Here's a detailed **Software Requirements Specification (SRS)** document for your **Mathematical Symbol Recognition** project:

Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

The purpose of this project is to develop a Python-based system that can recognize mathematical symbols from both printed and handwritten sources, such as PDF documents and images. The system will employ optical character recognition (OCR), image processing, and machine learning models to extract and convert mathematical expressions into structured digital formats like LaTeX or MathML.

1.2 Scope

This system aims to streamline the digitization of mathematical documents by automating the recognition of mathematical symbols and expressions. Potential applications include:

- Educational tools for students and teachers.
- Automated grading systems for mathematical assessments.
- Document digitization for academic research and archival purposes.

1.3 Definitions, Acronyms, and Abbreviations

- **OCR**: Optical Character Recognition.
- **CNN**: Convolutional Neural Network.
- **LaTeX**: A typesetting system commonly used for mathematical and scientific documents.
- **MathML**: Mathematical Markup Language, an XML-based standard for displaying mathematical notations.

1.4 References

- Python Libraries: OpenCV, TensorFlow/Keras, Tesseract OCR, pdf2image.
- Datasets: CROHME dataset for handwritten math recognition.
- Documentation for tools like Tesseract and Poppler.

---

2. Overall Description

2.1 Product Perspective

The system is a standalone application designed to process images and PDF documents, extract mathematical content, and convert it into a digital format. It integrates existing OCR tools and machine learning frameworks.

2.2 Product Functions

The system will:

1. Preprocess input PDFs and images.
2. Detect and segment mathematical symbols.
3. Recognize symbols using OCR and machine learning models.
4. Export results in LaTeX and MathML formats.

2.3 User Characteristics

The primary users are:

- Students and educators who require digitized mathematical content.
- Developers integrating this functionality into larger platforms.
- Researchers digitizing historical or handwritten mathematical texts.

2.4 Constraints

- Performance may depend on input quality (e.g., resolution, noise).
- Limited by the availability of training data for handwritten symbols.
- Compatibility with operating systems (Windows, macOS, Linux).

2.5 Assumptions and Dependencies

- Users will provide clear images or well-scanned PDFs.
- Python and required libraries (e.g., Tesseract, TensorFlow) are installed and configured.
- GPU is available for training machine learning models (optional).

---

3. Specific Requirements

3.1 Functional Requirements

1. **Input Handling**:
    - Accepts images (JPEG, PNG, BMP) and PDF files as input.
    - Extracts pages from PDFs for further processing.
2. **Preprocessing**:
    - Binarization, noise reduction, and contour detection for clean input.
    - Segmentation of mathematical symbols and expressions.

3. **Recognition**:
    - Uses Tesseract OCR for printed symbols.
    - Employs a trained CNN model for handwritten symbols.
4. **Output Generation**:
    - Converts recognized symbols into LaTeX and MathML.
    - Saves outputs to a specified folder.
5. **User Interface**:
    - Command-line interface (CLI) for running tasks.
    - (Optional) GUI for uploading files and viewing results.

3.2 Performance Requirements

- **Accuracy**: At least 90% for printed symbols; 80% for handwritten symbols.
- **Processing Speed**: Process one page of a PDF in under 5 seconds on average (depends on hardware).

3.3 Non-functional Requirements

1. **Usability**:
    - Simple CLI/GUI with clear documentation.
    - Support for error handling and user feedback.
2. **Scalability**:
    - Capable of handling multi-page PDFs and batch processing.
3. **Portability**:
    - Runs on Windows, macOS, and Linux systems.
4. **Maintainability**:
    - Modular codebase for easy updates and improvements.

3.4 External Interface Requirements

- **Hardware**:
    - Minimum 4GB RAM, recommended 8GB for smooth performance.
    - Optional GPU for training and recognition.
- **Software**:
    - Python 3.8 or later.
    - Required libraries: OpenCV, TensorFlow/Keras, Tesseract, pdf2image, and PyMuPDF.

---

4. Appendices

4.1 Tools and Technologies

- Python 3.8+
- Libraries: OpenCV, TensorFlow/Keras, PyTesseract, pdf2image
- Utilities: Tesseract OCR, Poppler
- Datasets: CROHME for handwritten math.

4.2 Glossary

- **Binarization**: Converting an image into a binary (black-and-white) format.
- **Contour Detection**: Identifying the boundaries of symbols in images.

---

5. Data Flow Diagram (DFD)

Below is a high-level **Data Flow Diagram (DFD)** for the system:

1. **User Inputs PDF/Image** → 2. **Preprocessing Module (Binarization, Segmentation, Noise Reduction)** → 3. **Symbol Recognition (OCR/CNN Model)** → 4. **Output Formatter (LaTeX/MathML Generator)** → 5. **User Receives Digital Output**

---

6. Entity-Relationship (ER) Diagram

The **Entity-Relationship (ER) Diagram** represents the relationship between key entities in the system:

- **User** (uploads) → **Document** (contains) → **Symbols** (recognized by) → **OCR/CNN Model** (classified into) → **Math Expression** (converted into) → **LaTeX/MathML Output**

This diagram helps visualize data flow and entity interactions in the recognition process.

Let me know if you want adjustments or further elaboration!