1.0 Core Strategy: Rule-Based Heuristic Approach

Our primary strategy is to build a high-performance, rule-based system. This approach analyzes the stylistic and positional properties of text within the PDF to identify headings. It avoids the overhead and size of machine learning models, ensuring we meet the hackathon's strict constraints on execution time (≤10s) and size (≤200MB).

Key Library: PyMuPDF (fitz) will be the core of our solution. It is a powerful Python library known for its speed and ability to extract detailed text metadata, such as font size, font name, bold flags, and coordinates.

2.0 Step-by-Step Implementation Plan

2.1 Environment Setup

The solution will be containerized using Docker. The project structure will include a Dockerfile, a requirements.txt file, and the main Python script.

File: requirements.txt

```
PyMuPDF==1.24.1
```

File: Dockerfile

```
# Use a lightweight, amd64-compatible Python base image
FROM --platform=linux/amd64 python:3.10-slim

# Set the working directory
WORKDIR /app

# Copy dependency list and install them
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the solution code into the container
COPY . .

# The command to run your solution. This script will handle the I/O logic.
CMD ["python", "main.py"]
```

2.2 Main Script (main.py)

This script will manage the file I/O operations, processing all PDFs from /app/input and writing JSON results to /app/output.

File: main.py

```
import os
import json
import fitz  # PyMuPDF

# Define input and output directories
INPUT_DIR = "/app/input"
OUTPUT_DIR = "/app/output"
```

```
# The core logic will be in this function
def extract_outline_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    title = doc.metadata.get('title', 'Untitled')
    blocks = []

    # 1. Collect all text blocks with their properties
    for page_num, page in enumerate(doc):
        page_blocks = page.get_text("dict")["blocks"]
        for block in page_blocks:
            if "lines" in block:
                for line in block["lines"]:
                    for span in line["spans"]:
                        blocks.append({
                            "text": span["text"].strip(),
                            "size": span["size"],
                            "font": span["font"],
                            "page": page_num + 1
                        })

    if not blocks:
        return {"title": title, "outline": []}

    # Find the most common font size (likely body text)
    sizes = [b['size'] for b in blocks]
    most_common_size = max(set(sizes), key=sizes.count)

    # 2. Filter for potential headings
    heading_candidates = [b for b in blocks if b['size'] > most_common_size and b['text']]

    # 3. Identify and rank unique heading styles by size
    unique_styles = sorted(list(set(b['size'] for b in heading_candidates)), reverse=True)

    style_map = {}
    if len(unique_styles) > 0: style_map[unique_styles[0]] = "H1"
    if len(unique_styles) > 1: style_map[unique_styles[1]] = "H2"
    if len(unique_styles) > 2: style_map[unique_styles[2]] = "H3"

    # 4. Build the final outline
    outline = []
    for block in heading_candidates:
        if block['size'] in style_map:
            outline.append({
                "level": style_map[block['size']],
```

```
            "text": block['text'],
            "page": block['page']
        })

    # Use largest font on first page for title if metadata is poor
    if not title or len(title) < 4:
        first_page_blocks = [b for b in blocks if b['page'] == 1]
        if first_page_blocks:
            title = max(first_page_blocks, key=lambda x: x['size'])['text']

    return {"title": title, "outline": outline}

if _name_ == "_main_":
    if not os.path.exists(OUTPUT_DIR):
        os.makedirs(OUTPUT_DIR)

    for filename in os.listdir(INPUT_DIR):
        if filename.lower().endswith(".pdf"):
            pdf_path = os.path.join(INPUT_DIR, filename)
            outline_data = extract_outline_from_pdf(pdf_path)

            output_filename = os.path.splitext(filename)[0] + ".json"
            output_path = os.path.join(OUTPUT_DIR, output_filename)

            with open(output_path, 'w', encoding='utf-8') as f:
                json.dump(outline_data, f, indent=2, ensure_ascii=False)
```

3.0 Meeting Constraints and Scoring

* Execution Time (≤ 10s): The PyMuPDF library is highly optimized and will easily process a 50-page PDF in under a second.
* Model Size (≤ 200MB): This approach uses no machine learning models, so the constraint is met by default.
* Network: The solution is entirely self-contained and requires no internet access.
* Accuracy: Accuracy can be further improved by adding heuristics for font weight (e.g., "Bold" in the font name), text numbering (1.1, A.), and line length.
* Bonus (Multilingual): PyMuPDF correctly extracts Unicode text. For languages like Japanese, where bold flags are uncommon, the system's reliance on font size as the primary differentiator is a significant advantage.