Overview

Patent searching is a critical step in the innovation lifecycle. It helps inventors, companies, and legal professionals assess the novelty of an idea, avoid infringement, and gain insights into technological trends. A central part of this process involves analyzing *claims*—the legally enforceable definitions of an invention—and mapping them to other patents or applications to determine overlap, similarity, or potential conflicts. Accurate and efficient claim mapping enables stronger patent prosecution, more effective prior art searches, and better strategic decision-making. This project applies search techniques to the claims and descriptions within patent filings, with the goal of improving how relevant information is discovered and compared in a growing sea of intellectual property.

In other words, this project applies directly to the cornerstone of Thinkstruct's technology and is an accurate representation of the type of work that will be done during this summer.

Data

The data needed for this project can be found in a zip file, attached. It consists of patent data from patent applications, specifically those pertaining to vehicles from 2024-present. It is grouped into separate json files, all named `patents_ipa{DATE}.json` where the DATE field indicates the week in which that set of patent applications was filed. Each json file is a list of patent dictionaries containing the following fields: Title, Document Number, Abstract, Detailed Description (broken up in paragraphs), Claims, Bibtext Citation, Classification Code. Note that some patents may be missing some fields, you may choose to either exclude those patents or handle them separately. Document which choice you make in your final code submission.

Project Details

The project will compose of two parts. The first one will build a generic search tool and then the second part will allow you to pick one or two enhancements that you would like to include. This project should take you **2 hours** (1 hour for each part, roughly). However, the goal is not to speed-code but rather spend a bit of time thinking critically and creatively about how you might solve the problems presented. The project has a lot of freedom in it by design, it is up to you to think critically about what *specific* problem you'd like to solve and how you want to solve it. **You do not need to complete every problem listed.** After part 1, you are welcome to choose any of the challenges listed, should you have remaining time.

Part 1: Basic search engine

The goal of this part is to build a search engine for the information contained in the data given to you. You can decide what exactly the inputs and outputs of this search engine should be.

Example inputs: natural language sentence query, a specific claim of a patent, an entire patent (given by patent ID), etc.
Example outputs: a specific claim or paragraph from the detailed description of the patent, a patent's metadata, groupings of patents, etc.

Based on what you pick, you may have to scrape additional data. For example, if you allow for the input to be any patent application ID, you might have to develop a small scraper that pulls the text from that patent. *Hint: ChatGPT, Gemini, etc. are great for building these kinds of scrapers*.

Part 2: Enhancements to search engine **(pick 1-2, as time allows)**

- Evaluation and training: When training an AI model, training and evaluation are the cornerstones to good performance. When generating embeddings, models are trained using *positive examples* --pairs of text that are close to each other-- and *negative examples* --pairs of text that seem like they should be related, but aren't really. You can finetune your model by training it on these pairs. Furthermore, you can evaluate it based on how closely it rates the *positive examples* or how far it rates the *negative examples.* You should decide what training data to use (whether it comes from the data given to you, from other datasets such as bigpatent, or directly from the patent office). Note that positive examples are much easier to generate than negative examples and you may not have to do both. Generate some training data, evaluate your model based on the data, train the model, and report what the improved evaluation metric is. Note that since you are only working for 1 hr, the expectation is not to show significant improvement in the evaluation post-training, but rather construct the training/evaluation pipeline.
- Hybrid searching: Often, patent agents don't just want to look for *any* patent that is related to a query, they want to look for a patent who's abstract contains a few keywords, who's classification code is specific to vehicle wheels (meaning the classification starts with B60B), etc. Add to your search engine so that you can:
    - Place constraints on the classification codes
    - Search for keywords in the title or abstract
    - Search for a specific title

  Naive, performing a hybrid search can be quite computationally expensive, but there are ways to implement it so that it actually helps performance at large scales. Time how long your algorithm takes with and without the hybrid search enabled and provide some commentary on things you would or have used to make it more efficient.
- Two-phase searching: You may notice that while semantic search is much better than keyword searching, it could still be improved. Often what people do is use semantic searching to reduce the results to roughly 100 candidates, and then use some more sophisticated algorithm to reduce those to 30 candidates and rank them. There are many of these algorithms online to find. Some examples are simply just asking a language model to output rankings, graph based re-ranking, using a cross-encoder, etc.
- Efficiency and large data: This dataset is fairly small, it contains only on the order of 10^3 patents. In general, there are 10^7 patents filed *every year*. **To do this part, please request the large data sample**. Measure the time it takes to process the data

into the database and search it for this large dataset (or a portion of it). Then, make improvements to how you process and search the data. Record your improvements and submit the highest performing code.

- Interfaces and users: No algorithm is complete without some sort of implementation. Create an interface that users can easily use to interact with your system. This can be a graphical interface or a terminal interface. Additionally, you can opt to support user sessions, allowing users to see/revisit their most recent searches and/or log in. Lastly, you can design a method that allows multiple users to search at the same time. If you do this, take note of how parallel use affects performance. Can you simulate 100 users at the same time?

Submit the code either as a github link or a zip file of the code. Also include a short 2 minute screen recording of you using various features of your code. Additionally, make sure to include a README detailing at minimum:

- What the specific problem statement is you chose to solve.
- How the code addresses that problem statement, including which enhancement you picked and why.
- How to run your code

On using Gen AI: It doesn't matter if you choose to use Gen AI for this project. If you find it helpful, then you'll certainly be working with it, so why not allow it here. *Even if you don't type a single line yourself, that's completely OK*. The point of this project (and most other projects we work on) is **not** to type code quickly, but rather to think critically about high-level CS and AI problems.