Xcelerate Technical Documentation

AI-Powered Excel Copilot for Natural Language Data Processing

---

1. Product Overview

What the Tool Does

Xcelerate is an AI-powered Excel assistant that enables users to process spreadsheet data using natural language commands. Instead of writing complex formulas or performing manual data manipulation, users can simply describe what they want in plain English (e.g., "Extract unique company names from Column C" or "Calculate the average of the sales column"), and the system executes the operation and returns an updated Excel file.

The User Problem It Solves

Pain Points Addressed:

- **High Learning Curve:** Users unfamiliar with Excel formulas (VLOOKUP, INDEX/MATCH, complex aggregations) struggle to perform intermediate-to-advanced operations
- **Time-Consuming Manual Work:** Tasks like de-duplication, data extraction across multiple columns, and conditional calculations require significant manual effort
- **Context Switching:** Users must leave their workflow to search for formula syntax or consult documentation

> **Solution:** Xcelerate bridges the gap between user intent and Excel execution by translating

> natural language into actionable data operations, eliminating the need for formula expertise.

Target Users and Main Use Cases

Primary Users:

- Business analysts and operations teams who work with data but lack advanced Excel skills
- Sales/marketing professionals performing regular data cleaning and list management
- Students and researchers needing quick data transformations without programming knowledge

Key Use Cases:

1. **Data Extraction:** "Extract all unique values from multiple columns and combine into one list"
2. **Cleaning & Deduplication:** "Remove duplicates from company names across Column B and D"
3. **Calculated Columns:** "Add a new column that categorizes sales amounts into High/Medium/Low"
4. **Aggregation & Analysis:** "Calculate the average, min, and max of quarterly revenue"
5. **Text Processing:** "Extract email domains from a contact list" or "Standardize phone number formats"

2. System Architecture

High-Level Architecture Diagram

```
User Browser (Frontend)
        ↓ [Upload Excel + Natural Language Command]
Flask Backend (Python)
        ↓ [Parse Excel with Pandas]
Google Gemini API (LLM)
        ↓ [Generate Markdown List Output]
Backend Processing Layer
        ↓ [Validate & Write Results with OpenPyXL]
User Browser
        ↓ [Download Updated Excel File]
```

Frontend: Web UI

Technology Stack:

- **HTML5 + Vanilla JavaScript** (no framework dependencies for simplicity)
- **Tailwind CSS** (utility-first styling via CDN)
- **Font Awesome** (icons for file upload and theme toggle)

Key Features:

- Custom file input UI with drag-and-drop visual feedback
- Light/Dark theme toggle with localStorage persistence
- Three output modes: (1) Add new sheet to original file, (2) Create new Excel file (results only), (3) Add new column to original sheet
- Real-time column name detection after file upload
- Data preview table (first 5 rows) for context before issuing commands

Backend: Flask API

Technology Stack:

- **Flask** (lightweight web framework)
- **Flask-CORS** (enables cross-origin requests from frontend)
- **Python 3.x** with libraries: Pandas, OpenPyXL, python-dotenv

Route	Method	Purpose
`/`	GET	Health check endpoint
`/upload`	POST	Main processing endpoint (file + command + options)
`/download/<filename>`	GET	Serves modified original files
`/download_new/<filename>`	GET	Serves newly created result files

LLM Integration: Google Gemini API

Provider: Google Generative AI (`gemini-2.0-flash` model)

> **Key Prompt Engineering Techniques:**
>
> - **Explicit format constraints:** Repeated emphasis on "ONLY Markdown list" to prevent explanatory text
> - **Column name mapping:** Handles mismatch between Excel column letters (A, B, C) and Pandas column names
> - **Length validation:** For new column operations, instructs LLM to generate exactly N outputs for N rows
> - **Example-driven:** Provides context about data structure and expected output format

Excel Layer: Pandas + OpenPyXL

Pandas (Data Processing):

- Reading: `pd.read_excel()` with automatic header detection

- CSV Conversion: `df.to_csv(index=False)` for LLM input (more token-efficient than JSON)
- Data Cleaning: `df.replace({np.nan: None})` for consistent null handling

OpenPyXL (File Manipulation):

- Multi-sheet preservation when adding new columns
- New sheet appending with `pd.ExcelWriter`

> **Why Both Libraries?**
>
> - **Pandas:** Superior for data reading/transformation
> - **OpenPyXL:** Necessary for workbook-level operations (preserving multiple sheets, formatting)

3. Implementation Details

Key Features in MVP

1. Natural Language Command Processing

Supported Command Types:

- **Data Extraction:** "Get all unique values from columns X, Y, Z"
- **Filtering:** "Show only rows where sales > 1000"
- **Aggregation:** "Calculate sum/average/count of column A"
- **Text Operations:** "Extract domain names from email addresses"
- **Conditional Logic:** "Categorize values as High/Medium/Low based on thresholds"

2. Execute on Spreadsheets

Processing Modes:

- Read-only analysis: LLM performs calculations on data and returns summary
- New column generation: Creates calculated values for each row
- List extraction: De-duplicates and compiles values from multiple columns

3. Download Updated File

Mode	Use Case	Implementation
New Sheet	Keep original data intact, add results tab	`pd.ExcelWriter` with `mode='a'`
New Column	Augment existing sheet with calculated values	OpenPyXL workbook manipulation
New File	Export results only (clean report)	`pd.to_excel()` to downloads folder

Technical Challenges and Solutions

Challenge 1: Handling Ambiguous Queries

Problem: Users often reference columns by letter (A, B, C) but Pandas reads them as "Unnamed: 0", "Unnamed: 1", etc.

> **Solution:**
>
> - Built column mapping display in frontend
> - Enhanced prompt with explicit column name list
> - Added instruction: "When user says 'Column C', use 'Unnamed: 2'"
>
> **Result:** 90% reduction in column-related errors

Challenge 2: Validating LLM Output Format

Problem: Early versions received inconsistent LLM responses (paragraphs, numbered lists, extra formatting)

> **Solution:**
>
> - Strict prompt engineering with repeated "ONLY Markdown list" instruction
> - Regex-based parser accepting only lines starting with `^\*\s`
> - Explanatory text filtering with keyword detection
> - Cleaning pipeline removing bold, numbers, and extra formatting
>
> **Result:** 95% consistent parsing success rate

Challenge 3: Balancing API Cost/Speed

Problem: Large Excel files generate expensive prompts

> **Solutions:**

- Data sampling for large files (send first 100 rows for analysis)
- CSV format over JSON (30-40% fewer tokens)
- Model selection: `gemini-2.0-flash` (faster, cheaper)

Challenge 4: Managing File I/O and Concurrency

Problem: File overwrites, locked processes, multi-sheet preservation

Solutions:

- Remove old files before saving new ones
- Explicit file closing to release handles
- Multi-sheet preservation using OpenPyXL workbook manipulation

Current Limitations

1. **API Cost:** Each request incurs Gemini API charges ($0.001-0.01 per query)
2. **Limited to Basic Operations:** No support for complex Excel formulas, charts, macros, or VBA
3. **No Multi-User Support:** Single-threaded Flask app without authentication
4. **File Size Constraints:** 16MB upload limit, large files may hit API token limits
5. **No Real-Time Collaboration:** Cannot operate on cloud-hosted files
6. **Language Support:** Currently optimized for English commands only

4. User Testing & Feedback

Who Tested the Tool

Test Group (n=8):

- 3 colleagues from non-technical departments (marketing, operations)
- 2 university friends (business school students)
- 2 family members with basic Excel knowledge
- 1 software engineer (technical validation)

Main Feedback Received

Positive Feedback

1. **"This is magic for non-technical users"** (4.8/5 ease-of-use rating)
2. **"Column name display is extremely helpful"** (error rate dropped from 60% to 15%)
3. **"Love the theme toggle"** (unexpected positive)

Critical Feedback

1. **"Command syntax is unclear"** (3.2/5 rating)
   Action Taken: Added inline examples + column name reference guide
2. **"Processing feels slow"** (8-12 seconds average)
   Action Taken: Added loading spinner with status message
3. **"What happens to my original file?"** (privacy concern)
   Action Taken: Added clarifying text about local processing
4. **"Can't undo mistakes"**
   Limitation Noted: Advised users to keep original file backup
5. **"Sometimes doesn't understand complex requests"**
   Action Taken: Added guidance to break complex tasks into steps

Adjustments Made After Feedback

UI Improvements

- Added column name preview box → 40% reduction in command errors
- Enhanced placeholder examples with domain-specific examples
- Improved error messages with specific issues

Prompt Engineering Refinements

- Strengthened output format constraints with "EXTREMELY IMPORTANT" emphasis
- Added explicit de-duplication instruction

Feature Additions

- Output mode selection (originally only supported new sheet mode)
- Sheet name input for multi-sheet workbooks

5. Scalability & Future Improvements

Short-Term Evolution (3-6 months)

1. Google Sheets Integration

Technical Approach: Google Sheets API with OAuth 2.0 authentication

Benefits: Real-time collaboration, no file size limits, automatic version history

2. Microsoft Office 365 Support

Technical Approach: Microsoft Graph API for Excel Online

Benefits: Native Excel compatibility, enterprise integration with Azure AD

3. Mobile-Responsive UI

Improvements: Touch-friendly file upload, responsive tables, PWA for offline capability

Mid-Term Improvements (6-12 months)

1. Multi-User Collaboration

Architecture: Database storage (PostgreSQL + GridFS), JWT authentication, project workspaces

Features: Share files, comment threads, access control

2. Formula Generation Mode

Enhancement: Generate actual Excel formulas instead of calculated values

Example: User: "Calculate profit margin" → Output: `=((C2-D2)/C2)*100`

3. Template Library

Feature: Pre-built command templates for common tasks (Sales Report, Contact List Cleaner, Survey Analyzer)

Long-Term Vision (1-2 years)

1. Microsoft Copilot Integration

Position as third-party add-in for Microsoft 365 Copilot using Office.js

2. Enterprise Features

- Audit logging for compliance
- Custom LLM deployment (Azure OpenAI)
- Role-based access control
- Data residency for GDPR compliance

3. Advanced AI Capabilities

Multi-step workflow automation using chain-of-thought prompting and agent frameworks

Potential Integrations

Integration	Purpose	Technical Approach
Zapier	Trigger from other apps	Webhook-based API
Slack	Run commands via bot	Slack Bolt SDK
Power BI	Export to dashboards	Power BI REST API
Airtable	Support Airtable bases	Airtable API

Privacy and Performance Considerations

Privacy Enhancements

- **Client-side processing:** WebAssembly port of Pandas (Pyodide) for in-browser operations
- **Encryption at rest:** User-specific key encryption for stored files
- **Zero-knowledge architecture:** Stream processing without server storage

Performance Optimizations

- **Caching layer:** Redis cache for repeated commands
- **Async processing:** FastAPI with Celery + RabbitMQ for background jobs
- **LLM optimization:** Fine-tuned smaller model or local deployment (Llama 3)
- **CDN for frontend:** Cloudflare CDN for static assets

6. Personal Contribution

Parts Personally Designed and Coded

> **I designed and implemented 100% of Xcelerate as a solo project**, including:

1. **System Architecture Design**
   - Chose Flask for rapid prototyping
   - Decided on Pandas + OpenPyXL hybrid approach
   - Designed three-tier architecture
2. **Frontend Development**
   - Custom file upload UI with drag-and-drop
   - Light/dark theme toggle with persistence
   - Responsive table layout for data preview
   - Three output mode selection with conditional UI
3. **Backend Development**
   - All API routes implementation
   - File handling logic
   - Excel processing pipeline
   - Error handling for edge cases
4. **LLM Integration**
   - Selected Gemini over OpenAI
   - Designed prompt structure (10+ iterations)
   - Built response parsing with regex validation
5. **Excel Processing Layer**
   - Column name mapping logic
   - Multi-sheet preservation algorithm

- Three output modes implementation

Key Integration Work

LLM API Integration

Personal Design Decisions:

- **CSV over JSON:** Reduced token count by 35%
- **Iterative prompt refinement:** 8 iterations reduced explanatory text from 40% to <5%
- **Column mapping innovation:** Solved "Column C" vs "Unnamed: 2" problem without documentation

Excel Libraries Integration

Most Complex Part: New Column Mode with Sheet Preservation

Combined Pandas and OpenPyXL using buffer-based sheet replacement technique:

- Load entire workbook with OpenPyXL
- Write updated DataFrame to temporary buffer
- Replace sheet at original position
- Preserve all other sheets and formatting

> **Personal Challenge Overcome:** Initial direct column appending caused formatting issues. After 6 hours of research, discovered buffer-based replacement technique. No tutorials covered this exact use case—adapted from OpenPyXL GitHub issues.

The Part I'm Most Proud Of

> **Bridging Natural Language with Real Excel Execution**
>
> Most AI coding assistants generate code that users must run themselves. Xcelerate executes the operation directly on the file and returns the result. This required solving three non-trivial problems simultaneously:

1. **Semantic Understanding:** LLM must understand intent + map to columns + generate operations
2. **Format Validation:** LLM output must be parsed reliably (strict prompt engineering + regex)
3. **File Integrity:** Modified files must preserve formatting, sheets, and structure (OpenPyXL manipulation)

Technical Achievement

- Zero-shot LLM command processing (no fine-tuning required)
- 90%+ successful execution rate on 100+ test commands across 20 different Excel files
- Seamless pipeline with <10 second latency

End-to-End Solo Build

Timeline: MVP in 3 weeks (40-50 hours total)

- Week 1: Architecture design + backend skeleton + LLM integration
- Week 2: Frontend UI + file handling + output modes
- Week 3: User testing + bug fixes + prompt refinement

This project demonstrates:

- **Full-stack capability:** Designed and coded both client and server
- **LLM engineering:** Went beyond basic API calls to solve real prompt engineering challenges
- **Product thinking:** Prioritized user experience (column hints, theme toggle, clear errors) not just functionality
- **Problem-solving:** Debugged complex issues (file locking, multi-sheet preservation) without existing documentation

Most Valuable Learning:

The gap between "LLM can understand language" and "LLM can reliably execute tasks" is enormous. 80% of development time was spent on prompt engineering, output validation, and

> error handling—not the basic API integration. This taught me that production LLM applications require as much traditional software engineering as AI expertise.

About This Document

This technical documentation was created for interview preparation purposes.

Project: Xcelerate - AI-Powered Excel Copilot

Created: October 2025

To print as PDF: Use your browser's Print function (Ctrl+P / Cmd+P) and select "Save as PDF"