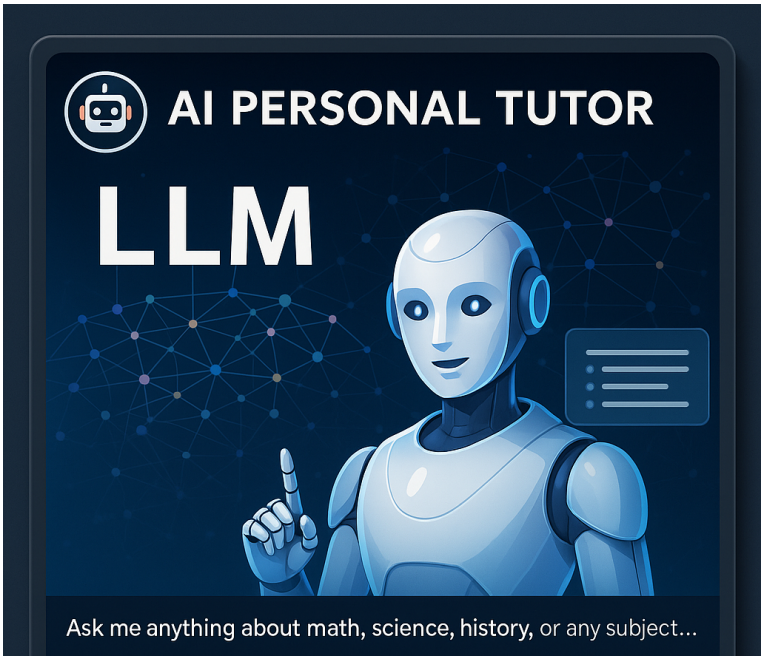


Personal Project

AI Personal Tutor using Mistral-7B

Réalisé par :

El Akramine Yassir Salim

April 2025

Contents

1 Introduction

1.1 Summary

The Mistral-7B-Instruct model runs using PyTorch, a popular machine learning framework. PyTorch handles the computations, especially for deep learning, by utilizing CUDA to accelerate processing on the GPU. The model and tokenizer come from Hugging Face, which provides pre-trained models like Mistral, making it easier to load and run without needing to train it from scratch. When you send input to the chatbot, the tokenizer breaks it into tokens, the model processes it, and PyTorch computes the response using your GPU, which stores and handles the model's data in VRAM for faster performance.

1.2 PC Requirements for Running the Project

To run the Mistral-7B model locally, certain hardware and software requirements must be met to ensure optimal performance.

1.2.1 GPU Requirements

Running the Mistral-7B model locally requires a **GPU with at least 8GB of VRAM** for feasible performance. Specifically, an **NVIDIA GPU** with CUDA support is necessary to leverage hardware acceleration during inference. The following GPUs are suitable for running the model:

- NVIDIA RTX 3060 (12GB VRAM)
- NVIDIA RTX 3070 (8GB VRAM)
- NVIDIA RTX 4080 (16GB VRAM)
- NVIDIA A100 (40GB VRAM)
- NVIDIA Tesla V100 (16GB VRAM)

Note: Although **8GB VRAM** is sufficient for running the model with **8-bit quantization**, **16GB VRAM** is recommended for optimal performance, especially when dealing with larger batch sizes or extended sequence lengths. For smaller GPUs, such as those with **4GB VRAM**, the model may struggle to run without significantly reducing performance or precision.

1.2.2 Alternatives for Limited Hardware

If your hardware does not meet the GPU requirements or if your GPU has less than 8GB VRAM, here are a few alternatives:

- **Quantization**: Using **4-bit or 8-bit quantization** significantly reduces the memory footprint of the model, enabling it to run on GPUs with lower VRAM (e.g., 4GB or 6GB).
- **Cloud Deployment**: For users with limited local resources, **cloud platforms** such as **Google Colab**, **AWS EC2**, or **Azure** provide access to GPUs with higher VRAM and computational power. You can deploy the model in the cloud and access it remotely for inference.
- **Model Optimization**: Techniques such as **model parallelism** or **offloading** parts of the model to the CPU can help distribute the computation load, though they may slow down processing.

With these alternatives, you can still run and experiment with the Mistral-7B model even on hardware with more limited resources.

1.3 Technology Choices

In this project, several key technologies were chosen based on the requirements of building a powerful, efficient, and scalable AI chatbot. Below is an explanation of why each technology was selected.

1.3.1 Mistral-7B-Instruct

Mistral-7B-Instruct was chosen because it is a fine-tuned version of the Mistral-7B base model, optimized for instruction-based tasks. It is specifically designed to understand and generate responses in natural language, making it ideal for building a conversational AI. The model offers:

- **High Performance:** Mistral-7B-Instruct is a large-scale model that can process complex inputs with high accuracy.
- **Instruct Tuning:** This version of Mistral has been fine-tuned for better understanding of instructions, making it more effective in handling a wide range of user queries.
- **State-of-the-Art Tokenizer:** It uses the latest tokenizer, ensuring efficient text processing and understanding.

1.3.2 PyTorch

PyTorch was selected as the deep learning framework because of its flexibility, efficiency, and ease of use. Key reasons for its selection include:

- **Dynamic Computation Graphs:** PyTorch allows for dynamic computation graphs, which makes it easier to modify and debug models during training and inference.
- **CUDA Support:** PyTorch provides seamless integration with CUDA for GPU acceleration, making it ideal for running large models like Mistral-7B on GPUs.
- **Community and Support:** PyTorch has an active community and extensive documentation, making it easier to find resources and solutions to problems.

1.3.3 Hugging Face Transformers

Hugging Face's Transformers library was chosen for its easy-to-use API and vast collection of pre-trained models, including Mistral. The library simplifies model loading, tokenization, and inference. Key benefits include:

- **Pre-trained Models:** Hugging Face provides access to a wide variety of pre-trained models, including Mistral-7B-Instruct, which saves significant time and resources in training.
- **Simplified Integration:** The Transformers library integrates well with PyTorch, allowing for seamless processing and inference.
- **Community and Ecosystem:** Hugging Face's ecosystem offers tools like Model Hub, datasets, and optimizers that further enhance development.

1.3.4 Gradio

Gradio was used to create the user interface for the chatbot. The reasons for choosing Gradio include:

- **Ease of Use:** Gradio makes it simple to create interactive web applications with minimal code, allowing for rapid prototyping and deployment.
- **Integration with ML Models:** Gradio integrates seamlessly with models built using PyTorch, making it ideal for displaying the results of the Mistral-7B-Instruct model.
- **Real-time Interaction:** It enables real-time interactions with the model, offering users an intuitive chat interface.

1.3.5 CUDA and GPU Usage

CUDA is used to leverage the GPU for processing large models like Mistral-7B-Instruct. The choice to use GPU acceleration ensures:

- **Faster Inference:** Running the model on a GPU speeds up computations, making the chatbot responsive even when processing large inputs.
- **Efficient Memory Management:** CUDA optimizes memory usage, allowing for better performance and less strain on system resources.

2 Environment Setup

2.1 Prepare WSL2 Environment

```
# In Windows PowerShell (as Administrator)
wsl --install    # If you haven't already installed WSL
```

2.2 Create Python Virtual Environment

```
# In WSL Ubuntu terminal
python3 -m venv ~/mistral-tutor-env
source ~/mistral-tutor-env/bin/activate

# Upgrade pip
python -m pip install --upgrade pip
```

Problem encountered: When trying to create the virtual environment using Python 3.12, the following error appeared:

```
salim@MSI:~$ python3 -m venv ~/mistral-tutor-env
The virtual environment was not created successfully because ensurepip is not
available. On Debian/Ubuntu systems, you need to install the python3-venv
package using the following command.

    apt install python3.12-venv

You may need to use sudo with that command. After installing the python3-venv
package, recreate your virtual environment.

Failing command: /home/salim/mistral-tutor-env/bin/python3
```

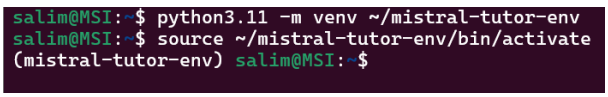
Explanation: This error happens because Python 3.12 is installed by default on Ubuntu 24.04, but it is not fully compatible yet with many AI libraries like `transformers`, `bitsandbytes`, etc.

Correction: To avoid future compatibility problems, we decided to install Python 3.11 manually.

```
# Install dependencies
sudo apt update
sudo apt install software-properties-common

# Add deadsnakes PPA (safe trusted repo)
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update

# Install Python 3.11 and venv module
sudo apt install python3.11 python3.11-venv python3.11-dev

# Verify Python 3.11 installation
python3.11 --version
```

After installing Python 3.11, we create and activate the virtual environment again (notice that the terminal prompt changes, confirming activation):

```
# Create and activate the virtual environment
python3.11 -m venv ~/mistral-tutor-env
source ~/mistral-tutor-env/bin/activate
```



2.3 Fix Missing `pip` inside the Virtual Environment

Problem encountered: After creating and activating the virtual environment, trying to upgrade `pip` failed with the following error:

```
(mistral-tutor-env) salim@MSI:~$ python -m pip install --upgrade pip
/home/salim/mistral-tutor-env/bin/python: No module named pip
```

Explanation: This happens because in Ubuntu 24.04 with Python 3.11, the virtual environment does not automatically include `pip`. We need to manually install `pip` inside the environment using the official `get-pip.py` script.

Correction: First, ensure that `python3.11-distutils` is installed, then manually install `pip`:

```
# Ensure distutils is installed
sudo apt install python3.11-distutils

# Activate the virtual environment (if not already)
source ~/mistral-tutor-env/bin/activate

# Install pip inside the venv using Python 3.11
curl -sS https://bootstrap.pypa.io/get-pip.py | python3.11
```

After successful installation, verify that `pip` is correctly installed:

```
pip --version
```

You should see output similar to:

```
pip 25.1 from /home/salim/mistral-tutor-env/lib/python3.11/site-packages/pip (python
    3.11)
```

This confirms that `pip` is now correctly associated with Python 3.11 inside the virtual environment.

2.4 Install PyTorch with CUDA Support

To install PyTorch with GPU acceleration, we use the following command:

```
pip install torch torchvision torchaudio
```

This command automatically installs PyTorch 2.7.0 built for CUDA 12.6, which is fully compatible with our RTX GPU and driver version 566.14.

Verification: After installation, we verify that PyTorch correctly detects the GPU by running the following Python code:

```
import torch

print(f"CUDA available: {torch.cuda.is_available()}")
print(f"PyTorch version: {torch.__version__}")
print(f"GPU: {torch.cuda.get_device_name(0)}")
```

Expected Output:

```
CUDA available: True
PyTorch version: 2.7.0+cu126
GPU: Your GPU
```

This confirms that PyTorch is correctly installed and the GPU is available for model training and inference.

2.5 Install Required Libraries

Once the Python environment and PyTorch are set up, we proceed to install the libraries required for model loading, quantization, and web interface:

```
# Install Hugging Face libraries
pip install transformers accelerate huggingface_hub einops safetensors sentencepiece
    protobuf

# Install BitsAndBytes for 8-bit quantization
pip install bitsandbytes

# Install Gradio for web UI
pip install gradio

# Install protobuf and sentencepiece library, which is required for the tokenizer
pip install protobuf sentencepiece
```

Explanation:

- `transformers`: for loading large language models (LLMs).
- `accelerate`: for optimized device placement and model execution.
- `huggingface_hub`: to connect and download models.
- `einops`, `safetensors`: helper libraries for tensor operations and secure weight loading.
- `bitsandbytes`: to enable 8-bit or 4-bit model quantization, making it feasible to run large models on limited VRAM.
- `gradio`: to build an easy-to-use web interface for the chatbot.

All packages were successfully installed without errors inside the virtual environment `(mistral-tutor-env)`.

3 Model Setup

3.1 Option 1: Automatic Download from Hugging Face

The code will automatically download the model on first run using the command `python chatbot_app.py`:

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

# Configuration
MODEL_NAME = "mistralai/Mistral-7B-Instruct-v0.3"  # Latest Mistral 7B Instruct model
MAX_NEW_TOKENS = 512  # Maximum number of tokens to generate
MAX_HISTORY_LENGTH = 5  # Maximum number of conversation turns to keep

# Set up quantization configuration for 4-bit loading (minimizes VRAM usage)
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True  # Further reduces memory footprint
)

# Load model and tokenizer
print(f"Loading {MODEL_NAME}...")
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    device_map="auto",
    quantization_config=quantization_config,
)
print("Model loaded successfully!")

# Check CUDA availability
if torch.cuda.is_available():
    print(f"Using GPU: {torch.cuda.get_device_name(0)}")
    print(f"GPU Memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:.2f} GB"
    )
else:
    print("CUDA not available. Using CPU.")

def format_prompt(message, history=None):
    """Format prompt according to Mistral instruction format."""
    prompt = ""

    # Include condensed history if provided
    if history:
        for user_msg, bot_msg in history[-MAX_HISTORY_LENGTH:]:
            prompt += f"<s>[INST] {user_msg} [/INST] {bot_msg}</s>\n"

    # Add the current message
    prompt += f"<s>[INST] {message} [/INST]"

    return prompt

def generate_response(prompt):
```

```
    """Generate a response from the model given a prompt."""
    try:
        inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_new_tokens=MAX_NEW_TOKENS,
                do_sample=True,
                temperature=0.7,
                top_p=0.9,
                pad_token_id=tokenizer.eos_token_id
            )

        full_response = tokenizer.decode(outputs[0], skip_special_tokens=True)

        # Extract only the generated response (after the prompt)
        response = full_response[len(tokenizer.decode(inputs.input_ids[0],
    skip_special_tokens=True)):]

        # Clean up any remaining instruction tokens that might be in the response
        response = response.replace("[/INST]", "").strip()

        return response
    except Exception as e:
        return f"An error occurred: {str(e)}"

def respond(message, history):
    """Process the user message and update the conversation history."""
    prompt = format_prompt(message, history)
    response = generate_response(prompt)

    # Memory management: explicitly clear CUDA cache every few interactions
    if len(history) % 3 == 0:
        torch.cuda.empty_cache()

    return response

# Create Gradio interface with chat history
with gr.Blocks(title="AI Personal Tutor") as demo:
    gr.Markdown("#  AI Personal Tutor using Mistral 7B")
    gr.Markdown("Ask any question and get personalized tutoring!")

    chatbot = gr.Chatbot(height=500)
    msg = gr.Textbox(
        placeholder="Ask me anything about math, science, history, or any subject...",
        container=False,
        scale=7
    )
    submit_btn = gr.Button("Send", variant="primary", scale=1)

    clear_btn = gr.Button("Clear Chat", scale=1)

    def user_input(user_message, history):
        if not user_message:
            return gr.update(value=""), history

        bot_response = respond(user_message, history)
        history = history + [[user_message, bot_response]]
        return "", history

    msg.submit(user_input, [msg, chatbot], [msg, chatbot])
    submit_btn.click(user_input, [msg, chatbot], [msg, chatbot])
    clear_btn.click(lambda: None, None, chatbot, queue=False)

    gr.Markdown("""
    ## About this Tutor

    This AI tutor is powered by the Mistral 7B Instruct model. It can:
    - Explain complex concepts in simple terms
    - Answer questions across various subjects
    - Provide step-by-step solutions to problems
```

```
    Note: While the AI tries to provide accurate information, always verify important
    facts.
    """)

if __name__ == "__main__":
    demo.launch(share=False)  # Set share=True to create a public link
```

3.2 Option 2: Manual Download

```
# Log in to Hugging Face
huggingface-cli login

# Install Git LFS for large files
git lfs install

# Clone the model repository
git clone https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
```

4 Model Agreement and Token Setup

4.1 Agreeing to the Terms

The Mistral-7B-Instruct model requires accepting its license terms through Hugging Face:

- Visit the model card: https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3
- Click "Agree and access repository"
- Log in with your Hugging Face account

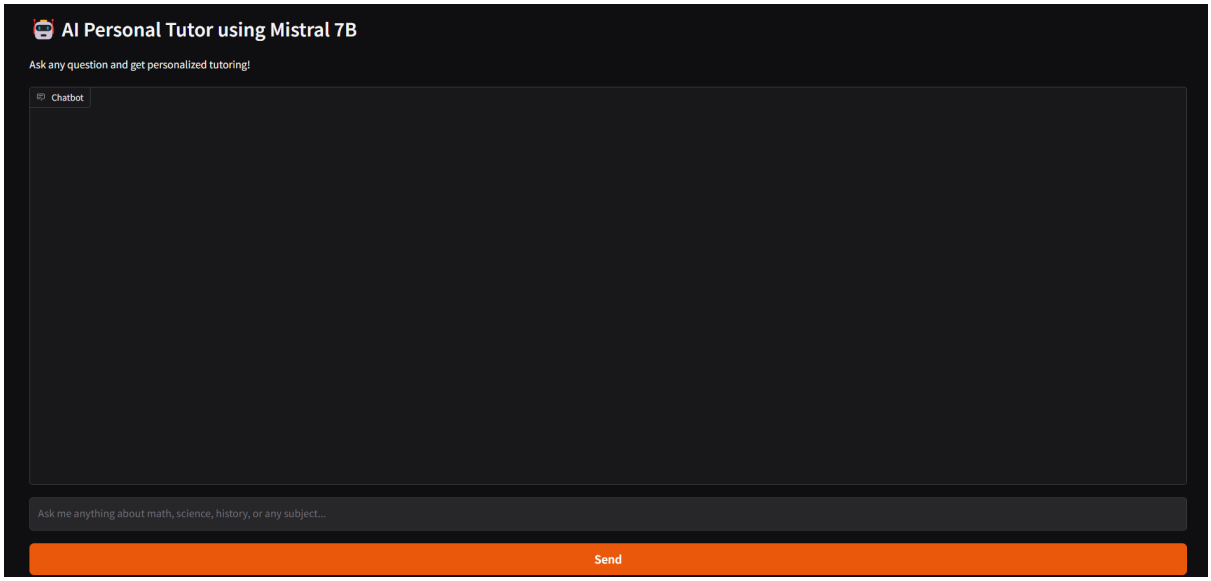
4.2 Token Configuration

After agreeing to the terms, configure your access token:

1. Retrieve your Hugging Face token from https://huggingface.co/settings/tokens
2. Authenticate via command line:
   ```
   huggingface-cli login
   ```
3. When prompted, paste your token (will not be visible)
4. It will give you a verification message that your token is being used, otherwise verify successful authentication (token stored at `~/.cache/huggingface/token`)

4.3 Running the Chatbot

1. Save the `chatbot_app.py` file to your working directory
2. Run the application:
   ```
   python chatbot_app.py
   ```
3. Open the displayed URL (usually http://127.0.0.1:7860) in your web browser



5 Troubleshooting

5.1 Out of Memory Errors

- Reduce `MAX_NEW_TOKENS` in the code
- Consider 4-bit quantization:

```
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_quant_type="nf4"
)
```

5.2 Slow Generation

- Reduce `temperature` and `top_p` values
- Reduce `MAX_NEW_TOKENS`
- Use `do_sample=False`

5.3 CUDA Not Available

- Verify NVIDIA drivers
- Ensure WSL2 GPU passthrough
- Check with `nvidia-smi`

```
watch -n 1 nvidia-smi
```

6 Model Information

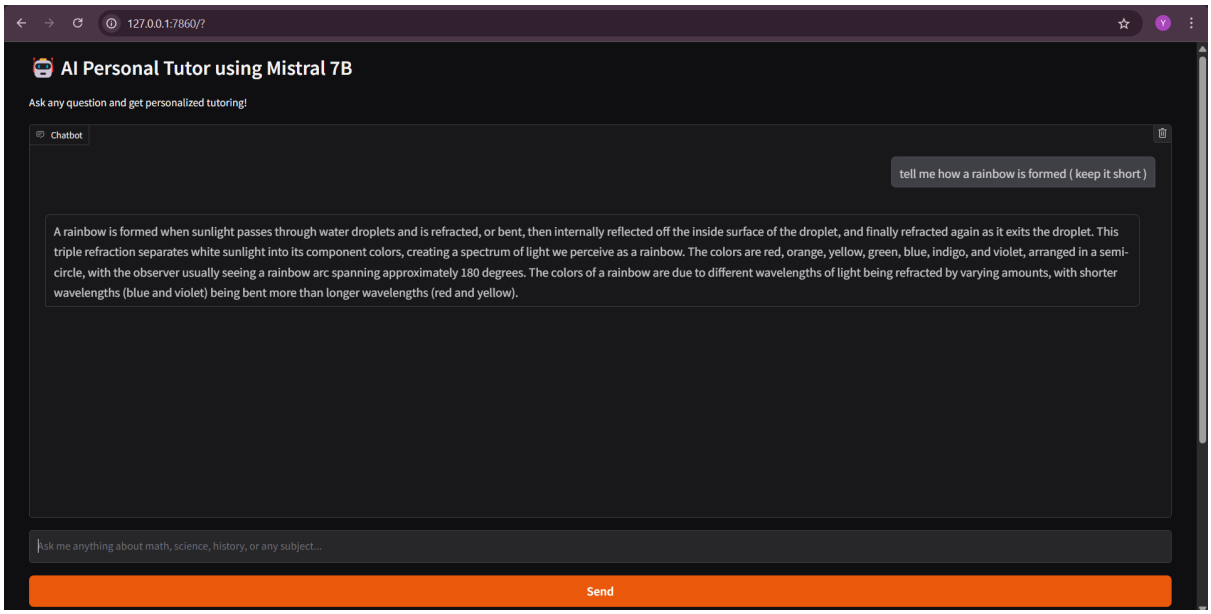
Model: Mistral 7B Instruct v0.3

Size: 7 billion parameters (quantized to 4GB (VRAM) in 4-bit mode)

Context window: 8192 tokens

Specialization: Instruction-following and conversational tasks

7 Results


127.0.0.1:7860/?
AI Personal Tutor using Mistral 7B
Ask any question and get personalized tutoring!
Chatbot
tell me how a rainbow is formed (keep it short)
A rainbow is formed when sunlight passes through water droplets and is refracted, or bent, then internally reflected off the inside surface of the droplet, and finally refracted again as it exits the droplet. This triple refraction separates white sunlight into its component colors, creating a spectrum of light we perceive as a rainbow. The colors are red, orange, yellow, green, blue, indigo, and violet, arranged in a semi-circle, with the observer usually seeing a rainbow arc spanning approximately 180 degrees. The colors of a rainbow are due to different wavelengths of light being refracted by varying amounts, with shorter wavelengths (blue and violet) being bent more than longer wavelengths (red and yellow).
Ask me anything about math, science, history, or any subject...
Send

8 Conclusion and Future Applications

We have successfully run the Mistral-7B-Instruct model locally, which is a significant achievement and an important step towards building more sophisticated AI applications in the future. This successful implementation opens up many opportunities for further development and integration into various domains.

Some potential future applications include:

- **API Integration:** We can link the model to an API, turning it into a service that can be used by various applications. This would allow other systems or businesses to easily integrate the model for natural language processing tasks, such as customer support, content generation, and more.
- **Cloud Deployment:** The model can be deployed on the cloud, making it accessible to users through a web interface. This would allow businesses to offer scalable AI-powered services, accessible from anywhere, at any time.
- **Subject-Specific Fine-Tuning:** The model can be fine-tuned to become hyper-specific to particular domains or subjects, such as medical, legal, or technical fields. This would allow the chatbot to provide more specialized and accurate responses tailored to specific industries.
- **Chatbot for Businesses:** The model can be integrated into customer service solutions, providing businesses with automated, real-time support for their customers. It could assist with answering frequently asked questions, troubleshooting common problems, and more.
- **Educational Applications:** The model can be used as an educational assistant, helping students learn new concepts, offering explanations, and providing personalized tutoring in various subjects.

8.1 Key Words

LLM, local, hosting, Mistral, deployment, finetuning, NLP, cloud, chatbot, API, ML, agent, automation, education, tutoring, transformer