Preliminary Research on Possible Models and Implementation for Beyond QWERTY Project

(Anurupa Saha)

---

1. Objective of the Research

The project aims to revolutionize form-filling processes by leveraging voice-based technologies. This research identifies potential models, databases, and methods required to develop a multilingual, speech-enabled form-filling system. The focus is on models for speech-to-text conversion, translation, and integration into form workflows.

---

2. Possible Models and Technologies

2.1 Speech-to-Text Models

1. **OpenAI Whisper**:
    - **Description**: A state-of-the-art speech-to-text model with multilingual capabilities and robust transcription accuracy.
    - **Strengths**: Handles diverse accents and noisy environments. Open-source, allowing offline processing.
    - **Limitations**: Requires powerful hardware (GPU) for larger models.
2. **Azure Cognitive Services Speech SDK**:
    - **Description**: Cloud-based solution offering speech-to-text and speech recognition.
    - **Strengths**: Scalable and integrates seamlessly with other Azure services.
    - **Limitations**: Requires an internet connection and incurs API usage costs.

2.2 Translation Models

1. **Azure Cognitive Services Translator API**:
    - Provides real-time multilingual translation for over 70 languages.
    - Can process text outputs from speech models for language conversion.
2. **Google Translate API**:
    - Similar functionality to Azure's Translator API, with extensive language support and easy integration.

2.3 Integration and Workflow Automation

1. **Python Libraries**:
    - **Whisper API**: For integrating OpenAI Whisper.
    - **Speech SDK**: To use Azure's Speech-to-Text API.
    - **Flask/Django**: Backend frameworks for workflow integration.
2. **Platforms**:
    - **Google Colab**: GPU-powered environment for testing models.
    - **Local Jupyter Notebook**: For small-scale development and debugging.

---

3. Proposed Architecture

1. **Input Layer**:
   - User speaks into a microphone or uploads an audio file.
2. **Speech-to-Text Conversion**:
   - Audio is transcribed into text using either OpenAI Whisper (for offline/local processing) or Azure Speech SDK (for cloud processing).
3. **Translation (Optional)**:
   - The transcribed text is translated into the desired language using Azure Translator API.
4. **Form Filling Workflow**:
   - Text is parsed and mapped to corresponding form fields.
5. **Output Layer**:
   - Completed form is presented to the user for review and export.

---

4. Basic Implementation Steps

Step 1: Speech-to-Text Setup

- **For Whisper**:

1. Install dependencies:

```
pip install openai-whisper
sudo apt install ffmpeg
```

2. Transcribe audio:

```
import whisper
model = whisper.load_model("base")
result = model.transcribe("audio_file.mp3")
print(result["text"])
```

- **For Azure Speech SDK**:

1. Create a Speech resource in Azure.
2. Install the SDK:

```
pip install azure-cognitiveservices-speech
```

3. Use the SDK:

```
import azure.cognitiveservices.speech as speechsdk

speech_config = speechsdk.SpeechConfig(subscription="You
recognizer = speechsdk.SpeechRecognizer(speech_config=sp
result = recognizer.recognize_once()
print(result.text)
```

Step 2: Translation Setup

- **Azure Translator API**:
    1. Set up Translator resource in Azure.
    2. Make API calls:

```
import requests

endpoint = "https://api.cognitive.microsofttranslator.com/translate"

headers = {

 "Ocp-Apim-Subscription-Key": "YourKey",
"Content-Type": "application/json"
}
body = [{"text": "Hello World!"}]
params = {"api-version": "3.0", "to": ["es"]}
response = requests.post(endpoint, headers=headers, json=body, params=params)
print(response.json())
```

Step 3: Form Mapping and Filling

1. Parse the transcribed text to extract relevant information using Natural Language Processing (NLP) tools (e.g., spaCy).
2. Map the extracted data to form fields in JSON or XML format.

Step 4: Integration and Testing

1. Create a Flask/Django app to manage inputs, processing, and outputs.
2. Test the app with sample audio and various accents/languages.

---

5. Key Considerations

- **Hardware**: Use GPUs for larger models or cloud services for scalability.
- **Data Privacy**: Ensure compliance with GDPR or other local data protection laws if using cloud services.
- **Error Handling**: Implement mechanisms to handle transcription and translation inaccuracies.
- **User Interface**: Develop an intuitive UI for easy adoption by frontline workers.

---

6. Conclusion

This preliminary research identifies Whisper and Azure Cognitive Services as strong candidates for building a voice-enabled form-filling system. The choice between them depends on factors like hardware availability, internet access, and processing needs. A detailed implementation plan ensures the solution is robust, scalable, and user-friendly.