HIPSTER

We strongly believe in assessing talent based on real-world problem-solving skills and practical abilities rather than solely focusing on experience, degrees, or certifications. Here's why:

Why Practical Assessments Matter
In today's rapidly evolving AI and ML landscape, the ability to implement, adapt, and deliver innovative solutions is critical. While traditional qualifications highlight foundational knowledge, real-world assessments:

1. **Demonstrate Hands-On Expertise**: They show your ability to apply concepts in practical scenarios.
2. **Show Problem-Solving Skills**: They allow us to see how you approach challenges, navigate complexities, and create solutions.
3. **Focus on Deliverables**: By evaluating actual outcomes, we emphasize results and impact over credentials.
4. **Foster Growth**: This process helps identify individuals who are not just skilled but also ready to learn and grow with us.

Your Task
We have designed three tasks, each reflecting real-world challenges you might encounter in this role. Out of the three tasks listed, **please select the one** where you feel most confident and align with your skills.

Once you've selected your task, you can begin working on it. After completing it, please respond to this email with your readiness.

What Happens Next
We will schedule a video meeting where you will:

1. **Showcase Your Work**: Demonstrate your setup, approach, and solution.
2. **Discuss Challenges**: Share your thought process, any obstacles faced, and how you resolved them.
3. **Receive Feedback**: We will evaluate your work based on technical skills, problem-solving, and scalability of your solution.

Why This Process is Beneficial
This process is designed not only to help us assess your skills but also to give you a platform to shine through your work. It reflects the kind of projects you'll work on here and ensures a mutual alignment of expectations.

HIPSTER

Task 1: Translation Model Fine-tuning and Deployment

Prompt:
Translation models often fail to account for cultural nuances, idioms, and community-specific expressions. For example, translating रस्सी जल गयी, बल नहीं गया using a generic model like Google Translate may yield incorrect results. The accurate translation is *Even after one hits rock bottom, their arrogance remains unchanged.*

Your task is to fine-tune a large translation model to better handle such cultural idioms and nuances.

Requirements:

1. **Model Selection**: Choose a pre-trained translation model (e.g., MarianMT, T5, M2M-100).
2. **Dataset**: Use a small, labelled dataset with examples of culturally nuanced idioms and their translations. Examples can be sourced from:
    - Tatobbaidioms Dataset
    - OpenSubtitles Dataset
3. **Tasks**:
    - Fine-tune the translation model on this dataset using frameworks like Hugging Face Transformers.
    - Deploy the fine-tuned model on a cloud platform (AWS, Azure, or GCP).
    - Build a REST API using frameworks like FastAPI or Flask to expose the model for predictions.
4. **Evaluation**:
    - Use BLEU or METEOR as evaluation metrics to measure translation accuracy.
    - Demonstrate performance improvements by comparing pre- and post-fine-tuning metrics.

Evaluation Criteria:

- **Technical Skills**: Leveraging frameworks like Hugging Face, LangChain, and deploying on cloud platforms.
- **Problem-Solving**: Handling challenges related to fine-tuning and deployment.
- **Code Quality**: Clean, efficient, and well-documented.
- **API Design**: User-friendly documentation of REST API.
- **Deployment**: Scalable deployment ensuring reliability.

HIPSTER

Task 2: Building a Chatbot using a Large Language Model and Vector Database

Prompt:
Develop a chatbot application that leverages a Large Language Model (LLM) and a Vector Database to answer questions based on website content from:

- Changi Airport
- Jewel Changi Airport

Requirements:

1. **Data Processing**:
    - Scrape the website content and clean it for storage.
    - Vectorize the data using a model like OpenAI embeddings or Sentence Transformers (e.g., SBERT).
2. **Database**: Store the embeddings in a vector database (e.g., Pinecone, Weaviate, or Milvus).
3. **Chatbot Development**:
    - Use frameworks like LangChain to create a Retrieval-Augmented Generation (RAG) chatbot.
    - The chatbot should provide accurate and contextual answers based on the website data.
4. **Deployment**:
    - Deploy the chatbot on a cloud platform with a REST API for integration.

Evaluation Criteria:

- **Technical Skills**: Integration of LLMs with a vector database and cloud deployment.
- **Problem-Solving**: Handling data cleaning, embedding generation, and RAG implementation.
- **Code Quality**: Modular, efficient, and well-documented.
- **API Design**: Concise API documentation.
- **Deployment**: Scalable architecture to support growing user demand.

Task 3: Emotion Detection from Video using YOLO v8

Prompt:
Use YOLO v8 (or the latest version) to process real-time or recorded videos for detecting human facial emotions. This task combines object detection with emotion classification.

Requirements:

1. **Model Development**:
    - Fine-tune YOLO for face detection using a dataset like WIDER FACE.
    - Train an integrated emotion classification model using datasets like:
        - FER-2013
        - AffectNet
    - Classify emotions into categories: happy, sad, angry, surprised, neutral, etc.
2. **Pipeline**:
    - Preprocess videos to detect faces using YOLO.
    - Integrate YOLO with the emotion recognition module.
3. **Deployment**:
    - Deploy the pipeline on a GPU-enabled cloud platform.
    - Build a REST API to process video streams or files, returning emotions with bounding boxes for each frame.
4. **Real-Time Interface**:
    - Create a simple web app to display live results.

Evaluation Metrics:

- **Model Accuracy**: Precision, Recall, and F1-score on a test set.
- **Performance**: Real-time detection with a minimum FPS benchmark.

Evaluation Criteria:

- **Technical Skills**: Fine-tuning YOLO and integrating emotion recognition.
- **Problem-Solving**: Addressing challenges like lighting, occlusion, and frame rate optimization.
- **Code Quality**: Modular and efficient.
- **API Design**: Detailed API documentation.
- **Deployment**: Scalable and low-latency pipeline.