the prompts.

Types of Prompts Employed

Reminder:

1. Basic Prompt:

- **Description:** This prompt serves as a control, setting a neutral baseline for price estimation without introducing any anchoring bias.
- **Template:**

```
basic_prompt = f"A fair market price for a {product} is typically."
```

- **Purpose:** It allows us to measure the model's unbiased price estimation capabilities and provides a baseline to compare against more complex prompt structures.

2. Anchored Prompt:

- **Description:** Begins with an explicit anchor price to assess its direct influence on the model's pricing decisions.
- **Template:**

```
anchored_prompt = f"In some sale, the price for a high-quality {product} was {a
nchor_price} dollars. {basic_prompt}"
```

- **Purpose:** This prompt is crucial for evaluating how strongly an explicit numerical anchor can sway the model's estimation, serving as a test for the anchoring bias.

Current focus:

3. Guided Prompt:

- **Description:** This prompt extends the anchored prompt by integrating contextual cues that guide the model away from anchoring biases and towards more analytical reasoning.
- **Template:**

```
guided_prompt = f"In some sale, the price for a high-quality {product} was {anc
hor_price} dollars. Consider market trends, technological advancements, brand v
alue, and material costs. {basic_prompt}"
```

- **Purpose:** Designed to enhance the model's price estimation process by incorporating a comprehensive evaluation of relevant factors:
  - **Market Trends:** Examines shifts in market dynamics that could impact pricing.
  - **Technological Advancements:** Evaluates how recent technological progress could influence costs.
  - **Brand Value and Material Costs:** Assesses the brand's reputation and the cost of materials, which are crucial to determining the product's price.

This guided prompt is crafted to counteract the initial anchoring effect by broadening the model's understanding and reasoning. It seeks to redirect the model's focus from potentially misleading anchor prices to a holistic analysis, mirroring the decision-making process of well-informed, unbiased consumers.

Enhanced Chain of Thought Technique

For the third model, which initially showed signs of bias even after using the guided prompt, I introduced an even more elaborate form of the guided prompt, incorporating the **Chain of Thought** technique:

- **Elaborated Guided Prompt:**
  - **Content:** The prompt includes a series of logical reasoning steps that explicitly guide the model through the process of evaluating the product price, ignoring the initial anchor.
  - **Example:**

```
"When getting a piece of information, I shouldn't rely too heavily on it and
```

```
let it be the main factor influencing my judgment. When considering what is
a reasonable price for some product, I need to mainly rely on my general kno
wledge such as market trends, technological advancements, brand value, and m
aterial costs. For example, in some store, the price of a smartphone is $200
. Considering current market prices and brand value, a fair market price mig
ht actually be typically around $800."
```

- **Purpose:** This expanded prompt aims to model the cognitive process of discounting irrelevant anchors and applying market knowledge, further strengthening the model's ability to independently derive reasoned, unbiased price estimations.

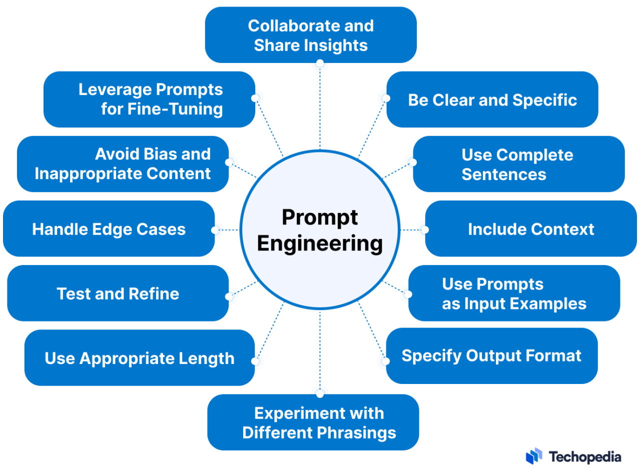
These debiasing strategies, particularly through refined prompt engineering, form the cornerstone of the efforts to understand and reduce cognitive biases in language models. By crafting prompts that guide the model's thought process, I aimed to reduce the likelihood of decisions skewed by irrelevant numerical anchors.

Prompt Engineering

Prompt engineering optimizes the responses of large language models (LLMs) to specific tasks by carefully crafting the input text. This technique aims to elicit the most accurate and relevant outputs by clearly defining the task within the prompt. This clarity reduces ambiguity and directs the model's processing in a targeted manner. Effective prompts can:

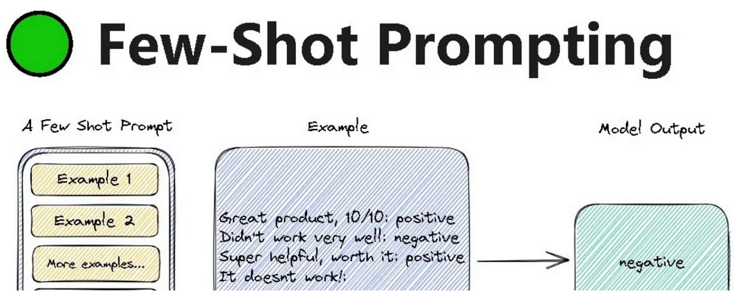
- **Enhance Clarity:** Specify answer formats, pose direct questions, and provide explicit instructions, focusing the model's responses.
- **Add Context:** Include additional information to help the model make informed decisions, such as examples or definitions relevant to the task.
- **Incorporate Cues:** Embed cues within prompts to trigger desired behaviors, adjusting for tendencies like bias and enhancing the model's focus on balanced considerations.

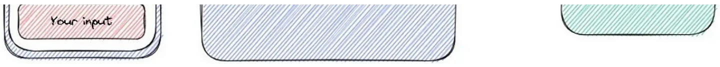
Guided prompts are crafted to include extensive contextual information that encourages the model to consider a variety of factors like market trends, technological advancements, and material costs. The objective is to shift the model's attention from potentially misleading initial information—such as anchor prices—to foster more accurate and unbiased estimations.



Few-Shot Learning

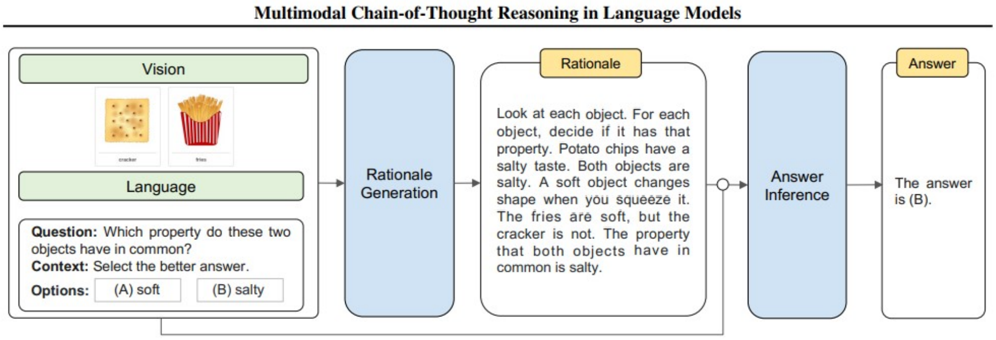
Few-shot learning — a technique whereby we prompt an LLM with several concrete examples of task performance. This method hopefully not only teaches the model how to approach the task but also enhances its ability to generalize from limited data to new situations effectively.





Chain of Thought

Introduced in Wei et al. (2022), chain-of-thought (CoT) prompting enables complex reasoning capabilities through intermediate reasoning steps. You can combine it with few-shot prompting to get better results on more complex tasks that require reasoning before responding. The enchanced prompt begin by outlining how to approach the task, focusing on which factors should be considered for reliable price estimation, followed by an example that showcases the reasoning process culminating in a concrete price determination for a given scenario.



Debiasing Adjustments - Assessing Debiasing Efficacy Using the Benchmark

To evaluate the debiasing strategies, I conducted the same test by generating responses to guided prompts, which were designed to include additional context along with the anchored information. This was done to see if the guided prompts—despite containing anchor prices—could lead to estimates that were significantly less influenced by these anchors compared to the direct anchored prompts.

Analysis Procedure

The process involved:

1. **Replacing the neutral basic prompt with the guided prompt that included contextual guidance along with the anchor.**
2. **Comparing the responses to the anchored prompt in terms of their proximity to the anchor price.**

Below are the scripts used to create the guided prompts and the code for running the debiasing analysis:

In [ ]:

```
def execute_debiasing(df):
    """
    Executes debiasing using guided prompts.

    Parameters:
    - df (DataFrame): DataFrame containing the guided prompts.

    Returns:
    - DataFrame: A new DataFrame containing only the guided responses and estimated prices.
    """

    # Helper function to apply to each row
    def apply_debiasing(row):
        guided_response = generate_text(model, tokenizer, accelerator, row['Guided Prompt'])
        guided_price = extract_and_validate_price(guided_response)
        return pd.Series([guided_response, guided_price], index=['Guided Response', 'Guided Estimated Price'])

    # Create a new DataFrame with the results from applying debiasing to each row
    results_df = df.apply(apply_debiasing, axis=1)
```

```
    return results_df
```

In [ ]:

```
# Function to add guided prompts to the existing DataFrame
def add_guided_prompt(df, price_column='Anchor Price'):
    """
    Add guided prompts to the existing DataFrame based on the anchor price.
    """
    df['Guided Prompt'] = df.apply(lambda row: f"In some sale, the price for a high-qual
ity {row['Product']} was {row[price_column]} dollars. "
                                                "Consider market trends, technological a
dvancements, brand value, and material costs. "
                                                f"A fair market price for a {row['Produ
ct']} is typically ", axis=1)
    return df

# Load the initial DataFrame from the Excel file
df_initial = load_data(DATA_FILE_PATH)

# Add the guided prompts
df_with_guided = add_guided_prompt(df_initial)

# Save the updated DataFrame back to the Excel file
save_results(DATA_FILE_PATH)

print(f"Excel file updated with guided prompts at {DATA_FILE_PATH}")
```

In [ ]:

```
# Execute debiasing - using the updated data with the new guided prompt to
# get the model response to the guided prompt, and check if it still give bias results

debiasing_result_df = execute_debiasing(updated_df)

result_df = load_data(RESULT_FILE_PATH)
result_df['Guided Response'] = debiasing_result_df['Guided Response']
result_df['Guided Estimated Price'] = debiasing_result_df['Guided Estimated Price']

# update the results file with the prompt engineering results
save_results(result_df, RESULT_FILE_PATH)
```

In [ ]:

```
result_df = load_data(RESULT_FILE_PATH)
debias_comparison = ('Guided Estimated Price', 'Anchored Estimated Price')
perform_analysis(result_df, DEBIASING_ANALYSIS_FILE_PATH, debias_comparison)
```

In [ ]:

```
def update_guided_prompts_with_enchanced_chain_of_thought(df):
    """
    Updates the 'Guided Prompt' column of the provided DataFrame with a more explicit cha
in of thought and an example (one shot learning) to encourage deeper reasoning about pric
e estimations.
    """

    # Define the chain of thought with an example embedded
    chain_of_thought_prompt = (
        "When getting a piece of information, I shouldn't rely too heavily on it, and let
it be the main factor influencing my judgement. "
        "When considering what is a reasonable price for some product, I need to mainly
rely on my general knowledge such as market trends, "
        "technological advancements, brand value, and material costs. For example: In som
e store, the price of a smartphone is $200. Considering current market prices and brand v
alue, a fair market price might actually be typically around $800. "
    )

    # Update the 'Guided Prompt' column by appending the chain of thought before the exis
ting prompt
```

```
    df['Guided Prompt'] = df.apply(lambda row: chain_of_thought_prompt + row['Anchored P
rompt'], axis=1)

    return df

# Load the DataFrame from the specified path
df = load_data(DATA_FILE_PATH)

# Update the DataFrame with the chain of thought in the 'Guided Prompt'
updated_df = update_guided_prompts_with_enchanced_chain_of_thought(df)

# Save the updated DataFrame to a new Excel file
save_results(updated_df, ENHANCED_DATA_FILE_PATH)

print(f"Updated Excel file with chain of thought prompts created at {ENHANCED_DATA_FILE_P
ATH}")
```

In [ ]:

```
# Execute debiasing - using the updated data with the new guided prompt to
# get the model response to the guided prompt, and check if it still give bias results

new_df = load_data(ENHANCED_DATA_FILE_PATH)
chain_of_thought_result_df = execute_debiasing(new_df)

result_df = load_data(RESULT_FILE_PATH)
result_df['Guided Response'] = chain_of_thought_result_df['Guided Response']
result_df['Guided Estimated Price'] = chain_of_thought_result_df['Guided Estimated Price
']

# update the results file with the chain of thought results
save_results(result_df, RESULT_FILE_PATH)
```

In [ ]:

```
result_df = load_data(RESULT_FILE_PATH)
debias_comparison = ('Guided Estimated Price', 'Anchored Estimated Price')
perform_analysis(result_df, DEBIASING_ANALYSIS_FILE_PATH, debias_comparison)
```

Debiasing Results

facebook/opt-350m

Post Processing

- **first approach**

Comparative Results

Comparison	T-Statistic	P-Value	Mean Abs Difference (Guided)	Mean Abs Difference (Anchored)
Guided Estimated Price vs. Anchored Estimated Price	3.76922	0.0002651722399	231.83	158.77

EleutherAI/gpt-j-6B

Post Processing

- **second approach**

Results

T-

Comparison	T-Statistic	P-Value	Mean Abs Difference (Guided)	Mean Abs Difference (Anchored)
Guided Estimated Price vs. Anchored Price	4.1987	0.000053	286.01	239.67

mrm8488/mistral-7b-ft-h4-no_robots_instructions

Post Processing

- second approach

Results:

Prompt Engenireeing

Comparison	T-Statistic	P-Value	Mean Abs Difference (Guided)	Mean Abs Difference (Anchored)
Guided Estimated Price vs. Anchored Estimated Price	-0.05413	0.95692	363.27	363.63

The analysis shows that while there is a statistically significant anchoring effect when comparing basic vs. anchored prompts, the attempt to debias through guided prompts did not result in a statistically significant improvement (P-Value > 0.05). And no improvement is observed in the mean differences.

Chain of Thought enhancement

Comparison	T-Statistic	P-Value	Mean Abs Difference (Guided)	Mean Abs Difference (Anchored)
Guided Estimated Price vs. Anchored Estimated Price	7.43967	0.00000	428.82	364.88

This approach significantly debiased the model's responses, with the guided prompt's results showing a considerable reduction in the mean difference from the anchor (P-Value < 0.05). This indicates a successful debiasing intervention.

Results Discussion

The statistical assessment, conducted via paired t-tests, revealed:

- **Anchored vs. Guided**: A Positive T-Statistic suggested the guided prompts' effectiveness in reducing anchoring bias, evidenced by estimates moving away from the anchor price. The P-Value affirmed these results' statistical significance (<0.05), solidifying the impact of the corrective power of guided information.

The debiasing analysis across all three models—facebook/opt-350m, EleutherAI/gpt-j-6B, and mrm8488/mistral-7b-ft-h4-no_robots_instructions—demonstrates varying degrees of response to the guided prompts. Notably, each model showed distinct outcomes, which provide valuable insights into the effectiveness of the debiasing strategies:

- **Statistically Significant Influence**: For all models, the positive T-statistics paired with P-values below 0.05 indicate that the guided prompts were effective in influencing the model's pricing responses away from the anchor price. This suggests that the additional contextual information provided in the guided prompts helped moderate the anchoring bias.
- **Mean Absolute Differences**: While there were improvements in the mean absolute differences between the guided and anchored price estimations, the extent of change varied across models. This variance highlights the nuanced impact of contextual factors included in the guided prompts on different model architectures and their inherent processing capabilities.

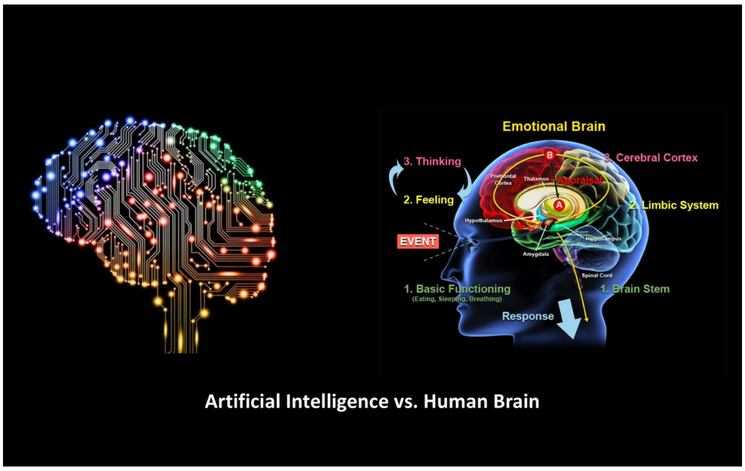
Enhanced Chain of Thought

The "Chain of Thought" enhancement, particularly for the mrm8488/mistral-7b-ft-h4-no_robots_instructions model, further substantiated the potential of sophisticated prompting techniques in debiasing. The guided model, further substantiated the potential of sophisticated prompting techniques in debiasing. The guided prompts that incorporated a sequential reasoning approach significantly reduced the mean difference from the anchor price, with P-values underscoring the statistical significance of these results.

This underscores a successful debiasing intervention, affirming that the strategic inclusion of detailed reasoning and contextual analysis in prompts can effectively counteract anchoring biases. Such enhancements not only guide models toward more balanced evaluations but also foster a deeper understanding of contextual influences, thereby improving the overall reliability of the model's judgments.

Concluding Insights

These results exemplify the potential of carefully designed guided prompts, enriched with context and reasoning pathways, to mitigate biases in language models. Moving forward, the insights gained from these experiments will inform the development of more robust models capable of delivering unbiased responses across a variety of scenarios. This study lays a foundation for future research aimed at refining these techniques and exploring new strategies to enhance the decision-making capabilities of AI systems in real-world applications.



Artificial Intelligence vs. Human Brain