Deployment Manual

This guide provides step-by-step instructions for deploying the AR Learner project, which consists of three main components: 1. Android Application 2. Website 3. LLM (Large Language Model) Server

1. Android Application Deployment

Prerequisites

- Android Studio (latest version)
- JDK 11
- Git

Build and Deploy Steps

1. Clone the repository:

   ```
   git clone <repository-url>
   cd SysEng/App
   ```

2. Open the project in Android Studio:

   - Launch Android Studio
   - Select "Open an Existing Project"
   - Navigate to the `App` directory

3. Build the application:

   ```
   # From terminal in the App directory
   ./gradlew assembleDebug
   ```

   Or use the "Build > Make Project" option in Android Studio

4. Install on device or emulator:

   - Connect an Android device via USB (with USB debugging enabled)
   - Or start an emulator in Android Studio
   - In Android Studio, click "Run > Run 'app' "

5. Distribute the APK:

   - The built APK will be in `App/app/build/outputs/apk/debug/`
   - Share this APK file with users for installation

2. Website Deployment

Prerequisites

- Node.js (v18 or higher)
- npm (comes with Node.js)
- MySQL database access (Azure MySQL)

Local Development Deployment

1. Clone the repository:

   ```
   git clone <repository-url>
   cd SysEng/Website
   ```

2. Install dependencies:

   ```
   npm install
   ```

3. Configure database connection:

   - Open `azureconnection.js`
   - Update the database connection parameters if needed:

     ```
     const connection = mysql.createConnection({
       host: 'nttdata.mysql.database.azure.com',
       user: 'nttdata',
       password: 'n13data!',
       database: 'reports_db',
       ssl: { rejectUnauthorized: true }
     });
     ```

4. Start the server:

   ```
   node server.js
   ```

5. Access the website:

   - Open a web browser
   - Navigate to `http://localhost:5000`
   - For HTML files not served by Express, use VS Code's Live Server extension
   - Right-click on `index.html` and select "Open with Live Server"

3. LLM Server Deployment

Prerequisites

- Docker Desktop (for containerized deployment)
- Azure subscription (for cloud deployment)
- Azure CLI
- A GGUF format LLM model file

Option 1: Using LM Studio for Local Development

LM Studio provides an easy way to run LLMs locally during development:

1. Download and Install LM Studio:
   - Download from LM Studio website
   - Install following the instructions for your operating system

2. Load a model:
   - Launch LM Studio
   - Go to the Models tab
   - Download a suitable model (e.g., Llama 3-8B Instruct)
   - Select the downloaded model
3. Configure CORS for API access:
   - Go to Server Settings
   - Enable "Allow CORS" option
   - Set allowed origins to include your development domains (e.g., `http://localhost:5000`)
4. Start the server:
   - Click "Start Server"
   - The server will be available at `http://localhost:8000`
   - API endpoints will be compatible with OpenAI format
5. Test the server:
   - Visit `http://localhost:8000/models` to see available models
   - The chat completions API will be at `/v1/chat/completions`

Option 2: Azure Container Deployment

For production deployment:

1. Setup Azure Storage:
   - Log in to Azure Portal
   - Create a new Storage Account
   - Create a container within the Storage Account
   - Upload your GGUF model file to the container
2. Generate SAS Token:
   - In Azure Portal, navigate to your Storage Account
   - Go to "Security + networking" > "Shared access signature"
   - Enable "Blob", "File", "Write", "Table" services
   - Enable "Container" and "Object" resource types
   - Enable "Read" permission
   - Set an expiry date
   - Click "Generate SAS and connection string"
3. Setup environment variables:

   ```
   cd SysEng/LLM
   ```

   Create a `.env` file with:

   ```
   SAS_TOKEN=<your-sas-token>
   STORAGE_ACCOUNT=<your-storage-account-name>
   CONTAINER_NAME=<your-container-name>
   LLM_FILE_NAME=<your-llm-file-name>
   ```

4. Deploy to Azure Container Instance:

```
az container create --resource-group <your-resource-group> \
                    --name llama-server-container \
                    --image <your-acr-name>.azurecr.io/llama-server:v1 \
                    --cpu 4 \
                    --memory 16 \
                    --ports 8000 \
                    --environment-variables \
                        STORAGE_ACCOUNT=$STORAGE_ACCOUNT \
                        SAS_TOKEN=$SAS_TOKEN \
                        CONTAINER_NAME=$CONTAINER_NAME \
                        LLM_FILE_NAME=$LLM_FILE_NAME
```

5. Access the deployed LLM server:
   - The server will be available at `http://<aci-dns-name>.azurecontainer.io:8000`
   - Use this URL in the Android app and website configuration

Option 3: GPU Acceleration for Docker Deployment

For enhanced performance with CUDA GPU acceleration:

```
FROM nvidia/cuda:12.8.0-devel-ubuntu24.04

# Install dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    git \
    cmake \
    python3 \
    python3-pip \
    python3-venv \
    curl \
    ca-certificates \
    gnupg \
    wget

# Create symlink for libcuda.so to fix build issues
RUN mkdir -p /usr/local/cuda/lib64/stubs && \
    touch /usr/local/cuda/lib64/stubs/libcuda.so && \
    ln -s /usr/local/cuda/lib64/stubs/libcuda.so /usr/local/cuda/lib64/stubs/libcuda.so.1 &&
    echo "/usr/local/cuda/lib64/stubs" > /etc/ld.so.conf.d/cuda-stubs.conf && \
    ldconfig

# Install Azure CLI
RUN curl -sL https://aka.ms/InstallAzureCLIDeb | bash
```

```
# Clone llama.cpp
WORKDIR /app
RUN git clone https://github.com/ggerganov/llama.cpp.git
WORKDIR /app/llama.cpp

# Build llama.cpp with explicit CUDA paths
RUN mkdir build && cd build && \
    export LD_LIBRARY_PATH=/usr/local/cuda/lib64/stubs:$LD_LIBRARY_PATH && \
    cmake -DGGML_CUDA=ON .. && \
    make -j$(nproc)

# Create a model directory
RUN mkdir -p /app/models

# Copy the env file
COPY .env /app/.env

# Create start server script
COPY start-server.sh /app/start-server.sh
RUN chmod +x /app/start-server.sh

# Expose the server port
EXPOSE 8000

# Set the entrypoint
ENTRYPOINT ["/app/start-server.sh"]

# Allow additional arguments to be passed to the server
CMD []
```

To deploy with GPU support on Azure, specify GPU resources:

```
az container create --resource-group <your-resource-group> \
                    --name llama-server-container \
                    --image <your-acr-name>.azurecr.io/llama-server:v1 \
                    --cpu 4 \
                    --memory 16 \
                    --gpu-count 1 \
                    --gpu-sku K80 \
                    --ports 8000 \
                    --environment-variables \
                        STORAGE_ACCOUNT=$STORAGE_ACCOUNT \
                        SAS_TOKEN=$SAS_TOKEN \
                        CONTAINER_NAME=$CONTAINER_NAME \
                        LLM_FILE_NAME=$LLM_FILE_NAME
```

Integration

After deploying all components:

1. Update the Android app with the correct LLM server URL:
   - Modify `Config.kt` with the deployed LLM server URL
2. Update the website with the correct API endpoints:
   - Modify any API connection settings to point to your deployed services
3. Verify all components can communicate correctly:
   - Test the Android app can connect to the LLM server
   - Test the website can connect to the database
   - Test report creation and viewing across platforms