# 

# Basic Flow

(Needs to be written more generically and needs polishing with better context i.e what is cpc, what dataset means, it’s source etc)

1. Download xml file for the latest version of patents from <https://data.uspto.gov/bulkdata/datasets/ptgrxml?fileDataFromDate=2025-01-1&fileDataToDate=2025-07-16> and split into individual chunks
2. Parse the individual chunks to form a dataframe which has individual patents , their section, class, subclass, main\_group and subgroup details as well as abstract, title, description. Combine the latter to a single text with some cleanup. Produce uspto\_patents\_cpc\_metadata.csv.
3. Load uspto\_patents\_cpc\_metadata.csv
4. Split into df\_cleaned (for train) and test\_df (for test) - 85:15
5. Use df\_cleaned to:
   1. Finetune a sentenceTransformer which can produce high quality embeddings using contrastive learning for **sections (techniques similar to CL)**
   2. Generate balanced positive and negative **section** pairs for each section to finetune 3.a
6. Optionally, train/test on df\_cleaned(or should it be test\_df) using RFC/LR/XGB etc without a finetuned sentencetransformer feature/embedding
7. Explain basic choice of classification models (especially why we go with 7b though we initially tried on 7a…….see in the ‘modeling the problem’ section below):
   1. Train and test using a classifier like RFC, LR, XGB etc using the finetuned model in 3 to generate embeddings/features on train/test split for test\_df
   2. MOE:
      1. Use the MOE section classifier on df\_cleaned for inference output
      2. Produce balanced dataset positive and negative pairs for each class within section
      3. Finetune the model loaded in 5 with class balanced data in 6
      4. Inference with test\_df/df\_cleaned for per section class level inference.

# Generating Paired Datasets

If we just use a pretrained SentenceTransformer to generate the embeddings for a pair of patents belonging to the same section say p1(G) and p2(G) and generate the embeddings for a pair of patents belonging to different section say p1(G) and p3(H), we see low difference in cosine similarity scores between the pairwise embeddings for the positive and negative pairs. This indicates that the embedding model (and embeddings) does not capture sharp differences between semantically similar pairs and semantically dissimilar pairs.

We could improve the semantic similarity and dissimilarity between corresponding pairs by moving them around in the embedding/vector space in such a way that semantically similar patents/descriptions are near each other and dissimilar patents/descriptions/section are far apart. This can be done by finetuning a pretrained sentence transformer with increasing cosine similarity for positive pairs belonging to the same section and decreasing the cosine similarity for dissimilar patent pairs from different sections.

In order to achieve this we create a finetuning dataset of positive and negative pairs per section carefully. While on one hand this dataset will help us improve the embedding quality, on the other it will help us address label/section imbalance, if done well. This approach shifts the problem from sample imbalance to pair imbalance and then addresses the latter by producing balanced positive and negative pairs per label. We implement **per–class/label pair balancing** rather than randomly sampling the pairs to avoid dominant classes flooding the pairs and prevent rare classes from not getting enough positive or negative examples. We intentionally generate enough positive pairs for rare classes, include rare classes in negative pairs, and balance pair counts so the loss function doesn’t ignore rare classes.

This approach, which is similar to contrastive learning, is naturally robust to class imbalance. By generating balanced pairs, there is nooversampling/undersampling needed on raw data. Pair generation is our main balancing tool and produces enough samples per section for finetuning for the model to be fair.

Label → # publications mapping:

Label H: 3217 publications

Label G: 2993 publications

Label B: 1092 publications

Label C: 524 publications

Label D: 38 publications

Label A: 1505 publications

Label F: 490 publications

Label E: 201 publications

Label H: Kept 2000 positive pairs

Label G: Kept 2000 positive pairs

Label B: Kept 2000 positive pairs

Label C: Kept 2000 positive pairs

Label D: Kept 703 positive pairs

Label A: Kept 2000 positive pairs

Label F: Kept 2000 positive pairs

Label E: Kept 2000 positive pairs

Total positive pairs: 14703

Label H: Kept 2000 negative pairs

Label G: Kept 2000 negative pairs

Label B: Kept 2000 negative pairs

Label C: Kept 2000 negative pairs

Label D: Kept 190 negative pairs

Label A: Kept 2000 negative pairs

Label F: Kept 2000 negative pairs

Label E: Kept 1005 negative pairs

Total negative pairs: 13195

Balanced pair dataframe with 26390 rows.

The same technique is followed for finetuning classes within each of the sections i.e we generate positive and negative pairs for each of the classes in each section separately. Eg for section E we generate positive and negative pairs for each of the below classes

['E01', 'E02', 'E03', 'E04', 'E05', 'E06', 'E21'].

As mentioned earlier, instead of randomly sampling the pairs, we make sure each of the positive and negative pairs is unique for each of the classes, which will likely improve the information gain per unique pair.

# Modeling the problem

A snapshot of the dataframe of sample observations (patents) is shown below:



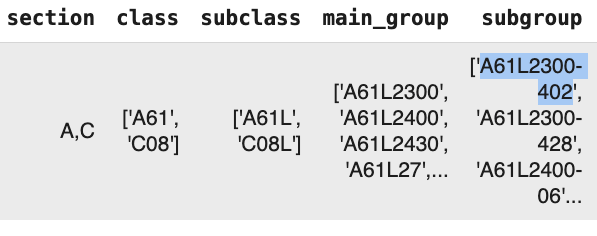
**Labels**

Here we have a hierarchical multilabel classification problem at 5 levels:

1. Sections - A, B, C, D, E, F, G, H, Y
2. Class - within each section, we have further classes corresponding to those sections - Say for section A the classes are: ['A01', 'A21', 'A22', 'A23', 'A24', 'A41', 'A42', 'A43', 'A44', 'A45', 'A46', 'A47', 'A61', 'A62', 'A63']
3. Subclass - Class A61 for example can have further subclass like L, B, K
4. Main\_group - Subclass A61L can have further Main\_group like A61B5, A47K10
5. Subgroup - Main\_group finally can have further subgroup like A61B5-0022, A47K10-42, A61L2300-402

Please note that each of the observations (patents) can belong to multiple labels in each of the above levels. However, in this project, we limit ourselves to only two hierarchy levels:

1. Section prediction
2. Class prediction



**Attributes (Feature Engineering)**

There are several attributes for each of the patents as indicated by the column names in the dataframe above. However, we only consider the following features for this problem.

1. Abstract
2. Title
3. Description

## Modeling and Choice of Models

The core idea is to produce embeddings for each of the patents using a concatenation of Abstract, Title, and Description patent attributes. We further finetune these embeddings using techniques similar to contrastive learning with positive and negative pairs at section as well as class level classification. Finally, we use these embeddings as features for a downstream classifier.

1. Pretrained Models for Finetuning: For finetuning there are a variety of pretrained SentenceTransformers to choose from like
   1. all-MiniLM-L6-v2,
   2. AI-Growth-Lab/PatentSBERTa,
   3. Multi-qa-mpnet-base-dot-v1,
   4. intfloat/e5-base-v2,
   5. allenai/longformer-base-4096.

The choice of any of the above for our problem is a bit arbitrary because we empirically experimented with some of the above instead of conducting a thorough comparison study. This is because the key emphasis for this project is on modeling a real world problem which can use any of the relevant models, as well as building a framework which can be further tuned at every stage than evaluation of the best pretrained models. We are also constrained by time and GPU resources in this project.

1. Classification Models: There are lot of choices for classification models like Logistic Regression, XGBoost, SVM, Random Forest etc. While these models land themselves naturally to many traditional classification problems, they are not easy to fit directly into multilabel classification with a hierarchical structure. We tried some of these traditional models (write results).

However, Mixture Of Experts (MOE) aligns naturally with hierarchical classification as well as long tail distribution like some labels like G,H are dominant and others like D,Y are rare. MOE enables rare label focus as some experts can specialize in rare categories as well as avoid catastrophic forgetting of underrepresented classes.

A **Mixture of Experts** model splits a big problem into **smaller sub-problems**, and lets specialized models (called “experts”) handle parts of it. Instead of one big model trying to learn everything, MoE builds many smaller “expert” models, each good at certain things. A “gate” decides which experts to trust for each input.

In general, MoE can assign **specialized experts to different branches** of tree of labels: Section → Class → Subclass → Main Group → Subgroup. Traditional models treat everything as flat and can’t model this hierarchy unless you build it manually. MoE (with sigmoid activation + BCE loss) is **designed for multilabel**. Patents often belong to multiple sections/classes. Traditional models like XGBoost and SVM need either: Many One-vs-Rest models (slow, unscalable) Or reduce multilabel to multiclass (incorrect). Rare sections like D, Y suffer in traditional models.In MoE, some experts can **specialize in rare classes**.In Random Forests or Logistic Regression, rare classes often get ignored or misclassified, even with class weights. MoE can learn **context-aware routing**, activating different experts for chemical vs. electronics patents. XGBoost/Logistic Regression builds a single monolithic model that must work for all domains.

There are multiple ways to model MOE. We could have local experts for each node and train an MoE at each level where the gating network routes inputs to the right branch and each branch has specialized experts. Another way is to build a single MoE with multiple specialized experts instead of a separate model at each node. Each expert specializes in a branch of the hierarchy or a certain combination of fine-grained classes. The gating network learns which experts to activate depending on the input. Yet another way is to model Multi-task MoE where each task learns its own gating over shared experts and experts can specialize in features relevant to parts of the hierarchy.

Given the time and other constraints for this project, we train a simple or flat MoE for section classification. Also we also train separate simple MoEs for each of the classes within each section. In a Flat MoE, the gating network and experts operate at a single level, and the final output is directly computed as a weighted combination of all expert outputs for the target task (e.g., predicting CPC Section).

Explain config of our simple MoE here.

While the section-level MoE handles multilabel classification, the class-level MoEs classify labels within a section. We currently use the section labels ground truth to feed the data to the respective class level MoE, though a production inference pipeline would invoke the respective class MoE based on the predicted section. The reason for the current implementation is to just evaluate the class MoE performance and optimize it with proper training instead of building a complete production pipeline. Also, we do not currently evaluate multilabel scores for class MoEs, though it is trivial to collect those from the respective sections that they belong to.

We do, however, fine-tune the fine-tuned section embedding model further on pairwise positive and negative examples for each class MoE. There is a clear possibility of taking a pretrained sentence transformer and maintaining separate fine-tuned models for the class within each section, though it is difficult to maintain all of those models without a model zoo or ML platform/pipeline.

## Evaluation

### Section Prediction

We first discuss the baseline and finetuning results for multilabel section classification. We drop the rows which have no section information and then split the dataset into 80:20 ratio. The 80% set is called df\_cleaned and the 20% set is called test\_df. We then setup the pipeline with the following options:

1. Choice of pretraining models to produce embedding features based on concatentation of title, abstract and description for a patent. We have chosen all-MiniLM-L6-v2 and intfloat/e5-base-v2. all-MiniLM-L6-v2 is a very basic pretrained sentence transformer while e5-base-v2 has comparable performance to other similar models like AI-Growth-Lab/PatentSBERTa, multi-qa-mpnet-base-dot-v1
2. Choice of classification model for multilabel section prediction
   1. MultilabelBinarizer with OneVsRestClassifier and RandomForestClassifier - **MultilabelBinarizer** converts text labels to binary format.**OneVsRestClassifier** trains one classifier **per label**.Together, they enable robust multilabel classification with any base model (e.g. LogisticRegression, XGBoost, etc.)
   2. MultilabelBinarizer with OneVsRestClassifier and LogisticRegression
   3. Mixture of Experts Classifier is run only on the best of the two pretrained model which produces a better baseline results.
3. We use first df\_cleaned to produce embeddings from both the pretrained models mentioned above(1) and then run all the classifiers mentioned above (2). Df\_cleaned embeddings are split into train and test set again when training and predicting from the classifiers.
4. We repeat the exact same process mentioned above (3) for test\_df
5. Next we use df\_cleaned to generate a balanced dataset across sections with positive and negative pairs as explained in section ‘Generating paired dataset’. This produces a new dataframe df\_pairs, which is used to finetune each of the pretrained models mentioned above (1).
6. We then repeat step 3 and 4 by using embeddings generated via the finetuned model in step 5 above.

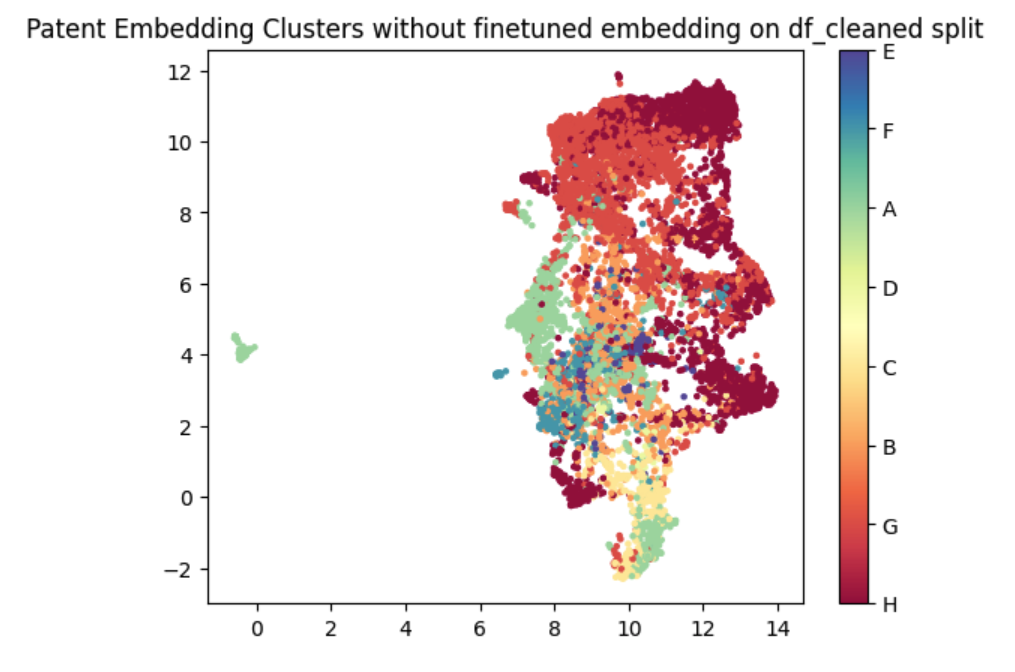
**Baseline Metric**

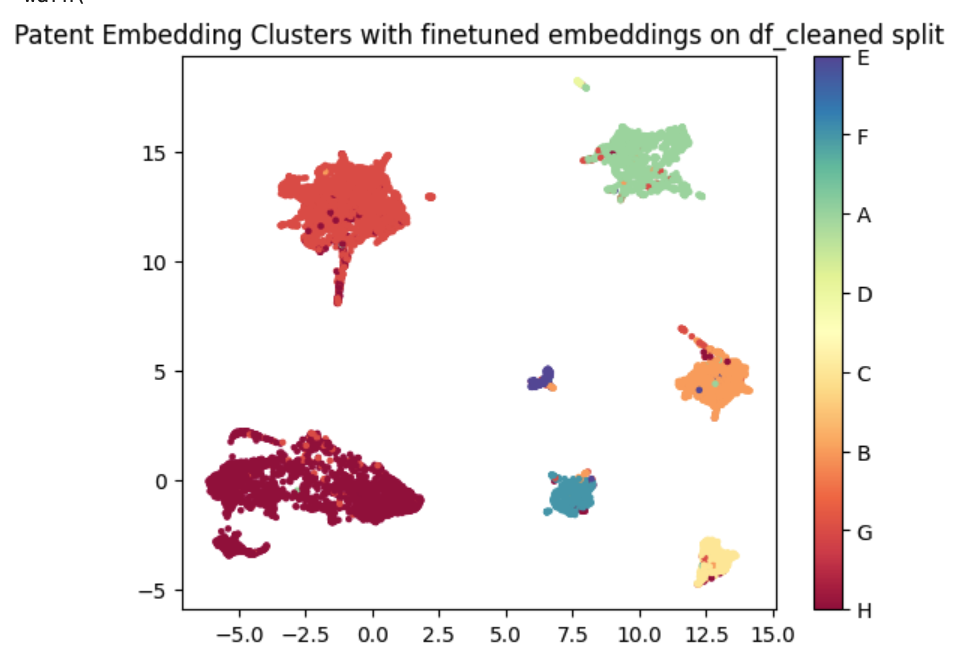
We have 2 pretrained models (MiniLLM, e5-base) \* 2 classifiers in Patent NB(RFC, LR) \* 2 datasets (df\_cleaned, test\_df) = 8 metrics to show

**Finetuned Metric**

We have 2 finetuned models (MiniLLM, e5-base) \* 3 classifiers in Patent NB(RFC, LR, MoE) \* 2 datasets (df\_cleaned, test\_df) = 12 metrics to show - 1(as we do not run MoE on MiniLLM) = 11 metrics to show

For e2-base\_v2, the difference between baseline and finetuned embeddings is show belowon df\_cleaned:





**Class prediction**

For each of the sections we train a class-level MoE classifier with per-class pair-level finetuned embedding on test\_df split. We basically skip evaluation on df\_cleaned set, just because we care about the final performance on test\_df split.

# Appendix

There are two parallel programs:

1. Patents.ipynb: <https://colab.research.google.com/drive/1boqLgMtNXYRWPbjD0-NySTnHRcqqEh8n#scrollTo=Jel4h9sDKYoy>
2. MOE.ipynb: <https://colab.research.google.com/drive/1MBy1Z_JwO9JMLCXCDRMGUBdFkB4QTxxv#scrollTo=47alHv7b3guR>

# Basic Flow (Patents.ipynb)

1. Download xml file for the latest version of patents from <https://data.uspto.gov/bulkdata/datasets/ptgrxml?fileDataFromDate=2025-01-1&fileDataToDate=2025-07-16> and split into individual chunks
2. Parse the individual chunks to form a dataframe which has individual patents , their section, class, subclass, main\_group and subgroup details as well as abstract, title, description. Combine the latter to a single text with some cleanup. Produce uspto\_patents\_cpc\_metadata.csv.
3. Load uspto\_patents\_cpc\_metadata.csv
4. Split into df\_cleaned (for train) and test\_df (for test) - 85:15
5. Use df\_cleaned to:
   1. Finetune a sentenceTransformer which can produce high quality embeddings using contrastive learning for **sections**
   2. Generate balanced positive and negative **section** pairs for each section to finetune 3.a
6. Optionally, train/test on df\_cleaned(or should it be test\_df) using RFC/LR/XGB etc without a finetuned sentencetransformer feature/embedding
7. Train and test using a classifier like RFC, LR, XGB etc using the finetuned model in 3 to generate embeddings/features on train/test split for test\_df

# Basic Flow (MOE.ipynb)

1. Load uspto\_patents\_cpc\_metadata.csv
2. Determine the complete list of sections and classes present in the dataset.
3. Split into df\_cleaned (for train) and test\_df (for test) - 80:20 (may be it should be same split as above 85:15 to avoid data leak)
4. Load the section finetuned model from step 5 above.
5. Use the MOE section classifier on df\_cleaned for inference output
6. Produce balanced dataset positive and negative pairs for each class within section
7. Finetune the model loaded in 5 with class balanced data in 6
8. Inference with test\_df/df\_cleaned for per section class level inference.