

# LEXICAL MULTI-DIMENSIONAL ANALYSIS SOLUTION

## High-Level Design

Tony Berber Sardinha                      Rogério Yamada

2025

### Abstract

This document describes the high-level design of a Python-based application for Lexical Multi-Dimensional Analysis (LMDA), an extension of the Multi-Dimensional (MD) Analysis framework developed by Biber in the 1980s (“Multi-Feature Multi-Dimensional Analysis”) to study register variation. Through the identification of (lexical) dimensions or sets of correlated lexical features, LMDA enables the analysis of lexical patterning from a multi-dimensional perspective. These lexical dimensions represent a variety of latent, macro-level discursive constructs. (Berber Sardinha & Fitzsimmons-Doolan, 2025)

**Keywords:** discursive constructs; Lexical Multi-Dimensional Analysis; lexical patterning; Multi-Dimensional Analysis

### List of Figures

1	Lexical Multi-Dimensional Analysis (Berber Sardinha & Fitzsimmons-Doolan, 2025) .	3
2	Lexical Multi-Dimensional Analysis Procedures . . . . .	3
3	From text to factor . . . . .	3
4	Lexical variables selection via keywords (Scott, 1997) . . . . .	5
5	Lexical variables selection by correlation . . . . .	6

### Contents

1	Project Overview	2
1.1	Purpose . . . . .	2
1.2	Scope . . . . .	2
1.3	Stakeholders . . . . .	4
2	Functional Requirements	4
2.1	Corpus Ingestion . . . . .	4
2.2	Preprocessing . . . . .	4
2.3	Feature Extraction . . . . .	4
2.4	Dimensional Analysis . . . . .	5
2.5	Visualisation and GUI . . . . .	5
2.6	Export and Reporting . . . . .	5
2.7	Automation and Scripting . . . . .	5

<b>3</b>	<b>Non-Functional Requirements</b>	<b>7</b>
3.1	Performance . . . . .	7
3.2	Usability . . . . .	7
3.3	Reliability and Robustness . . . . .	7
3.4	Portability and Compatibility . . . . .	7
3.5	Security . . . . .	7
<b>4</b>	<b>System Architecture</b>	<b>7</b>
4.1	Overview . . . . .	7
4.2	Components . . . . .	7
4.3	Data Flow . . . . .	8
<b>5</b>	<b>Technology Stack</b>	<b>8</b>
<b>6</b>	<b>Interfaces and APIs</b>	<b>9</b>
6.1	Python Module Interfaces . . . . .	9
6.2	Plug-in Interface . . . . .	9
6.3	CLI . . . . .	9
6.4	Optional REST API (FastAPI) . . . . .	9
<b>7</b>	<b>Security and Privacy Considerations</b>	<b>9</b>
7.1	Data Protection . . . . .	9
7.2	Access Control (Server Mode) . . . . .	10
7.3	Supply Chain . . . . .	10
<b>8</b>	<b>Risks and Mitigations</b>	<b>10</b>
<b>9</b>	<b>Timeline and Milestones</b>	<b>10</b>
<b>A</b>	<b>Appendices</b>	<b>12</b>
A.1	work in progress . . . . .	12

# 1 Project Overview

## 1.1 Purpose

This document describes the high-level design of a Python-based application for Lexical Multi-Dimensional Analysis (LMDA) (Figure 1) with an interactive graphical user interface (GUI). The system ingests text corpora, performs linguistic preprocessing, extracts lexical features, projects texts into a multi-dimensional factor space via Factor Analysis (FA), and presents results through interactive visualisations and reports (Figures 2 and 3).

## 1.2 Scope

The application supports:

- Corpus ingestion from files, folders, and CSV/TSV and JSONL columns;

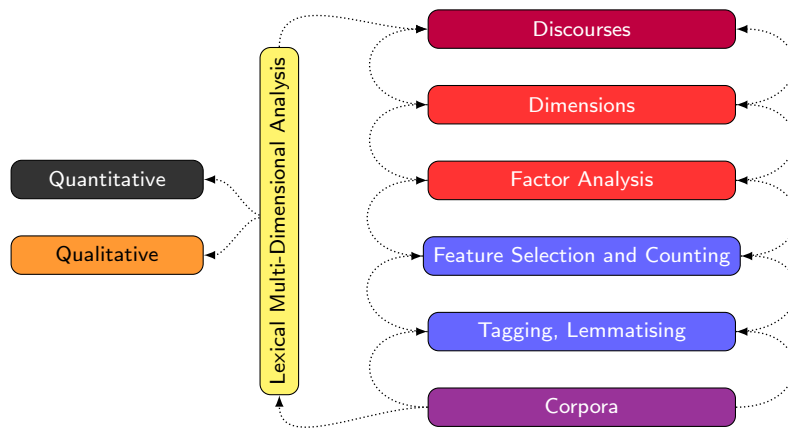


Figure 1: Lexical Multi-Dimensional Analysis (Berber Sardinha & Fitzsimmons-Doolan, 2025)

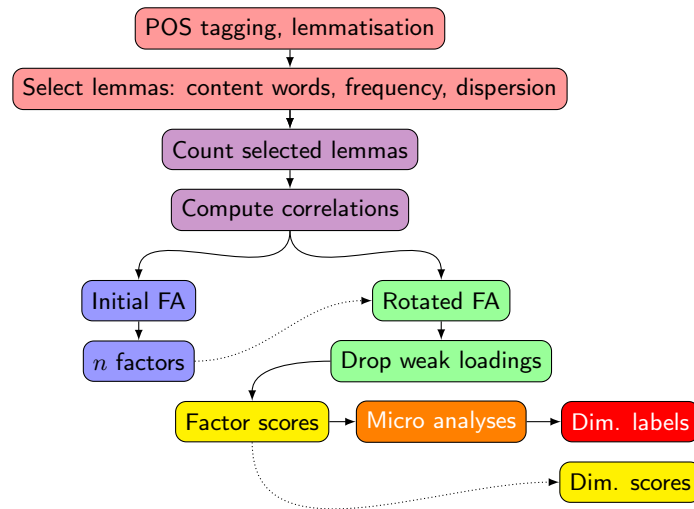


Figure 2: Lexical Multi-Dimensional Analysis Procedures

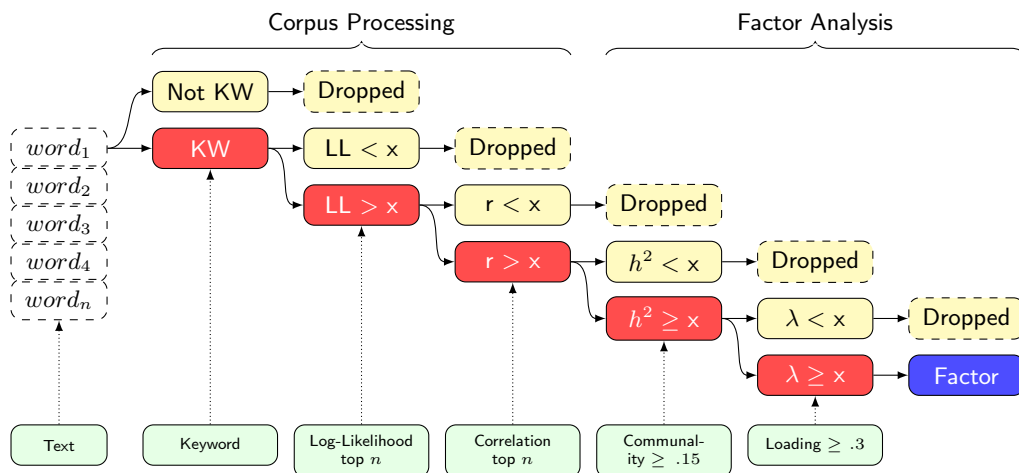


Figure 3: From text to factor

- Language-aware preprocessing (tokenisation, lemmatisation, POS tagging);
- Feature extraction (lexical richness, frequency profiles, function/content word ratios, POS n-grams, domain lexicons);
- Dimensionality reduction via Factor Analysis to derive interpretable dimensions;
- Interactive exploration (scatter plots, loadings, factor scores), filtering, and labelling;
- Exportable reports (PDF/HTML) and machine-readable outputs (CSV/TSV/JSON/JSONL).

### 1.3 Stakeholders

- Linguists and corpus analysts;
- Data scientists working on stylistic and register analysis;
- Educators and applied researchers;
- Software maintainers and operations personnel (if deployed as a desktop or server app).

## 2 Functional Requirements

### 2.1 Corpus Ingestion

- FR-1: Import plain text files, folders, and CSV/TSV and JSONL with a designated text column;
- FR-2: Detect/allow selection of text encoding (UTF-8 default), handle BOM (Byte Order Mark), and report decoding errors;
- FR-3: Optional metadata import (e.g., author, source, label) to support grouping and filtering.

### 2.2 Preprocessing

- FR-4: Tokenise, normalise (case, punctuation policies), and sentence-segment texts;
- FR-5: Lemmatise and POS-tag (TreeTagger; spaCy models; fallback to NLTK if unavailable);
- FR-6: Language selection and multi-language support; warn if models are missing;
- FR-7: Stopword management (built-in lists and user-defined additions).

### 2.3 Feature Extraction

- FR-8: Compute lexical richness metrics (TTR, MTLD, Maas, HDD);
- FR-9: Frequency features: word and lemma frequency profiles; function word distributions;
- FR-10: POS and POS n-grams; content/function word ratios; lexical sophistication bands;
- FR-11: Custom feature plug-ins via a documented Python interface.

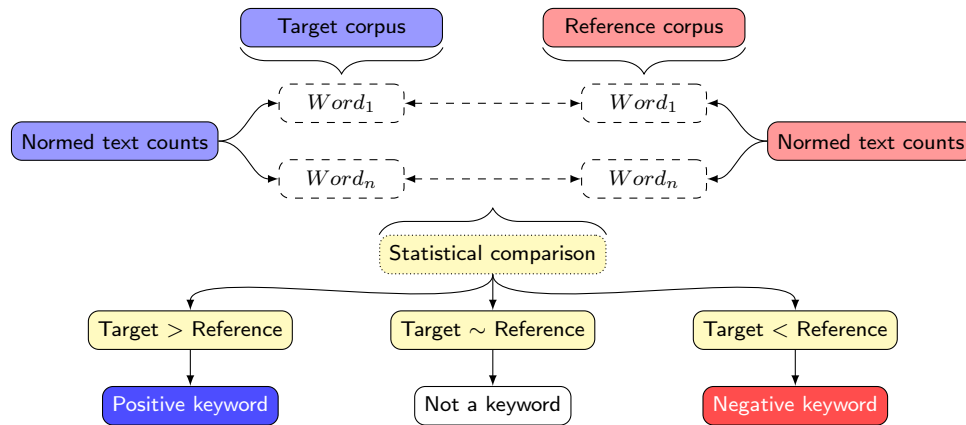


Figure 4: Lexical variables selection via keywords (Scott, 1997)

## 2.4 Dimensional Analysis

- FR-12: Build a document-feature matrix (DFM) with configurable weighting (raw, tf-idf, z-scores);
- FR-13: Perform dimensionality reduction (PCA) and optional factor analysis (EFA/Varimax);
- FR-14: Compute factor loadings, factor scores, and explained variance;
- FR-15: Save and load trained models/pipelines for reproducibility.

## 2.5 Visualisation and GUI

- FR-16: Interactive scatter/biplots of documents in 2D/3D factor space with tooltips and labels;
- FR-17: Loading plots, scree plots, and feature contribution charts;
- FR-18: Brushing/filtering by metadata, search, and selection for detailed inspection;
- FR-19: Theming, layout persistence, and session autosave.

## 2.6 Export and Reporting

- FR-20: Export factor scores, loadings, and projections to CSV/JSON;
- FR-21: Generate PDF/HTML reports with key plots and tables;
- FR-22: Export model artefacts (preprocessing config, feature schema, PCA/FA components).

## 2.7 Automation and Scripting

- FR-23: Batch processing via CLI and/or headless mode;
- FR-24: Optional REST API for programmatic access (server mode).

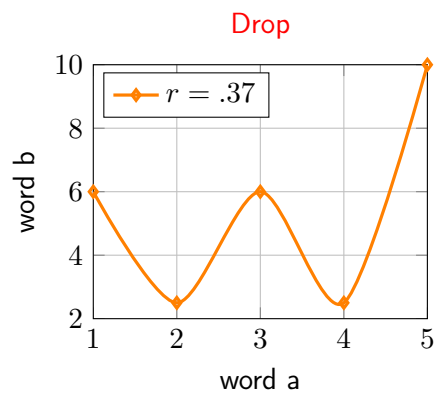
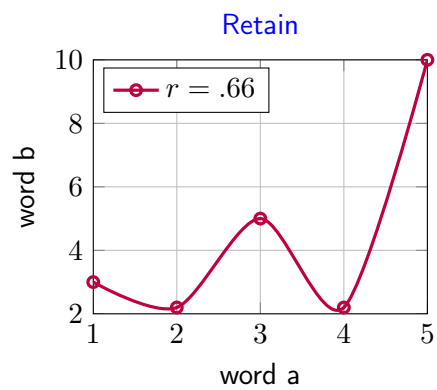
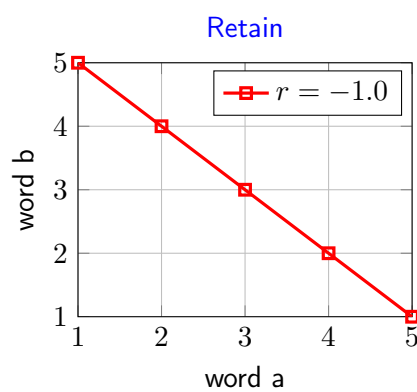
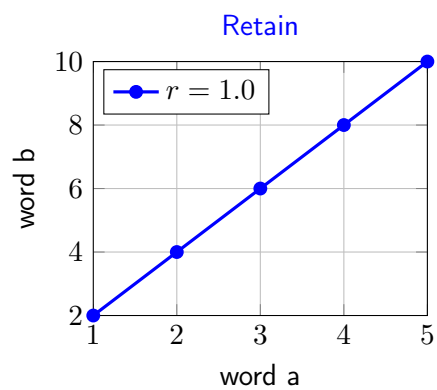


Figure 5: Lexical variables selection by correlation

## 3 Non-Functional Requirements

### 3.1 Performance

- NFR-1: Handle corpora up to 1M tokens on a standard laptop with reasonable latency (<10s for PCA on 10k documents x 1k features, assuming sparse ops);
- NFR-2: Incremental processing and caching to avoid recomputation.

### 3.2 Usability

- NFR-3: Intuitive GUI with sensible defaults; accessible color schemes;
- NFR-4: Contextual help and onboarding tips; undo/redo for key actions.

### 3.3 Reliability and Robustness

- NFR-5: Deterministic pipelines when seeds are fixed; versioned model artifacts;
- NFR-6: Error handling with clear diagnostics and recovery suggestions.

### 3.4 Portability and Compatibility

- NFR-7: Cross-platform support (Windows, macOS, Linux);
- NFR-8: Python 3.10+; packaged with virtualenv/Poetry; optional standalone (PyInstaller).

### 3.5 Security

- NFR-9: Local-only data processing by default; no data leaves the machine;
- NFR-10: If server mode is enabled, enforce TLS and authentication.

## 4 System Architecture

### 4.1 Overview

The system follows a modular, pipeline-oriented architecture separating ingestion, preprocessing, feature extraction, modelling, and presentation layers. A central configuration orchestrates reproducible runs; artefacts are persisted for reuse.

### 4.2 Components

**GUI Layer** Desktop GUI (PySide6/PyQt) for user workflows: project management, corpus loading, configuration, visualisation, and export.

**Controller / Orchestration** Coordinates pipeline stages, manages state, triggers incremental re-computation when inputs change, and logs provenance.

**Preprocessing Service** Language detection, sentence segmentation, tokenisation, lemmatisation, POS tagging, stopwords filtering; caches normalised representations.

**Feature Service** Computes metrics and vectorises documents into a DFM with configurable normalisation and weighting; supports plug-ins.

**Modeling Service** Performs PCA/EFA, computes loadings and scores, applies rotations (e.g., Varimax/Promax), and persists models.

**Visualisation Service** Generates interactive plots (2D/3D scatter, biplots, scree, loadings) and data tables; supports selections and linked views.

**Persistence Layer** Stores corpora metadata, preprocessed artefacts, features, and models (e.g., SQLite for metadata, parquet/npz for matrices, joblib for models).

**API Layer (Optional)** FastAPI-based REST endpoints for batch/remote processing in server mode.

### 4.3 Data Flow

1. Import corpus → normalise text → linguistic annotation.
2. Extract features → construct DFM → apply weighting/normalisation.
3. Fit PCA/EFA → compute scores/loadings → store artifacts.
4. Visualise and interact → export results and reports.

## 5 Technology Stack

- Language: Python 3.11 (or 3.10+);
- GUI: PySide6 or PyQt6; alternative: Qt for Python widgets + matplotlib/plotly;
- NLP: TreeTagger (via `treetaggerwrapper`), spaCy (core models: `en_core_web_sm/md/lg`), NLTK (fallback, resources), langdetect/fastText for language ID;
- Data: pandas, numpy, scipy (sparse matrices), scikit-learn for PCA, factor analysers (`factor_analyzer`) for EFA/rotations;
- Visualisation: matplotlib, seaborn, plotly; pyqtgraph for fast interactive plots;
- Persistence: SQLite (metadata), parquet/CSV (tables), npz (sparse matrices), joblib/pickle (models), YAML/TOML (config);
- Packaging: Poetry or pip-tools; PyInstaller for desktop bundle; Docker for server mode;
- Testing: pytest, hypothesis for property-based testing, tox/nox for matrix runs;
- CI/CD: GitHub Actions or GitLab CI for tests, linting (ruff, black), and packaging.



## 6 Interfaces and APIs

### 6.1 Python Module Interfaces

- **Preprocessor**: `process(corpus, config) → annotations`;
- **FeatureExtractor**: `fit_transform(annotations) → DFM`; `transform(...)`;
- **Modeler**: `fit(DFM) → model`; `transform(DFM) → scores`;
- **Visualiser**: `plots(scores, loadings, metadata)`;
- **Persistence**: `save_artifact(obj, kind)` and `load_artifact(kind)`.

### 6.2 Plug-in Interface

Third-party feature extractors implement `IFeaturePlugin`:

- `schema()` returns feature names and dtypes;
- `compute(doc)` yields a sparse/compact feature vector;
- Registration via entry points or a plugins folder.

### 6.3 CLI

```
lmda run --config config.yaml --input corpus/ --output artifacts/
lmda visualise --project my_project.lmda
```

### 6.4 Optional REST API (FastAPI)

- POST `/v1/projects`: create project
- POST `/v1/projects/id/ingest`: upload texts/metadata
- POST `/v1/projects/id/run`: execute pipeline
- GET `/v1/projects/id/scores`: factor scores
- GET `/v1/projects/id/loadings`: factor loadings
- GET `/v1/projects/id/report`: PDF/HTML report

## 7 Security and Privacy Considerations

### 7.1 Data Protection

- Local-first processing; no external calls by default;
- Optional at-rest encryption for project files; secure temp directories;
- Configurable redaction of personally identifiable information (PII) in exports.

## 7.2 Access Control (Server Mode)

- Authentication (OAuth2/JWT) and role-based access control (RBAC);
- TLS enforcement; secure headers and CSRF protections for Web GUI (if used).

## 7.3 Supply Chain

- Pin dependencies; verify wheels; use vulnerability scanning (pip-audit, Safety);
- Sandboxed plug-ins with permission checks.

# 8 Risks and Mitigations

- **R1: Model interpretability** — Factors may be hard to interpret.  
*Mitigation:* Offer rotations, feature contribution charts, and glossary tooltips.
- **R2: Performance on large corpora** — High memory/CPU usage.  
*Mitigation:* Sparse structures, incremental processing, caching, chunked I/O.
- **R3: NLP model availability** — TreeTagger and spaCy models not installed.  
*Mitigation:* Guided download, fallback tokenisers, graceful degradation.
- **R4: Data privacy** — Sensitive texts accidentally exported.  
*Mitigation:* Local-first defaults, explicit consent prompts, export redaction.
- **R5: Cross-platform GUI quirks** — Rendering differences.  
*Mitigation:* UI testing matrix, use of Qt styles, theming validation.
- **R6: Reproducibility drift** — Version changes affect outputs.  
*Mitigation:* Lockfiles, artefact versioning, seeds, environment capture.

# 9 Timeline and Milestones

1. **Week 1–2: Foundations**  
Project scaffolding, env setup, CI/CD, basic data model, configuration schema;
2. **Week 3–4: Preprocessing**  
Language detection, tokenisation, lemmatisation, POS tagging; caching;
3. **Week 5–6: Features**  
Core lexical metrics, function word profiles, POS n-grams; plug-in API;
4. **Week 7–8: Modeling**  
PCA/EFA pipeline, rotations, persistence; basic CLI;
5. **Week 9–10: GUI v1**  
Data loading, configuration panels, 2D scatter, scree and loadings plots;
6. **Week 11–12: Reporting & Export**  
CSV/JSON exports, PDF/HTML reports; theming and preferences;

**7. Week 13–14: Hardening**

Performance tuning, testing, error handling, and documentation;

**8. Week 15+: Optional Server Mode**

FastAPI endpoints, auth/TLS, Docker packaging.

## References

- Berber Sardinha, T., & Fitzsimmons-Doolan, S. (2025). *Lexical Multi-Dimensional Analysis: Identifying Discourses and Ideologies*. Cambridge: Cambridge University Press.
- Scott, M. (1997, June). PC analysis of key words — And key key words. *System*, 25(2), 233–245.  
doi: 10.1016/S0346-251X(97)00011-0

## **A Appendices**

### **A.1 work in progress**