# 1 White Paper Title

**An Example Technical Document**

**Author** Author Name
**Organization** Organization
**Email** author@example.com
**Date** 2025-11-23

## Abstract

This is an example abstract for the white paper. It provides a brief overview of the document's contents, methodology, and key findings. The abstract should be concise yet informative, typically 150-250 words.

This document demonstrates the use of Pandoc for professional technical writing with automated figure and table generation using Python scripts.

한글도 완벽하게 지원됩니다! Korean text is fully supported!

**Keywords:** technical writing, pandoc, markdown, automation, multilingual

## Contents

chapters/01-introduction chapters/02-methodology chapters/03-results chapters/04-conclusion chapters/05-korean-example chapters/06-system-overview chapters/90-appendix-documentation

## 1.1 Indices and tables

- `genindex`
- `search`

## 1.2 Introduction

This document serves as an example white paper template using Pandoc and Markdown. It demonstrates how to structure a professional technical document with automated figure and table generation using Python scripts.

### 1.2.1 Background

Technical writing often requires the integration of dynamically generated content such as figures, tables, and data visualizations. Traditional document preparation systems can make this process cumbersome and error-prone. This template provides a modern, reproducible approach to technical writing that combines:

- **Markdown** for easy-to-write, human-readable content
- **Pandoc** for professional document conversion
- **Python** for automated content generation
- **Version control** for collaborative editing

As demonstrated in previous work [@example2024], automated documentation workflows significantly improve reproducibility and reduce errors in scientific and technical publications.

### 1.2.2 Objectives

The primary objectives of this white paper template are:

1. **Simplicity**: Enable writers to focus on content rather than formatting
2. **Reproducibility**: Ensure all figures and tables can be regenerated consistently
3. **Flexibility**: Support multiple output formats (PDF, HTML, DOCX) from the same source
4. **Professional Quality**: Produce publication-ready documents with proper numbering and cross-references

### 1.2.3 Document Structure

This white paper is organized as follows:

- **Chapter 1 (Introduction)**: Provides context and objectives
- **Chapter 2 (Methodology)**: Describes the technical approach
- **Chapter 3 (Results)**: Presents findings with figures and tables
- **Chapter 4 (Conclusion)**: Summarizes key takeaways

Each chapter is maintained as a separate Markdown file in the `chapters/` directory, making it easy to edit and manage content modularly.

### 1.2.4 Cross-References

This template supports automatic cross-referencing of:

- **Sections**: Reference Section @sec:methodology for technical details
- **Figures**: See Figure @fig:example_plot for a visual example
- **Tables**: Refer to Table @tbl:comparison for data comparison
- **Equations**: Equation @eq:example demonstrates mathematical notation

These cross-references are automatically updated when the document is built, ensuring consistency throughout the paper.

## 1.3 Methodology

This chapter describes the technical approach and methodology used in this white paper template system.

### 1.3.1 System Architecture

The documentation system consists of several integrated components:

1. **Content Layer**: Markdown files containing the written content
2. **Generation Layer**: Python scripts that create figures and tables
3. **Build Layer**: Pandoc and Make for document compilation
4. **Output Layer**: Generated documents in various formats

This modular architecture ensures that each component can be developed and maintained independently, as discussed by @examplebook2023.

### 1.3.2 Content Generation

**Figure Generation** Figures are generated programmatically using Python's matplotlib library. The generation process follows these steps:

1. Define the data and visualization parameters
2. Create the plot using matplotlib
3. Apply styling for professional appearance
4. Save to the `../_static/figures/` directory at high resolution (300 DPI)

For example, the sine wave shown in Figure @fig:example_plot is generated using:

```
x = np.linspace(0, 10, 1000)
y = np.sin(x)
plt.plot(x, y)
plt.savefig('../_static/figures/example_line_plot.png', dpi=300)
```

**Table Generation** Tables are generated using pandas DataFrames and exported as Markdown format. This ensures:

- **Data integrity**: Tables reflect actual data values
- **Consistency**: Formatting is uniform across all tables
- **Reproducibility**: Tables can be regenerated with updated data

Table @tbl:comparison shows an example of a comparison table generated from the `generate_tables.py` script.

Table 1: Comparison of different methodological approaches

| Method | Accuracy (%) | Speed (ms) | Memory (MB) | Complexity |
|---|---|---|---|---|
| Method A | 92.5 | 12.3 | 256 | Low |
| Method B | 94.3 | 15.7 | 512 | Medium |
| Method C | 91.8 | 10.2 | 128 | Low |
| Method D | 95.1 | 18.4 | 1024 | High |

### 1.3.3 Mathematical Notation

The system supports mathematical equations using LaTeX syntax. For instance, the Gaussian function is defined as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

{#eq:gaussian}

where $\mu$ is the mean and $\sigma$ is the standard deviation.

More complex equations can also be included. The general form of a neural network layer computation is:

$$\mathbf{y} = \sigma(W\mathbf{x} + \mathbf{b})$$

{#eq:neural}

where $W$ is the weight matrix, $\mathbf{b}$ is the bias vector, and $\sigma$ is the activation function.

### 1.3.4 Build Process

The document build process is automated using a Makefile:

```
make all      # Generate figures, tables, and PDF
make pdf      # Build PDF only
make html     # Build HTML version
make clean    # Remove generated files
```

This automation ensures reproducibility and simplifies the workflow for authors, as noted in recent conference proceedings [@exampleconf2024].

## 1.4 Results

This chapter presents the results obtained using the methodologies described in Section @sec:methodology. All figures and tables are generated automatically from Python scripts to ensure reproducibility.

**Automatic Table Generation**

Tables in this chapter are generated using the TableManager system. They are automatically created by running:

```
python scripts/generate_tables_v2.py
```

Tables are saved to `../_static/tables/chapter-03/` and can be easily included or updated. See TABLE_WORKFLOW.md for details.

### 1.4.1 Visualization Results

**Line Plots** Figure @fig:scatter shows the relationship between variables with a fitted trend line. The scatter plot reveals a strong linear correlation with coefficient of determination $R^2 = 0.89$.

**Categorical Comparisons** Figure @fig:bars presents a categorical comparison across five different categories. Category D shows the highest value at 78 units, while Category E demonstrates the lowest at 32 units.

**Distribution Analysis** The histogram in Figure @fig:histogram illustrates the distribution of measured values. The data closely follows a normal distribution with mean $\mu = 100.2$ and standard deviation $\sigma = 14.8$.

**Correlation Analysis** Figure @fig:heatmap presents the correlation matrix between six different variables. Strong positive correlations (>0.7) are observed between Variables 1 and 3, while Variables 2 and 5 show weak correlation (<0.3).

### 1.4.2 Tabular Results

**Experimental Data** Table @tbl:experimental presents the experimental results across five different conditions. The data shows a clear trend of increasing yield with temperature, from 78.3% at 20.5°C to 92.5% at 40.5°C.

Table 2: Experimental results under different temperature and pressure conditions

| Experiment | Temperature (°C) | Pressure (kPa) | Yield (%) | Time (min) |
|---|---|---|---|---|
| Exp-1 | 20.5 | 101.3 | 78.3 | 45 |
| Exp-2 | 25.0 | 105.2 | 82.1 | 42 |
| Exp-3 | 30.5 | 110.1 | 85.7 | 38 |
| Exp-4 | 35.0 | 115.8 | 89.2 | 35 |
| Exp-5 | 40.5 | 121.3 | 92.5 | 32 |

**Statistical Summary** Table @tbl:stats provides a statistical summary of three datasets. Dataset B exhibits the highest mean value (119.64) and largest standard deviation (20.31), indicating greater variability compared to the other datasets.

Table 3: Statistical summary of experimental datasets

| Dataset | Mean | Std Dev | Min | Max | Median |
|---|---|---|---|---|---|
| Dataset A | 100.18 | 15.07 | 60.42 | 139.62 | 100.35 |
| Dataset B | 119.64 | 20.31 | 61.70 | 176.47 | 120.17 |
| Dataset C | 89.82 | 9.91 | 62.04 | 116.93 | 89.95 |

**Performance Metrics** Model performance across different data splits is summarized in Table @tbl:performance. The results demonstrate good generalization, with test set performance (F1-Score: 0.915) closely matching validation set performance (F1-Score: 0.918).

Table 4: Performance metrics evaluated on training, validation, and test sets

| Metric | Training Set | Validation Set | Test Set |
|---|---|---|---|
| Precision | 0.953 | 0.921 | 0.918 |
| Recall | 0.947 | 0.915 | 0.912 |
| F1-Score | 0.950 | 0.918 | 0.915 |
| Accuracy | 0.948 | 0.916 | 0.913 |
| AUC-ROC | 0.982 | 0.965 | 0.961 |

### 1.4.3 Key Findings

The analysis presented in this chapter yields several important findings:

1. **Strong Linear Relationships**: As shown in Figure @fig:scatter, the linear

model provides excellent fit to the experimental data
2. **Temperature Dependence**: Table @tbl:experimental demonstrates a clear positive correlation between temperature and yield
3. **Model Robustness**: Performance metrics in Table @tbl:performance indicate minimal overfitting
4. **Variable Independence**: The correlation analysis (Figure @fig:heatmap) reveals which variables can be treated independently

These findings support the methodology outlined in Chapter @sec:methodology and provide a foundation for the conclusions drawn in Chapter @sec:conclusion.

## 1.5 Conclusion

This white paper has demonstrated a comprehensive approach to technical document authoring using Markdown, Pandoc, and Python for automated content generation.

### 1.5.1 Summary of Contributions

The key contributions of this template system include:

1. **Automated Workflow**: Integration of Python-based figure and table generation with Pandoc document compilation
2. **Reproducibility**: All content can be regenerated consistently from source data
3. **Professional Output**: High-quality documents suitable for academic and technical publications
4. **Multi-format Support**: Single source generates PDF, HTML, and DOCX outputs
5. **Version Control Friendly**: Plain text format enables effective collaboration using Git

As discussed in Section @sec:methodology, the modular architecture separates content creation, visualization, and document compilation into distinct, manageable components.

### 1.5.2 Results Overview

The results presented in Chapter @sec:results demonstrate the capabilities of this system:

- **Visualizations**: High-quality figures (Figures @fig:example_plot, @fig:scatter, @fig:bars, @fig:histogram, @fig:heatmap) generated at publication quality (300 DPI)
- **Tables**: Automatically formatted tables (Tables @tbl:comparison, @tbl:experimental, @tbl:stats, @tbl:performance) with consistent styling
- **Cross-references**: Seamless linking between sections, figures, tables, and equations

- **Mathematical Notation**: Professional typesetting of equations (e.g., Equations @eq:gaussian, @eq:neural)

### 1.5.3 Best Practices

Based on the implementation of this system, several best practices emerge:

#### Content Management

- **Modular Chapters**: Maintain each chapter in a separate file for easier editing and collaboration
- **Meaningful IDs**: Use descriptive identifiers for cross-references (e.g., `#fig:methodology-f` rather than `#fig1`)
- **Version Control**: Commit source files (.md, .py, .yaml) but exclude generated outputs (.pdf, .png)

#### Figure Generation

- **High Resolution**: Save figures at 300 DPI or higher for publication quality
- **Consistent Styling**: Define standard plot styles in a configuration file or module
- **Descriptive Names**: Use clear, descriptive filenames for generated figures

#### Table Generation

- **Data-Driven**: Generate tables from actual data rather than manual entry
- **Standard Format**: Use pandas DataFrames for consistency across all tables
- **Precision Control**: Specify appropriate decimal places for numerical values

#### Build Automation

- **Makefile Targets**: Define clear targets for common build operations
- **Dependency Tracking**: Ensure figures and tables are regenerated before document compilation
- **Clean Operations**: Provide targets to remove generated files and start fresh

### 1.5.4 Limitations and Future Work

While this system provides a robust foundation for technical writing, several areas merit future development:

1. **Template Customization**: Additional LaTeX templates for specific publication styles
2. **Interactive HTML**: Enhanced HTML output with interactive plots using Plotly or Bokeh
3. **Collaborative Editing**: Integration with real-time collaboration tools

4. **Automated Testing**: Unit tests for Python scripts to ensure correct figure/table generation
5. **Bibliography Management**: Tighter integration with reference management systems (Zotero, Mendeley)

### 1.5.5 Final Remarks

This white paper template provides a modern, efficient approach to technical writing that addresses the common challenges of reproducibility, version control, and multi-format output. By combining the simplicity of Markdown with the power of Pandoc and Python, authors can focus on content creation while maintaining professional standards.

The complete source code and documentation for this template are available in the project repository. Users are encouraged to customize and extend the system to meet their specific requirements.

As noted in the Pandoc documentation [@exampleweb2024], the future of technical writing lies in formats that are both human-readable and machine-processable. This template embraces that philosophy while maintaining compatibility with traditional publishing workflows.

### 1.5.6 Acknowledgments

This template builds upon the excellent work of the Pandoc development team, the Python scientific computing community, and countless contributors to open-source documentation tools. Special thanks to the maintainers of pandoc-crossref for enabling sophisticated cross-referencing capabilities.

---

**For questions or contributions, please refer to the project repository documentation.**

## 1.6 한글 사용 예제 / Korean Example

이 장에서는 한글과 영어를 함께 사용하는 방법을 보여줍니다.

This chapter demonstrates how to use Korean and English together in your white paper.

### 1.6.1 혼합 언어 작성 / Mixed Language Writing

**기본 텍스트 / Basic Text** 한글과 영어를 자연스럽게 섞어서 사용할 수 있습니다. You can naturally mix Korean and English text. 예를 들어, **머신러닝** (machine learning)이나 **딥러닝** (deep learning)과 같은 기술 용어를 함께 사용할 수 있습니다.

기술 문서에서는 종종 한글 설명과 영어 원어를 병기합니다:

- **인공지능** (Artificial Intelligence, AI)

- **자연어 처리** (Natural Language Processing, NLP)
- **컴퓨터 비전** (Computer Vision, CV)

**수식과 한글 / Equations with Korean** 한글 설명과 함께 수학 공식을 사용할 수 있습니다. Mathematical equations can be used with Korean explanations.

정규분포의 확률밀도함수는 다음과 같이 정의됩니다:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

{#eq:korean-normal}

여기서 $\mu$는 평균 (mean), $\sigma$는 표준편차 (standard deviation)입니다.

### 1.6.2 표와 그림 / Tables and Figures

**한글 표 / Korean Tables** 한글로 작성된 표의 예시입니다. Here is an example of a table with Korean text:

Table 5: 온도에 따른 실험 결과 / Experimental results by temperature

| 실험 번호 | 온도 (°C) | 압력 (kPa) | 수율 (%) | Experiment | Temperature | Yield |
|---|---|---|---|---|---|---|
| 실험-1 | 20.5 | 101.3 | 78.3 | Exp-1 | 20.5°C | 78.3% |
| 실험-2 | 25.0 | 105.2 | 82.1 | Exp-2 | 25.0°C | 82.1% |
| 실험-3 | 30.5 | 110.1 | 85.7 | Exp-3 | 30.5°C | 85.7% |
| 실험-4 | 35.0 | 115.8 | 89.2 | Exp-4 | 35.0°C | 89.2% |
| 실험-5 | 40.5 | 121.3 | 92.5 | Exp-5 | 40.5°C | 92.5% |

표 @tbl:korean-exp 에서 볼 수 있듯이, 온도가 증가함에 따라 수율도 함께 증가했습니다.

As shown in Table @tbl:korean-exp, the yield increased with temperature.

**방법론 비교 표 / Method Comparison Table**

Table 6: 다양한 방법론의 성능 비교 / Performance comparison of different methods

| 방법 | 정확도 (%) | 속도 (ms) | 메모리 (MB) | 복잡도 |
|---|---|---|---|---|
| 방법 A | 92.5 | 12.3 | 256 | 낮음 |
| 방법 B | 94.3 | 15.7 | 512 | 중간 |
| 방법 C | 91.8 | 10.2 | 128 | 낮음 |
| 방법 D | 95.1 | 18.4 | 1024 | 높음 |

### 1.6.3 교차 참조 / Cross-References

**섹션 참조 / Section References** 한글로 된 섹션도 쉽게 참조할 수 있습니다. Korean sections can be easily referenced.

- 서론은 Section @sec:introduction 을 참조하세요
- 방법론은 Section @sec:methodology 를 참조하세요
- 이 장의 표 섹션은 Section @sec:korean-tables 에 있습니다

**그림 참조 / Figure References** 영어 장의 그림도 참조할 수 있습니다:

- Figure @fig:example_plot 는 삼각함수 그래프를 보여줍니다
- Figure @fig:scatter 는 산점도와 회귀선을 보여줍니다
- Figure @fig:heatmap 은 상관관계 히트맵입니다

**수식 참조 / Equation References** Equation @eq:korean-normal 은 정규분포를 나타내며, Equation @eq:gaussian 은 같은 공식의 영어 버전입니다.

### 1.6.4 인용 / Citations

한글로 작성할 때도 인용을 동일하게 사용할 수 있습니다. Citations work the same way in Korean text.

최근 연구에 따르면 [@example2024], 자동화된 문서 작성 시스템이 재현성을 크게 향상시킨다고 합니다.

여러 문헌을 동시에 인용할 수도 있습니다 [@example2024; @examplebook2023].

### 1.6.5 코드 블록 / Code Blocks

한글 주석이 포함된 코드 예제:

```
# 한글 주석도 잘 표시됩니다
import numpy as np
import matplotlib.pyplot as plt

# 데이터 생성
x = np.linspace(0, 10, 100)
y = np.sin(x)

# 그래프 그리기
plt.plot(x, y)
plt.xlabel('시간 (초)')  # X축 레이블
plt.ylabel('진폭')       # Y축 레이블
plt.title('사인파 그래프')
plt.show()
```

### 1.6.6 목록 / Lists

**순서 있는 목록 / Ordered List** 연구 진행 단계:

1. **문헌 조사** (Literature Review)
    - 관련 연구 검색
    - 선행 연구 분석
2. **실험 설계** (Experimental Design)
    - 변수 선정
    - 측정 방법 결정
3. **데이터 수집** (Data Collection)
    - 실험 수행
    - 결과 기록
4. **분석 및 해석** (Analysis and Interpretation)
    - 통계 분석
    - 결과 해석

**순서 없는 목록 / Unordered List** 주요 기여사항:

- **재현성 향상**: 모든 그림과 표를 자동으로 생성
- **다국어 지원**: 한글, 영어, 일본어, 중국어 등 지원
- **버전 관리**: Git을 통한 효율적인 협업
- **다양한 출력 형식**: PDF, HTML, DOCX 등

### 1.6.7 강조와 서식 / Emphasis and Formatting

한글에서도 다양한 서식을 사용할 수 있습니다:

- **굵게** (bold)
- 기울임 (italic)
- [STRIKEOUT: 취소선] (strikethrough)
- 코드 (inline code)
- 링크 (link)

### 1.6.8 주요 결과 / Key Findings

이 예제 장에서 다룬 내용:

1. **혼합 언어 작성**: 한글과 영어를 자연스럽게 섞어 사용
2. **표와 그림**: 한글 레이블과 캡션 사용 (Tables @tbl:korean-exp, @tbl:korean-methods)
3. **수식**: 한글 설명과 함께 수학 공식 사용 (Equation @eq:korean-normal)
4. **교차 참조**: 한글 텍스트에서 섹션, 그림, 표 참조
5. **코드**: 한글 주석이 포함된 코드 블록
6. **서식**: 다양한 마크다운 서식 기능

### 1.6.9 결론 / Conclusion

Pandoc과 XeLaTeX를 사용하면 한글과 영어를 완벽하게 혼용할 수 있습니다. Using Pandoc with XeLaTeX enables perfect mixing of Korean and English.

한글 폰트가 제대로 설치되어 있다면, 별도의 추가 설정 없이도 한글 문서를 작성할 수 있습니다. With proper Korean fonts installed, you can write Korean documents without additional configuration.

더 자세한 정보는 KOREAN_SUPPORT.md 문서를 참조하세요. For more information, see the KOREAN_SUPPORT.md guide.

## 1.7 System Overview and Use Cases

This chapter describes the design philosophy, features, and practical use cases of this documentation system.

### 1.7.1 Design Philosophy

This documentation system is built around four core principles that address common pain points in technical writing:

**Easy to Edit** Content is written in plain text formats (Markdown or reStructuredText) that require no special software. Any text editor works, and the syntax is human-readable even without rendering.

**Easy to Automate** Python scripts generate figures and tables programmatically. When underlying data changes, a single command regenerates all derived content. This ensures reproducibility and eliminates manual copy-paste errors.

**Easy to Render** A single source produces multiple output formats: PDF for printing, HTML for web viewing, DOCX for collaboration with Word users, and EPUB for e-readers. The Makefile handles all conversion automatically.

**Easy to Process** Plain text sources with semantic markup are ideal for processing by language models and other automated tools. The structured organization and metadata-rich tables preserve context for AI-assisted workflows.

### 1.7.2 Key Features

**Dual Build System** The system supports two documentation toolchains:

1. **Pandoc workflow**: Write in Markdown, convert to PDF/HTML/DOCX using Pandoc with pandoc-crossref for cross-references.
2. **Sphinx workflow**: Write in reStructuredText, build with Sphinx for full-featured HTML documentation with search, PDF via LaTeX, and EPUB.

Both workflows share the same Python scripts for figure and table generation, allowing you to choose the toolchain that best fits your needs.

```
# Pandoc workflow
```

```
make all          # Build PDF and HTML
make pdf          # Build PDF only

# Sphinx workflow
make sphinx-html  # Build HTML documentation
make sphinx-pdf   # Build PDF via LaTeX
```

**Automated Content Generation** Figures and tables are generated by Python scripts rather than created manually:

**Figures**: The `generate_figures.py` script creates publication-quality plots using matplotlib at 300 DPI. When data or visualization requirements change, regenerating figures is a single command.

**Tables**: The `TableManager` class creates tables with YAML frontmatter containing metadata (caption, label, description). The preprocessor automatically replaces placeholders like `{{table:tbl:03-results}}` with actual table content during build.

```
from scripts.table_manager import TableManager

tm = TableManager(chapter=3)
tm.save_table(
    dataframe,
    name="experimental-results",
    caption="Experimental Results Summary",
    description="Results from the primary experiment series."
)
```

**Self-Contained Tables** Each generated table is a self-contained file with YAML frontmatter:

```
---
label: tbl:03-experimental-results
caption: Experimental Results Summary
chapter: 3
description: Results from the primary experiment series.
generated: 2025-01-15T10:30:00
---

| Metric | Value | Unit |
|--------|-------|------|
| Accuracy | 95.2 | % |
| Precision | 94.8 | % |
```

This metadata travels with the table, providing context even when processed by external tools or language models.

**Korean/CJK Support** Full support for Korean, Japanese, and Chinese text:

- Pre-configured Noto CJK fonts for PDF output
- Korean matplotlib utilities for properly rendered plot labels
- Mixed-language documents work seamlessly
- UTF-8 throughout the entire pipeline

```
from scripts.utils.korean_plot import setup_korean_font

setup_korean_font()
plt.xlabel('측정값 (Measurements)')
plt.title('실험 결과 분석')
```

**Adaptive Toolchain** The Makefile automatically detects your Pandoc version and adjusts accordingly:

- Uses `--citeproc` for Pandoc 2.11+ or `--filter pandoc-citeproc` for older versions
- Gracefully degrades when optional filters (pandoc-crossref) are unavailable
- Provides warnings but continues building when possible

This ensures the system works across different environments without manual configuration.

### 1.7.3 Use Cases

**Technical Reports and White Papers** The primary use case: writing technical documents where figures and tables derive from data analysis.

**Workflow**:

1. Perform analysis in Python, generating figures and tables
2. Write narrative content in Markdown/RST
3. Reference figures and tables using cross-reference syntax
4. Build the complete document with `make pdf`

When the analysis updates, regenerate content and rebuild. The document stays synchronized with your data.

**Reproducible Research** For academic papers or research reports where reproducibility matters:

- All figures regenerable from source data via Python scripts
- Git tracks both content and generation scripts
- Anyone can clone the repository and rebuild the exact same document
- Changes to methodology are reflected automatically in output

**LLM-Assisted Documentation** The plain text format and structured organization make this system ideal for AI-assisted workflows:

**Content generation**: Language models can read the source files, understand the document structure, and generate new sections that follow existing patterns.

**Data analysis**: LLMs can write Python scripts that generate figures and tables, which integrate seamlessly into the document.

**Review and editing**: The semantic markup (cross-references, labeled sections) provides context that helps LLMs understand relationships between document parts.

**Automation**: LLM agents can trigger build commands, check for errors, and iterate on content.

Example LLM workflow:

1. Provide the LLM with source files for context
2. Request new analysis or content
3. LLM generates Python scripts and/or document sections
4. Build and review the output
5. Iterate as needed

**Multi-Format Publishing** When the same content must reach different audiences:

- **PDF**: For formal distribution, printing, archival
- **HTML**: For web publishing with navigation and search
- **DOCX**: For collaboration with reviewers who use Word
- **EPUB**: For reading on mobile devices

Write once, publish everywhere. The Makefile handles all conversions.

**Collaborative Writing** Git-friendly plain text enables effective collaboration:

- Multiple authors work on different chapters simultaneously
- Pull requests for review before merging changes
- Clear diff views show exactly what changed
- No binary file conflicts

### 1.7.4 Getting Started

Quick setup for new users:

```
# Clone or create project
cd your-project

# Set up Python environment
uv venv
uv sync --all-extras

# Generate content and build
make all
```

The document builds to `whitepaper.pdf` and `whitepaper.html`.

For Sphinx output:

```
make sphinx-html
```

Output appears in `output/html/index.html`.

### 1.7.5 Summary

This documentation system addresses the core challenges of technical writing:

Table 7: System Capabilities

| Capability | Description |
|---|---|
| Easy editing | Plain text Markdown/RST with any editor |
| Reproducibility | Python scripts regenerate all derived content |
| Multi-format output | PDF, HTML, DOCX, EPUB from single source |
| LLM compatibility | Structured plain text ideal for AI processing |
| Automation | Single command builds via Makefile |
| Collaboration | Git-friendly with clear diffs |
| Internationalization | Full Korean/CJK support |

The result is a modern documentation workflow that scales from simple reports to complex technical publications, supports both human and AI authors, and produces professional output in any required format.

## 1.8 Appendix: Project Documentation

This appendix contains comprehensive documentation for the Pandoc white paper authoring system.

### 1.8.1 A. Quick Start Guide

#### Installation (Linux)

```
# System dependencies
sudo apt update
sudo apt install pandoc texlive-full fonts-noto-cjk build-essential

# Python environment
curl -LsSf https://astral.sh/uv/install.sh | sh
cd pandoc_report
uv venv
source .venv/bin/activate
uv pip install matplotlib numpy pandas seaborn scipy pillow tabulate pyyaml
```

#### Daily Workflow

```
# 1. Activate environment
source .venv/bin/activate

# 2. Generate content
python scripts/generate_figures.py
python scripts/generate_tables_v2.py

# 3. Build document
make pdf              # or: make html, make all
```

#### Common Commands

| Command | Description |
|---|---|
| `make all` | Generate figures, tables, and PDF |
| `make pdf` | Build PDF only |
| `make html` | Build HTML only |
| `make clean` | Remove generated documents |
| `make help` | Show all available commands |

### 1.8.2 B. Build System Guide

**Makefile vs Shell Script Use “make pdf“ (Recommended):** - Simpler commands - Automatic dependency handling - Includes pandoc-crossref filter - Industry standard

**Use “./build_example.sh“ (For learning):** - Shows exact Pandoc commands - Useful for debugging - Educational purposes

#### Build Process

```
1. make figures  →  2. make tables  →  3. make preprocess  →  4. Pandoc build
   (Python)            (Python)           (Replace {{table:..}})    (PDF/HTML)
```

### 1.8.3 C. Korean/CJK Language Support

#### Setup

```
# Install Korean fonts
sudo apt install fonts-noto-cjk fonts-noto-cjk-extra

# Fonts are pre-configured in metadata.yaml:
# CJKmainfont: "Noto Serif CJK KR"
```

**Usage** Simply write Korean in your markdown:

```
# 서론 {#sec:intro}

한글과 English를 자유롭게 섞어 사용할 수 있습니다.

{{table:tbl:05-korean-data}}
```

#### Korean in Matplotlib

```
from utils.korean_plot import setup_korean_font
setup_korean_font()

plt.xlabel('한글 레이블')
plt.ylabel('값')
```

### 1.8.4 D. Automatic Table System

**Table Format (Self-Contained Objects)** Each table is a `.md` file with YAML frontmatter:

```
---
id: tbl:03-results
caption: "Experimental results"
description: "Detailed context about the table"
chapter: 3
tags: [experimental, temperature]
---

| Column 1 | Column 2 |
|----------|----------|
| Data     | Data     |
```

#### Creating Tables

```
from table_manager import TableManager

tm = TableManager(chapter=3)
tm.save_table(
    data,
    name="my-results",
    caption="Experimental results",
    description="Results from temperature experiments",
    tags=["experimental"]
)
# Saves to: ../_static/tables/chapter-03/table-my-results.md
```

**Using Tables in Chapters  Placeholder syntax:**

```
## Results

We conducted experiments:

{{table:tbl:03-my-results}}

The results show...
```

**Modes:** - `{{table:tbl:03-name}}` - Full mode (includes description) - `{{table:tbl:03-name|inline` - Inline mode (table only)

**Build Process**

```
make pdf  # Automatically replaces placeholders!
```

The preprocessor: 1. Scans chapters for `{{table:...}}` placeholders 2. Loads tables from `../_static/tables/` 3. Replaces placeholders with actual content 4. Builds document

**Manual Preprocessing**

```
# Preview changes
python scripts/table_preprocessor.py --dry-run

# Replace placeholders
python scripts/table_preprocessor.py

# List available tables
python scripts/table_preprocessor.py --list-tables
```

### 1.8.5 E. Path Configuration

**Path Rules** All paths in markdown files are **relative to project root**:

```
pandoc_report/              ← Run `make pdf` here
├── chapters/
│   └── 03-results.md       ← Your file
├── ../_static/
│   ├── figures/
│   │   └── plot.png        → ../_static/figures/plot.png
│   └── tables/
│       └── chapter-03/     → {{table:tbl:03-name}}
└── assets/
    └── images/
        └── logo.png         → assets/images/logo.png
```

#### Correct Paths

```
<!-- ✅ CORRECT -->
![Plot](../_static/figures/example_plot.png){#fig:plot}
![Logo](assets/images/logo.png)
{{table:tbl:03-results}}

<!-- ❌ WRONG -->
![Plot](../../_static/figures/example_plot.png)
```

#### Fixing Path Issues

```
# Fix all image paths
find chapters/ -name "*.md" -exec sed -i 's|../../_static/|../_static/|g' {} \;
```

### 1.8.6 F. Troubleshooting

**Common Issues Makefile: "source: not found"** - Fixed! Makefile now uses bash shell directly - Python called via `.venv/bin/python`

**Images not found** - Check paths are from project root: `../_static/figures/plot.png` - Not relative to chapter: `../../_static/figures/plot.png`

**Korean text shows as boxes**

```
sudo apt install fonts-noto-cjk
fc-cache -fv
```

#### Table placeholder not replaced

```
# List available tables
python scripts/table_preprocessor.py --list-tables

# Check placeholder syntax
{{table:tbl:03-name}}  # ✅ Correct
{{tbl:03-name}}        # ❌ Wrong
```

**pandoc-crossref version warning** - Minor warning, usually safe to ignore - Cross-references still work - Update pandoc-crossref to match Pandoc version if needed - See: https://github.com/lierdakil/pandoc-crossref/releases

#### Python ModuleNotFoundError

```
source .venv/bin/activate
uv pip install matplotlib numpy pandas seaborn scipy pillow tabulate pyyaml
```

### 1.8.7 G. Project Architecture

#### Directory Structure

```
pandoc_report/
├── chapters/              # Markdown chapter files
│   ├── 01-introduction.md
│   ├── 02-methodology.md
│   ├── 03-results.md
│   └── 90-appendix-documentation.md
├── scripts/              # Python scripts
│   ├── generate_figures.py
│   ├── generate_tables_v2.py
│   ├── table_manager.py
│   ├── table_preprocessor.py
│   └── utils/
│       └── korean_plot.py
├── ../_static/               # Generated content
│   ├── figures/
│   └── tables/
│       └── chapter-{NN}/
├── assets/               # Static resources
│   └── images/
├── templates/            # Pandoc templates
├── references/           # Bibliography
│   └── references.bib
├── metadata.yaml         # Document metadata
├── Makefile             # Build automation
├── pyproject.toml       # Python dependencies
└── README.md            # Quick start guide
```

**File Organization** **Source files (commit to git):** - `chapters/*.md` - Your content - `scripts/*.py` - Analysis and generation - `metadata.yaml` - Document configuration - `Makefile` - Build system - `pyproject.toml` - Dependencies

**Generated files (don't commit):** - `../_static/figures/*.png` - Generated plots - `../_static/tables/**/*.md` - Generated tables - `*.pdf`, `*.html` - Build outputs - `.venv/` - Virtual environment

### 1.8.8 H. Advanced Features

#### Cross-References

```
# Chapter {#sec:chapter}

## Results {#sec:results}

![Plot](../_static/figures/plot.png){#fig:plot width=80%}

| A | B |
|---|---|
| 1 | 2 |
```

```
: Caption {#tbl:data}

$$E = mc^2$$ {#eq:einstein}

See Section @sec:results, Figure @fig:plot,
Table @tbl:data, and Equation @eq:einstein.
```

**Citations**

```
Recent work [@smith2020; @jones2021] shows...
@smith2020 demonstrated that...
```

Add entries to `references/references.bib`:

```
@article{smith2020,
  title={Example Article},
  author={Smith, John},
  journal={Example Journal},
  year={2020}
}
```

**Custom Pandoc Options** Edit `Makefile` to add options:

```
COMMON_OPTS = --number-sections --toc --toc-depth=3 \
              --highlight-style=tango \
              --variable=geometry:margin=1in
```

**Multiple Output Formats**

```
make pdf          # PDF via LaTeX
make html         # Standalone HTML
make docx         # Microsoft Word
make all-formats  # All formats
```

### 1.8.9 I. Best Practices

**1. Version Control**

```
# Commit source files
chapters/
scripts/
metadata.yaml
Makefile
pyproject.toml

# Don't commit generated files
../_static/
```

```
*.pdf
*.html
.venv/
```

**2. Table Management**

- Use descriptive names: `table-temperature-results.md`
- Include descriptions for context
- Tag tables for organization
- Regenerate when data changes

**3. Figure Quality**

```
# Save at high DPI for publication
plt.savefig('../_static/figures/plot.png', dpi=300, bbox_inches='tight')
```

**4. Modular Chapters**

- One chapter per file
- Use meaningful section IDs
- Keep chapters focused

**5. Reproducibility**

```
# Document your workflow
make clean-all    # Remove everything
make all          # Rebuild from scratch
```

### 1.8.10 J. Quick Reference

**Build Commands**

```
make pdf          # Build PDF
make html         # Build HTML
make all          # Everything
make clean        # Remove outputs
make help         # Show options
```

**Python Scripts**

```
# Generate content
python scripts/generate_figures.py
python scripts/generate_tables_v2.py
python scripts/my_analysis.py

# Manage tables
python scripts/table_preprocessor.py --list-tables
python scripts/table_preprocessor.py --dry-run
```

#### Markdown Syntax

```
# Heading {#sec:id}
![Caption](path){#fig:id width=80%}
: Caption {#tbl:id}
$$equation$$ {#eq:id}
{{table:tbl:03-name}}
[@citation]
@fig:id, @tbl:id, @sec:id
```

#### File Paths

| Resource | Path Format |
|---|---|
| Figures | `../_static/figures/name.png` |
| Tables | `{{table:tbl:NN-name}}` |
| Static images | `assets/images/name.png` |
| Bibliography | `references/references.bib` |

### 1.8.11 K. Resources

#### Documentation

- Pandoc Manual: https://pandoc.org/MANUAL.html
- Pandoc-Crossref: https://lierdakil.github.io/pandoc-crossref/
- Markdown Guide: https://www.markdownguide.org/
- UV: https://github.com/astral-sh/uv
- Matplotlib: https://matplotlib.org/

#### Project Files

- `scripts/table_manager.py` - TableManager API
- `scripts/table_preprocessor.py` - Preprocessor
- `scripts/demo_table_workflow.py` - Working demo
- `Makefile` - Build system reference

#### Getting Help

```
# Show available make targets
make help

# List available tables
python scripts/table_preprocessor.py --list-tables

# Check versions
pandoc --version
python --version
make --version
```

### 1.8.12 L. Summary of Features

**✅ Implemented Features**

1. **Automated Build System**
   - Makefile with dependency management
   - Automatic figure/table generation
   - Multi-format output (PDF, HTML, DOCX)
2. **Korean/CJK Language Support**
   - Full UTF-8 support
   - Korean fonts configured
   - Mixed language documents
   - Korean matplotlib plots
3. **Automatic Table System**
   - Self-contained table objects
   - YAML frontmatter metadata
   - Placeholder replacement
   - Multiple modes (full/inline)
4. **Reproducible Workflow**
   - Python-based generation
   - Version control friendly
   - UV virtual environment
   - Documented dependencies
5. **Professional Output**
   - Publication-quality figures (300 DPI)
   - Automatic numbering
   - Cross-references
   - Bibliography management
6. **Developer-Friendly**
   - Modular architecture
   - Clear documentation
   - Example scripts
   - Troubleshooting guides

This system provides a complete, professional authoring environment for technical white papers with automated content generation and multi-language support.