# Digitization and Logic Extraction of the British Columbia Building Code (BCBC)

Timothy Fang, Ryan Bradshaw, Jonathan Doyle

February 4, 2025

## Summary

The aim of this project is to digitize the British Columbia Building Code (BCBC) to improve its accessibility, usability, and efficiency for architects, engineers, and other professionals. By leveraging large language models and graph database technology, the project explored the feasibility of transforming the complex, lengthy BCBC document into a machine-readable format, capturing both its structural hierarchy and logical relationships. Integrating a graph database (Neo4j) enabled the visualization and traversal of these relationships, while a front-end demo showcased the potential for interactive navigation and filtering. This first phase lays the groundwork for modernizing how professionals interact with building codes, offering a more streamlined and user-friendly approach to compliance and design.

## Project Overview

This project has four main goals:

- Digitizing the British Columbia Building Code (BCBC),
- Extracting the logical relationships between different codes,
- Integration with a graph database management system to store the hierarchal structure and logical relationships of the BCBC
- Developing an interactive front-end demo for navigating and filtering the extracted codes.

The project leveraged the large language model OpenAI GPT-4 to extract the structural hierarchy (digitization) and logical relationships of the BCBC. The graph database management system Neo4j was employed to streamline interactions with the BCBC, making it more accessible and easier to understand for users. Below is a detailed breakdown of the key phases of the project.

### 1. Proof of Concept with ChatGPT

The first step in this project was to determine whether data and logic from the BCBC could be reliably extracted. Two specialized ChatGPT models were developed:

- **Data Extraction**: Converting the BCBC's complex text into JSON while preserving the structural hierarchy of articles, sentences, and clauses.
- **Logic Extraction**: Mapping logical relationships between these elements to capture dependencies and cross-references.

By applying ChatGPT to smaller sections of the BCBC and confirming accurate extractions, the project moved forward with larger-scale automation.

### 2. Scaling with the GPT-4 API

Following the successful proof of concept, the team transitioned to using the GPT-4 API to handle larger sections of the BCBC. This move brought two key advantages:

- **Faster Processing**: The advanced model architecture handled longer and more complex text with greater efficiency, accelerating turnaround times.

- **Scalability**: By integrating the GPT-4 API into automated pipelines, the team could parse large volumes of content with minimal manual intervention.

Overall, this transition significantly boosted both the speed and accuracy of capturing the BCBC's intricate legal text and complex logical structure.

### 3. Integration with Neo4j

Once large sections of the BCBC were digitized and its logic extracted, the data was ingested into a Neo4j graph database. Neo4j was chosen for its ability to represent complex relationships in a graph structure, allowing the BCBC's logical connections to be stored and queried in an intuitive manner. This phase involved writing an ingestion script that utilizes Neo4j's declarative quary language, Cyphers, to embed the BCBC as a graph:

- **Nodes**: Each building code element is embedded as a node, which has properties to hold the type (article, sentence, table, etc.) and descriptions (code text).
- **Hierarchy**: The structural hierarchy is represented by directed edges named CONTAINS. For example, a Sentence node will connect to each of its Clause nodes using these CONTAINED directed edges.
- **Logical Relationships**: These are also represented by directed edges. For example, if a Sentence requires all of its Clauses to be satisfied (the AND logical connective), AND edges connect each Clause node that must be traversed back to the parent Sentence (via a RESULT directed edge) for that Sentence to be considered satisfied.

Here is an example:

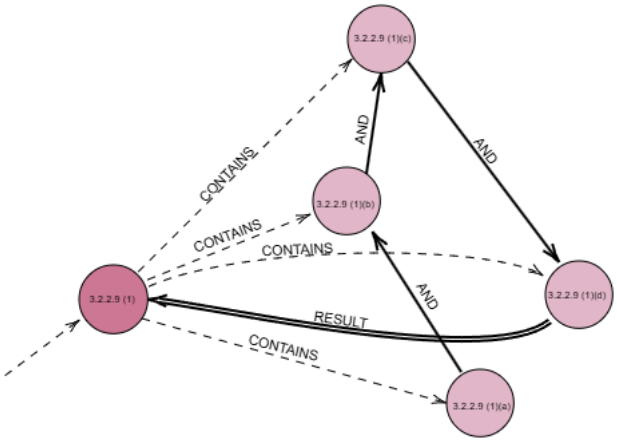


Figure 1: graph-example

### 4. Front-End Demo Development

The final phase of the project focused on building a front-end user interface that would allow users to interact with the digitized BCBC. This demo was built to:

- Visualize the BCBC data and its logic as a graph, showcasing the structure of the building code and how various codes are interrelated.
- Provide filtering capabilities so users could explore and narrow down the building codes based on the extracted logic.
- Demonstrate the power of graph traversal, enabling users to navigate between interconnected sections of the building code seamlessly.

## Challenges and Future Work

Although this beginning phase of the project had great progess, several challenges and areas for improvement were identified during the process. Addressing these issues will ensure greater accuracy and efficiency in the future. Below are key difficulties encountered with proposed solutions, and discussion on future work.

### Challenges

**1. Data Chunk Extraction from the BCBC** One of the major challenges was accurately extracting correct chucks of data (blocks of codes) from the BCBC, particularly when codes spanned multiple pages. PDF extraction can be problematic due to inconsistencies in formatting, where sections or tables break across pages, making it difficult to capture these in a unified format. This was particularly an issue when attempting to extract tables, and lower level codes (sentences, clauses, subclauses) that spilled onto multiple pages. Such cases led to incomplete or malformed data.

**Solution:** Instead of solely relying on the GPT-4 API for data extraction, future iterations of this project should consider integrating an LLM model from Sherpa, which is specifically trained for these types of tasks. Initial tests with Sherpa have shown promising results in dealing with multi-page data and table extractions more accurately. Given that Sherpa operates under an MIT license, this makes it an appealing option for future work, as it offers robust capabilities while remaining cost-effective.

**2. Imperfect Logic Extraction** While the logic extraction generally performed well, there were instances where the model failed to correctly identify the logical connectives (OR, AND, NOT) between codes. This was most apparent with codes requiring a list of lower level codes (clauses, subclauses) to be satisfied. The logical connectives connecting these lower level codes were occasionally misinterpreted; AND instead of OR, and vise versa.

**Solution:** A three-step improvement process is proposed:

1. **Breaking Down Logic Extraction:** First, breaking down the task of logic extraction into smaller, more manageable components would break the logic into parts (to be concatenated at a later stage) and shorten the length of prompts. This could allow the model to focus on less complicated logic structures, improving accuracy.
2. **Fine-Tuning the Model:** Second, fine-tuning the GPT model on a dataset of building code-specific logic would further enhance the accuracy of extractions, as it would learn the particular nuances of the BCBC's logical structure.
3. **Tuning the Temperature Parameter:** The third improvement involves adjusting the temperature parameter during the logic extraction process. The temperature setting in GPT models controls the randomness of the output: a higher temperature (e.g., 0.8-1.0) generates more creative and varied responses, while a lower temperature (e.g., 0.2-0.4) leads to more deterministic and focused output. For logic extraction, lowering the temperature could help the model focus on producing consistent, well-defined logical structures rather than introducing variability. Fine-tuning this parameter for the specific task of extracting building code logic will likely result in more accurate and reliable results.

**4. Algorithm Accuracy Measurement** Given the size of the BCBC, which spans over 1800 pages, calculating the overall accuracy of the extraction algorithm remains a complex and unresolved issue. Without a clear benchmark for measuring the accuracy of logic extraction and code structuring across such a large document, assessing the quality of the results is difficult.

**Solution:** Fine-tuning the algorithm on logic conversion tasks should provide better accuracy metrics. This would allow the team to establish a clearer understanding of how well the extraction algorithm is performing. Additionally, creating a sample set of manually validated logic conversions could serve as a baseline for evaluating the model's overall accuracy. By refining this process, future work will be able to quantitatively assess improvements and pinpoint specific areas that need further optimization.

### Future Work

**1. Efficient Filtering of Building Codes** One of the significant challenges in dealing with the British Columbia Building Code (BCBC) is its sheer size and complexity. With over 1800 pages and multiple sections, visualizing the entire code at once or querying large parts of the database can be inefficient and computationally expensive. Without an effective filtering mechanism, users might face delays and difficulty navigating through relevant building codes.

To address this, a filtering mechanism based on key criteria can be implemented. By answering a set of targeted questions, users can significantly narrow down the relevant sections of the BCBC, particularly the occupancy

classifications. This method not only reduces the complexity of visualizations but also optimizes the querying process before traversing code requirements using the extracted logic. The six criteria that can be used to filter codes include:

1. Usage (assembly, industrial, residential, or commercial use)
2. Building Square Footage
3. Number of Storeys
4. Construction Type (combustible or non-combustible)
5. Sprinklered (yes or no)
6. Street Exposure (number of streets a building faces)

By using these six criteria, it is possible to narrow down the occupancy classification, which serves as a powerful filter to reduce the dataset significantly before engaging with the logic-extraction layer. For example, just by answering the first four questions, the system can narrow down Section 3.2 (which deals with 86 different occupancy classifications) to a handful of relevant classifications—perhaps as few as five. This drastically reduces the amount of data that needs to be processed and visualized, making the system faster and more user-friendly.

The process of extracting the necessary data from the BCBC to apply these six filtering questions will be handled using the GPT-4 API. By passing GPT-4 a specific building code, it will be prompted to automatically identify and extract relevant information for each of the six criteria above. This newly required information will be then be added as a property to the Neo4j node representing the building code in question. Thus allowing efficient filtering of building codes.

**2. Front-End Development** A basic React demo was built at the end of summer of 2024. The demo demonstrated how Neo4j can be used as an interactive tool using Vis.js for graph visualization.

Development is ongoing; but here is a mockup of the envisioned tool.

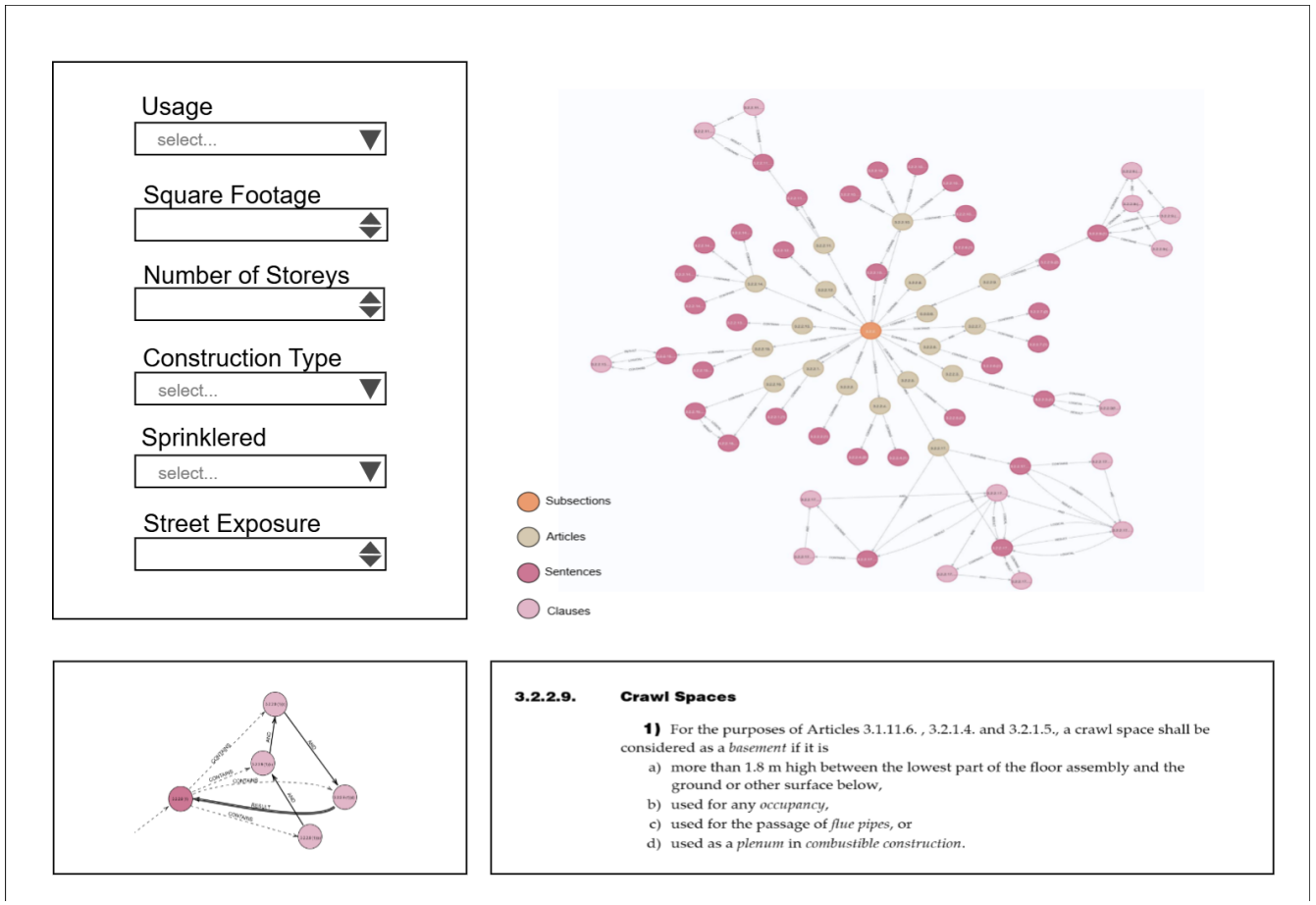


Figure 2: front-end