# Integrating Graph and Large Language Model with AOP-Wiki for Contextual and Semantic Parsing of Adverse Outcome Pathway Information

**OR**

AOP-Wiki Explorer: A third-party tool for the scientific community to search complex queries related to Adverse Outcome Pathway (AOP)

Saurav Kumar1,2, Deepika Deepika1,2, Vikas Kumar1,2,3\*

*1Environmental Analysis and Management Group, Chemical Engineering Department, Universitat Rovira I Virgili, Tarragona, Spain.*

*2IISPV, Hospital Universitari Sant Joan de Reus, Universitat Rovira I Virgili, Reus, Spain*

*German Federal Institute for Risk Assessment (BfR), Berlin, Germany*

## Abstract

Adverse Outcome Pathways (AOPs) provide the basis for non-animal testing, outlining the series of molecular and cellular events initiated with exposure to stressors and leads to adverse effects. Currently, AOPs are considered an integral part of the New Approach Methodology (NAMs) to aid in human health risk assessment. Over the last few years, the scientific community has shown immense interest in developing AOPs with crowdsourcing, which are being archived in the AOP wiki. AOP wiki is a centralized repository coordinated by the OECD consisting of around 512 AOPs with a few endorsed and others currently in the development stage. However, the AOP-wiki platform currently lacks the capability to provide a versatile querying system, that empowers developers to inquire into and analyse the AOPs network in a flexible manner. Therefore, the objective of this work is to leverage the full potential of the AOP wiki archive by adapting its data into a graph database and simultaneously providing a natural language interface for querying.

On top

A large language model (LLM) was incorporated into the AOP-wiki explorer so that users can query in natural language and further networks can be generated.

A case study was taken with three levels of use case scenarios (simple, moderate, and complex query) for evaluating the prediction power of the new tool.

We observed that the tool can generate the network based on the query taking data from multiple AOPs. Another advantage of this tool is that it can save your previous query and then generate stepwise results.

This can help toxicologists to explore AOPs ake complex networks taking data from multiple AOPs simultaneously.

Overall, this tool has a huge potential to explore the AOP wiki and reduce the time and resources utilized to search the vast database.

This tool is freely available on GitHub for wider community usability and further enhancement.

Keywords: Adverse outcome pathway, network analysis, large language model,

## Introduction

The concept of Adverse Outcome Pathways (AOPs) was laid in 2010 by Ankely et.al to streamline the idea of Next Generation Risk Assessment (NGRA). AOPs provide the abstract representation of cascade events initiated by the perturbation of stressors at the molecular and cellular level. Events in AOPs are broadly categorized into three categories i.e., molecular initiating events (MIEs), key events (KEs), and adverse outcomes (AO) with KEs being connected to form a key event relationship (KERs). AOP also contributes towardsthe development of new approach methodologies (NAMs) by providing mechanistic information related to perturbation at biological levels making them extremely important for 21st century toxicology.

Over the years, scientific experts have developed multiple AOPs covering fields like immune-, neuro-, reproductive- toxicology etc. which are being stored in AOP Knowledgebase (AOP-KB). AOP-KB stores machine-readable textual information in MySQL database according to the current data model and complied in the format of XML mark-up language. The information has been stored in AOP wiki which is hosted as a central repository for all AOPs developed as part of the OECD AOP development Programme. AOP-Wiki provides a platform to crowd-source and organize available knowledge and also provides read/write access to AOP-KB in accordance with OECD EAGMST guidelines. To assist the AOP-wiki, various third-party tools have been developed to enrich and support the AOP development or search.

A third-party tool called AOP wiki RDF was developed by Martens et al. where AOP data was converted into RDF format with the help of standard ontology to make it machine-readable. This RDF data can be queried by any user using SPARQL (SPARQL Protocol and RDF Query Language), which allows to search, filter and extract desired information. The challenging part is that querying with SPARQL requires remembering the exact identifier of the nodes and also complex query nature of SPARQL make it difficult for biologist to use especially the ones with non-coding background. The tabular response generated using such query make it difficult to understand the interconnections between different KEs and generate a network. Another idea can be utilizing graph algorithms with natural language processing to create an easy-to-use user interface.

If analyzed deeply, AOP in itself is a graph as it represents the cascade of events in the form of nodes (K.E) connected by edges (KER). Having AOP data not being adapted into a “queryable graph”, limits its usability and ease with which users can interact and retrieve desired information. Another additional point is that converting the AOP is graph database can also assist in fairification process. The concept of FAIR data i.e., Findable, accessible, Interoperable, and reusable data becomes crucial if data is being shared, integrated, and utilized across multiple disciplines of scientific research. Catia et.al works on the fairification in the AOPs, by dividing the key event term of AOP into three sub-terms i.e., process, object, and action each assigned with respective ontologies. Such kind of work can help in making AOPs more machine readable and acceptable by regulatory bodies with both fairification and graph database.

The objective of this work was to develop a graph database for AOPs to capture the multifaceted relationship that underlies within AOPs, while efficiently exploiting the wealth of unstructured information associated with each AOPs. Crafting queries either in SPARQL or Cypher (graph) for data retrieval is a challenge, particularly for non-technical users. To make the interface user friendly, we integrated a powerful Large Language Model (LLM) which can help in generating queries from natural language queries. The tool developed using graph database and LLM can simplifying query composition and also the interpretation of extracted data in an interactive network to simplify AOPs network analysis. This tool can be highly helpful for AOP developer’s community simplifying the search and reducing time required to create networks.

## Methods

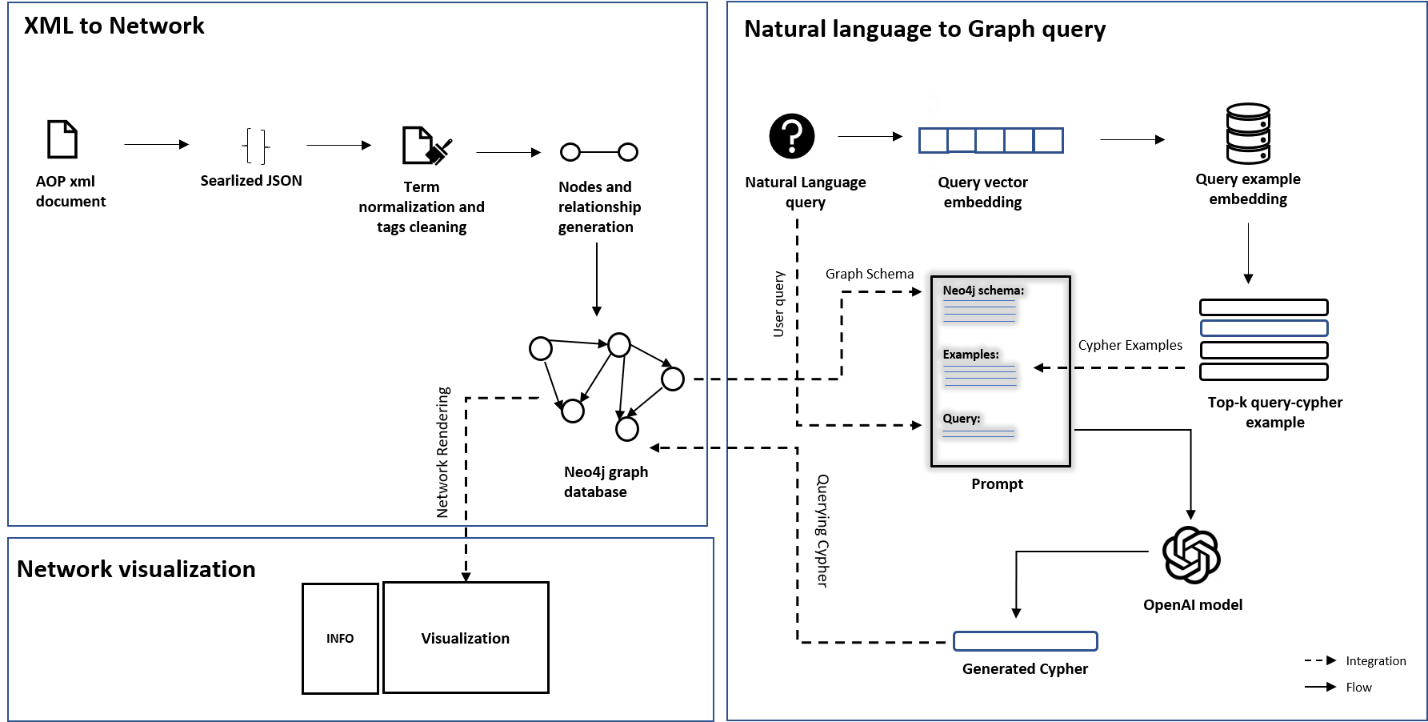


Figure : Overall workflow showing a) XML to graph conversion, b) natural language to graph query and c) network visualization using Neo4J graph database and natural language query prompt.

The workflow for developing AOP wiki explorer was divided into 3 steps:

1. XML to network conversion
2. Natural language to graph query
3. network visualization

All three components seamlessly interconnect to build a cohesive tool. In this interconnection the schema of the generated graph database in step 1 is useful for prompt generation and finally, the visualization component, which renders the network as the graph or natural language query.

### XML to network conversion

XML version of the latest AOP data released in April 2023 was downloaded from <https://aopwiki.org/donwloads>. This data get updated quarterly in a year. XML file was then parsed into a Python dictionary object using the xmltodict library (ref). The converted dictionary object consists of elements, such as AOP, key-event, key-event-relationship, chemical, stressor, and taxonomy which are the building block of the whole AOP network. These data elements hold unique IDs, which helps in referencing and restructuring the block to develop the whole AOP from scratch. For building a queryable graph database, the Neo4j platform had been used (ref).

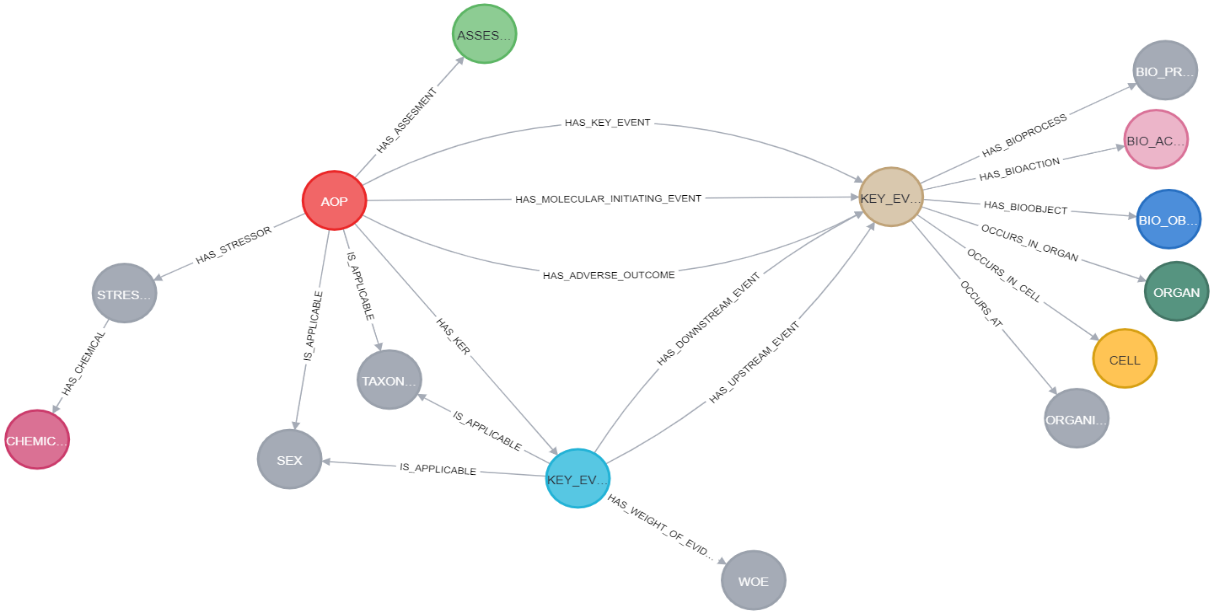


Figure . Neo4j graph model schema for AOP data. The schema contains key events, sex, stressors, cell, organ, taxonomy, and other blocks for AOP network.

Neo4j is a widely used graph database management system, designed to store, manage, and query large amounts of data organized in graph structure. Neo4j relies on cypher query language to interact with graph data. It is designed to express simple to complex queries, that can transverse and retrieve the relationship within the graph. As Neo4j, at its core uses Java to run the engine, to make it accessible programmatically Py2neo had used. Py2neo is a Python library, which acts as an interface between Python and Neo4j and makes programmatic access to neo4j with ease. With Py2neo, nodes, and relationships are built while keeping a consistent naming convention of node and edge labels and their properties. Nodes and edge labels are all in uppercase and the labels having more than one word are joined with snake naming convention. Properties of nodes and edges are written in lowercase. The strict naming convention provides robustness to the graph data model.

**## More to add about the duplication and normalization of different properties-**

The schema of the graph in Figure 1. provides the abstract view of different nodes and their relationship with each other. Typically, in AOP graphical representation, there are mainly three types of nodes i.e. AOP, key event, and Key event relationship present. In this schema, it has been extended to capture information like taxonomy, sex, biological organization level, assessment methodology, and many more. Also, while populating the properties of nodes, textual information was processed using regex to clean the HTML tags to make the text human readable. The extended schema gives flexibility to the user to retrieve different aspects of AOP to put in their analysis. Some of the crucial textual information of AOP such as their assessment methodology and weight of evidence are also represented as nodes, so it will be helpful in context-based queries.

|  |  |  |
| --- | --- | --- |
| S.NO | NODES | PROPERTIES |
| 1. | AOP | 1. ID 2. AUTHORS 3. NAME 4. SHORT\_NAME 5. ABSTRACT 6. POTENTIAL\_APPLICATIONS 7. ESSENTIALITY-SUPPORT 8. AUTHORS 9. REFRENCES |
| 2. | KEY\_EVENT | 1. ID 2. NAME 3. SHORT\_NAME 4. DESCRIPTION 5. MEASUREMENT\_METHODOLOGY 6. URL 7. REFRENCES |
| 3. | KEY\_EVENT\_RELATIONSHIP | 1. NAME 2. QUANTITATIVE UNDERSTANDING 3. EVIDENCE SUPPORTING TAXONOMIC APPLICABILITY 4. REFRENCES |
| 4. | BIO\_ACTION |  |
| 5. | BIO\_OBJECT |  |
| 6. | BIO\_PROCESS |  |
| 7. | STRESSORS |  |
| 8. | CHEMICAL |  |
| 9. | ORGAN |  |
| 10. | ORGANIZATION\_LEVEL |  |

Table . Description of AOP nodes and their relations

In the AOP wiki portal, each aop has brief information contained in the abstracts, description, and detailed information about KE, and KER in specific sections of that AOP. Sections containing descriptive textual information, such as abstract, description, the weight of evidence, and assessment method methodology, etc. have been combined into a single document. Over the combined document Named Entity Recognition (NER) was applied to extract the biological concepts such as gene, protein, chemical, and disease using BERN2. BERN2 is a neural biomedical named entity recognition and normalization tool developed by dmis-lab, south Korea. The extracted entities were normalized using glida grounder, to bring the entity name in grounded form using ontology assistance. Normalized entities were converted into nodes and integrated into the network.

To make the AOPs network, contextually queryable, textual embedding was implemented. Embedding is a fundamental concept of natural language processing and machine learning. Embedding represents words and sentences as a dense vector in a continuous vector space. Vector embeddings transform high-dimensional discrete data (words and phrases) into a lower-dimensional continuous vector while preserving the meaningful relationship. Hence it allows algorithms to capture the semantic, syntactic, and contextual relation between words and sentences.

Textual information of three foundational elements of AOPs i.e. AOP, KE, and KER has been considered for embedding generation. For AOP embedding, AOP’s name, abstract, potential application, background, and short name were taken. While merging information of aop as mentioned above, pre-text such as “AOP with id”, “the abstract of aop”,” title of aop” etc. was pre-appended to keep information descriptive. Similarly, for KE embedding generation, its name, short name, description, measurement methodology, and evidence supporting taxonomic applicability were taken, which were merged together with pre-text. For KER, its weight of evidence contains crucial information, hence the weight of the evidence node, which was connected directly to KER was also merged with the properties of KER to generate embeddable text. The openAI’s most capable embedding model, “text-embedding-ada-002” was used to generate the embedded context of textual content of AOP, KE and KER. The text-embedding model has a context input length of up to 8192 tokens, and returns reduced embedding with 1536 dimensions. The embedded content is matched with a context-based query asked by the user using the Graph Data Science Library of Neo4j. The cosine similarity method was used to calculate the closeness of query and context, on the basis of the similarity score, the top matching component asked by the user is returned.

#### Natural Language to graph query

To bridge the technical gap and provide flexibility to query information in natural language form, OpenAI’s GPT-4 model was used. OpenAI’s GPT-4 is a large language model designed to understand and generate human-like text. The GPT-4 model is trained on an extensive and varied corpus of text derived from the internet, encompassing sources like books, articles, websites, documentation, and code repositories. The training on such a massive amount of data enables the model to learn patterns, syntax, semantics and contextual nuances. GPT-4 model inherently does not possess specific knowledge about Cypher queries, instead, they acquire knowledge through extensive training data or prompts they are exposed to. Description and examples of cypher query provided in the prompt enable the model to generate coherent patterns. However, the response from the model is based on statistical patterns, they have learned while training, rather than a deep understanding of the Neo4j cypher.

The quality and correctness of generated cypher queries depend on how precise the prompt is provided. Precise prompt here signifies explicit instruction, relevant context, illustrative examples of desired response and clear specification of the output format. To build precise prompts, in this work dynamic prompt generation methodology has been adopted. In dynamic methodology, a few examples in the prompt are provided similar to the query. Similar examples make the model well aware of the syntax and variables to be used for generating the cypher. The variable part in the dynamic prompt is their provided example query and query itself, whereas other components are static. Static components of prompts include graph schema, context and output format. Graph schema holds information about how the database is structured and what are properties mentioned in nodes and edges. Context provides precise instruction to the model, while output format provides schema such as JSON, and XML to generate parseable output.

**## The overall flow of prompt generation**

In this work, the prompt is being updated as per the user’s query. Each time the query asked by the user is embedded with a sentence transformer model for similarity matching with already indexed examples in the database.

### Interactive network visualization

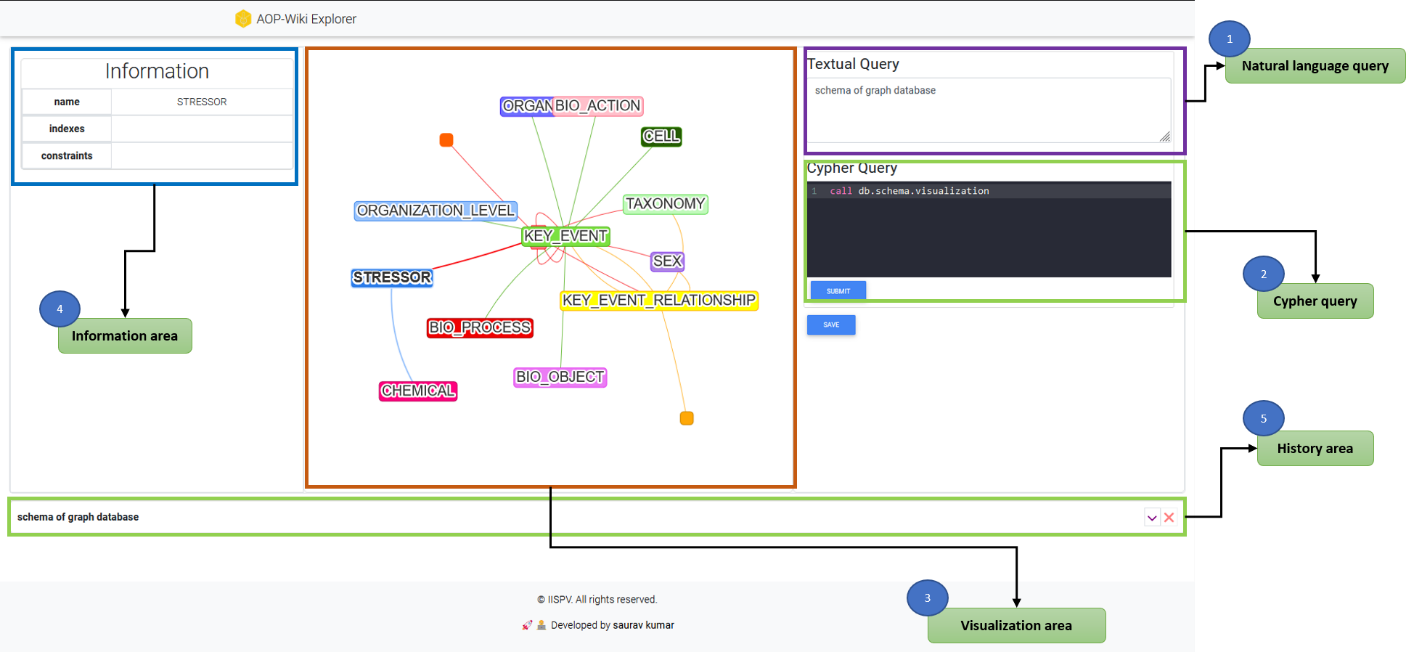


Figure . AOP-Wiki Explorer user interface. The AOP Wiki Explorer user interface provides an interactive playground to get graphical insights into AOP. The user interface rationally designs into 4 components i.e. 1.) Natural Language query area, 2.) Cypher query area 3.) Visualization area 4.) Information area and 5.) History area

AOP-wiki Explorer comes with an interactive user interface, The interface is thoughtfully structured into five distinct components, and each component plays a pivotal role in enhancing user interaction and understanding. These components orchestrate in a harmonious fashion and provide users with an adaptable playground to analyze and design AOPs. The components and their role are as follows:

**Natural language query:** It takes user-initiated queries in natural language form. Input to this component is mandatory as it helps to keep track of human-understandable queries. Along with this, it helps to generate cypher from natural language.

**Cypher query:** Takes direct Neo4j cypher query as input. Here users with technical knowledge can craft intricate cypher queries, and able to leverage the full potential of a graph database to extract precise insights and patterns. Syntax highlighting is supported by the cypher which provides readability to the code.

**Visualization Area:** It’s a visual output component that renders the pattern given by the cypher query. Visual representation empowers the users to unravel complex relationships, dependencies, and trends within AOPs and helps developers to foster informed decision-making. Visualization is powered by the Neovis library, which is a part of the Vis JavaScript library specialized in handling Neo4j queries.

**Information Area:** It provides comprehensive details stored as properties in nodes and edges. When tapped over the graph component in the interactive visualization, it shows the tabular list of properties and URLs to navigate the source of origin i.e. AOP-wiki.

**History Area:** This component allows users to retrace their steps and revisit previous interactions. This feature facilitates iterative exploration and dynamic data investigation.

The processing of raw data to the network has been done with Python3. Neo4J version \_ has been used as the graph database platform. Interactive visualization has been developed with React, a JavaScript framework for UI development. Along with this other JavaScript library such as Neovis.js, codeMirror.js etc. has been used. The backend has developed with a Flask micro server in Python.

## Result and Discussion

### Adopting AOP-wiki XML as a Graph database

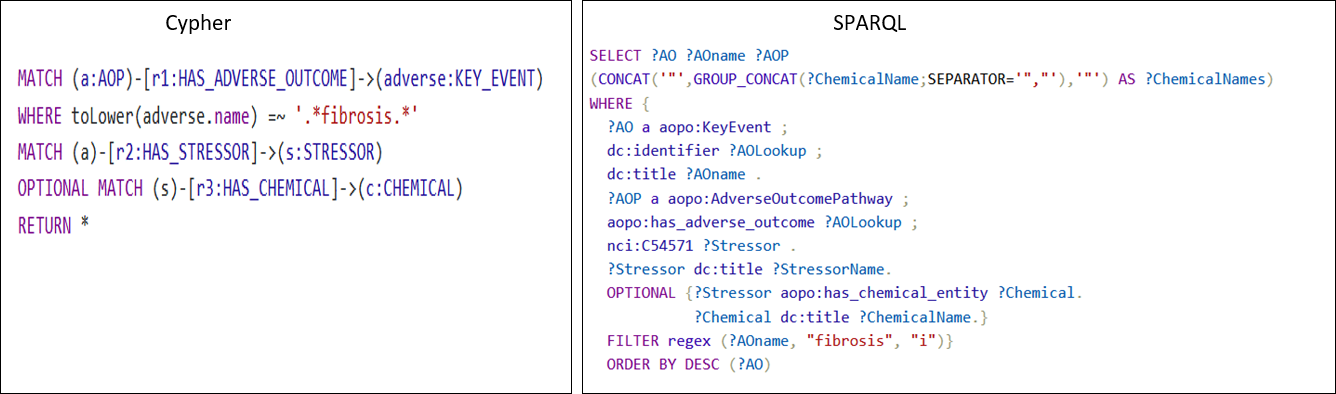
This work adept AOP-Wiki XML information into a graph data structure. Transforming XML data to graphs enables the researchers to query AOPs in their more natural way. Along with this AOP wiki information was further enriched with information about genes/proteins, chemicals and diseases mentioned in each AOP using NER. A total of unique 1138 biological entities were identified, and out of that 439 were gene/protein, 378 chemicals and 321 diseases. Each of the entities was attached to their respective ontology such as CHEBI is associated with chemicals, MIM is related to disease, NCBIGene for genes and MESH is a general purpose which covers multiple domains. With ontology association, the problem of duplication of the same entity with a synonym has been resolved. The enrichment allows users to search AOPs with the very fundamental information of their work such as genes or chemicals associated with them and gives more flexibility to information retrieval. The enriched information is available as supplementary information. To make the AOP graph data free of redundant information and more FAIR, information such as life stages and taxonomy has been filtered and attached to its consistent identifier. Out of the 23 identified life stages, redundant ones like "Adults" and "Adult," as well as "Foetal" and "Fetal," have been filtered and attached to their permanent identifier. Similarly, lots of other terms like “Adult, reproductively mature”, “1 to < 3 months” and many more were carefully assigned with specific ontology and definition. FAIRified life stages are available as supplementary.

## How the better-verified data will enable

## Discuss how the schema of the graph is designed like that (Add this information in supplementary)

## Finally, as discussion write about, how a graph can handle a surge number of nodes and edges

### Addressing query crafting with LLM integration

Figure 4. Query complexity comparison between cypher and SPARQL. Both queries retrieve the same information about the chemicals which leads to adverse outcomes of fibrosis.

Each database comes up with its own syntax or query language to facilitate efficient information retrieval. SPARQL Protocol and RDF Query LanguageSQL (SPARQL) are used for managing and retrieval from databases mapped in RDF format. Similarly, the cypher query is used with Neo4j graph databases. In terms of complexity, the neo4j query is far less complex and more intuitive than the SPARQL query. As an example, to retrieve the chemicals which is related to “fibrosis” as an adverse outcome, the syntax mentioned in Fig 4. is used, to retrieve the same information with both the cypher and SPARQL query. The cypher query pattern resembles to graph, where the parenthesis “()” represents the node, while “-[]->” represents the edge. Filtering the nodes and relationships on the basis of certain criteria can be done using the WHERE clause or directly imputing property as key and value in the nodes and relationship. Static clauses used in cypher and SPARQL both are comprehensible and represent the intended process.

|  |  |
| --- | --- |
| **Natural Language Query** | **Visual representation of network generated from query** |
| **Simple:** Stressors related to AOP on learning and memory impairment. |  |
| **Moderate:** Provide me with a network of AOPs related to neurotoxicity and connected to key events mentioning calcium influx. |  |
| **Complex:** key events which lie in the shortest path connecting AOPs related to neurotoxicity. |  |

Table Different variations of query simple, moderate and complex are shown in natural language and cypher format. A graphical representation of the query is present in the supplementary. The network result of this query is presented in Figure 4.

Querying connected information from graph databases provides significant flexibility, the flexibility entails the inherent nature of graph databases. Graph databases consider the relation between data points as first-class information in the data model, which is crucial for biological information. Mostly the biological facts are non-linear, one fact influences other facts in multiple ways, and the same is applicable to AOPs as well. Graph transversal is one of the key properties of graph databases, it makes it efficient to navigate from one node to another and capture the intermediate relationship between them.

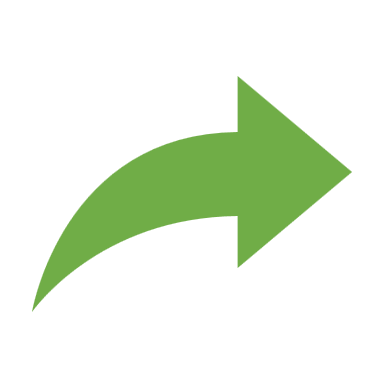
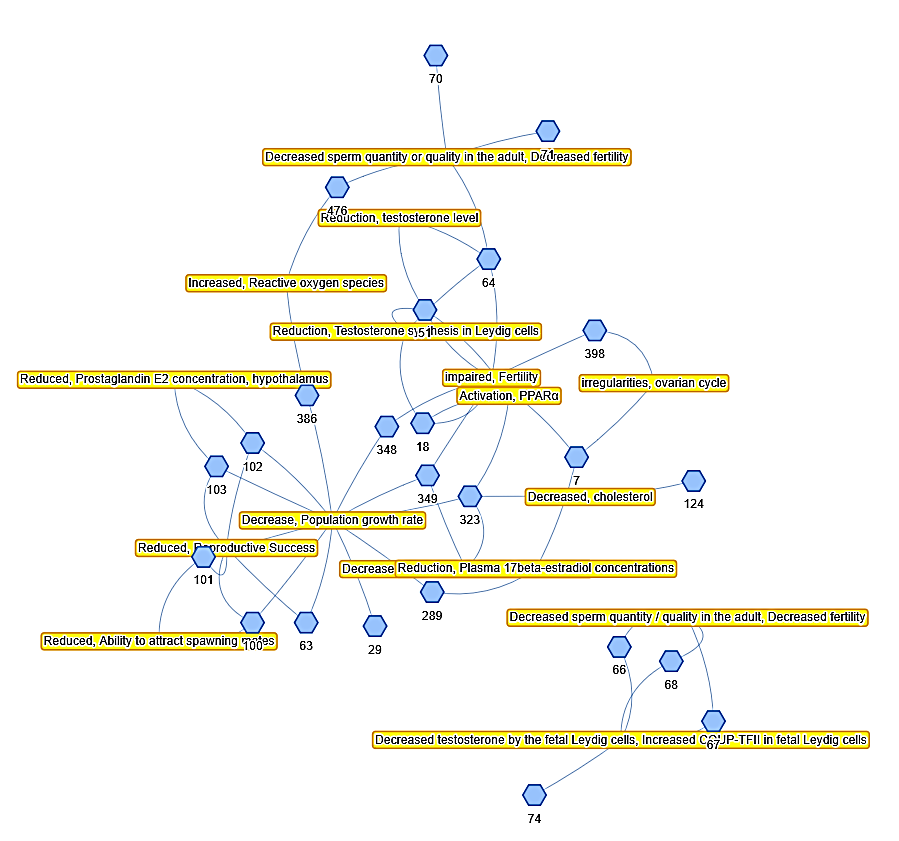
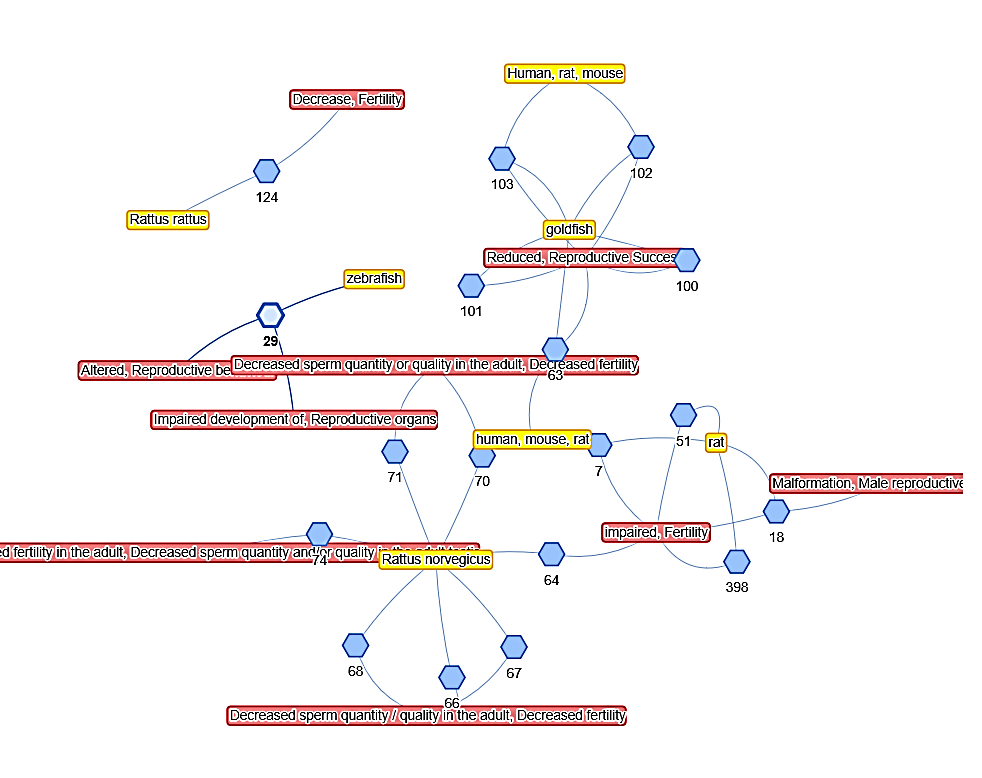
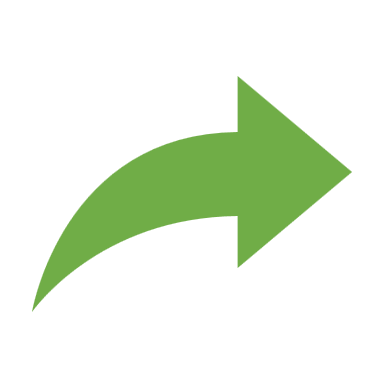
Depending on the research question, the complexities of database queries can range from relatively simple to highly complex. Table 2 and Figure 4 illustrate the diversity of questions and queries researchers may need to address while building or evaluating the AOP. Crafting complex queries is not an easy task for researchers from non-technical backgrounds. The advent of Large Language models (LLM) emerges as a valuable resource and offers a solution to tackle the problem of crafting queries. With a few examples of natural language and cyphers in the form of a prompt, the model adapts the pattern and syntax of cyphers and is able to generate the cyphers for real-time queries asked by the user. LLM acts as a wrapper and bridges the gap of technical need. Well-curated natural language query in accordance with the schema of a graph database is enough to generate precise cypher. Having no communication barrier between the information source and the users will increase the accessibility and usability of the tool. The growing usability of the tool results in the addition of diverse queries in the AOP context from the users leading to an increase in precision and accuracy of cypher generation. The diverse queries asked by the developers will also reflect the needs and expectations they have from the AOP wiki; hence this will be also helpful in modulating the AOP wiki as per the user’s requirement in future updates.

In this study we used OpenAI’s GPT-4 with 8k context size closed source LLMs, for each request to the language model charges $0.03/1k tokens. On average our prompt consists of 5k tokens, which costs us $0.15 for each natural language to cypher generation. These expenses are anticipated to reduce over time, as the database accumulates relevant examples, we expect to rely on prompt examples resembling user queries, thereby reducing the need of imputing multiple examples. Furthermore, with the rapid advancement of large language model development in the open-source community, the problem concerning cost will be addressed. The recent release of open open-source LLMA model by Facebook shows the capacity to generate code and has garnered attention. In the future fine-tuning models like LLMA on cypher queries will be a cost-effective alternative to the OpenAI’s model.

### Interactive and stepwise query

The interface for exploring AOPs is designed to facilitate interactive and stepwise queries, acknowledging the inherently exploratory nature of AOP development.

In this section, we will explain with the help of an example query i.e. “What are the key events that are connecting the AOPs which is applicable to fish and rat taxonomy and has adverse outcome related to fertility or reproductive issue”. From the query, you can infer that the long and complex query can be broken down into multiple stepwise queries such as 1.) AOPs which are applicable to “fish” and “rat” 2.) Filter the AOPs, which mention “fertility” or “reproductive” issues in the adverse outcome 3.) Find the shortest path between the filtered AOPs which are connected by key events only. The above-mentioned queries are rationally divided into steps and queried separately, which allows users to analyze the results of the query individually and make further decisions as per requirement. In Figure 5. above mentioned natural language query is presented in cypher query and the visual representation of the query is present in supplementary.



**Query 1.**

**AOPs applicable to “fish” and “rat” taxonomy?**

**Query 2.**

**Filter the AOPs, which mention “fertility” or “reproductive” issues in the adverse outcome**

**Query 3.**

**Find the shortest path between the filtered AOPs which are connected by key events only.**

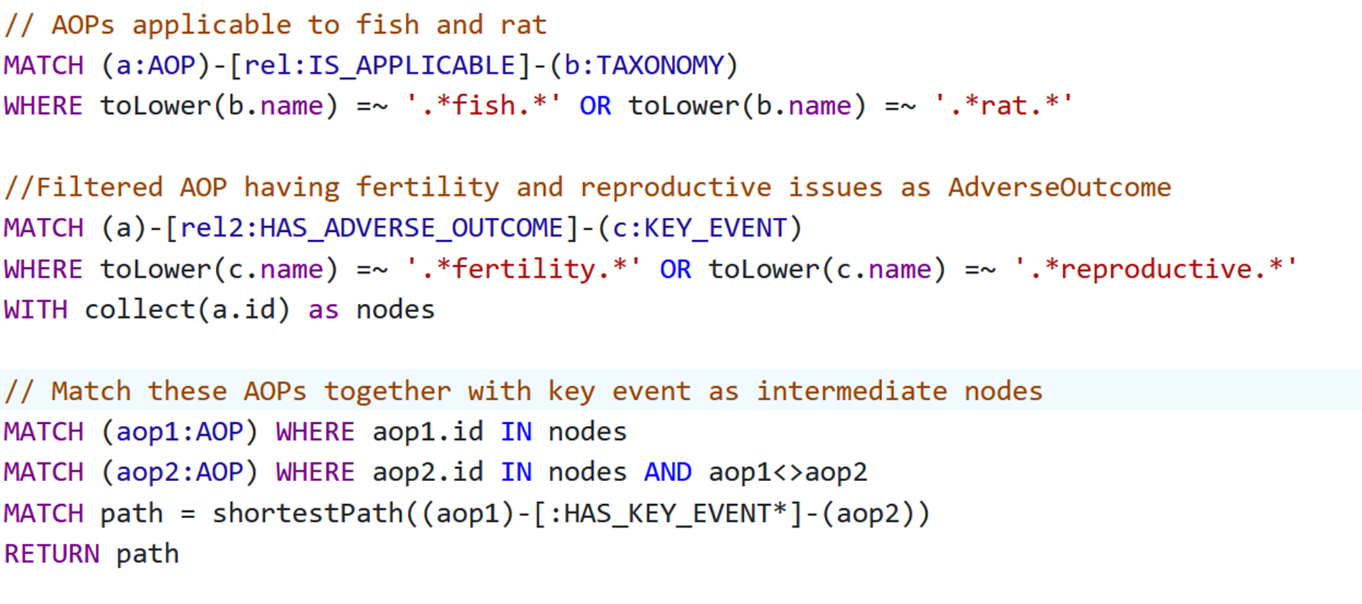
Figure Network based view of step wise query

## Conclusions

In this work we adept AOP-wiki data in its natural representation as a graph enabling the users to query and link information in a very flexible way. With databases comes the technical bottleneck of crafting queries to retrieve desirable information and hence it reduces the usability of the tool among domain users. In this work, we fill the technical gap and implement the large language model’s ability to generate cypher queries based on the natural language query provided in the context of graph database schema. Having an exploratory nature to the AOP development process, we thoughtfully designed the interface which allows users to retrieve their complex research queries in multiple steps and also keep track of individual queries separately. The results of the queries will be retrieved in interactive network form. The Detailed information on AOPs, key events and their relation is accessible in tabular form and also provides the functionality to retrace the information source.

## Now we will talk about the extension of this work, in future

|  |  |
| --- | --- |
| **Natural Language Query** | **Cypher Query** |
| **Simple:** Stressors related to AOP on learning and memory impairment. | MATCH (a:AOP)-[rel1:HAS\_STRESSOR]-(b:STRESSOR)  WHERE toLower(a.name) =~ '.\*learning.\*' OR toLower(a.name) =~ '.\*memory.\*'  RETURN \* |
| **Moderate:** Provide me with a network of AOPs related to neurotoxicity and connected to key events mentioning calcium influx. | MATCH (a: AOP)-[rel1:HAS\_KEY\_EVENT]-(b:KEY\_EVENT)  WHERE toLower(a.name) =~ '.\*neuro.\*'  WITH a  MATCH (a)-[rel2:HAS\_KEY\_EVENT]-(c:KEY\_EVENT)  WHERE toLower(c.name) =~ '.\*calcium.\*'  RETURN \* |
| **Complex:** key events which lie in the shortest path connecting AOPs related to neurotoxicity. | MATCH (a:AOP)-[:HAS\_ADVERSE\_OUTCOME]-(c:KEY\_EVENT)  WHERE toLower(a.name) =~ '.\*neuro.\*'  WITH collect(a.id) as nodes  MATCH (aop1:AOP) WHERE aop1.id IN nodes  MATCH (aop2:AOP) WHERE aop2.id IN nodes AND aop1<>aop2  MATCH path = shortestPath((aop1)-[:HAS\_KEY\_EVENT\*..2]-(aop2))  RETURN path |

Figure . Stepwise query of information.

## References