EXP 3: Evaluation of Prompting Tools Across Diverse AI Platforms: ChatGPT, Claude, Bard, Cohere Command, and Meta

Aim:

- To compare the performance, user experience, and response quality of different AI platforms (ChatGPT, Claude, Bard, Cohere Command, and Meta) within a specific use case, such as summarizing text or answering technical questions.

Procedure:

1. **Define Use Cases:** Select a variety of text samples and technical questions.
2. **Create Evaluation Criteria:** Establish criteria for assessing accuracy, relevance, completeness, coherence, fluency, speed, and user experience.
3. **Conduct Experiments:** Input prompts into each AI platform, record responses, and evaluate them against the criteria.
4. **Data Analysis and Reporting:** Analyze the collected data, identify trends, and prepare a comprehensive report.
5. **Additional Considerations:** Consider ethical implications, contextual understanding, and user feedback.

PROMPT	CHATGPT	BLACK BOX
Write a code using Python to increase accuracy for hate speech detection using the BERT embeddin g model.	import pandas as pd import numpy as np from sklearn.model_selection import train_test_split from sklearn.metrics import classification_report, accuracy_score import torch from transformers import BertTokenizer, BertModel from sklearn.linear_model import LogisticRegression from sklearn.pipeline import make_pipeline from sklearn.preprocessing import StandardScaler	import torch from torch import nn from torch.utils.data import DataLoader, Dataset from transformers import BertTokenizer, BertModel, AdamW, get_linear_schedule_with_warmup from sklearn.model_selection import train_test_split from sklearn.metrics import accuracy_score, classification_report import pandas as pd # Load the dataset df = pd.read_csv('hate_speech.csv')

```
# Load dataset

data = pd.read_csv('hate_speech_data.csv')  # Replace with your dataset path

data['label'] = data['label'].map({'hate': 1, 'not_hate': 0})  # Convert labels to binary



# Split the dataset

X = data['text']

y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Load BERT tokenizer and model

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

model = BertModel.from_pretrained('bert-base-uncased')



# Function to get BERT embeddings

def get_bert_embeddings(texts):

    encoded_inputs = tokenizer(texts.tolist(), padding=True, truncation=True, return_tensors='pt', max_length=512)
```

```
# Split the data into training and validation sets

train_texts, val_texts, train_labels, val_labels = train_test_split(df['text'], df['label'], test_size=0.2, random_state=42)



# Load a pre-trained BERT model and tokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

model = BertModel.from_pretrained('bert-base-uncased')



# Create a custom dataset class for text classification

class TextClassificationDataset(Dataset):

    def __init__(self, texts, labels, tokenizer, max_length):

        self.texts = texts

        self.labels = labels

        self.tokenizer = tokenizer

        self.max_length = max_length



    def __len__(self):

        return len(self.texts)



    def __getitem__(self, idx):

        text = self.texts[idx]
```

	with torch.no_grad():
	
	outputs = model(**encoded_inputs)
	
	# We can use the output from the [CLS] token as the sentence embedding
	
	return outputs.last_hidden_state[:, 0, :].numpy()
	
	# Get BERT embeddings for training and test sets
	
	X_train_embeddings = get_bert_embeddings(X_train)
	
	X_test_embeddings = get_bert_embeddings(X_test)
	
	# Train a classifier (Logistic Regression in this case)
	
	classifier = LogisticRegression(max_iter=1000) # You can tune this or choose a different model
	
	classifier.fit(X_train_embeddings, y_train)
	
	# Make predictions
	
	y_pred = classifier.predict(X_test_embeddings)
	
	# Evaluate the model
	
	print("Accuracy:",
		label = self.labels[idx]
		
		encoding = self.tokenizer(text, return_tensors='pt', max_length=self.max_length, padding='max_length', truncation=True)
		
		return {'input_ids': encoding['input_ids'].flatten(), 'attention_mask': encoding['attention_mask'].flatten(), 'label': torch.tensor(label)}
		
		# Create a data loader for the dataset
		
		train_dataset = TextClassificationDataset(train_texts, train_labels, tokenizer, max_length=128)
		
		val_dataset = TextClassificationDataset(val_texts, val_labels, tokenizer, max_length=128)
		
		train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
		
		val_dataloader = DataLoader(val_dataset, batch_size=16)
		
		# Define a BERT classifier model
		
		class BERTClassifier(nn.Module):
		
		def __init__(self, bert_model_name, num_classes):
		
		super(BERTClassifier, self).__init__()
		
		self.bert = BertModel.from_pretrained(bert_model_name)
		
		self.dropout = nn.Dropout(0.1)
		
		self.fc = nn.Linear(self.bert.config.hidden_size,

	accuracy_score(y_test, y_pred))
	
	print(classification_report(y_test, y_pred))
		num_classes)
		
		def forward(self, input_ids, attention_mask):
		
		outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
		
		pooled_output = outputs.pooler_output
		
		x = self.dropout(pooled_output)
		
		logits = self.fc(x)
		
		return logits
		
		# Define a function to train the model
		
		def train(model, data_loader, optimizer, scheduler, device):
		
		model.train()
		
		for batch in data_loader:
		
		optimizer.zero_grad()
		
		input_ids = batch['input_ids'].to(device)
		
		attention_mask = batch['attention_mask'].to(device)
		
		labels = batch['label'].to(device)
		
		outputs = model(input_ids=input_ids, attention_mask=attention_mask)
		
		loss = nn.CrossEntropyLoss()(outputs, labels)
		
		loss.backward()
		
		optimizer.step()
		
		scheduler.step()

```
# Define a function to evaluate the model

def evaluate(model, data_loader, device):

    model.eval()

    predictions = []

    actual_labels = []

    with torch.no_grad():

        for batch in data_loader:

            input_ids =
batch['input_ids'].to(device)

            attention_mask =
batch['attention_mask'].to(device)

            labels = batch['label'].to(device)

            outputs = model(input_ids=input_ids,
attention_mask=attention_mask)

            _, preds = torch.max(outputs, dim=1)


predictions.extend(preds.cpu().tolist())


actual_labels.extend(labels.cpu().tolist())

    return accuracy_score(actual_labels,
predictions),
classification_report(actual_labels,
predictions)



# Train the model

device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")

model = BERTClassifier('bert-base-uncased',
2).to(device)

optimizer = AdamW(model.parameters(),
```

		lr=2e-5)
		
		total_steps = len(train_dataloader) * 4
		
		scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)
		
		for epoch in range(4):
		
		  print(f"Epoch {epoch + 1}/4")
		
		  train(model, train_dataloader, optimizer, scheduler, device)
		
		  accuracy, report = evaluate(model, val_dataloader, device)
		
		  print(f"Validation Accuracy: {accuracy:.4f}")
		
		  print(report)

Conclusion:

Comparing the above two code, the black box code is more accurate than chatgpt. Hence Blackbox.ai is better AI than chatgpt

JANANI M

212222050021