RESEARCH PAPER IMPLEMENTATION - CASE STUDY

Problem Statement and Objectives

Problem Statement:- To select a research paper from a reputable source and implement it on their own. I have selected a research paper titled “A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges” from IEEE Xplore.

Objective:- The primary objective is to implement the research paper by explaining Large Language Models in simple terms along with practical implementations through programming the models, so that learners can understand the concepts of Large Language Models with ease and implement one or two on their own.

Implementation of Large Language Models

Data Pre-Processing

The data that was taken during the implementation of Large Language Models was mostly text-based data. I have pre-processing techniques such as converting text into tokens, mapping characters to indexes, tensors and vectors, converting tokens into IDs, etc.

Model Selection and Development

The models that are selected for implementation are:

1. Phi-2 by Microsoft

   It is a pre-trained model developed by Microsoft, which I have used for Question-Answering Conversation. It takes the user’s prompt as input and converts into tokens and generates continuing text from the prompt, answering the question.

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

model_name = "microsoft/phi-2"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

prompt = input("Enter your prompt: ")
inputs = tokenizer(prompt, return_tensors="pt")
outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task % python3 phi2_model.py
Loading checkpoint shards: 100%|| 2/2 [00:15<00:00,  7.79s/it]
Enter your prompt: what is the capital of India?
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
The capital of India is New Delhi.
```

2. BERT (Bidirectional Encoder Representation from Transformers) by Google

It is a pre-trained model developed by Google, which I have used for Text Summarization. It takes a long paragraphed-based text as an input, and summarizes the paragraph into two-lined text.

```
from transformers import pipeline

# Load pre-trained summarization pipeline
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")

# Your long text input
text = """
Language models are a key component of many modern NLP systems. They are trained to predict the next word in a
sequence given the previous words, enabling applications such as text generation, machine translation, and question
answering.
Recent advancements in transformer-based models like BERT and GPT have significantly pushed the boundaries of what
language models can achieve, thanks to their ability to model long-range dependencies and large-scale training on
massive datasets.
"""

# Get the summary
summary = summarizer(text, max_length=60, min_length=25, do_sample=False)

# Print the summary
print("Summary:", summary[0]['summary_text'])
```

```
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task % python3 bert.py
config.json: 100%|| 1.80k/1.80k [00:00<00:00, 2.36MB/s]
pytorch_model.bin: 100%|| 1.22G/1.22G [00:32<00:00, 37.3MB/s]
tokenizer_config.json: 100%|| 26.0/26.0 [00:00<00:00, 114kB/s]
vocab.json: 100%|| 899k/899k [00:00<00:00, 1.39MB/s]
merges.txt: 100%|| 456k/456k [00:00<00:00, 1.88MB/s]
model.safetensors:  11%|| 136M/1.22G [00:03<00:27, 39.4MB/s]
Device set to use mps:0
model.safetensors:  14%|| 168M/1.22G [00:04<00:29, 35.3MB/s]
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/transformers/pytorch_utils.py:329: UserWarning: To copy construct from a tensor, it is recommend
ed to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  test_elements = torch.tensor(test_elements)
model.safetensors:  53%|| 650M/1.22G [00:17<00:15, 36.0MB/s]
Summary:  Language models are a key component of many modern NLP systems . They are trained to predict the next word in a sequence given the previous words . Transformer-based
models like BERT and GPT have pushed the boundaries of what language models can achieve .
model.safetensors: 100%|| 1.22G/1.22G [00:34<00:00, 35.9MB/s]
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task %
```

3. Llama 3.2:1b by Meta

It is a Large Language Model that runs on local machine, containing 1 Billion parameters, which i have used for Conversational-based interaction. The model takes the user's prompt as input, sends as a chat-style message to the model, and the model generates output based on the listed conversations.

```
import ollama

model_name = "llama3.2:1b"
prompt = input("Enter your prompt: ")

response = ollama.chat(
    model=model_name,
    messages=[{"role": "user", "content": prompt}]
)

print("AI Response:", response["message"]["content"])
```

```
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task % python3 llama_chat.py
AI Response: Agent AI, also known as autonomous agent or artificial general intelligence (AGI), refers to a type of artificial intelligence that can act independently and make
decisions without being explicitly programmed.

Think of an agent like a robot or a computer program that can:

* Learn from experience
* Understand its environment and the rules that govern it
* Make choices and take actions based on those choices

In other words, an agent AI is a self-contained system that can navigate through different situations, make decisions, and interact with its environment without needing to be e
xplicitly told what to do.

For example, imagine a chatbot that can play chess against a human opponent. The chatbot has been trained on millions of games and can learn from experience to improve its chan
ces of winning. It doesn't need explicit instructions or directions to make moves; it's like the AI is an agent that can act autonomously in the game of chess.

The goal of developing Agent AI is to create intelligent systems that can:

* Learn, adapt, and generalize
* Reason and problem-solve
* Interact with humans and other agents

Agent AI has many potential applications, including robotics, virtual assistants, autonomous vehicles, and more.
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task % 
```

4. TinyLLM - A model built from scratch

This is a tiny-level character-level language that I have built for Text generation using Python. The input text is converted into a list of characters using Encoder and Decoder, processed through Embedding and Linear layers of the model with Adam optimization, Cross entropy, trained for a 1000 steps and the long sequence of characters is generated with Softmax activation function

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

# === Data Setup ===
text = "hello world! welcome to the world of language models."
chars = sorted(list(set(text)))
vocab_size = len(chars)

char_to_idx = {ch: i for i, ch in enumerate(chars)}
idx_to_char = {i: ch for i, ch in enumerate(chars)}

def encode(s): return [char_to_idx[ch] for ch in s]
def decode(l): return ''.join([idx_to_char[i] for i in l])

data = torch.tensor(encode(text), dtype=torch.long)

# === Model Definition ===
class TinyLLM(nn.Module):
    def __init__(self, vocab_size, embed_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.linear = nn.Linear(embed_dim, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        x = self.linear(x)
        return x

model = TinyLLM(vocab_size, embed_dim=32)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)
loss_fn = nn.CrossEntropyLoss()

# === Training Setup ===
block_size = 4

def get_batch():
    ix = torch.randint(0, len(data) - block_size - 1, (1,))
    x = data[ix:ix + block_size]
    y = data[ix + 1:ix + block_size + 1]
    return x.unsqueeze(0), y.unsqueeze(0)

# === Training Loop ===
for step in range(1000):
    x_batch, y_batch = get_batch()
    logits = model(x_batch)
    loss = loss_fn(logits.view(-1, vocab_size), y_batch.view(-1))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if step % 100 == 0:
        print(f"Step {step}, Loss: {loss.item():.4f}")

# === Text Generation ===
context = torch.tensor([char_to_idx['h']], dtype=torch.long).unsqueeze(0)

for _ in range(100):
    logits = model(context)
    last_logits = logits[0, -1]
    probs = F.softmax(last_logits, dim=0)
    next_char_idx = torch.multinomial(probs, num_samples=1)
    context = torch.cat([context, next_char_idx.unsqueeze(0)], dim=1)

print("Generated text:", decode(context[0].tolist()))
```

```
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task % python3 tinyllm.py
Step 0, Loss: 3.5380
Step 100, Loss: 1.3157
Step 200, Loss: 1.6961
Step 300, Loss: 1.2366
Step 400, Loss: 0.6637
Step 500, Loss: 0.4957
Step 600, Loss: 1.6336
Step 700, Loss: 0.6944
Step 800, Loss: 0.9221
Step 900, Loss: 0.4093
Generated text: helanguaguahe melo worlange taguagelanguange wo the wele to corlanguanguaguanguangelanguanguanguaguag
sachinkarthikeya@Vs-MacBook-Pro-2 Megaminds Task %
```

Recommendation and Conclusion

From the above implementations, we can say that further improvements and advancements can be performed on the existing Large Language Models, which can be suitable for complex tasks such as Translation, Information Retrieval, etc.

As of now, the aim is to help upcoming learners understand the basic architectures and applications of the Large Language Models, so that they can learn the basics with ease and improve their knowledge and implementations in their LLM developing further.