Johannes Gutenberg University Mainz
FB08
Institute of Computer Science
Data Management Group

Bachelor's Thesis

Graph database for Circuit neuroscience - frontend, editor tool

Anh Viet Ngo

1. Reviewer Jun.-Prof. Dr. Panagiotis Bouros
Institute of Computer Science - Data Management Group
Johannes Gutenberg University Mainz

2. Reviewer Dr. Shimpei Ishiyama
Institute of Pathophysiology - University Medical Center
Johannes Gutenberg University Mainz

Supervisors Jun.-Prof. Dr. Panagiotis Bouros
Dr. Shimpei Ishiyama

August 27, 2025

Anh Viet Ngo
Graph database for Circuit neuroscience - frontend, editor tool
Bachelor's Thesis, August 27, 2025
Reviewers: Jun.-Prof. Dr. Panagiotis Bouros and Dr. Shimpei Ishiyama
Supervisors: Jun.-Prof. Dr. Panagiotis Bouros and Dr. Shimpei Ishiyama

Johannes Gutenberg University Mainz
Data Management Group
Institute of Computer Science
FB08
Staudingerweg 9
55128 Mainz

Abstract

This thesis presents an approach to integrate Natural Language Processing (NLP)-based information extraction techniques into a web-based editor tool, addressing the challenge of a continuous workflow by enabling neuroscientists to efficiently convert unstructured data from neuroscience publications into visual representations of experimental data. This pipeline is implemented within the extended web application that now offers multiple extraction methods (traditional NLP, ChatGPT-based, and hybrid approaches with secure API key management), an integrated user-specific cloud workspace, a user-friendly and intuitive entity review interface, and additional features such as interactive PDF highlighting, error feedback, and enhanced editor functionalities.

Abstract (german)

Diese Arbeit präsentiert einen Ansatz zur Integration von Natural Language Processing (NLP)-basierten Informationsextraktionstechniken in ein webbasiertes Editor-Tool. Sie adressiert die Herausforderung eines durchgängigen Workflows, indem sie Neurowissenschaftler*innen ermöglicht, unstrukturierte Daten aus neurowissenschaftlichen Publikationen effizient in visualisierte Darstellungen experimenteller Daten umzuwandeln. Diese Pipeline ist in die erweiterte Webanwendung integriert, die nun mehrere Extraktionsmethoden (traditionelle NLP-, ChatGPT-basierte und hybride Ansätze mit sicherem API-Schlüssel-Management), eine integrierte nutzerspezifische Cloud-Arbeitsumgebung, eine benutzerfreundliche und intuitive Oberfläche zur Überprüfung der extrahierten Entitäten sowie zusätzliche Funktionen wie interaktive PDF-Hervorhebung, Fehlerrückmeldungen und erweiterte Editor-Funktionalitäten bietet.

Acknowledgement

I would like to sincerely thank my supervisors, Jun.-Prof. Dr. Panagiotis Bouros and Dr. Ishiyama Shimpei, for their continuous support and guidance during my thesis. Their advice, encouragement, and constructive feedback were incredibly helpful and made a big difference in my work.

I am also very grateful to Dr. Shimpei Ishiyama for kindly allowing me to use the "Gnoilinx" name and logo that he created for this web interface.

Contents

1 Introduction

This chapter introduces the motivation and problem statement for developing an end-to-end pipeline to extract and visualize structured knowledge from neuroscience literature, outlines the main contributions of this thesis, and provides an overview of the thesis structure.

1.1 Motivation and Problem Statement

Neuroscience research produces complex, multi-scale data that is often hidden inside publications [1]. These publications are typically unstructured documents [12], making systematic analysis and integration a significant challenge that limits the pace of discovery in the field.

Despite advances in automated extraction, existing workflows remain fragmented and lack seamless connections between information extraction, organization, and interactive visualization [20]. Most current platforms focus either on automated extraction or on editor and visualization tools. As a result, valuable insights are often lost or overlooked. This shows a clear need for integrated solutions that can transform unstructured knowledge from neuroscience literature into graph representations for knowledge discovery.

This thesis addresses that need by presenting an end-to-end pipeline that extracts knowledge from neuroscience literature and enables interactive graph-based visualization. By integrating and leveraging existing information extraction methods [7] with a modern web-based editor tool [8], this approach reduces manual effort and accelerates neuroscientific discovery.

1.2 Contributions

The primary contributions of this thesis are:

- Design and implementation of a semi-automated, end-to-end pipeline that integrates advanced information extraction techniques with an interactive, web-based graph editor tool for extracting and visualizing structured neuroscience knowledge from publications.

- Modular integration of multiple information extraction methods (traditional NLP, ChatGPT, and hybrid) with flexible method selection and API key management.
- Extension and optimization of LLM prompts for neuroscience-specific entity and relationship extraction.
- Implementation of an interactive entity review and curation interface, including editing and color-coded PDF highlighting.
- Implementation of comprehensive error handling and user feedback mechanisms for a user-friendly workflow.
- Standardization and conversion of extracted data into interoperable formats, including automated node positioning and layout for clear visualization.
- Extension of the existing web application with secure, user-specific authentication and cloud-based project management.
- Enhancement of the editor tool with additional features and improved usability (e.g., node reassignment, undo/redo).

1.3 Thesis Structure

The thesis is divided into the following chapters:

Chapter 2: Background and Related Work
The chapter reviews neuroscience data workflows, challenges, and existing extraction and visualization tools.

Chapter 3: System Architecture and Integration
The chapter describes the extended system architecture, data model, technology stack, detailed pipeline workflow, and expected functionalities of the proposed solution.

Chapter 4: Pipeline Implementation
The chapter details the pipeline implementation in the backend with processing setup, modular extraction methods, output structuring and validation, and project file conversion.

Chapter 5: Demonstration
The chapter demonstrates the end-to-end pipeline in action by walking through each step of the user workflow and highlighting new features for improved usability.

Chapter 6: Conclusion
The final chapter summarizes the work discussed in this thesis and proposes possibilities for future work on the system.

2 Background and Related Work

This chapter introduces key concepts in circuit neuroscience and highlights challenges such as fragmented workflows, particularly the lack of integration between extraction and visualization. Existing resources are reviewed to demonstrate current limitations and motivate the need for a unified, end-to-end pipeline in the web application developed in this thesis.

2.1 The Neuroscience Data Pipeline

Neuroscience research generates complex and multi-scale data that include anatomical brain structures, diverse neuron types with specialized properties, stimuli, and related behavioral responses [1]. Neural circuits are specific ensembles of interconnected neurons that work together to process information [1, 13]. These circuits transform external and internal stimuli into structured patterns of neural activity [13], forming the basis for perception and behavior [1]. For example, external stimuli such as tickling in rats produce distinct neural activity patterns in the somatosensory cortex, which correlate with behavioral outputs including ultrasonic vocalizations, approach, and unsolicited jumps [10].

The primary method through which such knowledge is communicated is neuroscience publications, which describe the experimental setup, observed results, and conclusions drawn from the data [12]. Automated information extraction techniques from the field of Natural Language Processing (NLP) can convert neuroscience publications into structured, machine-readable data. By detecting relevant entities and relationships within the text, these methods make it possible to systematically capture complex experimental findings described in the publications [7].

Graph representations, such as knowledge graphs, provide an effective framework for modeling and visualizing these complex relationships. In these models, nodes represent entities including brain structures, neurons, stimuli, or behaviors, while edges denote relationships such as activation, inhibition, or anatomical projection [8]. This approach enables a more intuitive understanding of the interconnections of neural circuits and their components, and helps researchers recognize patterns and connections that might not be obvious just from looking at raw data or text descriptions.

An effective informatics pipeline in neuroscience can be conceptualized as a four-stage process:

1. Automated entities extraction from raw data in the form of PDF using NLP methods.
2. Human-in-the-loop review of extracted entities for correction or confirmation.
3. Automated relationships extraction from the confirmed extracted entities.
4. Interactive graph-based visualization for analysis and discovery.

However, as will be discussed in the following sections, current neuroscience workflows face significant limitations in achieving such seamless integration, particularly in connecting automated extraction methods with interactive visualization platforms.

2.2 Limitations in Current Neuroscience Workflows

Much of the neuroscientific knowledge is embedded in unstructured or semi-structured formats such as text, figures, and tables within publications [12], making it difficult to mine, analyze, or integrate. Relevant information is often sparse and extracting useful data typically requires considerable domain expertise for annotation [9], which is a challenge across scientific disciplines and particularly in neuroscience.

As a result, researchers must frequently rely on manual extraction, which remains time-consuming, labor-intensive, and error-prone. Indeed, extracting information from publications on a large scale is often viewed as a tedious process that few researchers are willing to dedicate their limited time and effort to, despite its importance for modern data-driven design practices [9].

Automation offers a promising solution by enabling the large-scale conversion of unstructured content into structured, analyzable data. Despite notable advances in automated information extraction, these techniques have yet to be integrated into scientific literature workflows due to technical and logistical challenges [9]. These challenges are particularly evident in neuroscience, where the complexity and diversity of data further complicate the seamless integration [1].

Most approaches still operate independently and rarely form comprehensive, systematic, and interoperable pipelines [20]. In practice, current neuroscience pipelines typically extend only as far as generating NLP-extracted outputs (e.g., [7, 12, 21]), while integration with modern web-based editor tools for

graph visualization is often minimal or entirely absent. Thus, valuable knowledge remains hidden and the research process is slowed by these persistent barriers.

This highlights the need for an integrated, user-friendly approach that streamlines data extraction, visualization, and analysis, allowing researchers to focus on discovery rather than being limited by the complexities of data processing.

2.3 Existing Resources

Several existing tools address parts of the neuroscience data pipeline. However, none currently deliver a seamless, end-to-end semi-automated or automated pipeline from literature to interactive graph-based visualization. For instance, Goraya [7] investigated and compared three NLP-based approaches for information extraction from circuit neuroscience publications, including a traditional NLP Pipeline, a Large Language Model (LLM) prompt-based, and a hybrid approach that combines both strategies. However, the work lacks integration with visualization platforms.

In parallel, Götz [8] developed a robust web-based editor tool designed specifically for circuit neuroscience data. This editor tool allows users to manually enter, manage, and visualize experimental data as interactive graphs, and offers various features such as project management, hierarchical views for anatomical brain structures, error checking, and integration of paper metadata from external sources such as PubMed [@18]. Similarly, platforms such as ResearchMaps.org [@19] provide valuable tools for graph-based experimental planning, while Allen Brain Atlas [@2] offers detailed anatomical data visualization. However, a major limitation is that these platforms do not automate the extraction of information directly from publications in PDF format.

In order to address these limitations, this thesis aims to bridge the gap by integrating advanced NLP extraction methods directly into an existing neuroscience web-based editor tool to semi-automate the entire process from raw PDF to interactive graph. Specifically, this work:

1. Integrates three distinct extraction methods (traditional NLP, LLM-based ChatGPT, and hybrid approaches) within a unified processing framework.
2. Develops an automated conversion system that transforms extracted neuroscience entities and relationships into standardized project file formats compatible with the existing web-based editor tool.
3. Creates a seamless workflow from PDF upload through entity review by users to interactive graph visualization.

This integration enables researchers to move directly from publications to actionable visual insights without manual data transfer or format conversion.

3 System Architecture and Integration

This chapter provides a comprehensive overview of the extended system architecture developed in this thesis, including its data model, graph database, and the workflow of the information extraction pipeline. It also describes the technology stack and expected functionalities that support efficient neuroscience data management and interactive graph visualization.

3.1 System Architecture

The system architecture is built directly upon the web-based client-server architecture from Götz [8], which is extended with an end-to-end information extraction pipeline based on recent NLP advances from Goraya [7]. This section introduces the main architectural components and their roles in supporting the structured neuroscience data pipeline and personalized workflows.

3.1.1 Data Model

The data model (Figure 3.1) used in the system represents circuit neuroscience experiments in a structured, consistent, and interoperable way. The model defines core entities such as Brain Structures[1], Neuron, Stimulus, Behavior, and Experiment, each uniquely identified and associated with relevant attributes (e.g., name, description, metadata, method). These entities are designed to capture the essential components and context described in neuroscience publications.

Relationships between entities are also explicitly represented as directed edges in the graph database. More importantly, these relationships also play a central role in the information extraction pipeline, where every output relationship from the NLP extraction process is validated to ensure accuracy. This provides a consistent output result for the pipeline, which is compatible with the existing schema.

The key experimental relationship types modeled and extracted by the pipeline include:

[1] Referred to as *Structure* in the data model and schema.

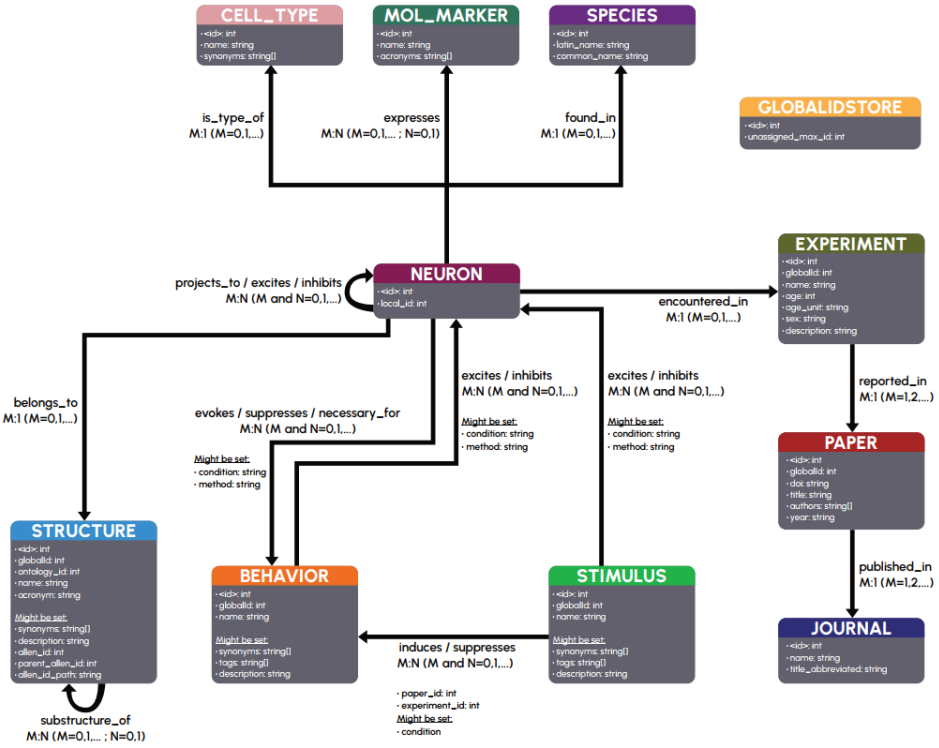


Fig. 3.1.: Data model for neuroscience experiments. *Adopted from* [8].

1. Structure *is_substructure_of* Structure.
2. Neuron *projects_to, excites, inhibits* Neuron.
3. Neuron *belongs_to* Structure.
4. Neuron *evokes, suppresses, necessary_for* Behavior.
5. Behavior *excites, inhibits* Neuron.
6. Stiumlus *induces, suppresses* Behavior.
7. Stimulus *evokes, inhibits* Neuron.

Although other relationships, such as administrative relationships (e.g., *encountered_in, reported_in, published_in*) and neuron properties (e.g., *is_type_of, suppresses, found_in*), are not the primary focus of the information extraction pipeline, they are essential for ensuring comprehensive data provenance and effective metadata management within the database.

In addition to the neuroscience experiment data model, this thesis introduces a user-specific project data model (Figure 3.2). This cloud-based approach enables the migration of all project-related data from local storage to the cloud database in Neo4j, ensuring that the projects are user-specific, securely managed, and accessible from any device with internet access. By linking

project data directly to individual user accounts, the system supports personalized workflows, improves data security, and enables future features such as project sharing and real-time collaboration.

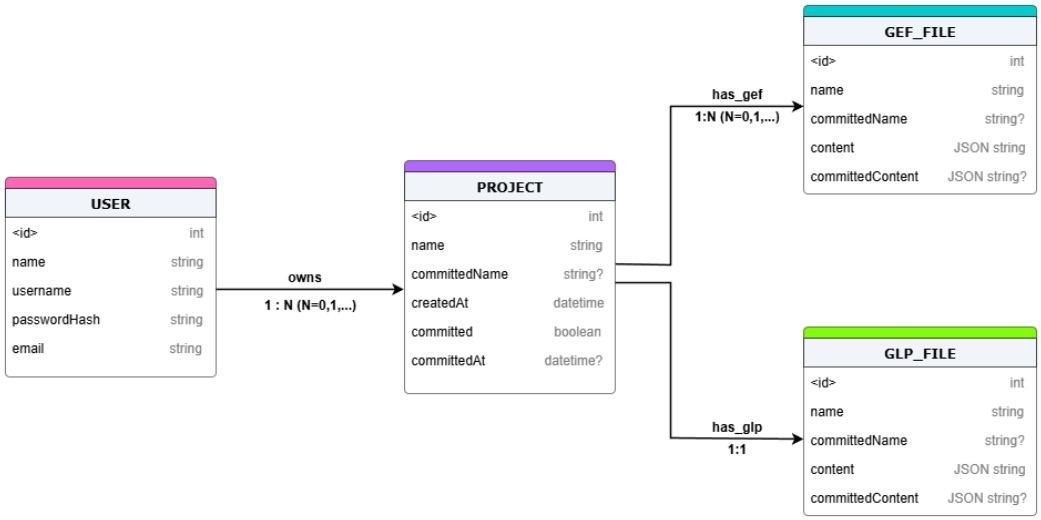


Fig. 3.2.: Data model for user management.

Each user is uniquely identified and has attributes such as name, username, hashed password, and email, which enable authentication and personalized access. Users can own multiple projects, each defined by a name, creation date, and committed status.

Projects can include multiple GEF[2] files and a single GLP[3] file, each with its name and the content stored as a JSON-formatted string. Projects and files also support optional metadata for committed versions, including the committed name, content, and date.

The committed information for projects, GEF files, and GLP files is used to track the latest committed version. If a committed project or file is subsequently modified, users can easily revert to the most recent committed state. This provides reliable version control, allowing users to restore their work to a previously finalized version whenever necessary.

3.1.2 Graph Database

The system utilizes Neo4j using Cypher Query Language to efficiently store and manage the structured data extracted from neuroscience publications. In addition, the graph database manages user accounts, project files, and version control by representing users, projects, and associated files as nodes linked via explicit relationships (see Figure 3.2).

[2]Gnoilinx Experiment File.
[3]Gnoilinx Linking Paper.

When a new user registers, their information (username, name, email, and a securely hashed password) is stored in the graph database as a USER node. This operation is the starting point for associating projects and files with individual users (Listing 3.1).

```
CREATE (u:USER {username: "bob2002",
                name: "Bob",
                email: "bob@example.com",
                passwordHash: "hashed_password"
        })
```

Lst. 3.1: Cypher example for creating a new user.

Users can access their project data stored in the database by retrieving a specific project along with its associated GLP and GEF files. The query matches the USER and PROJECT nodes, then gathers the contents of the GLP file and a collection of GEF files, allowing the application to display the current state of the project and its related files to the user (Listing 3.2).

```
MATCH (u:USER {username: "bob"})
      -[:owns]->(p:PROJECT {name: "project1"})
OPTIONAL MATCH (p)-[:has_glp]->(glp:GLP_FILE)
OPTIONAL MATCH (p)-[:has_gef]->(gef:GEF_FILE)
RETURN glp.content AS glpContent,
    collect({name: gef.name, content: gef.content}) AS gefFiles
```

Lst. 3.2: Cypher example for retrieving a project folder.

After users modify their project files, the database is updated to reflect these changes. The update operation ensures that both the GLP file and GEF files are correctly linked to the project with their contents set to the latest change by users (Listing 3.3).

```
MATCH (u:USER {username: "bob"})
      -[:owns]->(p:PROJECT {name: "project1"})
MERGE (p)-[:has_glp]->(glp:GLP_FILE)
SET glp.content = '{ "paperData": { ... },...}'
MERGE (p)-[:has_gef]->(gef:GEF_FILE {name: "gef1"})
SET gef.content = '{ "experimentName": "gef1",...}'
```

Lst. 3.3: Cypher example for updating GLP and specific GEF Files.

If a user decides to permanently remove a project, the system deletes the PROJECT node along with all associated GLP and GEF nodes. This process eliminates all related data and relationships from the database, ensuring that unwanted projects are fully erased (Listing 3.4).

```
MATCH (u:USER {username: 'bob'})
      -[:owns]->(p:PROJECT {name: 'project1'})
```

```
OPTIONAL MATCH (p)-[:has_glp]->(glp:GLP_FILE)
OPTIONAL MATCH (p)-[:has_gef]->(gef:GEF_FILE)
DETACH DELETE p, glp, gef
```

Lst. 3.4: Cypher example for deleting project and related files.

For the NLP pipeline, Cypher queries were developed to support robust entity linking[4] with progressive matching strategies [7], in which all entity identifiers are adjusted to use custom globalID instead of Neo4j's internal id property to fit the requirements of the integrated NLP pipeline and the specific schema of the current system.

In addition, this thesis introduces new Cypher queries to manage global ID assignments and maintain database consistency, specifically to support the correct handling of extracted output when creating project files for visualization.

For example, whenever a new set of nodes is created, the global ID counter is incremented to maintain unique and sequential assignment for future entries, regardless of their committed status. This approach ensures that all new entities receive unique identifiers without conflicts (Listing 3.5).

```
MATCH (idStore:GLOBALIDSTORE)
SET idStore.unassigned_max_id = $initial_id + $increment
```

Lst. 3.5: Updating the global ID counter.

To maintain the integrity of hierarchical relationships, the system checks if a brain structure node already has a parent. Since each brain structure is allowed only one parent in the database (see Figure 3.1), this step helps prevent the creation of duplicate or conflicting relationships (Listing 3.6).

```
MATCH (child:STRUCTURE)-[:substructure_of]->(parent:STRUCTURE)
WHERE child.globalId = $child_id
RETURN parent
LIMIT 1
```

Lst. 3.6: Checking for an existing parent-child brain structure relationship.

3.1.3 Technology Stack

The technology stack builds on the previously established frameworks for NLP processing and knowledge graph visualization. The web application is built using React.js, TypeScript, and Vite for the frontend with React Konva for interactive graph editing and visualization, Node.js and Express.js for server-side functionality, and Neo4j for graph database management. The

[4] The process of mapping extracted terms to existing database entities.

NLP processing uses tools such as PyPDF2 used for PDF text extraction and PyTorch, spaCy, SciBERT, OpenAI API, fuzzywuzzy, and custom fine-tuned models to extract neuroscience knowledge from publications.

While the core technologies from both works remain central to the system, this thesis introduces new tools and technologies to improve the integration of NLP processing and file management within the pipeline. These improvements are designed to better address the specific needs of this work and further strengthen the overall functionality of the system.

PDF Processing

- **Multer**: Handles file uploads and manages temporary storage of PDF files. It simplifies the process of receiving and storing user-uploaded files, ensuring they are accessible for subsequent processing.
- **PDF.js**: Renders and displays PDF documents in the web interface, allowing users to view the uploaded publications directly in the browser. Additionally, it is used to highlight the extracted entities within the PDF, providing a clear visual indication of the information identified by the NLP pipeline.

NLP Environments

Two different Python environments are used to run different NLP tasks.

- **Python 3.7.12**: This environment is required because the traditional NLP pipeline depends on neuralcoref and older `spaCy` versions that are incompatible with the newer versions of Python.
- **Python 3.8+**: Python 3.11.7 is used in this work and serves as the primary environment for the NLP pipeline. It supports modern libraries and NLP models such as OpenAI [@17].

The system uses two separate Python environments to combine traditional NLP techniques and modern LLM-based approaches. This setup provides flexibility and allows the system to effectively extract information from neuroscience publications. The use of different environments ensures compatibility, as some models or libraries are only supported in specific Python versions.

User Authentication and Security

- **bcrypt**: Used for secure password hashing.

- **JSON Web Token (JWT)**: Used for both stateless user authentication/session management and secure handling of OpenAI API keys. All API requests in terms of user authentication require valid JWT for secure access to user-specific data and project files. For OpenAI integration, API keys are securely stored on the server and access is authorized using temporary server-validated JWTs.

Process Management

- **subprocess**: Triggers the entire NLP pipeline and captures `stdout`, `stderr` for script output and error handling, allowing the full process to run smoothly without the need to execute each individual script separately. It also executes traditional NLP tasks in an isolated environment, ensuring compatibility with old Python libraries.
- **spawn**: Used in the `Node.js` server to run Python scripts as child processes. It allows for asynchronous execution and provides real-time communication between the server and NLP processing in the backend.

3.2 Pipeline Workflow

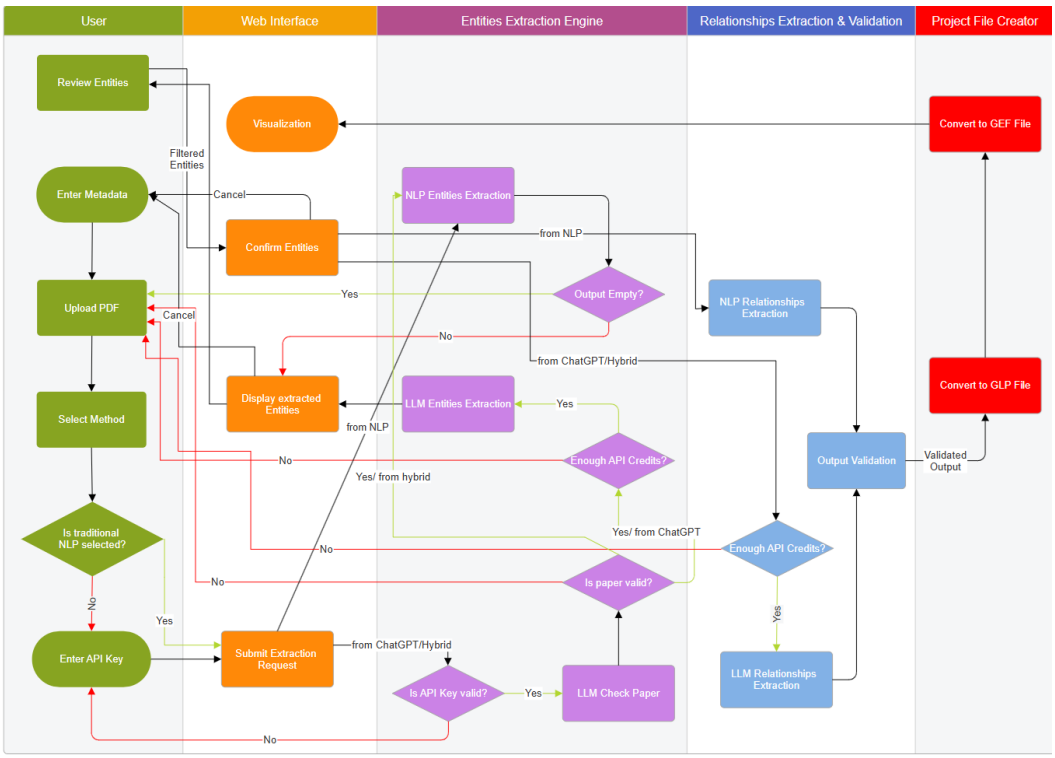


Fig. 3.3.: Pipeline workflow for end-to-end information extraction and visualization.

The pipeline workflow, shown as a multi-lane swimlane diagram in Figure 3.3, visually separates the responsibilities of the user, the web interface, the entities extraction engine, the relationships extraction and validation stage, and the project file creation module. Each lane groups related activities, clarifying the flow of information and decision points throughout the semi-automated extraction and visualization process.

The pipeline transforms raw neuroscience publications into interactive, structured knowledge graphs through a sequence of steps. After users manually enter essential metadata such as DOI, paper title, journal, authors, year, and project name, they can upload a PDF file via the web interface. If the user selects ChatGPT-based or hybrid extraction, they must also provide a valid OpenAI API key to enable LLM processing.

The system then extracts relevant text sections and processes them using the chosen extraction method: traditional NLP, ChatGPT, or a hybrid approach. Once initial entity extraction is complete, the identified entities (such as brain structures, behaviors, and stimuli), which are linked to the existing database entries through entity linking, are sent back to the user for review and selection, allowing manual filtering or correction before proceeding.

After the user has selected the relevant entities, corresponding relationships are extracted, validated to ensure correctness, and then compiled into a structured text file (Figure 4.6). Once finished, the data is converted into standardized formats (GLP and GEF) for visualization. The entire workflow is orchestrated automatically, requiring minimal user intervention beyond the initial metadata entry, extraction method selection, API key input (if applicable), and the review and confirmation of extracted entities.

Throughout this process, errors such as invalid API keys, API keys with insufficient credits, or extraction failures are detected and handled in a method-specific way with informative feedback provided to the user. For example, when using ChatGPT-based or hybrid extraction, the system checks for valid API keys and sufficient credits before proceeding. This adaptive error handling ensures that users will be notified of issues specific to their chosen workflow.

This streamlined workflow ensures that neuroscience knowledge can be rapidly extracted, validated, and visualized, serving as the backbone of the system's pipeline described in this thesis. Furthermore, the modular design allows for easy integration of new extraction methods in the future, ensuring that the platform can adapt to evolving research needs and technological advancements in neuroscience.

3.3 Expected Functionalities

The system developed in this thesis is designed to provide a semi-automated, end-to-end, and cloud-based pipeline for extracting, processing, and visualizing neuroscience information from neuroscientific literature. The following functionalities are expected:

PDF Processing & Data Handling

- PDF file upload with a user-friendly file selection interface that accepts PDF documents and validates them server-side for neuroscience content processing.
- Multiple extraction method selection, including traditional NLP, ChatGPT-based, and hybrid approaches, with clear descriptions of each method to help users choose the most appropriate approach for their purpose.
- Secure API key management that ensures user-provided OpenAI API keys are protected and never exposed to unauthorized parties, maintaining privacy and security throughout the extraction process.
- Temporary file handling for storing PDFs and intermediate processing data.

Entity Extraction & Review

- An interactive entity review system that allows users to manually edit and select extracted entities, featuring a hierarchical display for brain structures.
- Synchronized color-coded highlighting based on entity type in the original PDF for improved accuracy and user experience.

Advanced NLP Extraction Methods

- **Traditional NLP** processing uses Named Entity Recognition (NER) models specifically trained on neuroscience literature to identify brain structures, behaviors, and stimuli. The system incorporates coreference resolution to replace pronouns with proper antecedents and relation extraction algorithms to automatically identify connections between detected entities.

- **Large Language Model** integration utilizes ChatGPT models with specific prompts that guide extraction. It also features automatic model selection based on API availability and confidence scoring using logprobs to assess extraction reliability.
- **Hybrid** combines the precision of traditional NER with the contextual understanding of LLMs. It uses NER to consistently identify entities, while LLMs help capture more complex relationships between those entities.

Automatic Knowledge Graph Generation & Data Standardization

- Automated extraction of entities such as brain structures, behaviors, and stimuli.
- Direct integration with the graph database, automatically linking new entities to existing knowledge.
- Identification and validation of relationships between extracted entities and automatic creation of knowledge graphs.
- Generation of GLP project files from NLP extraction results for compatibility with the existing platform architecture.
- Automatic placement and rendering of extracted entities and relationships from GEF experiment files on the canvas with proper positioning, type-specific styling for knowledge graph visualization.

Error Handling & Feedback

- Comprehensive error handling at each stage of the workflow with error notifications guiding users through issues such as invalid API keys or failed extractions.
- Ability to retry or switch extraction methods if one fails, minimizing user effort.

User Management & Cloud Project Storage

- Secure user registration, login, and authentication, ensuring that only authorized users can access the system and their data.
- All user projects and experiment files are stored in the cloud (Neo4j database), enabling access from any device and eliminating reliance on local storage.

- Users can create, open, rename, and delete projects through a web interface, with all changes reflected in real time in the cloud database.
- Each user's data is isolated and protected, with strict access control enforced by authentication middleware.
- A project can be reverted to its last committed version, ensuring data integrity and easy recovery from unwanted changes.

In summary, the system architecture and integrated workflow presented in this chapter lay the foundation for transforming unstructured neuroscientific literature into structured knowledge graphs. By combining graph-based data modeling, NLP-driven entity and relationship extraction, and secure cloud-based project management, the system sets the stage for the implementation and demonstration of the pipeline in the following chapters.

4 Pipeline Implementation

This chapter details the backend implementation of the end-to-end neuroscience knowledge extraction pipeline. It covers PDF upload, secure API management, multi-method entity and relationship extraction, output structuring and validation, and conversion to GLP and GEF files. The implementation emphasizes modular design for easy extensibility and robust error handling in diverse processing scenarios. [1]

4.1 File Upload and Processing Setup

This section establishes the infrastructure for PDF document uploads and multi-language processing pipeline coordination between `Node.js` server operations and Python-based NLP processing. The implementation includes file management, secure API management, and subprocess execution chains.

4.1.1 File Upload and API Key Management

The file upload infrastructure was implemented using `Multer` [@16] middleware with `diskStorage` configuration to handle PDF uploads properly. The system applies timestamp-based filename generation to prevent naming conflicts. Directory creation is handled automatically, while file type selection in the upload dialog defaults to PDF files for user convenience. The system implemented automatic cleanup of temporary files, including uploaded PDFs and intermediate processing files, upon successful completion or error conditions to maintain server storage efficiency.

Each method selection logic implements conditional parameter handling that distinguishes between processing approaches. Traditional NLP processing continues without requiring an API key, while ChatGPT and Hybrid methods require valid authentication tokens. For this, a custom JWT-based `SecureTokenManager` class (Listing 4.1) was developed using crypto-generated secrets for token creation and validation [@5, 11].

[1] All figure outputs in this chapter are generated from [4].

```
const tokenManager = new SecureTokenManager();

const token = tokenManager.createToken(openaiApiKey);

// Token validation only - entity extraction phase
openaiApiKey = tokenManager.validateTokenOnly(processingToken);

// Token validation and consumption - final processing
openaiApiKey = tokenManager.validateAndUseToken(processingToken);
```

Lst. 4.1: Token management methods across different phases.

This approach enables secure token reuse in the multi-step pipeline: tokens are created during upload, validated during entity extraction, and finally consumed during the processing phase. The `validateTokenOnly` method allows same token to be safely reused across pages, while `validateAndUseToken` ensures one-time consumption to prevent replay attacks after the workflow is complete.

API keys are encoded using `Base64` within the JWT payload and configured with 60-minute expiration timeouts to limit exposure risk while providing sufficient time for processing. Even if the token expires before the final processing step, users are prompted to re-enter their API key when they continue at the end of the review process, ensuring no loss of progress.

4.1.2 Processing Setup

Server Pipeline Orchestration

The server triggers Python subprocess execution through dynamic argument construction based on the selected processing method (Listing 4.2). The Node.js `spawn` mechanism [@6] creates a non-blocking subprocess that executes the Python extraction pipeline while maintaining server responsiveness. The `stdout` serves as the primary communication channel between the Python scripts and the `Node.js` server. Its output is accumulated and parsed to extract JSON-formatted extracted entities data that is subsequently returned to the frontend interface for users to review.

```
const pythonArgs = [
    path.join(__dirname, '..', 'nlp', 'api', 'extract_entities.py'),
    pdfPath, '--extraction-method', processingMethod
];

if ((processingMethod === 'chatgpt' || processingMethod === 'hybrid')
    && openaiApiKey) {
    pythonArgs.push('--openai-api-key', openaiApiKey);
}

const pythonProcess = spawn('python', pythonArgs);

```

```
pythonProcess.stdout.on('data', (data: Buffer) => {
    pythonData += data.toString();
    console.log('Python output: ${data}');
});

pythonProcess.on('close', (code: number) => {
    ...
    const jsonMatch = pythonData.match(/(\{.*\})/s);
    if (jsonMatch && jsonMatch[0]) {
        const extractedEntities = JSON.parse(jsonMatch[0]);

        return res.json({
            extractedEntities: extractedEntities,
            uploadedPdfPath: pdfPath,
            message: 'Entity extraction completed successfully'
        });
    }
    ...
});
```

Lst. 4.2: Python subprocess execution for entity extraction.

Following the entity review phase, the server uses the same subprocess orchestration pattern for comprehensive processing of the main pipeline script with filtered entity data, PDF path, API key, and metadata as inputs. This consistent subprocess architecture enables the server to seamlessly manage both initial entity extraction and complete pipeline processing while maintaining uniform communication patterns and error handling throughout the multi-stage workflow.

Python Processing Chain

To ensure the seamless execution of the Python scripts, the system implements a cascading subprocess execution where the initial Python script triggers subsequent processing chains. All processing methods utilize Python's `subprocess.run` mechanism [@22] (Listing 4.3).

```
# This will also trigger convert_data_to_gef.py script
convert_script_path =
    os.path.join(script_dir, '../convert_data_to_glp.py')

cmd = [
    'python',
    convert_script_path, pdf_path,
    '--out', output_txt_path,
    '--filename', project_name,
    '--entity-id-map', json.dumps(entity_id_map)
]

if title:
    cmd.extend(['--title', title])
if doi:
```

```
    cmd.extend(['--doi', doi])
...
result = subprocess.run(cmd, capture_output=True, text=True)
```

Lst. 4.3: Example of Python subprocess execution for project file conversion with necessary inputs.

The Python processing pipeline is architected with a modular design, with each major processing step encapsulated in a dedicated script (Figure 4.1). The main orchestration and data transformation logic is implemented in `main_pipeline.py` with three distinct processing methods. Depending on the selected method, it invokes the corresponding module to perform extraction. After collecting the processing results, it continues with further data transformation and then triggers the project files conversion process by `convert_data_to_glp.py`, followed by `convert_data_to_gef.py`.

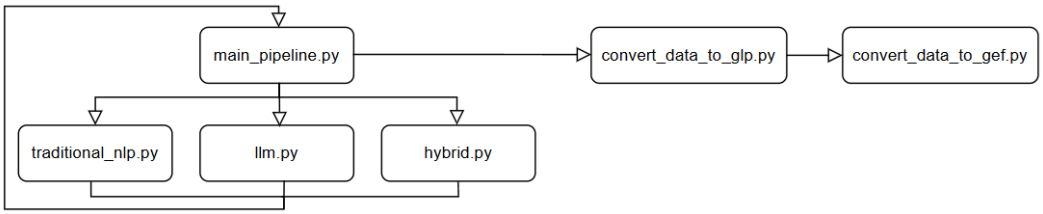


Fig. 4.1.: Process flow for Python script execution.

For traditional NLP processing, the system uses `conda` environment isolation (Listing 4.4) to handle dependency requirements of legacy libraries (e.g. `neuralcoref`) that are incompatible with modern Python environments.

```
# Custom conda environment name
conda_env = "neurocoreference"

cmd = [
    "conda", "run", "-n", conda_env, "python",
    nlp_script_path, input_path, output_path
]

if filtered_entities:
    filtered_entities_json = json.dumps(filtered_entities)
    cmd.extend(["--filtered-entities", filtered_entities_json])

result = subprocess.run(cmd, capture_output=True, text=True)

with open(output_path, 'r', encoding='utf-8') as f:
    results = json.load(f)
```

Lst. 4.4: Traditional NLP subprocess with conda environment isolation.

The output file is necessary because the conda environment isolation prevents direct data return from the subprocess, requiring JSON file serialization to transfer complex entity and relationship data structures between the isolated Python process and the main pipeline.

Error Handling

The system implements comprehensive error handling throughout the entire workflow, providing users with informative status messages and actionable guidance for resolving issues, including invalid credentials, insufficient quota, extraction error, paper content error, and other issues.

The error handling architecture operates at multiple critical points in the workflow. For example, OpenAI API key validation occurs before every LLM interaction by checking both credential validity and quota availability through the `check_openai_key` function (Listing 4.5). This validation ensures API calls only with verified credentials and sufficient quota for each extraction step, preventing failed requests and unnecessary API consumption during the multi-stage LLM processing pipeline.

```
def check_openai_key(client):
    try:
        response = client.models.list()
        print("API key is valid. Available models:")
        for model in response.data[:3]:
            print(f"- {model.id}")
        return True
    except Exception as e:
        print(f"API key is invalid or has quota issues: {e}")
        return False
```

Lst. 4.5: Function checks valid OpenAI API key.

The system also validates the document's relevance using checking mechanisms through a ChatGPT-based prompt from Goraya [7] to determine if a publication meets neuroscience criteria after checking for a valid and sufficient API key when users start the entities extraction process. For the traditional NLP method, irrelevant publications are handled by returning a message to users when no entities are extracted, as the method does not require an API key. When publications fail to meet neuroscience criteria, the system terminates the process with clear explanations and suggests alternative approaches to prevent unnecessary resource consumption.

4.2 Extraction and Output Structuring

This section describes the pipeline for extracting entities and relationships from neuroscience publications. While the traditional NLP, LLM, and hybrid methods are adapted from a scattered implementation of Goraya [7], this thesis unifies them into a cohesive pipeline with adjusted prompts. Extracted entities are linked and all data is validated and well-structured for downstream use.

4.2.1 Extraction Methods

Traditional NLP Method

The traditional NLP method implements a multi-stage processing pipeline that leverages established neuroscience-specific models and techniques for entity and relationship extraction:

1. SciBERT-based NER extraction for neuroscience-specific entities with categorization mainly into brain structures, behaviors, and stimuli.
2. Entity linking maps recognized entities to corresponding database IDs. Users then review and edit the updated list of entities in the interface.
3. Neuralcoref for coreference resolution to link pronouns and references to their respective entities.
4. Relationships extraction with a fine-tuned SciBERT model specifically trained for neuroscience relationship classification, loading custom weights onto a domain-adapted scientific language model.

LLM-based (ChatGPT) Method

The system implements a model selection strategy with *GPT-4o* as the primary model and fallback capabilities to other models in case of errors or unavailability. This thesis explicitly extends the prompts for comprehensive neuroscience entity and relationship extraction processing. LLM receives the prompt that contains the list of extracted entities, the PDF text, and detailed instructions for specific tasks.

The method begins with comprehensive entity extraction using the redesigned prompt (Appendix A.1). The system implements strict criteria for output with categorical organization into brain structures, stimuli, and behaviors. Critical extraction rules prevent the inclusion of parenthetical explanations, ensure exact name preservation, and exclude methodology equipment or general biological terms unless they represent main specific neuroscience entities.

The extracted entities are then linked to corresponding database IDs using the same entity linking as the traditional method, and are reviewed and modified by users in the interface.

For relationship processing, the method uses an entity-constrained prompt that provides the LLM with the extracted and reviewed entities (Appendix A.2). This approach prevents the hallucination of non-existent entities while allowing the model to focus on identifying main, real relationships between validated entities. It initially extracts relationships using brain structures rather than individual neurons, consistent with the traditional NLP method.

Subsequently, these brain structure-level tuples are transformed into neuron-level tuples during the next stage.

After initial relationships are extracted, the system leverages logprobs for confidence scoring, applying threshold-based filtering with **85%** confidence requirements to retain only those relationships with a high accuracy and relevance.

Following relationship extraction, the system converts the filtered relationships into structured tuples through a specialized tuple creation process using a dedicated prompt (Appendix A.3). It applies strict formatting, validation rules, relationship constraints, and converts brain structure-level relationships to neuron-level tuples, while creating corresponding *belongs_to* relationships. This ensures that complex relationship descriptions are converted into a list of standardized three-element tuples *(entity1, entity2, relation)*, suitable for data processing.

Hybrid Method

The Hybrid method implements a sequential processing architecture that combines the reliability of traditional NER with the contextual understanding capabilities of large language models for extraction. It begins with SciBERT-based NER for initial entity extraction (Figure 4.2), followed by the entity linking process and user review.


```
{"Structure": ["antprl", "anterior prelimbic cortex", "cg1 area 1",
               "Periaqueductal gray", "Lateral reticular nucleus",
               "dpc cortex", "medial prefrontal cortical areas", "Orbital area",
               "m1 motor cortex", "inl cortex", "m2 motor cortex", "dorsal areas",
               "Midbrain reticular nucleus", "Entorhinal area", "deep layers",
               "Medullary reticular nucleus", "cg2", "habenula", "vmc",
               "Nucleus retroambiguus", "ventrolateral part", "Striatum",
               "postprl prelimbic cortex", "parvocellular reticular formation",
               "Accessory trigeminal nucleus", "prefrontal cortices",
               "Ventral tegmental area", "dorsal prelimbic cortex",
               "ventrolateral periaqueductal gray", "posterior prelimbic cortex",
               "Medial pretectal area", "Intermediate reticular nucleus",
               "medial prefrontal cortex", "Secondary motor area",
               "cingulate area 1", "Midbrain reticular nucleus", "Prelimbic area",
               "Primary motor area", "ventral prelimbic cortex", "cingulate area 2",
               "Infralimbic area", "primary cingulate cortex", "superficial layers",
               "dorsopenducular cortex", "ventral orbitofrontal cortex",
               "Medial group of the dorsal thalamus", "retrosplenial cortex cortex"],
 "Behaviour": ["vocalizations", "fear calls", "calls", "USVs",  "tickling",
               "microstimulation", "nonresponsive"],
 "Stimulus":  ["stimulation", "usvs", "nacl", "microstimulation", "vo", "sounds",
              "vocalizations", "calls", "electrically"]}
```


Fig. 4.2.: Output from NER entity extraction with entity linking.

The method then applies the identical LLM-based relationship extraction processes as the ChatGPT approach with logprobs-based confidence scoring

and threshold filtering (Figure 4.3a), resulting in a set of high-confidence relationships (Figure 4.3b).

```
[['Word:', 'Relationships:',', logprobs:', -0.004038, ', linear probability:', 99.6, '%'],
 ['Word:', ' -posterior prelimbic cortex projects_to Intermediate reticular nucleus', ',
       logprobs:', -0.060096, linear probability:', 94.17, '%'],
 ['Word:', '- posterior prelimbic cortex projects_to Nucleus retroambiguus', ',
       logprobs:', -0.055225, ', linear probability:', 94.63, '%'],
 ['Word:', '- posterior prelimbic cortex projects_to Periaqueductal gray', ',
       logprobs:', -0.077094, ', linear probability:', 92.58, '%'],
 ['Word:', '- posterior prelimbic cortex is_substructure_of medial prefrontal cortex', ',
       logprobs:', -0.061185,', linear probability:', 94.06, '%'],
 ['Word:', '- ci ngulate area 2 is_substructure_of medial prefrontal cortex',',
       logprobs:', -0.043624, ', linear probability:', 95 .73, '%'],
 ['Word:', '- Periaqueductal gray projects_to Nucleus retroambiguus',',
       log probs: ' -0.060047, linear probability:', 94.17, '%'],
 ['Word:', '- posterior prelimbic cortex evokes vocalizations', ',
       logprobs:', -0.044645, ', linear probability:', 95.63, '%']]
```

(a) Log probability scores after applying an 85% confidence threshold.

```
Relationships:
- posterior prelimbic cortex projects_to Intermediate reticular nucleus
- posterior prelimbic cortex projects_to Nucleus retroambiguus
- posterior prelimbic cortex projects_to Periaqueductal gray
- posterior prelimbic cortex is_substructure_of medial prefrontal cortex
- cingulate area 2 is_substructure_of medial prefrontal cortex
- Periaqueductal gray projects to Nucleus retroambiguus
- posterior prelimbic cortex evokes vocalizations
- cingulate area 2 evokes vocalizations
```

(b) Pruned relationships.

Fig. 4.3.: Relationship extraction output: (a) Log probability scores and (b) pruned relationships.

Following the relationship identification, the same tuple creation process as in the ChatGPT method is applied to convert the relationships into a list of standardized three-element tuples (Figure 4.4).

```
RAW GPT OUTPUT:
(posterior prelimbic cortex neurons, Intermediate reticular nucleus neurons, projects_to)
(posterior prelimbic cortex neurons, Nucleus retroambiguus neurons, projects_to)
(posterior prelimbic cortex neurons, Periaqueductal gray neurons, projects_to)
(posterior prelimbic cortex neurons, posterior prelimbic cortex, belongs_to)
(Intermediate reticular nucleus neurons, Intermediate reticular nucleus, belongs_to)
(Nucleus retroambiguus neurons, Nucleus retroambiguus, belongs_to)
(Periaqueductal gray neurons, Periaqueductal gray, belongs_to)
(cingulate area 2, medial prefrontal cortex, is_substructure_of)
(cingulate area 2 neurons, cingulate area 2, belongs_to)
(posterior prelimbic cortex neurons, vocalizations, evokes)
(cingulate area 2 neurons, vocalizations, evokes)
```

Fig. 4.4.: LLM-generated relationship tuples.

The Hybrid approach addresses the key limitations of both pure traditional NLP method (limited contextual understanding) and pure LLM-based approach (entity hallucination tendencies) to create a more balanced extraction pipeline optimized for both accuracy and completeness.

Entity Linking

As described in each extraction method, entity linking is performed after entity extraction and before the user's review process [7]. It progressively matches extracted entities with entries in the Neo4j database by applying exact name matching, fuzzy similarity matching with a 75% threshold, synonym, acronym matching for brain structures, and substring matching for behaviors and stimuli. The function returns a list of tuples of *(entity_type, entity_id, entity_name)* for each processed entity, with successful matches providing database IDs and failed matches returning descriptive error messages.

Building on this, the pipeline creates two standardized data structures essential for downstream processing. The first output serves as the new extracted entities list with the same categorical format as the original extracted entities, but contains the updated entity names from successful database linking. This ensures that entities are represented using their canonical database names rather than the varied terminologies from the original text.

The second output transforms the linking results into a dictionary structure (Figure 4.5), where each entity name maps to a two-element list containing the entity type and database ID. Successfully linked entities receive their corresponding database IDs, while unlinked entities have null values. This format enables immediate identification of linking success and provides the necessary database references for subsequent steps in the pipeline.


```
{'antprl': ['STRUCTURE', None], 'anterior prelimbic cortex': ['STRUCTURE', None], 'cg1 area 1': ['STRUCTURE', None]
 'Periaqueductal gray': ['STRUCTURE', 335], 'Lateral reticular nucleus': ['STRUCTURE', 816],
 'dpc cortex': ['STRUCTURE', None], 'medial prefrontal cortical areas': ['STRUCTURE', None],
 'Orbital area': ['STRUCTURE', 59], 'm1 motor cortex': ['STRUCTURE', None], 'inl cortex': ['STRUCTURE', None],
 'm2 motor cortex': ['STRUCTURE', None], 'dorsal areas': ['STRUCTURE', None],
 'Midbrain reticular nucleus': ['STRUCTURE', 117], 'Entorhinal area': ['STRUCTURE', 305],
 'deep layers': ['STRUCTURE', None], 'Medullary reticular nucleus': ['STRUCTURE', 228], 'cg2': ['STRUCTURE', None],
 'habenula': ['STRUCTURE', None], 'vmc': ['STRUCTURE', None], 'Nucleus retroambiguus': ['STRUCTURE', 1332],
 'ventrolateral part': ['STRUCTURE', None], 'Striatum': ['STRUCTURE', 1243], 'postprl prelimbic cortex': ['STRUCTUR
 'parvocellular reticular formation': ['STRUCTURE', None], 'Accessory trigeminal nucleus': ['STRUCTURE', 114],
 'prefrontal cortices': ['STRUCTURE', None], 'Ventral tegmental area': ['STRUCTURE', 1211],
 'dorsal prelimbic cortex': ['STRUCTURE', None], 'ventrolateral periaqueductal gray': ['STRUCTURE', None],
 'posterior prelimbic cortex': ['STRUCTURE', None], 'Medial pretectal area': ['STRUCTURE', 884],
 'Intermediate reticular nucleus': ['STRUCTURE', 1088], 'medial prefrontal cortex': ['STRUCTURE', None],
 'Secondary motor area': ['STRUCTURE', 831], 'cingulate area 1': ['STRUCTURE', None], 'Prelimbic area': ['STRUCTURE
 'Primary motor area': ['STRUCTURE', 1002], 'ventral prelimbic cortex': ['STRUCTURE', None],
 'cingulate area 2': ['STRUCTURE', None], 'Infralimbic area': ['STRUCTURE', 1162],
 'primary cingulate cortex': ['STRUCTURE', None], 'superficial layers': ['STRUCTURE', None],
 'dorsopenducular cortex': ['STRUCTURE', None], 'ventral orbitofrontal cortex': ['STRUCTURE', None],
 'Medial group of the dorsal thalamus': ['STRUCTURE', 357], 'retrosplenial cortex cortex': ['STRUCTURE', None],
 'vocalizations': ['STIMULUS', None], 'fear calls': ['BEHAVIOR', None], 'calls': ['STIMULUS', None],
 'USVs': ['BEHAVIOR', 1329], 'tickling': ['BEHAVIOR', None], 'microstimulation': ['STIMULUS', None],
```


Fig. 4.5.: Entity linking output with mapped database IDs and null values.

4.2.2 Output Structuring & Validation

After the entity linking step, the updated list of entities is displayed to users for review and editing. They can modify properties such as synonyms, acronyms, descriptions, and hierarchical relationships for brain structures. The finalized entities are saved as a separate, comprehensive output file with user-refined

information. Following relationship extraction, all refined entities and relationships are combined into a standardized text summary (Figure 4.6).

The output has two main sections. All entries in both sections are sorted alphabetically with case sensitivity, and entity names are preserved in their original case as they were extracted from the source text, or as they appear in the database if linked:

- **Nodes** section contains extracted entities, with each entry following the format *[Entity Name] [type=CATEGORY]*. The section includes reviewed extracted entities alongside automatically generated neurons.
- **Edges** section documents all validated relationships with format *[Source Entity] -> [Target Entity] [relationship=RELATIONSHIP_TYPE]*. It captures explicit relationships from the extraction process and automatically generated structural relationships such as *belongs_to* between neurons and their parent brain structures.

```
# Nodes
Intermediate reticular nucleus [type=STRUCTURE]
Intermediate reticular nucleus neurons [type=NEURON]
Medullary reticular nucleus [type=STRUCTURE]
Midbrain reticular nucleus [type=STRUCTURE]
Nucleus retroambiguus [type=STRUCTURE]
Nucleus retroambiguus neurons [type=NEURON]
Periaqueductal gray [type=STRUCTURE]
Prelimbic area [type=STRUCTURE]
USVs [type=BEHAVIOR]
anterior prelimbic cortex [type=STRUCTURE]
cingulate area 2 [type=STRUCTURE]
cingulate area 2 neurons [type=NEURON]
dorsal prelimbic cortex [type=STRUCTURE]
fear calls [type=BEHAVIOR]
microstimulation [type=STIMULUS]
posterior prelimbic cortex [type=STRUCTURE]
posterior prelimbic cortex neurons [type=NEURON]
posterior prelimbic cortex_1 neurons [type=NEURON]
posterior prelimbic cortex_2 neurons [type=NEURON]
postprl prelimbic cortex [type=STRUCTURE]
vocalizations [type=BEHAVIOR]
...

# Edges
Intermediate reticular nucleus neurons -> Intermediate reticular nucleus [relationship=BELONGS_TO]
Nucleus retroambiguus neurons -> Nucleus retroambiguus [relationship=BELONGS_TO]
cingulate area 2 neurons -> cingulate area 2 [relationship=BELONGS_TO]
microstimulation -> cingulate area 2 neurons [relationship=EXCITES]
microstimulation -> posterior prelimbic cortex neurons [relationship=EXCITES]
posterior prelimbic cortex neurons -> posterior prelimbic cortex [relationship=BELONGS_TO]
posterior prelimbic cortex_1 neurons -> Intermediate reticular nucleus neurons [relationship=PROJECTS_TO]
posterior prelimbic cortex_1 neurons -> posterior prelimbic cortex [relationship=BELONGS_TO]
posterior prelimbic cortex_2 neurons -> Nucleus retroambiguus neurons [relationship=PROJECTS_TO]
posterior prelimbic cortex_2 neurons -> posterior prelimbic cortex [relationship=BELONGS_TO]
```

Fig. 4.6.: Structured output from entities and relationships extraction.

The extracted relationships are validated using two key criteria. First, it checks relationships against the data model (see Section 3.1.1) to ensure only valid entity type combinations are allowed (e.g., only neurons can *"evoke"* behaviors). Second, it verifies that both entities in each relationship exist in the extracted entity list with exact name matching, preventing invalid connections to non-existent entities. The function only retains neurons that

participate in functional relationships beyond the *belongs_to* connection with their parent brain structure, filtering out isolated neurons that would serve no meaningful purpose in the circuit representation.

The function addresses a key challenge in neuroscience literature: papers often describe brain structure connections without specifying which exact neurons are involved. To handle this uncertainty, the system implements a conservative approach by creating separate neuron populations when a brain structure has multiple different relationships. The numbering system begins with the base neuron (e.g., *"posterior prelimbic cortex neurons"*) and then creates additional numbered variants (e.g., *"posterior prelimbic cortex_1 neurons"*, *"posterior prelimbic cortex_2 neurons"*) when that neuron would have multiple outgoing relationships beyond the mandatory *belongs_to*.

If a brain structure name already contains a number (e.g., *"Cingulate area 1"*) and multiple neuron populations are required for that brain structure, additional numbering is appended next to that number (e.g., *"Cingulate area 1_1 neurons"*). Rather than assuming the same neurons perform all functions, each numbered variant represents a separate group within the same brain structure.

Subsequently, this structured extraction output serves as the primary input for the project file conversion step by providing all validated entities and relationships in a standardized format required for generating project files.

4.3 Project Files Conversion

Converting extracted data into the GLP and GEF format is essential for compatibility, as the current editor tool is specifically designed to read and process data structured in these formats [8]. Both GLP and GEF files serve as direct outputs that can be immediately loaded, visualized, and edited by the system without requiring additional transformation.

4.3.1 Gnoilinx Linking Paper (GLP)

GLP is a structured JSON-based file format designed to capture neuroscience knowledge metadata. During the conversion process, paper metadata, a list of newly created or extracted entities with their properties, hierarchical relationships, error tracking information, and references to GEF files are all added to the GLP file.

This conversion process requires several types of input data:

1. Validated and structured entities and relationships from Section 4.2.2, including nodes such as brain structures, behaviors, and stimuli, as well as the relationships between them.

2. Entity-to-database ID mappings, which link extracted entity names to their corresponding IDs and types. This mapping allows the converter to distinguish between new and existing entities and to reuse global IDs if the mapping is successful.

3. User-provided metadata, such as the paper title, authors, DOI, journal name, and publication year, which is included in the GLP file to provide detailed bibliographic information about the source paper.

4. Additional entity details such as synonyms, acronyms, descriptions, tags, and hierarchical parent relationships for brain structures (provided by the user during the review phase).

The main function is responsible for constructing the entities (nodes) that form the core of the GLP output.

Node Construction

The function first gathers all entities and assigns each a unique identifier. If an entity already exists in the database, as determined by the entity-to-database ID mapping, it reuses the existing global ID. For new entities, it generates a new unique ID by incrementing the last used global ID, which is retrieved from the database using a Cypher query (Listing 3.5). This approach ensures that all new entities receive sequential and unique identifiers, avoiding conflicts not only with existing entries in the database but also with entities from other projects that have not yet been committed.

Once unique IDs are assigned, the function gathers and assigns properties to each node such as name, type, acronym, synonyms, description, and tags. These properties are retrieved from both the database for existing nodes and the detailed entity information provided by the user. All nodes are constructed in an `allNodes` dictionary, which contains a flat mapping of them by their unique IDs and their properties for fast and convenient lookup.

Explicitly, the function determines hierarchical relationships for brain structures by assigning parent nodes based on user-provided or extracted information. For each node with a parent, the parent's details are recorded in the `newlyCreatedNodesParents` mapping, and the node is grouped under its parent in the `newlyCreatedNodes` structure. Nodes without a parent are placed under the `NEW_ROOT_NODES` category, indicating they are top-level entities. Additionally, the function checks whether a node already has a parent assigned in the database to prevent assigning multiple parents to the same node, as each brain structure can only have one parent (Listing 3.6).

Relationship Edge Processing

Next, the extracted relationships between entities are processed to create relationship edges for later use in the GEF file conversion. During this step, the function ensures that whenever an entity appears with multiple possible types (such as both a stimulus and a behavior), the correct type is selected for each relationship according to predefined rules, whereas the previous validation step only checks for the existence of a compatible type.

To ensure the integrity of the output and avoid circular parent-child relationships, the code checks both directions of substructure relationships before adding them. If a parent-child relationship is already defined by users in the web interface, the reverse relationship extracted from the edge list is skipped. This approach prioritizes the users' curated data, ensuring that only one consistent direction for each parent-child relationship is retained.

Temporary Data Cleanup

Finally, the function generates the GLP file locally in JSON format, which includes nodes (with hierarchy, new entities, and properties), metadata, and other relevant information. The `allNodes` dictionary and all extracted relationship edges are initially added to the GLP file for the GEF file generation, which exclusively uses the GLP file as its input data. However, after the whole project file conversion is complete, they are removed from the GLP file, as this information is not needed for further use within the GLP format.

4.3.2 Gnoilinx Experiment File (GEF)

GEF is a structured JSON-based file format designed for graphical visualization of neuroscience knowledge graphs. It contains graphical nodes (brain structures, neurons, stimuli, and behaviors) with position, size, and metadata, neuron elements with their properties linked to parent brain structures, arrows for relationships with type and coordinates, brain structure hierarchy mappings, experiment metadata (species, sex, age, and description), and visualization state (canvas size, scale, and selected elements).

To generate a GEF file, a GLP file is required as the primary input. Within this GLP file, the most important data used for conversion are the `allNodes` dictionary, which contains all entities along with their properties, and the list of relationships, which defines how these entities with types are connected. These two components provide the essential information needed to construct the graphical nodes, neurons, and arrows in the resulting GEF file.

A key challenge in generating the GEF file is the positioning of nodes and neurons on the canvas. To ensure a clear and organized representation, the

system must assign positions, sizes, and layouts that minimize overlap and maintain readability, even as the number of nodes and neurons increases. The following methods are applied for effective spatial layout and positioning:

Initial Positioning of Nodes

The algorithm utilizes fixed base coordinates and spacing values to establish a virtual grid for node placement. It sets width and height by node type, ensuring that brain structures, stimuli, and behaviors are sized appropriately.

The placement of each node is calculated by counting how many nodes of each type have already been placed. Brain structures are placed with up to four brain structures in a row and the next brain structure moves down to the next row below once the row is filled. Behaviors and stimuli are positioned above the brain structures, also with up to four nodes per row, and move to the next row above when the current row is filled (Figure 4.7).

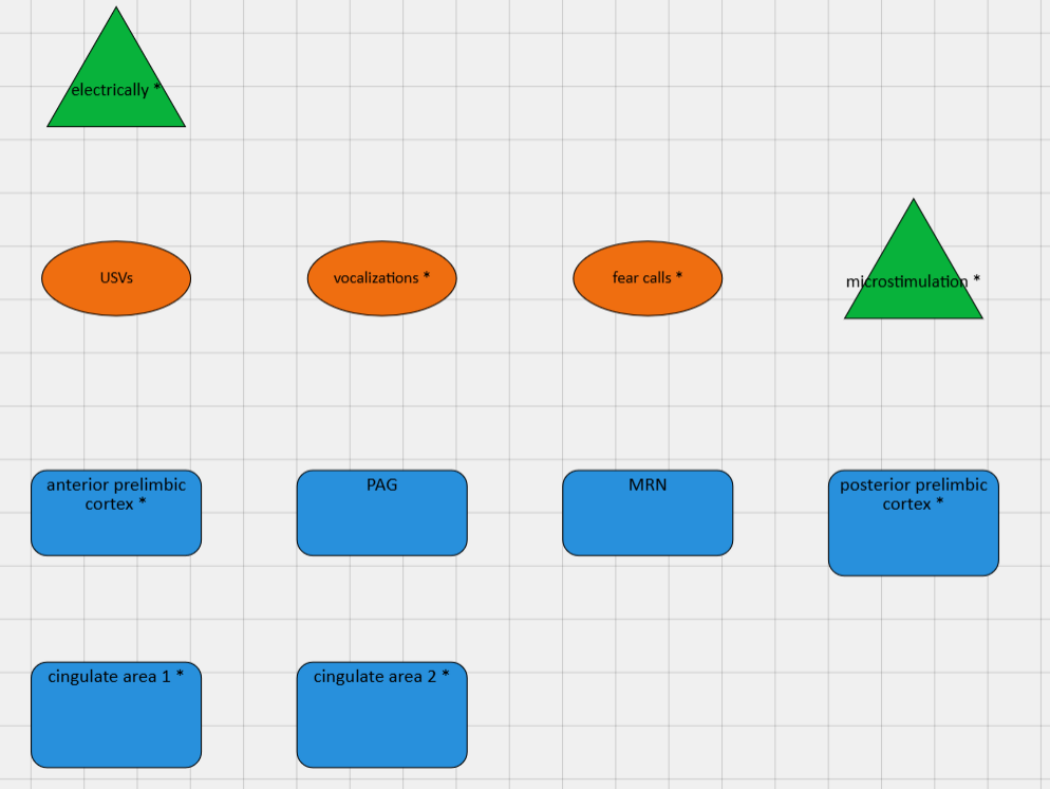


Fig. 4.7.: Example of grid-based node placement.

The final *(x, y)* position for each node is computed using the grid base, spacing, offsets, and the calculated row/column. During initial placement, the algorithm also checks for overlaps and adjusts node positions, if necessary, though collisions are uncommon due to the grid layout.

Resizing Brain Structures Based on Neuron Count

After neurons are assigned to brain structures, the algorithm recalculates each brain structure's size to fit all its neurons. The width and height of a brain structure are increased as needed, based on the number of neurons it contains with up to three neurons per row. It calculates the required space by considering the diameter of each neuron, the spacing between them, and padding for clear separation and visual clarity.

For brain structures with up to three neurons, the width is adjusted to fit the exact number of neurons in a single row. With more than three neurons, the width fits three neurons per row, and the height increases with the number of rows needed. Therefore, each brain structure remains visually distinct and proportionate to the number of neurons it contains.

Placement of Neurons Within Brain Structures

With the resized brain structure, neurons are positioned inside their parent brain structure using a grid layout. Neurons are arranged in rows of up to three, with each neuron's position calculated to ensure even spacing and visual balance. The algorithm centers the neurons' grid vertically and horizontally within the available space by considering padding and margins.

For incomplete rows with fewer than three neurons, the neurons are horizontally centered within the brain structure to maintain symmetry. As a result, all neurons are displayed in a clear and logical organization within their parent brain structure without any overlaps or misalignments (Figure 4.8).

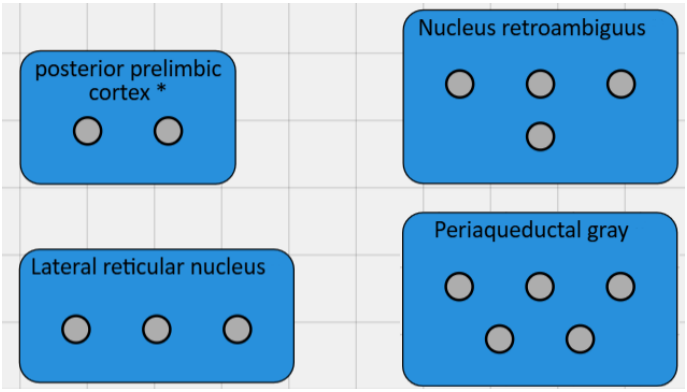


Fig. 4.8.: Example of neuron placement with different numbers of neurons.

Collision Detection & Resolution

After brain structures are resized to accommodate their neurons, the algorithm performs a collision resolution step to maintain clear separation

between them. It checks for vertical overlaps by comparing the coordinates of all brain structure nodes. If any brain structures overlap, the algorithm shifts the overlapping brain structure downward to make a minimum gap between them. This cascading adjustment is repeated as necessary so that all brain structures remain visually distinct and do not overlap, even as their sizes change to fit varying numbers of neurons.

The GEF file is constructed for graphical visualization in the web-based editor tool through the following main steps:

- **Creates Brain Structure, Stimulus, and Behavior Nodes:** Stores nodes' metadata and places them on a virtual grid by assigning size and position using the method described above for initial positioning.
- **Processes Neuron Assignments:** For each *belongs_to* relationship, generates a unique GEF neuron ID, creates a neuron object with default properties (position, radius, cell type, molecular marker), links it to its parent brain structure, and adds it to the brain structure's neuron list.
- **Resizes Brain Structures, Positions Neurons, and Resolves Collisions:** Adjusts brain structure sizes based on neuron count, arranges neurons within them, and resolves any collisions between brain structures to maintain a clear layout using the methods described above for resizing, positioning, and collision resolution.
- **Creates Arrows for Relationships:** For each relationship (except *belongs_to* and *is_substructure_of*), determines the source and target positions based on node or neuron coordinates, creates an arrow object with the relationship type, and updates the corresponding neurons' incoming and outgoing arrow lists when applicable.
- **Builds Brain Structure Hierarchy:** Constructs ancestor mappings for brain structures using hierarchical information from the GLP file.

After the project files are generated locally, they are removed from local storage and uploaded to the Neo4j database by calling a custom server-side function.

In summary, this chapter presented the complete backend implementation of the neuroscience knowledge extraction pipeline, highlighting its modular design for flexibility, robust error handling, multiple-method extraction, and effective data transformation. The resulting system enables effective and efficient transformation of neuroscientific literature into structured and visualizable knowledge graphs in the web-based editor tool.

5 Demonstration

This chapter extends the Gnoilinx web application built upon the work by Götz [8], with significant usability and workflow improvements. It demonstrates the end-to-end knowledge extraction pipeline, including secure authentication, PDF upload and processing, entity review, and graph visualization. [1]

5.1 Authentication Pages

This section demonstrates the user authentication system, including the modified *Home* page interface with the integrated sign-in functionality, the *Login* page for existing users, and the *Register* page for new users.

All user-specific operations in Gnoilinx are secured by a secure server-client authentication system using password hashing (`bcrypt`) [@15] and token-based session management (`JWT`) [@3]. This ensures that only authenticated users can access, modify, or manage their personal data and projects. All changes to user data and project information are now fully and securely shifted from local storage to the cloud using Neo4j database queries.

The *Home* page features the main application interface with the Gnoilinx logo and navigation buttons. A *Sign In* button has been added to the top-right corner (Figure 5.1), providing access to the authentication system. When users are not authenticated, all navigation buttons redirect to the *Login* page.

Fig. 5.1.: Modified *Home* page with sign-in functionality.

[1] All demonstrations in this chapter use [4] as the example input paper for processing.

5.1.1 Login Page

The *Login* page is designed for existing users to access their accounts securely. It features a simple form where users can enter their credentials, including their username and password. For enhanced usability, the password field includes a toggle button that allows users to show or hide their password as they type. The page also includes a *Register* link, allowing users to create a new account if they do not already have one (Figure 5.2).

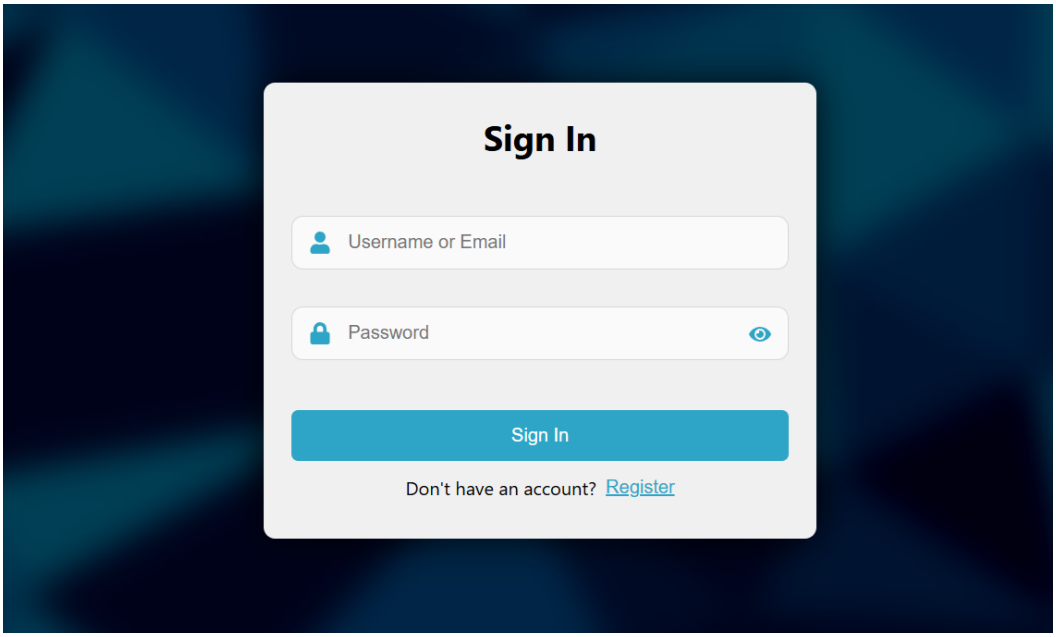
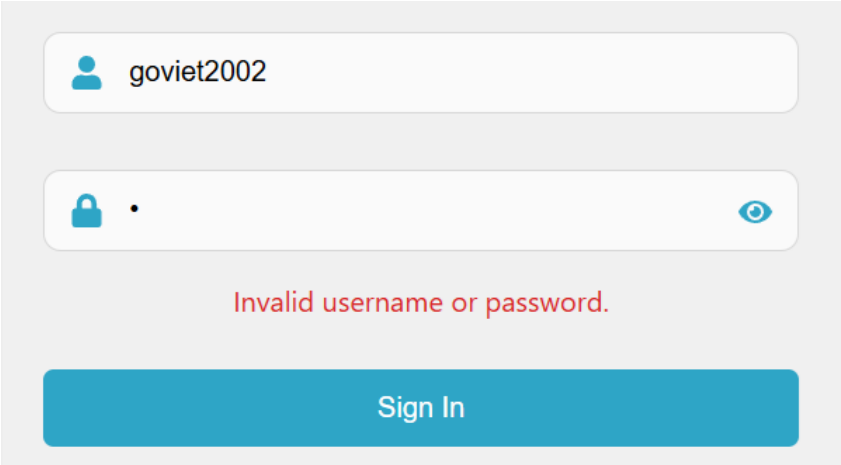


Fig. 5.2.: *Login* page for existing users.

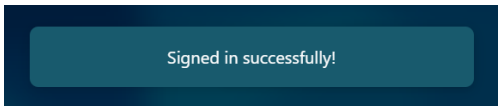
After users submit their credentials, the backend matches the entered password with the hashed password stored in the database using the `bcrypt` cryptographic hash function, ensuring secure authentication. The page also implements form validation by checking against the database and displays an error message *"Invalid username or password"* when users attempt to sign in with incorrect credentials (Figure 5.3a).

Upon successful login, users are redirected to the *Home* page with a confirmation toast[2] message displayed at the top right, temporarily replacing the login button (Figure 5.3b). The *Sign In* button transforms into a personalized welcome message showing the user's name (Figure 5.3d). When users hover over this, the welcome message reveals a dropdown menu with a *Sign Out* option (Figure 5.3e), providing access to logout functionality. When they sign out, another toast message will appear to confirm a successful logout (Figure 5.3c).

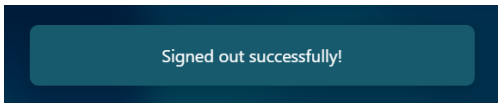
[2] A toast is a small, temporary notification that appears on the screen to inform users of an action's result.



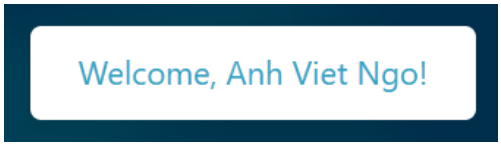
(a) Error message for invalid login attempts.



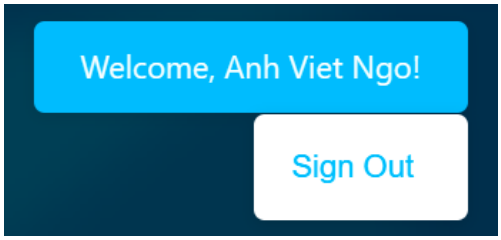
(b) Login success toast.



(c) Logout success toast.



(d) Personalized welcome message.



(e) Dropdown with *"Sign Out"* option.

Fig. 5.3.: Authentication feedback: (a) Login error message, (b) Login toast, (c) Logout toast, (d) Welcome message and (e) Sign out dropdown.

5.1.2 Register Page

If users are new to the web application, they can create an account and gain access to the platform via the *Register* page. It presents a user-friendly form requiring the name, username, email address, and password, along with a confirmation of the password to prevent typographical errors. Both the password and confirm password fields include toggle buttons, allowing users to show or hide their input for convenience (Figure 5.4).

Before storing the password in the database, the backend hashes it using the `bcrypt` cryptographic hash function, which automatically generates a unique random value for each password. Only the resulting hash is stored in the database, providing strong security for all registered accounts and preventing storage of plain-text passwords.

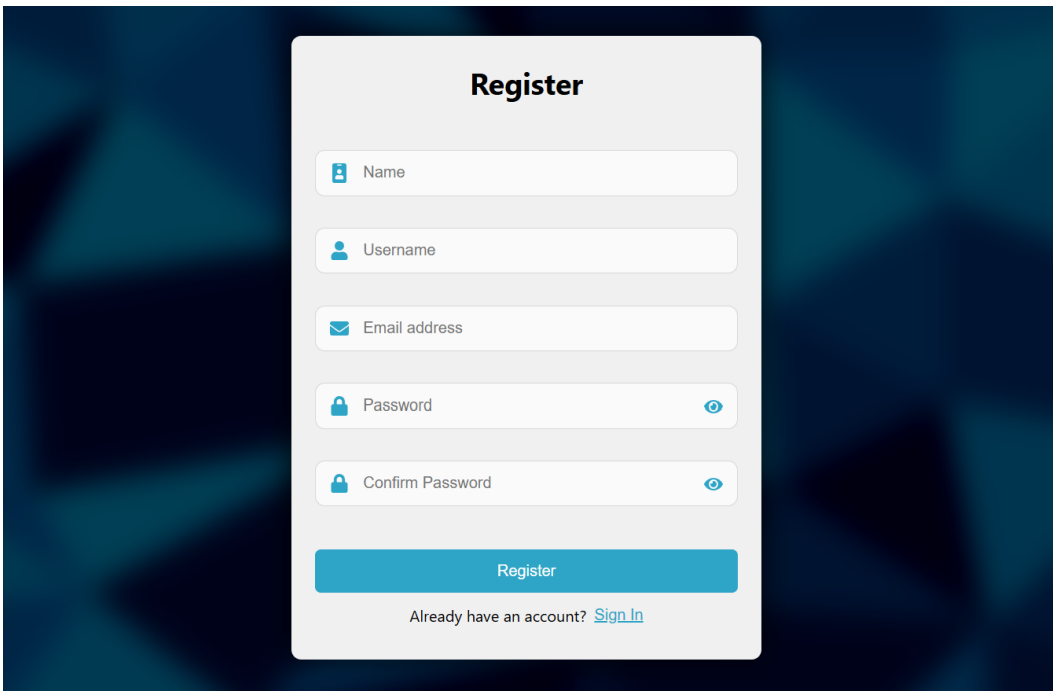


Fig. 5.4.: *Register* page for new users.

The page provides feedback on username and email availability in the database, as well as validation for the email format. If the given username or email address is already taken, the email pattern is invalid, or the passwords do not match, appropriate error messages are displayed and the registration button remains disabled until all requirements are met (Figure 5.5).

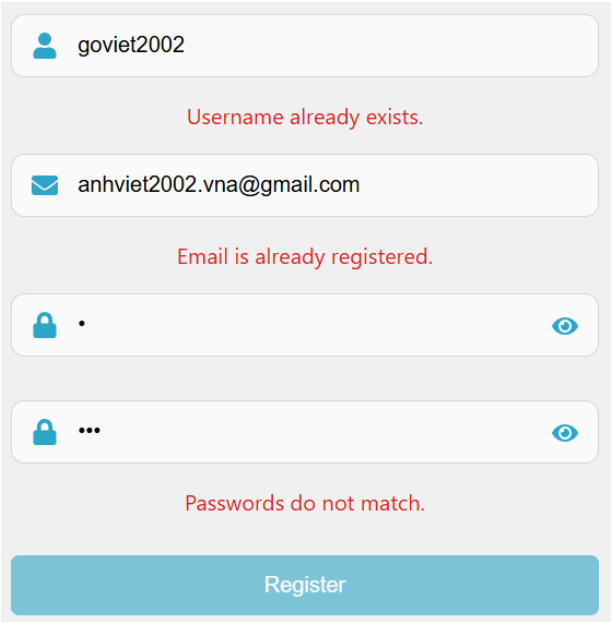


Fig. 5.5.: Error messages for invalid entries.

Upon successful registration, users are redirected to the *Login* page, where a toast notification confirming their registration appears at the top-right corner (Figure 5.6).

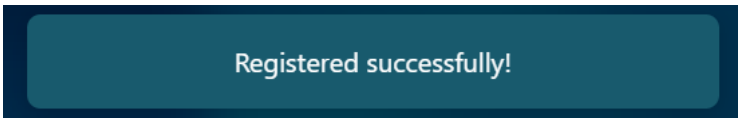


Fig. 5.6.: Successful registration toast notification.

5.2 PDF Upload Page

This section demonstrates the workflow for uploading and processing PDF documents within the web application, including the selection of different processing methods and the option to provide an OpenAI API key for AI-powered extraction.

Before reaching the page for uploading PDFs, users start on the modified *New Project* page, which is accessed from the *Home* page. Previously, the option to upload a PDF was available directly within the old *New Project* page (Figure 5.7). In the updated workflow, this functionality has been shifted to a dedicated *PDF Upload* page. The *New Project* page now features a *Create from PDF* button among its action buttons, which navigates users to the new upload page for PDF-based project creation only when all metadata is filled (Figure 5.8). This change separates the metadata entry from the document upload process.

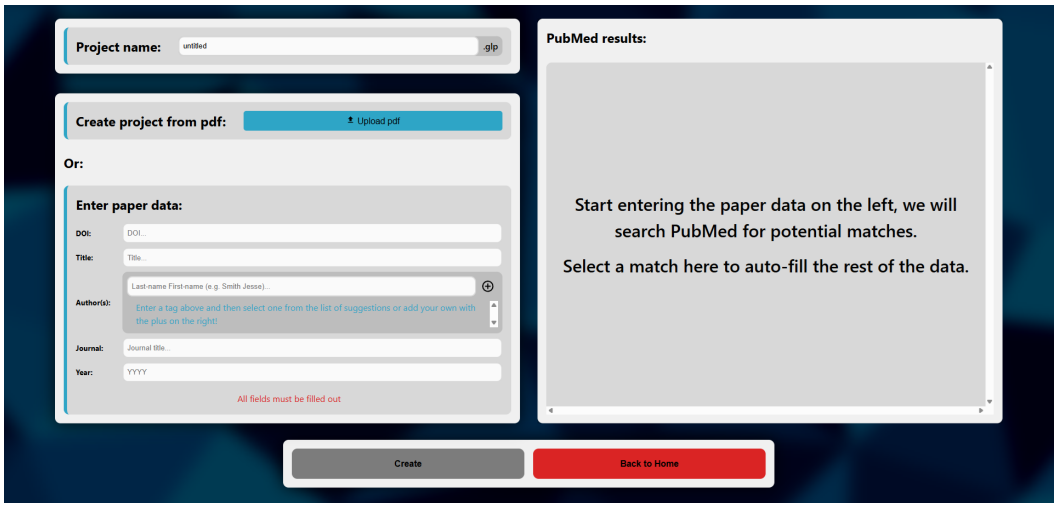


Fig. 5.7.: Old *New Project* page.

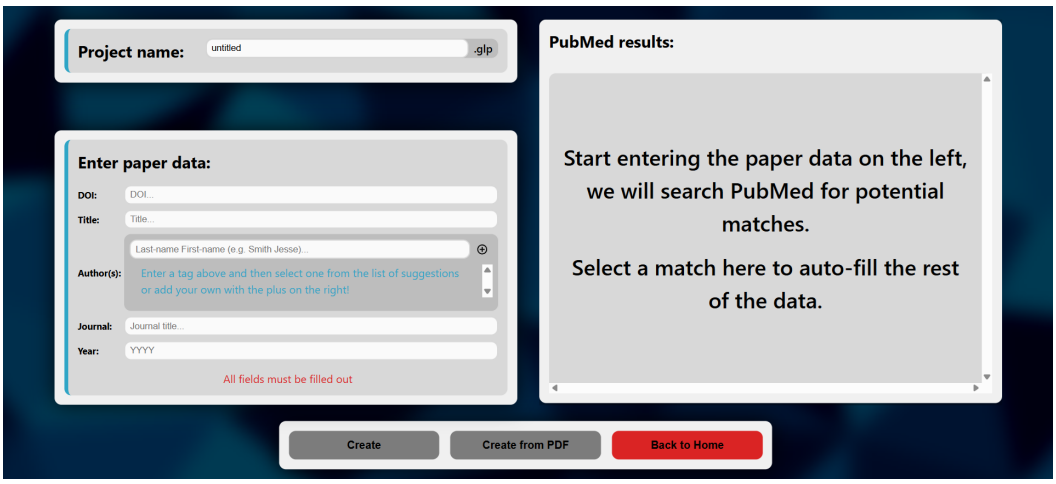


Fig. 5.8.: Modified *New Project* page.

5.2.1 Document Upload Interface

The *PDF Upload* page (Figure 5.9) is organized into two main sections. The left section provides buttons for uploading a PDF file, selecting the desired processing method, and entering an OpenAI API key if required. It also contains action buttons to start processing or cancel the operation. The right section displays a preview of the selected PDF document, allowing users to verify the file before proceeding.

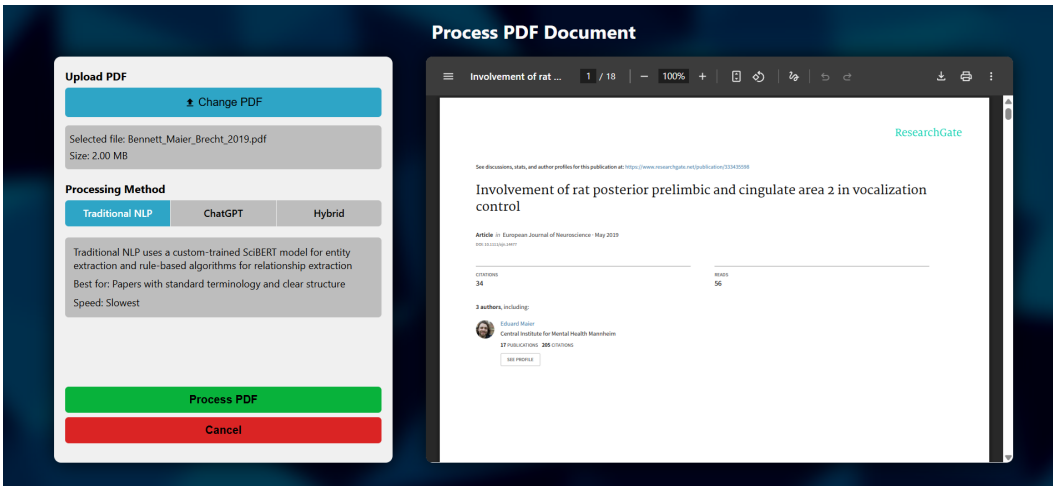
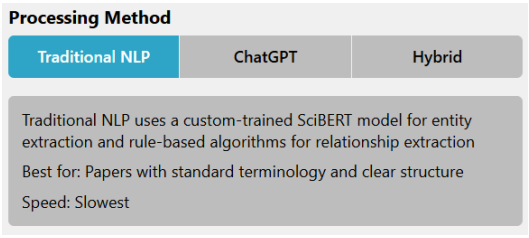


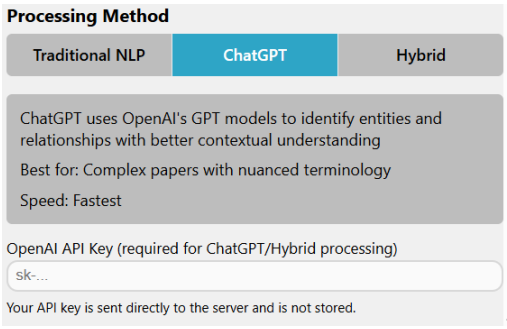
Fig. 5.9.: *PDF Upload* page.

On the left section, the filename and size of the uploaded PDF are displayed explicitly underneath the upload button, allowing users to confirm their selection at a glance. Users can replace the uploaded PDF file by simply reuploading a new PDF file. The processing method selection offers three extraction approaches: Traditional NLP, ChatGPT, and Hybrid (Figure 5.10). Each option has

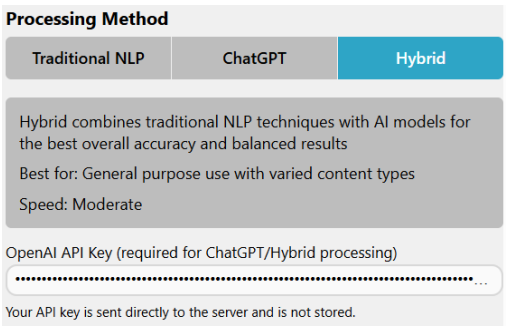
a brief description, recommended use cases, and an indication of processing speed. For the ChatGPT (Figure 5.10b) and Hybrid (Figure 5.10c) methods, users must provide an OpenAI API key to enable AI-powered extraction.



(a) Traditional method.



(b) ChatGPT method.



(c) Hybrid method.

Fig. 5.10.: Selection of processing methods for PDF extraction: (a) Traditional method, (b) ChatGPT method, and (c) Hybrid method.

On the right section, the PDF preview offers essential features of a standard PDF reader, including scrolling, zooming, page navigation, text selection, searching and more.

5.2.2 Error Handling

Once the user initiates PDF processing by clicking the *Process PDF* button, it is no longer possible to cancel the operation. At this point, the web application begins handling errors that may arise during the processing step, providing clear and immediate feedback to the user.

When processing with ChatGPT or Hybrid methods, if the provided API key is invalid or there are usage or billing issues, the page displays an *API Key Error* modal[3] with a detailed description (Figure 5.11). The users have the option to try again with a different key or cancel and return to the *New Project* page.

[3] A modal is a pop-up dialog box that appears in the center of the screen and requires the user to interact with it before continuing with the current process.

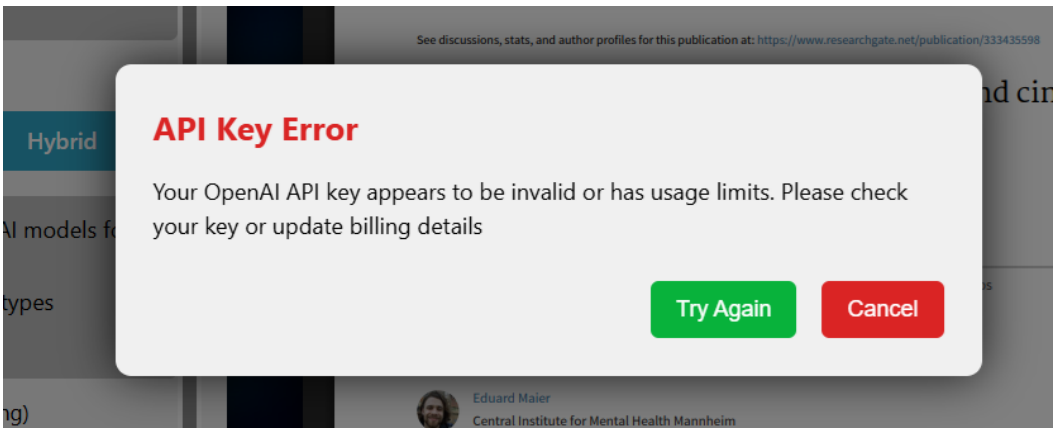


Fig. 5.11.: API key error on PDF upload.

If the backend determines that the uploaded PDF does not contain neuroscience research content based on the response from an LLM prompt, the page displays a *Document Validation Failed* modal (Figure 5.12). This occurs with ChatGPT and Hybrid methods, which use AI-powered technology. The user is presented with options to upload a different PDF or to try traditional processing, allowing them to recover from this error without restarting the entire workflow.

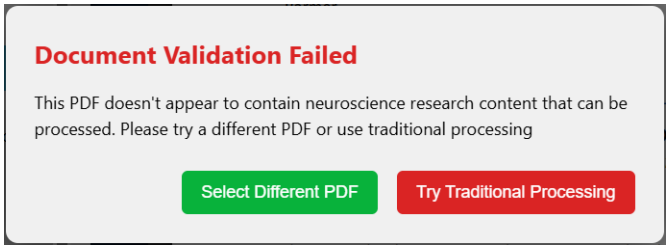


Fig. 5.12.: Document validation failed for ChatGPT/Hybrid methods when the uploaded PDF is not a neuroscience publication.

If the traditional method fails to extract any entities from the document, the page displays the same *Document Validation Failed* modal, but with a different description (Figure 5.13). The user can upload a different PDF or switch to the ChatGPT method.

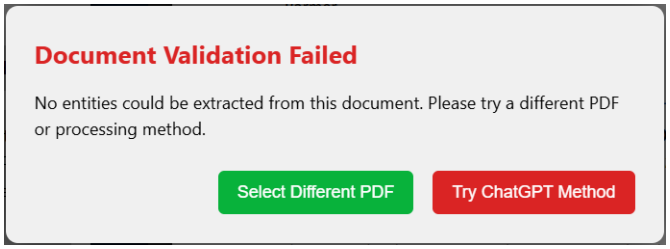


Fig. 5.13.: Document validation failed for traditional method when no entities are extracted from PDF.

Any unexpected server, upload, or processing failures would trigger a *Processing Error* modal. This generic error handling ensures that users are informed of issues not covered by the specific cases above. Users can try the operation again or cancel the process.

Throughout the PDF uploading process, users can upload, preview, and process PDF with clear and robust error handling at every step. This minimizes frustration and guides users toward a successful workflow.

5.3 Entity Review Page

This section demonstrates the *Entity Review* page, a critical step in the document processing workflow. The page enables users to review, curate, and finalize the set of entities extracted from an uploaded neuroscientific PDF document. This interface is designed to maximize transparency, user experience, and data quality before downstream processing.

5.3.1 Extracted Entities Review Interface

Upon successful entity extraction from PDF processing, users are presented with a structured interface that displays all entities automatically extracted from the document in the *Entity Review* page (Figure 5.14). The page is divided into 3 main sections:

- Entity Review Panel (upper left)
- Accepted Entities Panel (lower left)
- PDF Review Panel (right)

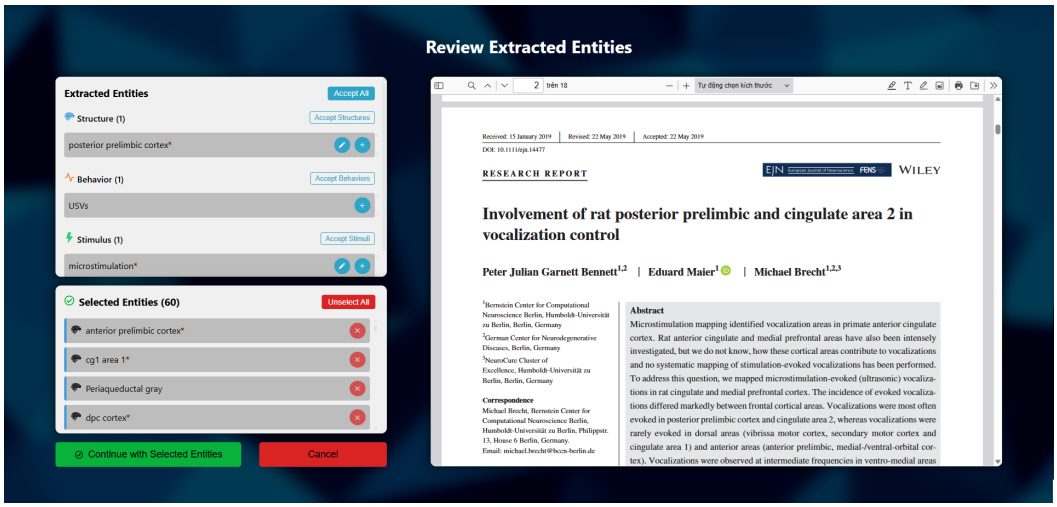


Fig. 5.14.: *Entity Review* page.

Extracted Entities Panel

The *Extracted Entities* panel presents all entities automatically extracted from the uploaded PDF during the previous processing step, which are grouped by type: Brain Structure, Behavior, and Stimulus. Each group is visually distinguished with a unique icon and color for clarity (Figure 5.15).

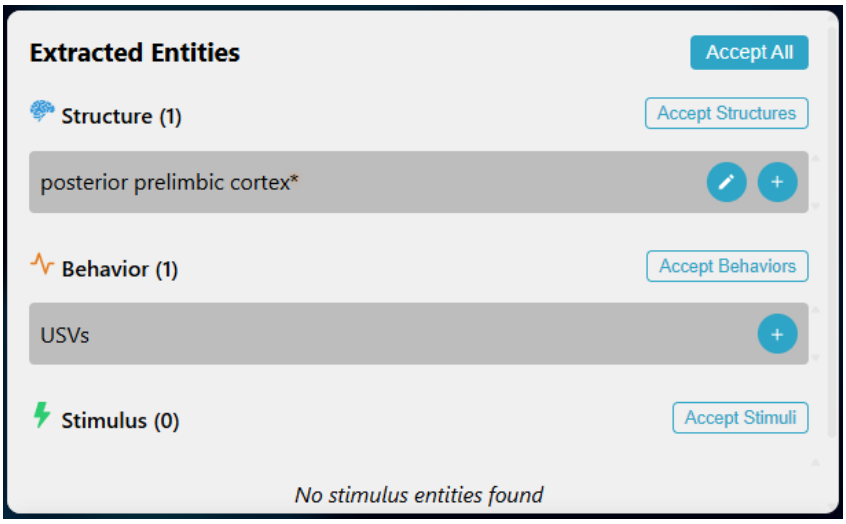


Fig. 5.15.: *Extracted Entities* panel.

Entities not linked to the database are visually indicated by an asterisk *(*)* next to their name. These entries also provide an edit button, enabling users to modify their details prior to acceptance. Brain structures are initially displayed as a flat list upon extraction, since no parent-child relationships are extracted during the automated extraction process. However, the interface supports a hierarchical tree display (Figure 5.16) and users can manually define parent-child relationships by editing entities through the edit modal.

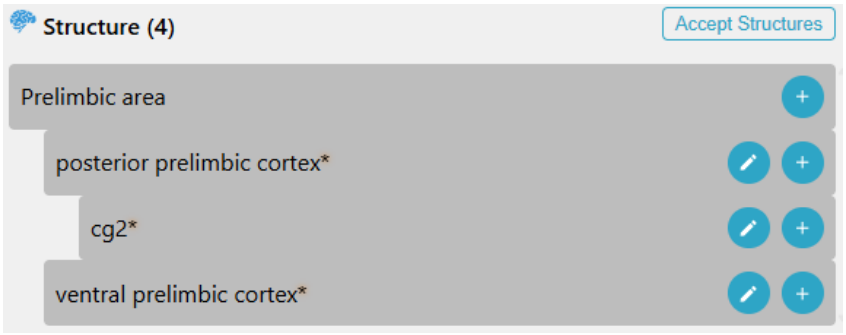


Fig. 5.16.: Example of a brain structure hierarchy.

The number of entities remaining in each category is displayed to provide users with a clear overview of the extraction results. Users can choose to

accept individual entities, accept all entities within a specific category, or use the *Accept All* button to move every entity across all categories at once to the *Accepted Entities* panel. When a brain structure entity with a hierarchical relationship is accepted, all of its descendant entities within the hierarchy are also automatically selected. If no entities of a certain type are left, the panel displays a corresponding message.

Accepted Entities Panel

The *Accepted Entities* panel displays all entities that users have selected for acceptance, along with an indication of the number of accepted entities. Entities have icons and colors for distinction, and an asterisk *(*)* for entities not linked to the database (Figure 5.17). Users can remove individual entities from the selection or unselect all entities at once using the *Unselect All* button. These changes will move the entities back to the *Extracted Entities* panel.

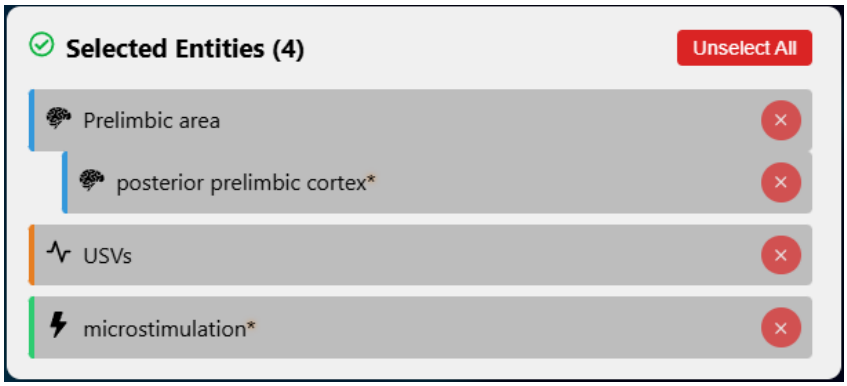


Fig. 5.17.: Example of the *Accepted Entities* panel.

Brain structures with hierarchical relationships are also displayed in a hierarchical tree structure. If a parent brain structure is removed from the *Accepted Entities* panel, all of its descendant entities within the hierarchy are also automatically removed.

PDF Review Panel

The *PDF Review* panel is implemented using `PDF.js` [@14], which provides in-browser rendering, navigation, and text selection for PDF documents, as well as features such as highlighting and searching within the web application.

When users select an entity from either the *Extracted Entities* or *Accepted Entities* panel, it will be visually emphasized with a red color. Simultaneously, all occurrences of the selected entity's name are automatically highlighted within the PDF preview, provided that such occurrences exist (Figure 5.18). If

the entity's name does not appear in the document, no highlighting is applied. Users can also unhighlight entities by clicking on them again.

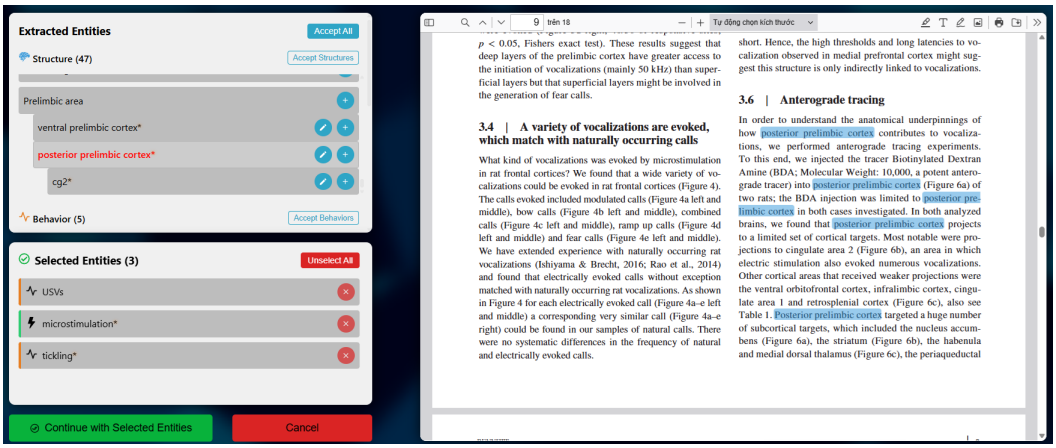
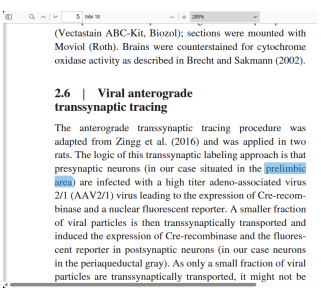
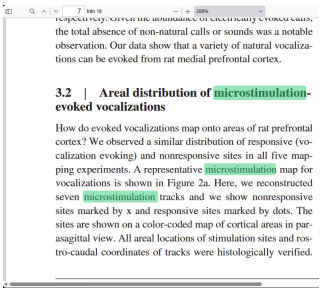


Fig. 5.18.: Highlighted entities in PDF.

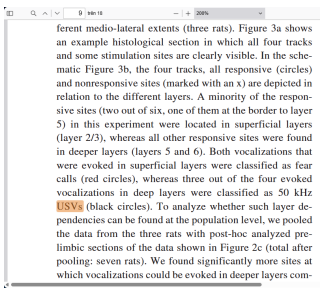
Color-coding is used to visually distinguish between different types of entities in the PDF. Specifically, brain structures are highlighted in blue (Figure 5.19a), stimuli in green (Figure 5.19b), and behaviors in orange (Figure 5.19c). This consistent use of color enhances the clarity of the review process by allowing users to quickly identify each highlighted entity's category.



(a) Blue highlight for brain structures.



(b) Green highlight for stimuli.

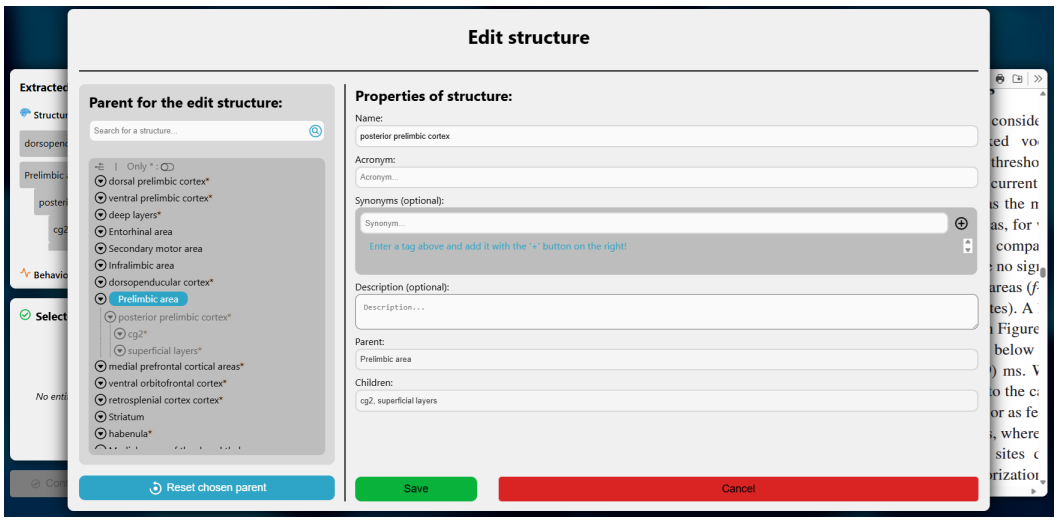


(c) Orange highlight for behaviors.

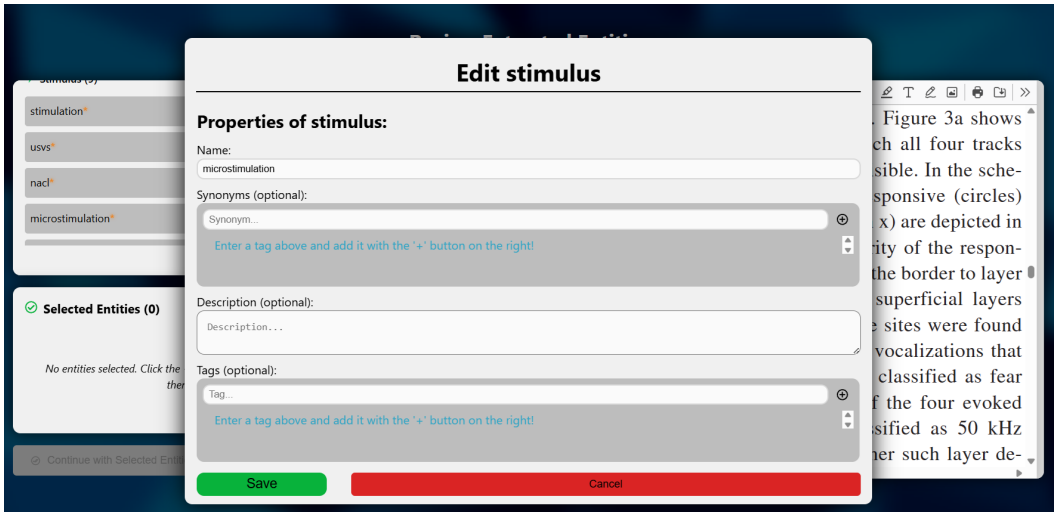
Fig. 5.19.: Color-coded highlights for different entity types in the PDF: (a) Blue for brain structures, (b) Green for stimuli, and (c) Orange for behaviors.

5.3.2 Edit Modal

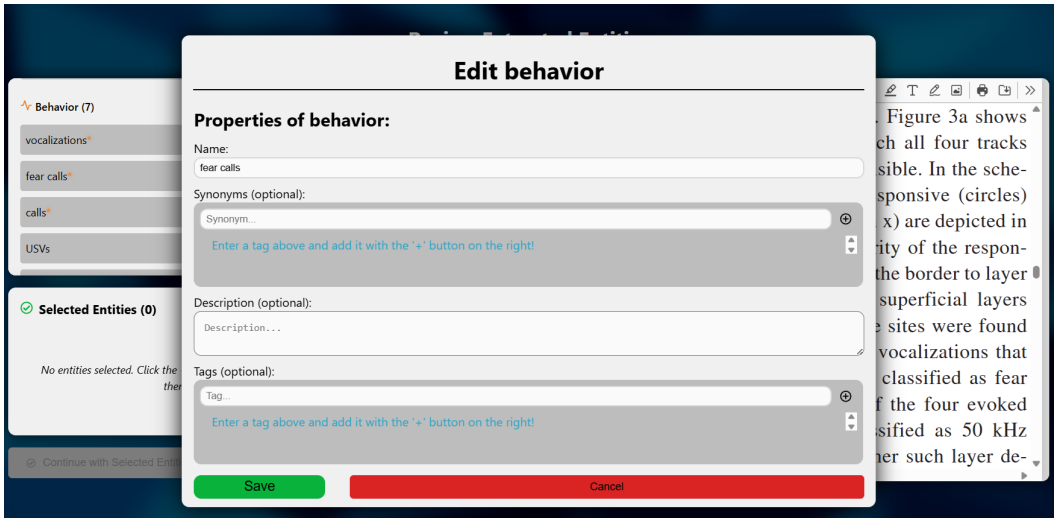
The *Edit* modal for extracted entities closely mirrors the design and functionality of the edit interface found in the *Main Canvas* page, but is specifically tailored to the requirements and workflow of the *Entity Review* page. It provides a comprehensive interface for modifying the properties and hierarchy of brain structures (Figure 5.20a), as well as the properties of stimuli (Figure 5.20b) and behaviors (Figure 5.20c).



(a) *Edit* modal for brain structures.



(b) *Edit* modal for stimuli.



(c) *Edit* modal for behaviors.

Fig. 5.20.: *Edit* modals for different entity types: (a) Brain Structures, (b) Stimuli, and (c) Behaviors.

The interactive hierarchical tree component (Figure 5.21) allows users to view the entire brain structure hierarchy from the *Extracted Entities* panel, highlighted assigned parent, change the parent brain structure of the currently edited node, and reset the chosen parent. The tree disables selection of the current brain structure and its descendants, ensuring only valid parent assignments. Additionally, it provides a search function and a toggle to display only nodes not linked to the database.

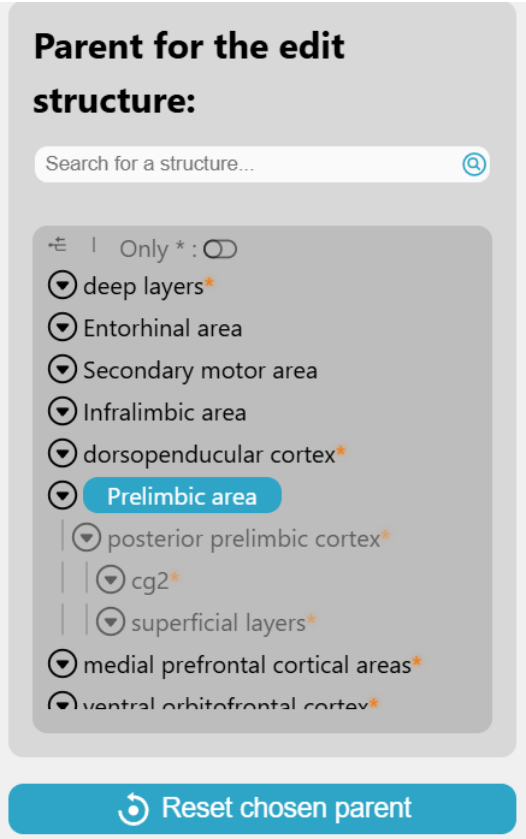


Fig. 5.21.: Parent-child relationship tree for brain structures.

In the edit modal for brain structures, users are able to modify several key properties of a brain structure entity (Figure 5.20a). Specifically, they can update the brain structure's name and acronym, add or remove synonyms using a tag input field, and edit the description in a dedicated text area. For greater clarity, the modal also displays the single assigned parent and children of the edited brain structure in read-only fields, which are immediately updated to reflect any changes made in the hierarchical tree.

For both stimuli and behaviors, users can modify the properties of the entity in a similar manner (Figure 5.20b). Unlike brain structures, stimuli and behaviors do not have hierarchical relationships or acronyms. The available fields include the entity's name, synonyms, description, and relevant tags using a tag input field.

(a) *Properties* panel for brain structures.

(b) *Properties* panel for stimuli and behaviors.

Fig. 5.22.: *Properties* panels for different entity types: (a) Brain Structures and (b) Stimuli and Behaviors.

All changes can be saved or discarded, and the *Extracted Entities* panel dynamically updates to present the hierarchy once these relationships are established. After processing, all action buttons become disabled to prevent further edits, but users can still highlight entities within the PDF for reference. This ensures data consistency and supports an efficient review process.

5.3.3 Error Handling

The *Entity Review* page implements robust error handling to ensure smooth user experience during entity review and processing. Error handling is primarily displayed through error modals, providing clear feedback and recovery options for users.

When users attempt to process and their OpenAI API key has expired or is no longer valid, the page shows the *API Key* modal (Figure 5.23), which is specifically designed to handle errors related to OpenAI API key authentication. This is the most common error encountered during entity processing. The modal prompts the user to enter a new API key, which is then sent to the backend to obtain a fresh processing token. Upon successful entry and validation, the previously failed operation is automatically retried with all changes preserved, allowing the user to continue their workflow with minimal interruption.

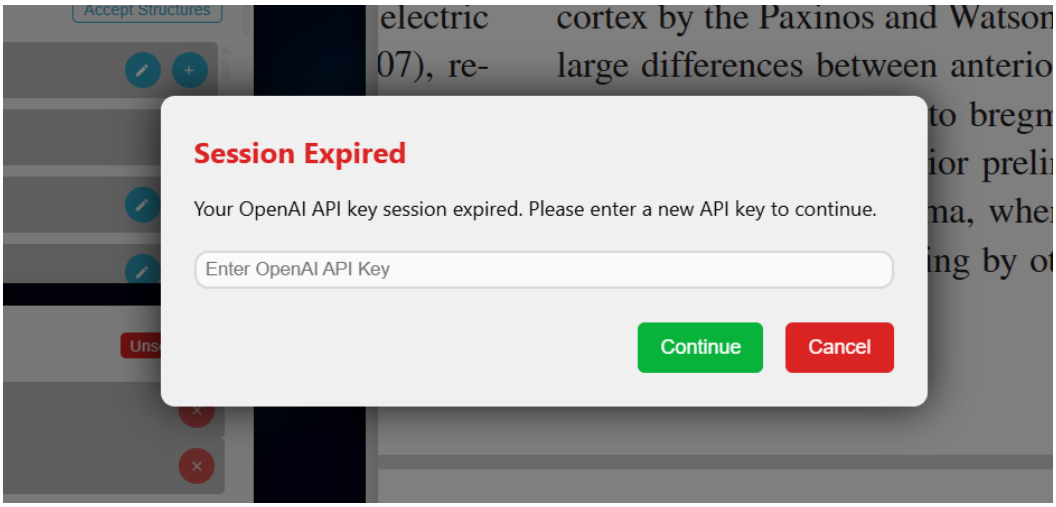


Fig. 5.23.: *API Error* modal.

If errors such as network failures, server errors, or unexpected exceptions occur during processing, the *General Error* modal (Figure 5.24) is displayed. This modal presents the error message to the user and provides the option to cancel the operation and go back to the *New Project* page.

Fig. 5.24.: *General Error* modal.

5.4 Main Canvas Page

This section describes how the results of the processing pipeline are integrated into the main canvas and highlights improvements to the representation and management of entities within the editor tool.

5.4.1 Entity Visualization and Interaction

Entity Visualization

After the pipeline is completed, the resulting project files are automatically loaded and visualized on the main canvas (Figure 5.25). Both the project name and tab name are set to the user-defined project name. The nodes on the canvas are arranged according to the layout and spatial organization described in Section 4.3.2, with newly created nodes that are not yet linked to the database visually marked by an asterisk *(*)* next to their names. This ensures a clear and structured visualization of all entities and their relationships.

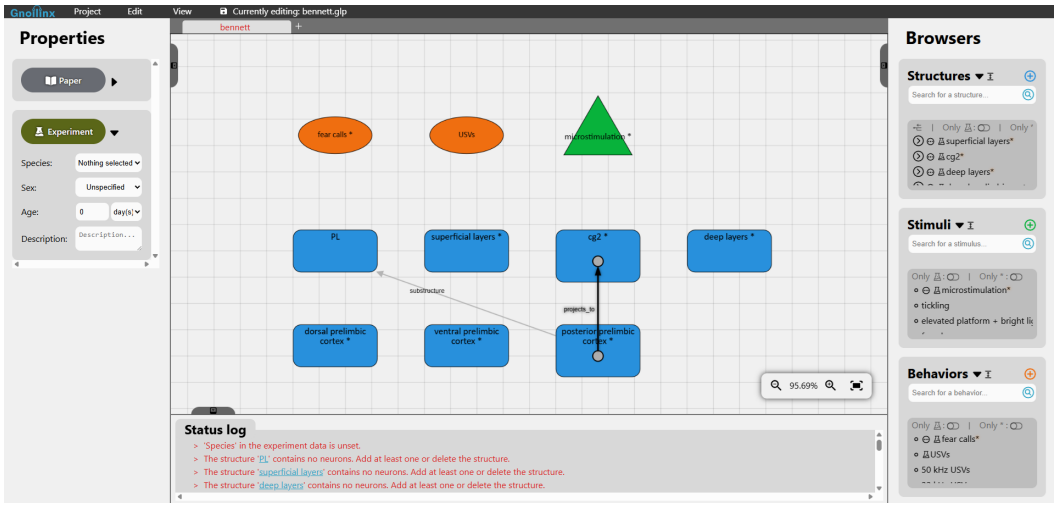


Fig. 5.25.: Resulting graph from pipeline processing in *Main Canvas* Page.

Modified Status Log

Additionally, the status log at the bottom of the canvas now displays information about unlinked entities as green informational messages (Figure 5.26). Together with the asterisk indicator on the canvas, this ensures that users are clearly informed about which entities still need linking, avoiding confusion with error warnings.

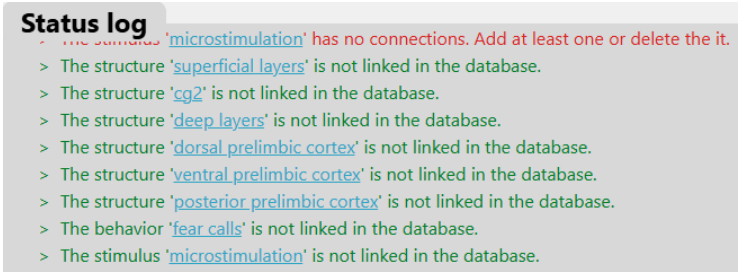
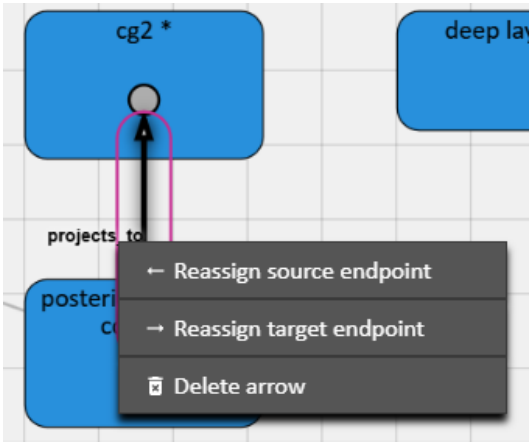


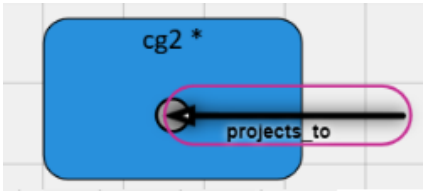
Fig. 5.26.: Modified status log with informational messages.

Node Reassignment

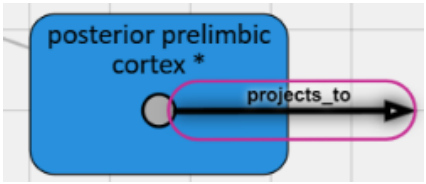
The system now allows users to reassign nodes directly within the canvas. By right-clicking on an arrow relationship, users can select to reassign either the source or target node and then use a simple click-to-move interaction to select a new valid node as defined by the data model (Figure 5.27). This intuitive functionality is particularly useful for large projects with many entities, as it eliminates redundant steps and enables users to quickly rearrange connections.



(a) Reassignment options.



(b) Reassigning source node.



(c) Reassigning target node.

Fig. 5.27.: Node reassignment on canvas: (a) Reassignment options, (b) Reassigning source node, and (c) Reassigning target node.

Edit Brain Structures

Previously, users could only assign parent relationships when creating a new brain structure node. The addition of the hierarchical tree component now allows users to edit existing brain structures and change their parent relationships directly the same way they would create new ones (Figure 5.28). This enhancement provides greater flexibility in managing the organizational hierarchy and makes it easier to update as project requirements evolve.

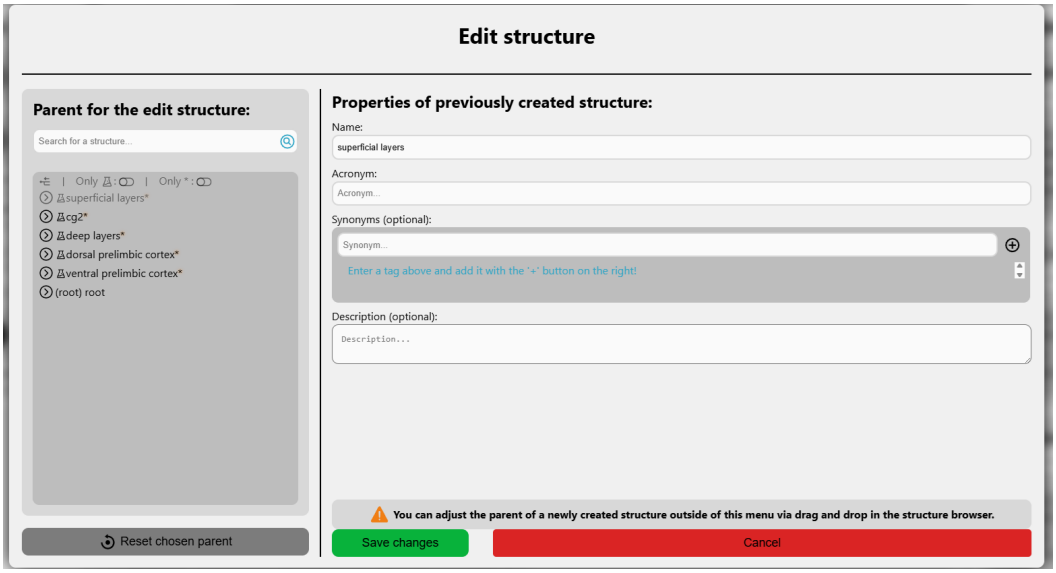


Fig. 5.28.: Editing brain structures in the hierarchical tree component with additional parent assignment.

5.4.2 Usability Enhancement

Shortcuts

The *Main Canvas* page now supports a wide range of keyboard shortcuts for common operations. Users can see which shortcuts are available by hovering over the corresponding buttons in the navigation bar. These shortcuts allow them to perform frequent actions quickly and efficiently, reducing reliance on mouse navigation and making the interaction with the page more productive.

The following operations are available via keyboard shortcuts:

- **Project operations:**
  - **New project**: `Ctrl + Alt + N`
  - **Open project**: `Ctrl + O`
  - **Save project**: `Ctrl + S`
  - **Save as**: `Shift + S`
- **Edit operations:**
  - **Copy**: `Ctrl + C`
  - **Paste**: `Ctrl + V`
  - **Delete selected entity**: `Del`

- **Select all**: `Ctrl + A`
- **Deselect all**: `Esc`
- **Undo**: `Ctrl + Z`
- **Redo**: `Ctrl + Y`

Undo/Redo Functionality

In addition, an advanced undo and redo system has been implemented, providing comprehensive historical context for all changes made within the canvas (Figure 5.29). Every significant modification to the project state is recorded up to 20 latest actions, including actions such as creating, editing, or deleting nodes in the hierarchical component tree, updating neuron and arrow properties, modifying paper metadata, and any changes made on the canvas. This robust feature allows users to easily revert or reapply actions as needed, preventing accidental changes or data loss.

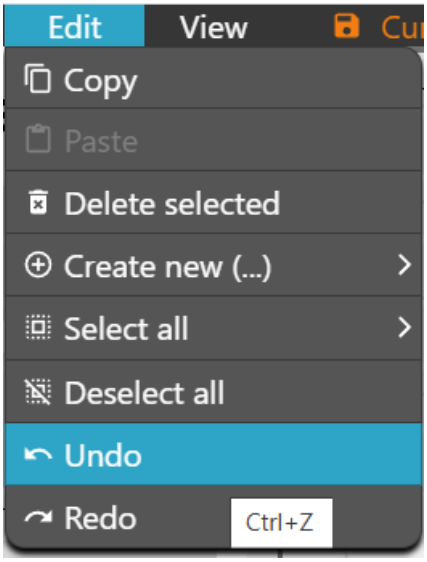


Fig. 5.29.: Undo/Redo functionality in the *Main Canvas* page.

In summary, the chapter demonstrated the user workflows of the end-to-end, semi-automated, and cloud-based knowledge extraction pipeline in the Gnoilinx web application with usability enhancements for improved user experience and efficiency. The extended web application provides a more streamlined and effective platform for neuroscience data, allowing researchers to efficiently extract insights from publications.

6 Conclusion

6.1 Summary

This thesis presents the design and implementation of an end-to-end, semi-automated, and cloud-based pipeline for extracting, organizing, and visualizing knowledge from neuroscientific publications. The developed system integrates advanced Natural Language Processing (NLP) techniques with a modern web-based editor tool, enabling researchers to efficiently convert raw PDF documents into structured, interactive knowledge graphs.

The solution features multi-method extraction, secure cloud-based project management, automated data structuring, user review, and well-organized knowledge graphs. All these components work together to ensure data quality, security, and a smooth workflow within an enhanced, user-friendly web interface.

Through these advancements, the thesis bridges the gap between automated information extraction and interactive data visualization for advanced analysis in neuroscience, significantly reducing manual effort and enabling more efficient, reproducible research workflows.

6.2 Future Work

While this thesis presents an advanced solution for a pipeline that extracts and visualizes neuroscience knowledge, several aspects could be further developed or improved in future work. These include:

- Improving NLP models for better accuracy and domain adaptation to extract more relevant entities and relationships.
- Enhancing NLP models to extract detailed neuron properties from neuroscientific texts.
- Implementing additional extraction methods to provide greater flexibility.
- Implementing automated extraction of metadata (e.g., title, journal, authors, doi, year) from PDF publications.
- Incorporating user feedback to iteratively refine extraction models and adapt to new terminology.

- Integrating with external neuroscience databases and ontologies to enrich extracted knowledge and enable cross-referencing.
- Implementing real-time collaboration and project sharing, allowing multiple users to work on the same project and share knowledge graphs.
- Implementing a mechanism to retain user progress and edits on the canvas page, so that data is not lost when the page is refreshed.
- Improving the performance and scalability of visualization to efficiently handle large and complex experiments.

Bibliography

[1]Huda Akil, Maryann E. Martone, and David C. Van Essen. “Challenges and Opportunities in Mining Neuroscience Data”. In: *Science* 331.6018 (2011), pp. 708–712 (cit. on pp. 1, 3, 4).

[4]Peter Bennett, Eduard Maier, and Michael Brecht. “Involvement of rat posterior prelimbic and cingulate area 2 in vocalization control”. In: *European Journal of Neuroscience* 50.7 (2019), pp. 3164–3180 (cit. on pp. 19, 35).

[7]Samia Mubarika Goraya. “Comparing State-of-the-Art NLP techniques for Information Extraction from Circuit Neuroscience Publications”. Master’s thesis. Johannes Gutenberg University Mainz, Sept. 2024 (cit. on pp. 1, 3–5, 7, 11, 23, 27).

[8]Patrick Götz. “Interactive Graphical User Interfaces for Neuroscience Data”. Bachelor’s thesis. Johannes Gutenberg University Mainz, Apr. 2024 (cit. on pp. 1, 3, 5, 7, 8, 29, 35).

[9]Zhi Hong, Logan Ward, Kyle Chard, et al. “Challenges and Advances in Information Extraction from Scientific Literature: a Review”. In: *JOM* 73 (2021), pp. 3383–3400 (cit. on p. 4).

[10]S. Ishiyama and M. Brecht. “Neural correlates of ticklishness in the rat somatosensory cortex”. In: *Science* 354.6313 (2016), pp. 757–760 (cit. on p. 3).

[11]Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. May 2015 (cit. on p. 19).

[12]David N. Kennedy. “The Dark Matter of the Bibliome”. In: *Neuroinformatics* 13.4 (2015), pp. 387–389 (cit. on pp. 1, 3, 4).

[13]Liqun Luo. “Architectures of neuronal circuits”. In: *Science* 373.6559 (2021), eabg7285 (cit. on p. 3).

[20]L. Schmidt, A. N. Finnerty Mutlu, R. Elmore, et al. “Data extraction methods for systematic review (semi)automation: Update of a living systematic review”. In: *F1000Research* 10 (May 2021), p. 401 (cit. on pp. 1, 4).

[21]Matthew Shardlow, Meihong Ju, Meng Li, et al. “A Text Mining Pipeline Using Active and Deep Learning Aimed at Curating Information in Computational Neuroscience”. In: *Neuroinformatics* 17 (2019), pp. 391–406 (cit. on p. 4).

Webpages

[@2]*Allen Brain Atlas*. 2025. URL: https://portal.brain-map.org/ (visited on Aug. 8, 2025) (cit. on p. 5).

[@3]Dev Balaji. *JWT Authentication in Node.js: A Practical Guide*. 2023. URL: https://dvmhn07.medium.com/jwt-authentication-in-node-js-a-practical-guide-c8ab1b432a49 (visited on Aug. 8, 2025) (cit. on p. 35).

[@5]OWASP Foundation. *JSON Web Token Security Cheat Sheet*. 2023. URL: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html (visited on Aug. 8, 2025) (cit. on p. 19).

[@6]GeeksforGeeks. *Run Python Script from Node.js using Child Process Spawn Method*. 2019. URL: https://www.geeksforgeeks.org/python/run-python-script-node-js-using-child-process-spawn-method/ (visited on Aug. 8, 2025) (cit. on p. 20).

[@14]Mozilla and individual contributors. *PDF.js*. 2025. URL: https://mozilla.github.io/pdf.js/ (visited on Aug. 8, 2025) (cit. on p. 45).

[@15]Inc. npm. *bcrypt*. 2025. URL: https://www.npmjs.com/package/bcrypt (visited on Aug. 8, 2025) (cit. on p. 35).

[@16]Inc. npm. *Multer*. 2025. URL: https://www.npmjs.com/package/multer (visited on Aug. 8, 2025) (cit. on p. 19).

[@17]*OpenAI*. 2025. URL: https://openai.com/ (visited on Aug. 8, 2025) (cit. on p. 12).

[@18]*PubMed*. 2025. URL: https://pubmed.ncbi.nlm.nih.gov/ (visited on Aug. 8, 2025) (cit. on p. 5).

[@19]*ResearchMaps.org*. 2025. URL: https://www.researchmaps.org/ (visited on Aug. 8, 2025) (cit. on p. 5).

[@22]Stanley Ulili. *An Introduction to Python Subprocess*. 2025. URL: https://betterstack.com/community/guides/scaling-python/python-subprocess/ (visited on Aug. 8, 2025) (cit. on p. 21).

List of Figures

List of Listings

Appendix

A

A.1 Extraction Prompt

You will receive a text about systems neuroscience. Your task is to extract **ONLY** these three types of entities that are **explicitly** mentioned in the text, especially those that are often mentioned:

STRUCTURE:

- List the names of the brain structures involved in the main findings of the paper.
- List also the hierarchies between structures (e.g., layer 5 has two distinct substructures L5a and L5b), which means if a structure is a substructure or part of another structure.

STIMULUS: List stimuli (cues given to subject)

BEHAVIOR: List behaviours (internal state or action of subject)

CRITICAL EXTRACTION RULES:

- **EXCLUDE** any entity whose name contains neurons, or refers to a specific cell type or neuronal population.
- Do **NOT** include parenthetical explanations or clarifications in the entity name. Only extract the main name as it appears in the text, without any text in parentheses.
- Keep the name of entities exactly as in the paper (but make sure it is correct) for later lookup.
- Do **NOT** add entities that aren't directly mentioned.
- Do **NOT** include negative findings, cited works, or background information.
- Do **NOT** include methodology equipment or general biological terms unless they're specific structures, stimuli, or behaviors.
- Include each entity only **ONCE** – no duplicates.

Output format:

Brain Structures:

- [structure 1]

- [structure 2]

Stimuli:

- [stimulus 1]

- [stimulus 2]

Behaviors:

- [behavior 1]

- [behavior 2]

If a category has no explicit mentions, list the heading and “- none specified”

A.2 Extract Relationships Prompt

You will receive a text about systems neuroscience and a list of entities extracted from the text.

CRITICAL RULES:

- Use **ONLY** the exact entity names from the Entities List (case-sensitive). Do not abbreviate, change, or add to these names.
- If a neuron/cell-type entity is not in the Entities List but its parent structure is, use the structure name.
- Output relationships **ONLY** where both entities are exact matches from the Entities List.

RELATIONSHIP TYPES: (use ONLY these, with correct node types)

- Structure → Behavior: *evokes, suppresses, necessary_for*
- Behavior → Structure: *excites, inhibits*
- Stimulus → Structure: *excites, inhibits*
- Stimulus → Behavior: *induces, suppresses*
- Structure → Structure: *excites, inhibits, projects_to, is_substructure_of*

FORMATTING:

- Do **NOT** number or group relationships.
- List all relationships under the heading: "*Relationships:*"

- Each relationship must start with a dash (-) and be on its own line.
- Format each relationship as: *Entity A relationship Entity B*
- Do **NOT** end relationships with a period.

EXTRACTION RULES:

- Only extract *is_substructure_of* if **BOTH** entities are labeled "Structure".
- Only extract main experimental findings with explicit evidence.
- Use *projects_to* **ONLY** for neuron populations or structures.
- Do **NOT** infer relationships or use vague terms (e.g., modulates, influences, involved in, etc.).
- Do **NOT** include interpretations, summaries, or results not from the text.

VERIFICATION:

- Both entities are exact, case-sensitive matches from the Entities List.
- Relationship type is appropriate for the entity types.
- There is explicit evidence for this relationship in the text.

Only output relationships that pass all verification checks.

A.3 Tuples Prompt

You will receive a list of entities and relationships. Convert these into unique, deterministic tuples of the form *(Entity 1 name, Entity 2 name, relation)*, preserving anatomical specificity and exact case as provided.

RELATIONSHIP TERMS: Use **ONLY** these: *expresses*, *evokes*, *excites*, *induces*, *inhibits*, *suppresses*, *projects_to*, *necessary_for*, *belongs_to*.

CONNECTION RULES:

- *evokes*: Neuron → Behavior
- *excites*: Neuron/Stimulus/Behavior → Neuron
- *inhibits*: Stimulus/Behavior/Neuron → Neuron
- *induces*: Stimulus → Behavior
- *suppresses*: Stimulus/Neuron → Behavior
- *necessary_for*: Neuron → Behavior
- *is_substructure_of*: Structure → Structure
- *belongs_to*: Neuron → Structure

- *projects_to*: Neuron → Neuron

SPLITTING & ANATOMICAL SPECIFICITY:

- Break complex sentences into atomic triples.
- For stimulation effects, create separate tuples for site activation and behavioral outcomes.
- For neural projections, always connect neurons to neurons, not structures.
- For structures with anatomical qualifiers (e.g., anterior), always generate *is_substructure_of* relationships to the parent structure (by removing the qualifier).
- Always use the exact entity names as provided.

NEURON HANDLING:

- For any structure-to-structure relationship found (except *belongs_to*, *is_substructure_of*), convert to neuron-to-neuron and add *belongs_to* tuples for both structures.
- For example, "A projects_to B" becomes:
  (A neurons, B neurons, projects_to)
  (A neurons, A, belongs_to)
  (B neurons, B, belongs_to)

VALIDATION & CONSISTENCY:

- Each tuple must use an approved relationship term.
- Each tuple must be unique.
- Ensure all tuples accurately represent the original relationships and preserve anatomical specificity.
- Always use consistent entity names and relationship directions.

FORMAT:

- Write each tuple on a new line, in the format: *(entity 1 name, entity 2 name, relation)*
- Do not include extra parentheses, quotes, numbering, or headings.
- Preserve the exact case of entity names.

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Adaptations to the style of the Institute of Computer Science can be found at `https://gitlab.rlp.net/institut-fur-informatik/cleanthesis-jgu`.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe (dazu zählen auch „KI-Werkzeuge“). Sämtliche wörtlichen und sinngemäßen Übernahmen und Zitate sind kenntlich gemacht und nachgewiesen. Auch beim Einsatz von KI-Werkzeugen[1] stellt die Arbeit klar dar, in welchem Umfang diese genutzt wurden und welche Inhalte dadurch erzeugt oder beeinflusst wurden[2]. Ich versichere, dass ich keine Hilfsmittel verwendet habe, deren Nutzung die Prüferinnen und Prüfer explizit ausgeschlossen haben.

Mit Abgabe der vorliegenden Leistung übernehme ich die Verantwortung für das eingereichte Gesamtprodukt. Ich verantworte damit auch alle KI-generierten Inhalte, die ich in meine Arbeit übernommen habe. Die Richtigkeit übernommener (KI-generierter) Aussagen und Inhalte habe ich nach bestem Wissen und Gewissen geprüft.

Ich habe die Arbeit nicht zum Erwerb eines anderen Leistungsnachweises in gleicher oder ähnlicher Form eingereicht. Von der Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten habe ich Kenntnis genommen.

Mir ist bekannt, dass ein Verstoß gegen die genannten Punkte prüfungsrechtliche Konsequenzen hat und insbesondere dazu führen kann, dass die Studien- und Prüfungsleistung als mit „nicht bestanden“ bewertet wird. Die Einschreibung kann für bis zu zwei Jahre widerrufen werden, wenn Studierende zweimal oder häufiger bei Prüfungsleistungen täuschen (§ 69 Abs. 4 und 5 HochSchG).

Mainz, August 27, 2025

Anh Viet Ngo

[1] Mit „KI-Werkzeugen“ werden computergestützte Systeme bezeichnet, die auf Basis maschinellen Lernens neue Inhalte generieren können (z. B. ChatGPT, Gemini, Claude, LLaMA, Midjourney, Stable Diffusion o. ä.).

[2] Die Nutzung von KI-Werkzeugen muss dann kenntlich gemacht werden, wenn Sie den Kern der Aufgabenstellung in von den Prüfern nicht anzunehmender Weise berührt: Sollte nichts anderes explizit mit den Prüferinnen und Prüfern vereinbart worden sein, so ist dies der Fall, sobald wesentliche Teile der eingereichten Arbeit (z. B. ganze Sätze des Textes, ganze Abbildungen, nicht-offensichtliche Teile von Rechnungen oder Beweisen, mehrere Zeilen von Programmcode am Stück) von solchen Werkzeugen generiert wurden und wörtlich oder in abgewandelter Form in die Arbeit übernommen wurden oder wesentliche Konzepte oder Ideen (z. B. eine Gliederung der Literatur oder ein Lösungsansatz für ein Problem) von KI-Werkzeugen generiert wurden.

Hinweis: Es wird im Falle von abweichenden Vereinbarungen mit den Prüferinnen und Prüfern empfohlen, jene im Vorfeld schriftlich festzuhalten und zusätzlich auch noch einmal in der eingereichten Arbeit zu benennen.

Use of AI tools.

AI Tool	Used for	Reason	When
Grammarly	Grammar, spelling, word choice, sentence fluency	Ensure proper grammar, correct spelling and smoother, more natural sentences.	Revision stage