# Contents



# 1 PDF Page Marker Preservation in Classification Pipeline

## 1.1 Overview

This document describes the implementation of transparent PDF page marker preservation through the SKOL classification pipeline. Page markers (format: `--- PDF Page N ---`) are now preserved in `.txt.ann` output files without being classified.

## 1.2 Problem

When generating `.txt.ann` files with YEDDA annotations, PDF page markers from the source `.txt` files were being: 1. Classified as regular text content 2. Lost or incorrectly formatted in the output 3. Breaking classification boundaries inappropriately

This caused page tracking to be lost in the annotated output.

## 1.3 Solution

Implemented transparent page marker handling at multiple layers:

### 1.3.1 1. Line-Level Marker Detection (`line.py`)

Added `is_page_marker` flag to the `Line` class: - New boolean attribute `_is_page_marker` - Constructor parameter to mark lines as page markers - Property method `is_page_marker()` for access

```
class Line(object):
    _is_page_marker: bool  # True if this line is a PDF page marker

    def __init__(self, line: str, fileobj: Optional[FileObject] = None, is_page_
        self._is_page_marker = is_page_marker
```

### 1.3.2 2. File Object Processing (`fileobj.py`)

Modified `read_line()` to create Line objects for page markers: - Previously: Page markers were skipped (continue) - Now: Create Line objects with `is_page_marker=True` - These lines are yielded to preserve ordering

```
# Check for PDF page marker
pdf_page_match = re.match(r'^---\s*PDF\s+Page\s+(\d+)\s*---\s*$', l_str.strip())
if pdf_page_match:
    self._pdf_page = int(pdf_page_match.group(1))
    # Create a special Line object for the page marker
    l = Line(l_str, self, is_page_marker=True)
    yield l
    continue
```

### 1.3.3 3. Extraction Mode (`extraction_modes/line.py`)

Added page marker detection in DataFrame: - Added `is_page_marker` column using regex matching - Column is boolean: True for page markers, False for regular content

```
return exploded_df.withColumn(
    "is_page_marker",
    col("value").rlike(r"^\s*---\s*PDF\s+Page\s+\d+\s*---\s*$")
)
```

### 1.3.4 4. Classifier (`classifier_v2.py`)

Modified `predict()` method to handle page markers: 1. **Separate**: Extract page markers before classification 2. **Classify**: Process only

non-page-marker lines 3. **Reinsert**: Merge page markers back into predictions

```
# Separate page markers
if "is_page_marker" in raw_data.columns:
    page_markers_df = raw_data.filter(col("is_page_marker") == True)
    raw_data = raw_data.filter(col("is_page_marker") == False)

# ... classification happens ...

# Reinsert page markers
if page_markers_df is not None:
    predictions_df = self._reinsert_page_markers(predictions_df, page_markers_df
```

#### 1.3.5 5. Output Formatter (`output_formatters.py`)

Enhanced `coalesce_consecutive_labels()` to handle page markers: - Page markers **break** coalescing chains - Page markers are output as raw text (no YEDDA formatting) - Maintains proper ordering via line numbers

```
for row in sorted_rows:
    row_num, value, label, is_page_marker = row

    if is_page_marker:
        flush_current_block()      # End current annotation block
        result.append(value)       # Add marker as-is
        current_lines = []          # Reset accumulator
```

### 1.4 Behavior

#### 1.4.1 Input `.txt` File

```
--- PDF Page 1 ---
Introduction to Fungi

This paper describes...

--- PDF Page 2 ---
Materials and Methods

We collected samples...
```

#### 1.4.2 Output `.txt.ann` File

```
--- PDF Page 1 ---
```

```
[@ Introduction to Fungi

This paper describes... #Description*]

--- PDF Page 2 ---
[@ Materials and Methods #Description*]

[@ We collected samples... #Description*]
```

## 1.5 Key Design Decisions

### 1.5.1 1. Page Markers Break Classification Boundaries

Page markers are treated as **boundaries** that prevent classification from spanning across them. This is appropriate because: - Pages are natural document boundaries - Paragraphs shouldn't span page breaks in the PDF - It simplifies the classification model

**Impact**: A paragraph that spans a page break will be split into two classification units.

### 1.5.2 2. Raw Marker Preservation

Page markers are preserved **exactly as they appear** in the source: - No YEDDA annotation formatting applied - No label assigned - Maintains exact text for parsing downstream

This allows: - Consistent parsing in `fileobj.py` - Clear visual page boundaries in annotated files - Easy extraction of page numbers

### 1.5.3 3. Line-Number Based Ordering

Page markers and predictions are merged using line numbers: - Maintains original document order - Works for line-level classification - Paragraph/section modes inherit this behavior

## 1.6 Limitations

### 1.6.1 Paragraphs Spanning Page Breaks

If a paragraph in the original PDF spans multiple pages, it will be: - Split at the page marker - Classified as separate units - Each part may receive a different label

**Example**:

```
--- PDF Page 1 ---
[@ This is a long paragraph that continues
across multiple pages and contains... #Description*]

--- PDF Page 2 ---
[@ ...important taxonomic information about
the species. #Nomenclature*]
```

The two parts might get different labels since they're classified independently.

**Mitigation**: This is generally acceptable because: - True page-spanning paragraphs are rare in scientific papers - The alternative (removing markers) loses critical metadata - Post-processing could merge consecutive blocks if needed

### 1.6.2 Section and Paragraph Modes

Currently implemented for **line mode** only. For section/paragraph modes: - Page markers are preserved in the source data - They're included in paragraph/section boundaries naturally - No special handling needed (paragraphs don't split on markers)

## 1.7 Files Modified

1. **`line.py`** - Added `is_page_marker` attribute to Line class
2. **`fileobj.py`** - Modified to yield Line objects for page markers
3. **`extraction_modes/line.py`** - Added is_page_marker column detection
4. **`classifier_v2.py`** - Added page marker filtering and reinsertion logic
5. **`output_formatters.py`** - Enhanced coalescing to respect page markers

## 1.8 Testing

To test page marker preservation:

```
# 1. Generate .txt file with page markers (if not already present)
cd /data/piggy/src/github.com/piggyatbaqaqi/skol
./bin/with_skol bin/regenerate_txt_with_pages.py \
    --database skol_dev \
    --doc-id YOUR_DOC_ID

# 2. Run prediction with line-level classification
./bin/with_skol bin/predict_classifier.py \
```

```
    --model logistic_sections \
    --pattern "*.txt"

# 3. Check that page markers appear in .txt.ann file
# They should be present as raw lines (not YEDDA formatted)
```

## 1.9 Future Enhancements

1. **Smart Paragraph Merging**: Detect and merge paragraphs that were split by page boundaries
2. **Page Context Features**: Add page number as a classification feature
3. **Section/Paragraph Modes**: Explicit page marker handling if needed
4. **Configurable Behavior**: Allow users to choose whether markers break classifications

## 1.10 Related Documentation

- `PDF_TXT_ATTACHMENT_SUPPORT.md` - PDF to text conversion with page markers
- `PDF_SECTION_EXTRACTOR_SUMMARY.md` - Section extraction with page tracking
- `LINE_NUMBER_TRACKING.md` - Line number tracking implementation