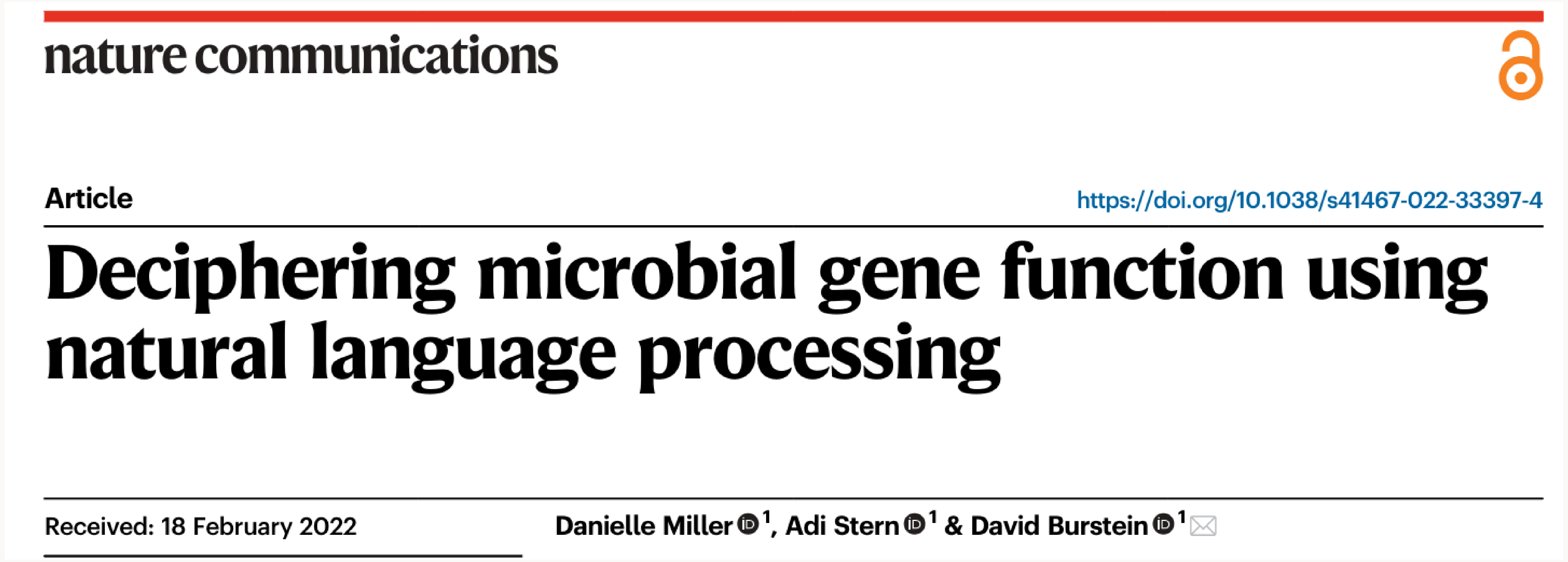
DECIPHERING MICROBIAL GENE FUNCTION USING NATURAL LANGUAGE PROCESSING

RAND FATOUH


nature communications

Article

https://doi.org/10.1038/s41467-022-33397-4

Deciphering microbial gene function using natural language processing

Received: 18 February 2022

Danielle Miller[1], Adi Stern[1] & David Burstein[1]

INTRODUCTION

- Vast amounts of genetic data, especially metagenomics
- Unknowen function
- Potential applications in biotechnology and medicine
- Genomic context, especially in prokaryotes, plays a crucial role in understanding gene function
- Co-functioning genes form clusters

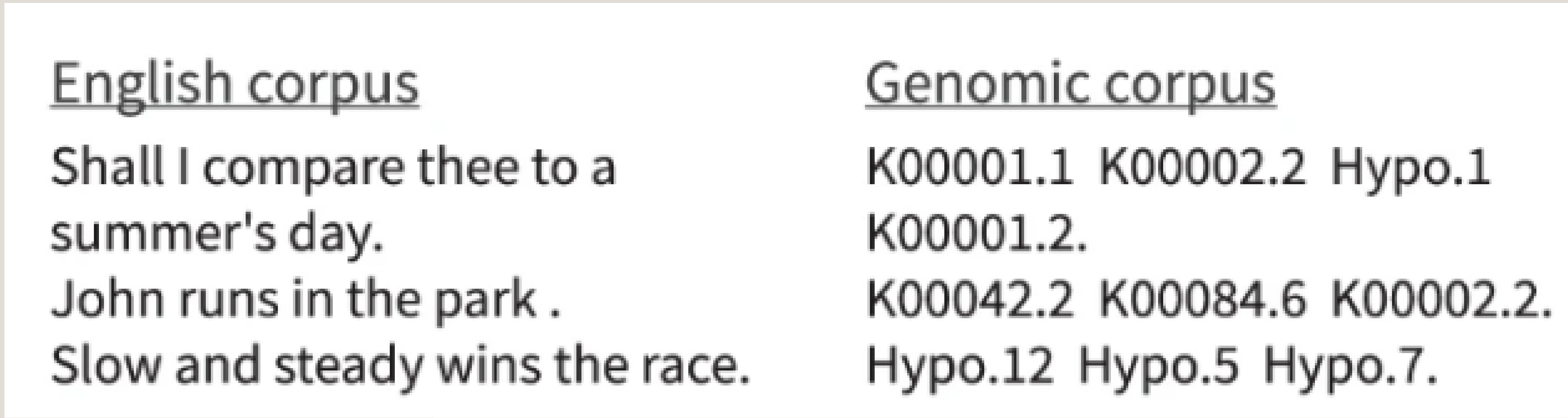
https://physicsnetwork.org/microbial-genetics-genetic-mechanisms-in-bacteria-and-archaea.html

GOAL

Find gene function based on their contextual surrounding

HOW

using natural language processing techniques



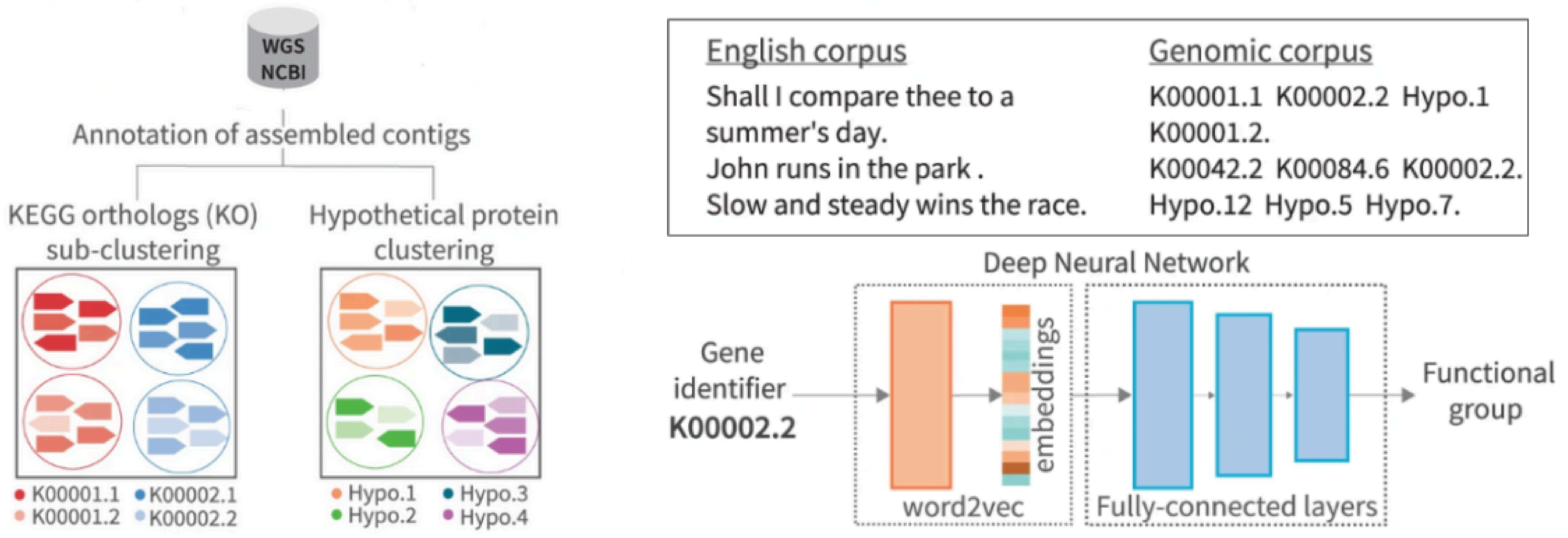
English corpus

Shall I compare thee to a summer's day.
John runs in the park .
Slow and steady wins the race.

Genomic corpus

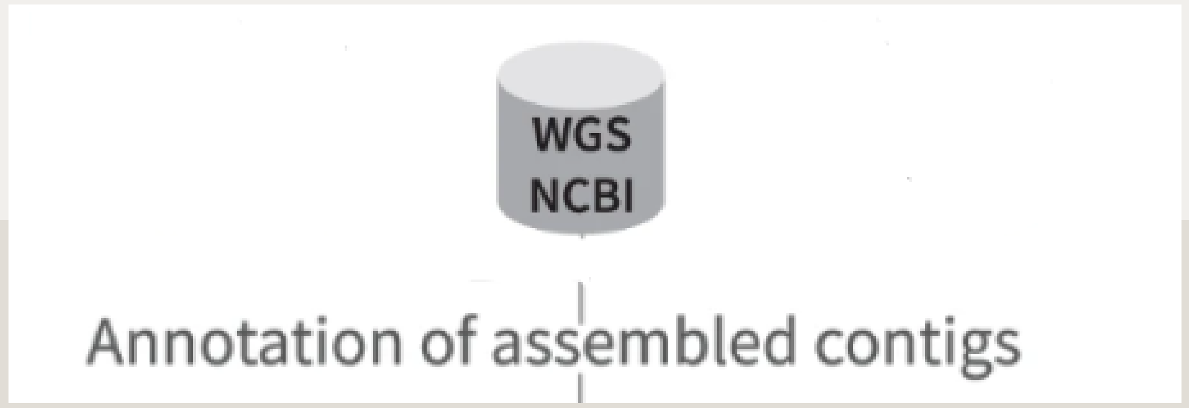
K00001.1 K00002.2 Hypo.1 K00001.2.
K00042.2 K00084.6 K00002.2.
Hypo.12 Hypo.5 Hypo.7.

STEPS OF THE ANALYSIS



English corpus	Genomic corpus
Shall I compare thee to a summer's day.	K00001.1 K00002.2 Hypo.1 K00001.2.
John runs in the park .	K00042.2 K00084.6 K00002.2.
Slow and steady wins the race.	Hypo.12 Hypo.5 Hypo.7.

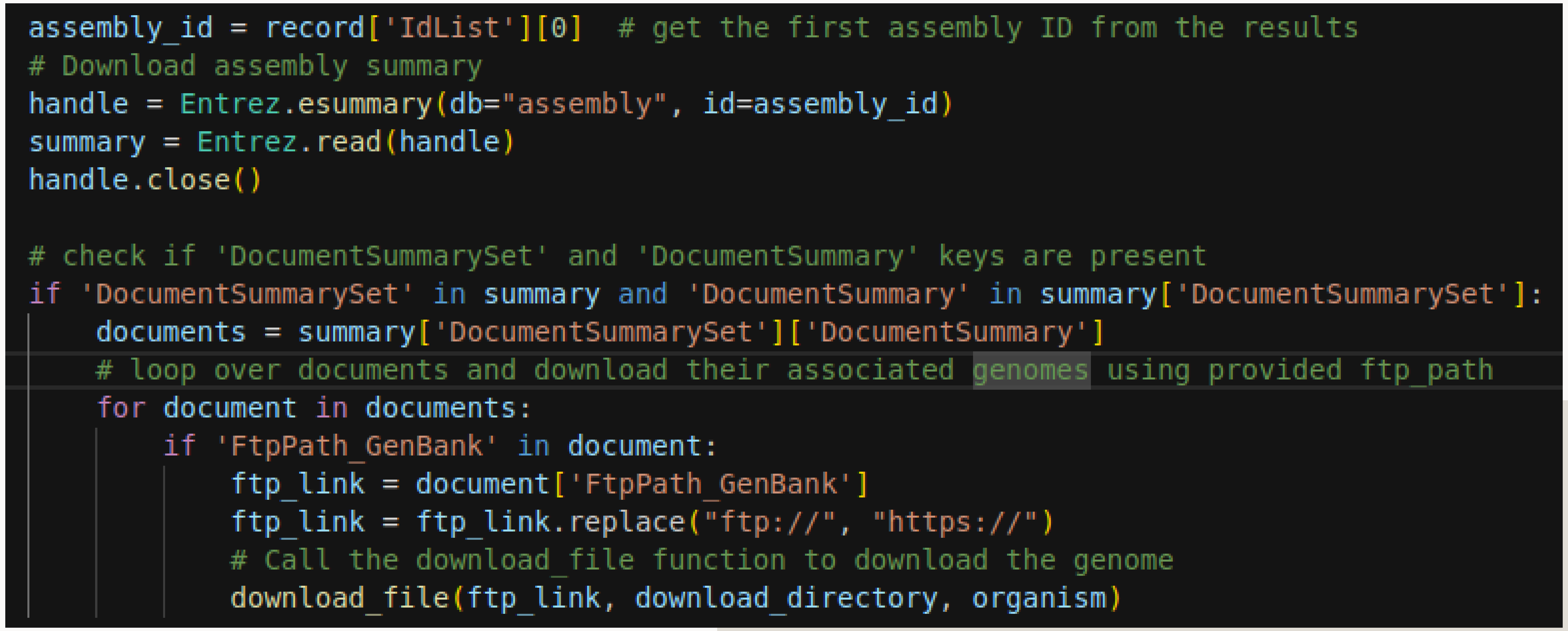
1- GENOMIC DATA COMPILATION



STEPS

1- Dowload all WGS information from NCBI dataset except for Euekaryotics, including biosample number

2- Scraped thier fasta files using Entrez from Biopython package and urllib.



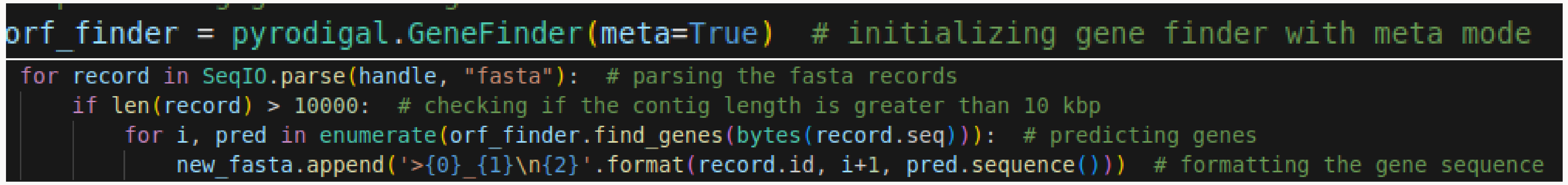
```python
assembly_id = record['IdList'][0]  # get the first assembly ID from the results
# Download assembly summary
handle = Entrez.esummary(db="assembly", id=assembly_id)
summary = Entrez.read(handle)
handle.close()

# check if 'DocumentSummarySet' and 'DocumentSummary' keys are present
if 'DocumentSummarySet' in summary and 'DocumentSummary' in summary['DocumentSummarySet']:
    documents = summary['DocumentSummarySet']['DocumentSummary']
    # loop over documents and download their associated genomes using provided ftp_path
    for document in documents:
        if 'FtpPath_GenBank' in document:
            ftp_link = document['FtpPath_GenBank']
            ftp_link = ftp_link.replace("ftp://", "https://")
            # Call the download_file function to download the genome
            download_file(ftp_link, download_directory, organism)
```

retrieving assemblies of biosamble

STEPS

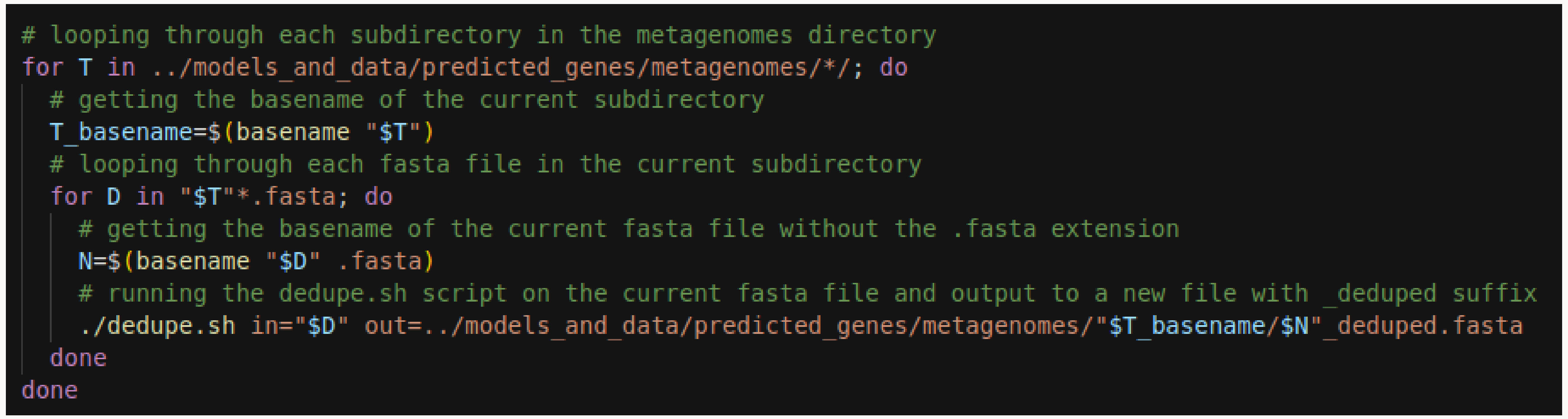
3 - Prodigal to predict genes for contigs from, filtering small ones.



```
orf_finder = pyrodigal.GeneFinder(meta=True)  # initializing gene finder with meta mode
for record in SeqIO.parse(handle, "fasta"):  # parsing the fasta records
    if len(record) > 10000:  # checking if the contig length is greater than 10 kbp
        for i, pred in enumerate(orf_finder.find_genes(bytes(record.seq))):  # predicting genes
            new_fasta.append('>{0}_{1}\n{2}'.format(record.id, i+1, pred.sequence()))  # formatting the gene sequence
```

predict genes with Prodigal

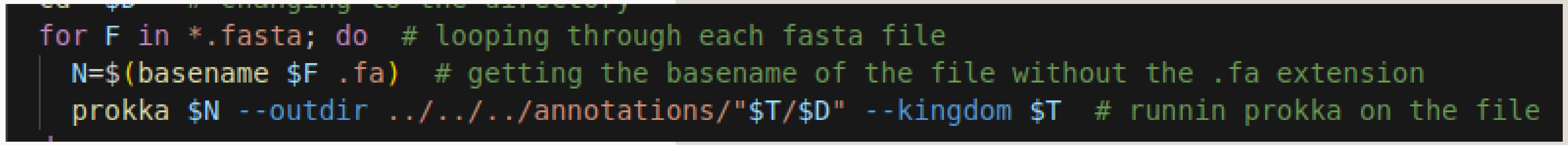
4 - Deduplicate metagenomes with BBMAP dedupe utility



```
# looping through each subdirectory in the metagenomes directory
for T in ../models_and_data/predicted_genes/metagenomes/*/; do
  # getting the basename of the current subdirectory
  T_basename=$(basename "$T")
  # looping through each fasta file in the current subdirectory
  for D in "$T"*.fasta; do
    # getting the basename of the current fasta file without the .fasta extension
    N=$(basename "$D" .fasta)
    # running the dedupe.sh script on the current fasta file and output to a new file with _deduped suffix
    ./dedupe.sh in="$D" out=../models_and_data/predicted_genes/metagenomes/"$T_basename/$N"_deduped.fasta
  done
done
```

deduplicat metagenomes with BBMAP

5 - Deduplications of rest and annotation for everytink with prokka

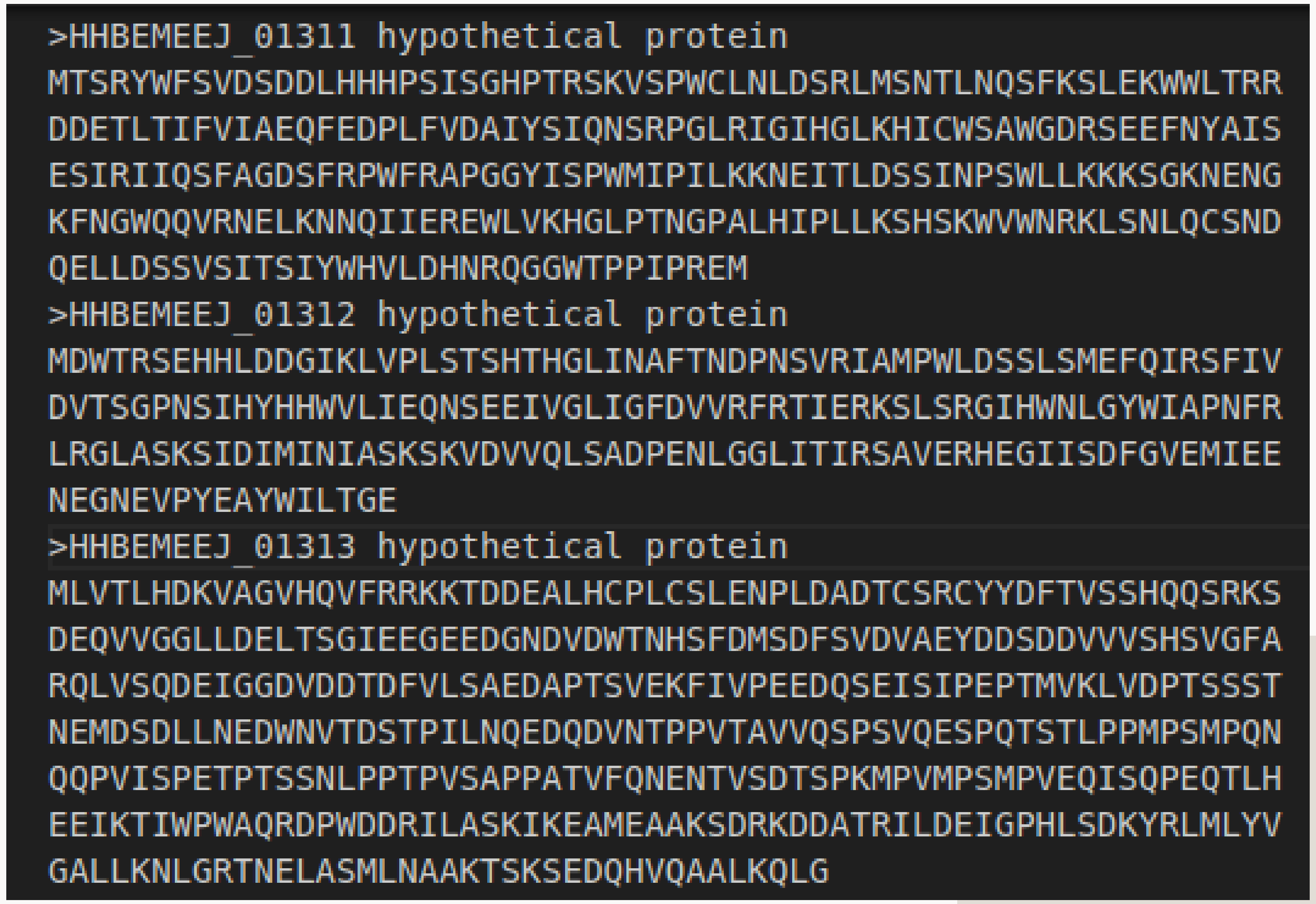


```
for F in *.fasta; do  # looping through each fasta file
  N=$(basename $F .fa)  # getting the basename of the file without the .fa extension
  prokka $N --outdir ../../../annotations/"$T/$D" --kingdom $T  # runnin prokka on the file
```

annotation with Prokka

RESULTING ANNOTATED FILES

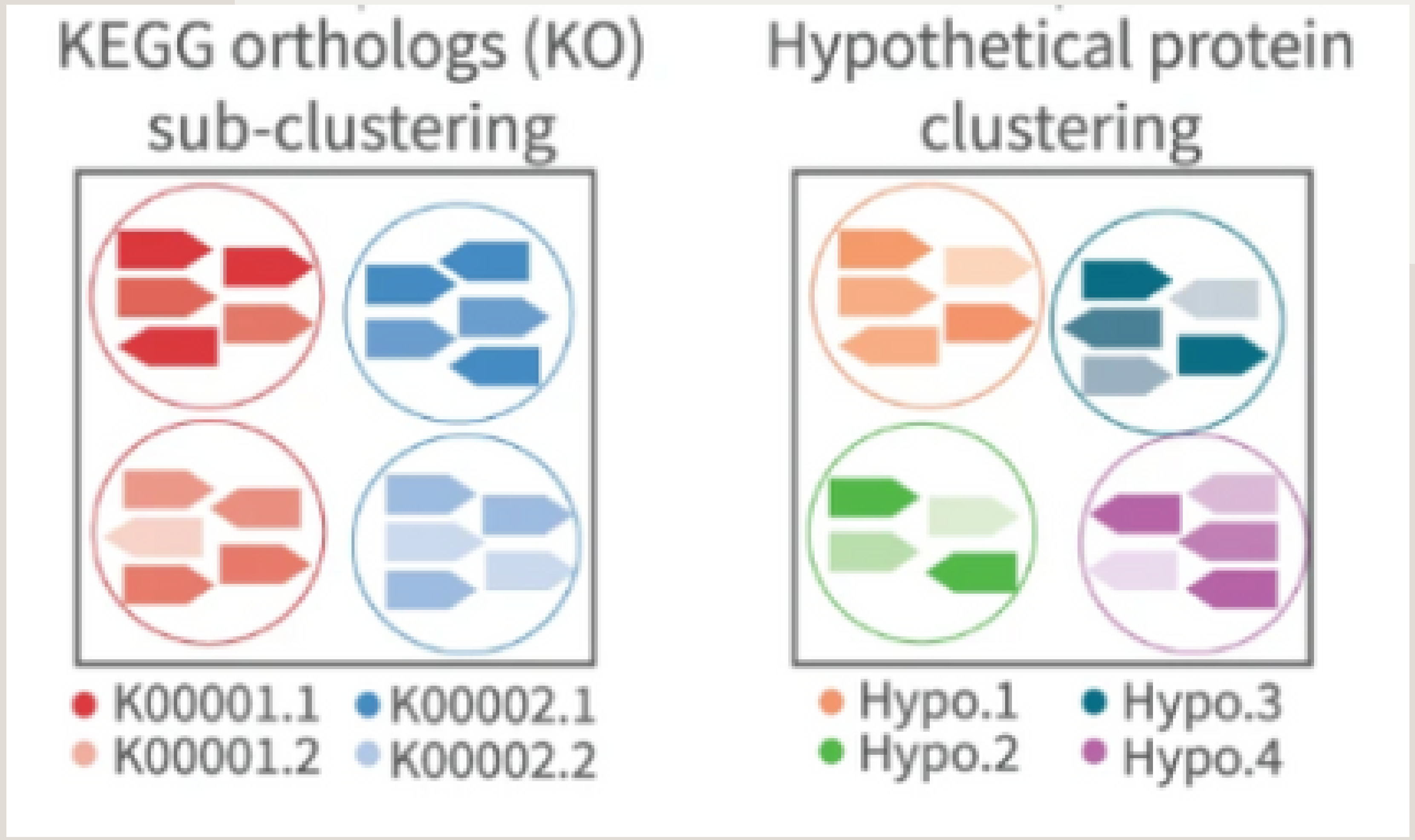
FAA files for each contig in a biosample



```
>HHBEMEEJ_01311 hypothetical protein
MTSRYWFSVDSDDLHHHPSISGHPTRSKVSPWCLNLDSRLMSNTLNQSFKSLEKWWLTRR
DDETLTIFVIAEQFEDPLFVDAIYSIQNSRPGLRIGIHGLKHICWSAWGDRSEEFNYAIS
ESIRIIQSFAGDSFRPWFRAPGGYISPWMIPILKKNEITLDSSINPSWLLKKKSGKNENG
KFNGWQQVRNELKNNQIIEREWLVKHGLPTNGPALHIPLLKSHSKWVWNRKLSNLQCSND
QELLDSSVSITSIYWHVLDHNRQGGWTPPIPREM
>HHBEMEEJ_01312 hypothetical protein
MDWTRSEHHLDDGIKLVPLSTSHTHGLINAFTNDPNSVRIAMPWLDSSLSMEFQIRSFIV
DVTSGPNSIHYHHWVLIEQNSEEIVGLIGFDVVRFRTIERKSLSRGIHWNLGYWIAPNFR
LRGLASKSIDIMINIASKSKVDVVQLSADPENLGGLITIRSAVERHEGIISDFGVEMIEE
NEGNEVPYEAYWILTGE
>HHBEMEEJ_01313 hypothetical protein
MLVTLHDKVAGVHQVFRRKKTDDEALHCPLCSLENPLDADTCSRCYYDFTVSSHQQSRKS
DEQVVGGLLDELTSGIEEGEEDGNDVDWTNHSFDMSDFSVDVAEYDDSDDVVVSHSVGFA
RQLVSQDEIGGDVDDTDFVLSAEDAPTSVEKFIVPEEDQSEISIPEPTMVKLVDPTSSST
NEMDSDLLNEDWNVTDSTPILNQEDQDVNTPPVTAVVQSPSVQESPQTSTLPPMPSMPQN
QQPVISPETPTSSNLPPTPVSAPPATVFQNENTVSDTSPKMPVMPSMPVEQISQPEQTLH
EEIKTIWPWAQRDPWDDRILASKIKEAMEAAKSDRKDDATRILDEIGPHLSDKYRLMLYV
GALLKNLGRTNELASMLNAAKTSKSEDQHVQAALKQLG
```

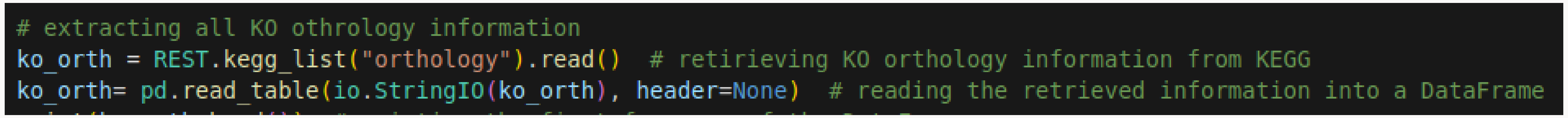
Resulting faa file

1- KO ORTHOLOGY CLUSTERING



STEPS

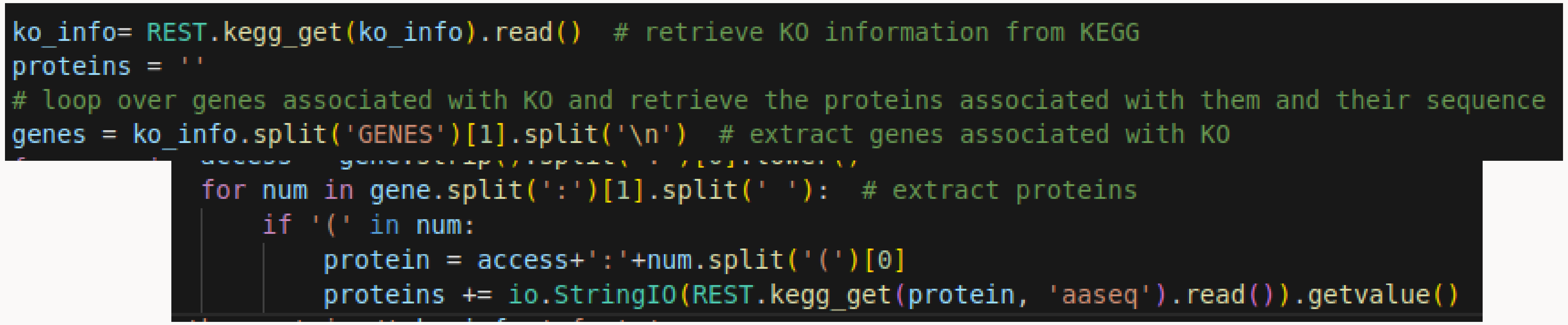
1- Downmoad all KO from KEGG database using using REST module from Bio kegg (biopython)



```
# extracting all KO othrology information
ko_orth = REST.kegg_list("orthology").read()  # retirieving KO orthology information from KEGG
ko_orth= pd.read_table(io.StringIO(ko_orth), header=None)  # reading the retrieved information into a DataFrame
```

download KO terms

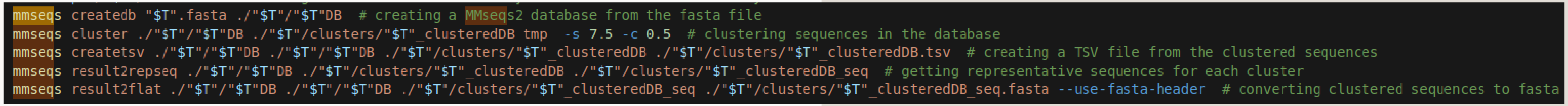
2- Get the list of proteins associated with each ko number.



```
ko_info= REST.kegg_get(ko_info).read()  # retrieve KO information from KEGG
proteins = ''
# loop over genes associated with KO and retrieve the proteins associated with them and their sequence
genes = ko_info.split('GENES')[1].split('\n')  # extract genes associated with KO
    for num in gene.split(':')[1].split(' '):  # extract proteins
        if '(' in num:
            protein = access+':'+num.split('(')[0]
            proteins += io.StringIO(REST.kegg_get(protein, 'aaseq').read()).getvalue()
```

scrape protein information

3- Subcluster each KO cluster into smaller groups usng mmseq2

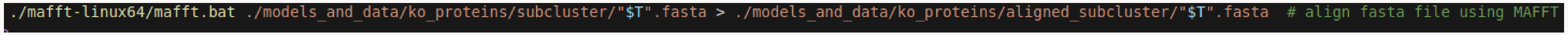


```
mmseqs createdb "$T".fasta ./"$T"/"$T"DB  # creating a MMseqs2 database from the fasta file
mmseqs cluster ./"$T"/"$T"DB ./"$T"/clusters/"$T"_clusteredDB tmp  -s 7.5 -c 0.5  # clustering sequences in the database
mmseqs createtsv ./"$T"/"$T"DB ./"$T"/"$T"DB ./"$T"/clusters/"$T"_clusteredDB ./"$T"/clusters/"$T"_clusteredDB.tsv  # creating a TSV file from the clustered sequences
mmseqs result2repseq ./"$T"/"$T"DB ./"$T"/clusters/"$T"_clusteredDB ./"$T"/clusters/"$T"_clusteredDB_seq  # getting representative sequences for each cluster
mmseqs result2flat ./"$T"/"$T"DB ./"$T"/"$T"DB ./"$T"/clusters/"$T"_clusteredDB_seq ./"$T"/clusters/"$T"_clusteredDB_seq.fasta --use-fasta-header  # converting clustered sequences to fasta
```

mmseq subclustering

STEPS

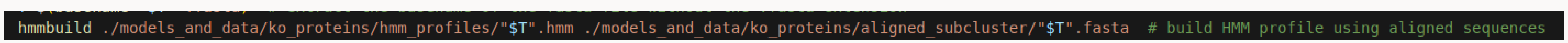
4 - Aligning subcluster with more than five KEGG proteins using MAFFT



```
./mafft-linux64/mafft.bat ./models_and_data/ko_proteins/subcluster/"$T".fasta > ./models_and_data/ko_proteins/aligned_subcluster/"$T".fasta  # align fasta file using MAFFT
```

Alignment with MAFFT

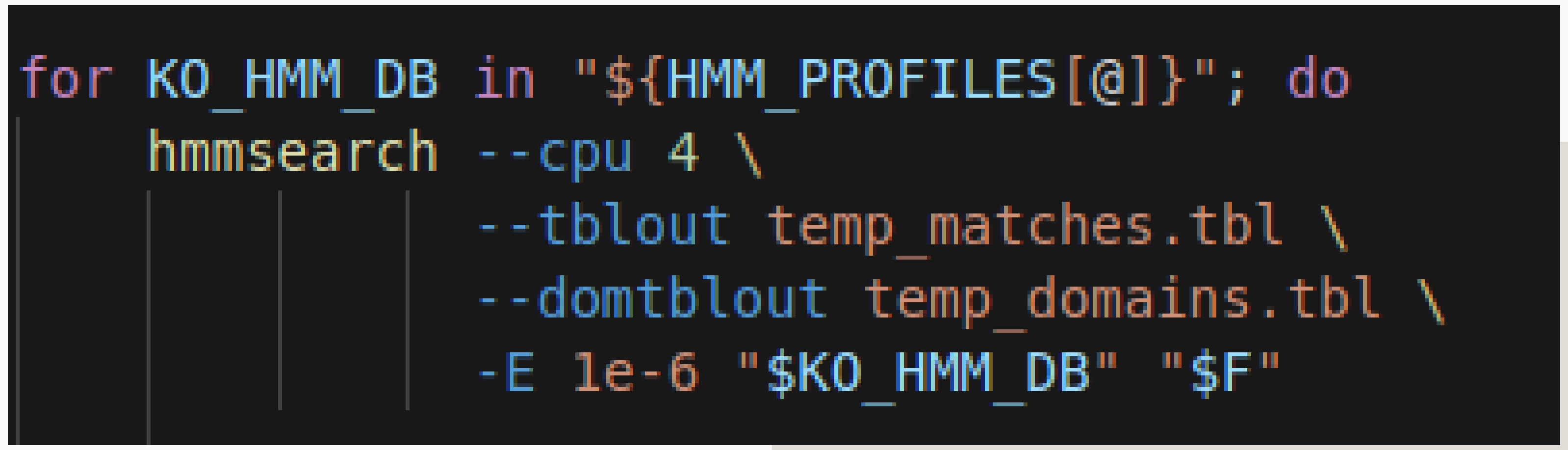
5 - Construction of profile HMM with HMMer suite



```
hmmbuild ./models_and_data/ko_proteins/hmm_profiles/"$T".hmm ./models_and_data/ko_proteins/aligned_subcluster/"$T".fasta  # build HMM profile using aligned sequences
```

HMM profile construction

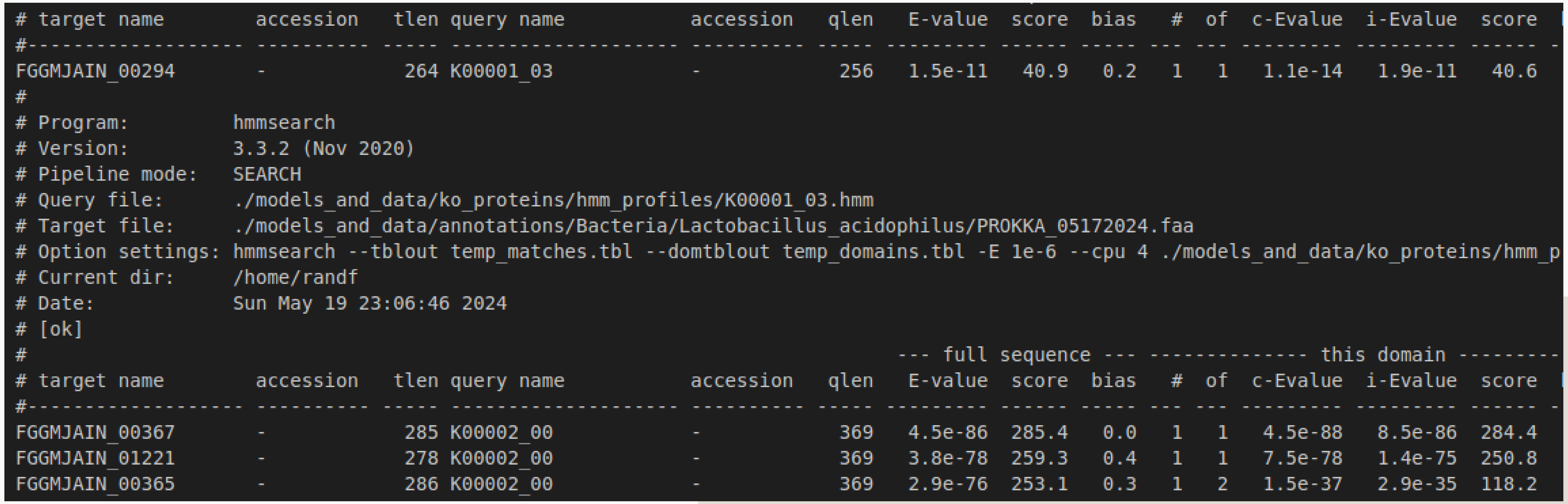
5 - Searching proteins against hmm profile



```
for KO_HMM_DB in "${HMM_PROFILES[@]}"; do
    hmmsearch --cpu 4 \
              --tblout temp_matches.tbl \
              --domtblout temp_domains.tbl \
              -E 1e-6 "$KO_HMM_DB" "$F"
```

Matching proteins with HMM database

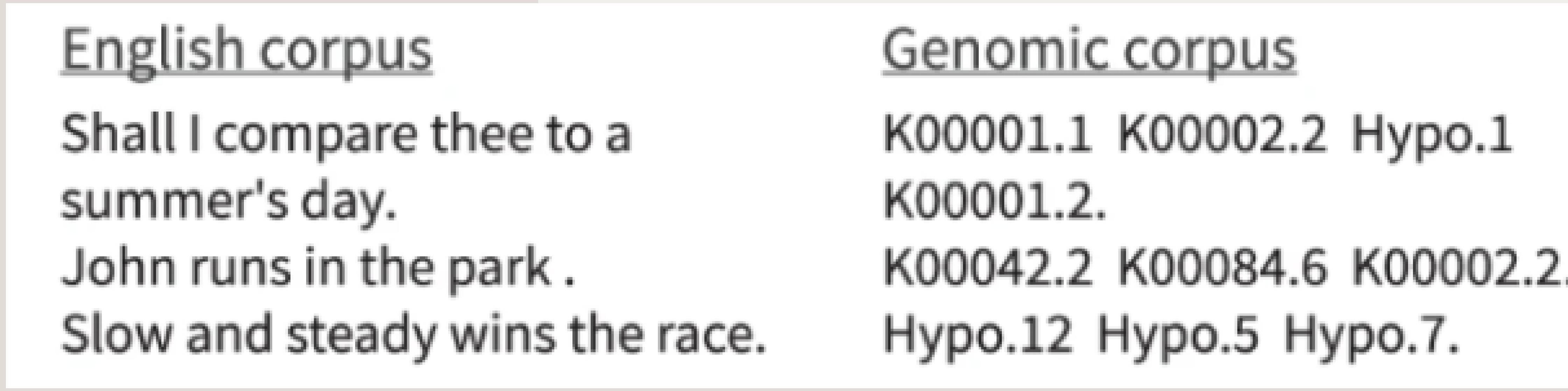
RESULTING HMM MATCH FILES



```
# target name        accession   tlen query name           accession   qlen   E-value  score  bias   #  of  c-Evalue  i-Evalue  score
#------------------- ---------- ----- -------------------- ---------- ----- --------- ------ ----- --- --- --------- --------- ------ -
FGGMJAIN_00294       -            264 K00001_03            -            256   1.5e-11   40.9   0.2   1   1   1.1e-14   1.9e-11   40.6
#
# Program:         hmmsearch
# Version:         3.3.2 (Nov 2020)
# Pipeline mode:   SEARCH
# Query file:      ./models_and_data/ko_proteins/hmm_profiles/K00001_03.hmm
# Target file:     ./models_and_data/annotations/Bacteria/Lactobacillus_acidophilus/PROKKA_05172024.faa
# Option settings: hmmsearch --tblout temp_matches.tbl --domtblout temp_domains.tbl -E 1e-6 --cpu 4 ./models_and_data/ko_proteins/hmm_p
# Current dir:     /home/randf
# Date:            Sun May 19 23:06:46 2024
# [ok]
#                                                                            --- full sequence --- -------------- this domain ---------
# target name        accession   tlen query name           accession   qlen   E-value  score  bias   #  of  c-Evalue  i-Evalue  score
#------------------- ---------- ----- -------------------- ---------- ----- --------- ------ ----- --- --- --------- --------- ------ -
FGGMJAIN_00367       -            285 K00002_00            -            369   4.5e-86  285.4   0.0   1   1   4.5e-88   8.5e-86  284.4
FGGMJAIN_01221       -            278 K00002_00            -            369   3.8e-78  259.3   0.4   1   1   7.5e-78   1.4e-75  250.8
FGGMJAIN_00365       -            286 K00002_00            -            369   2.9e-76  253.1   0.3   1   2   1.5e-37   2.9e-35  118.2
```

Results off HMM matching

3- CORPUS GENERATION



English corpus

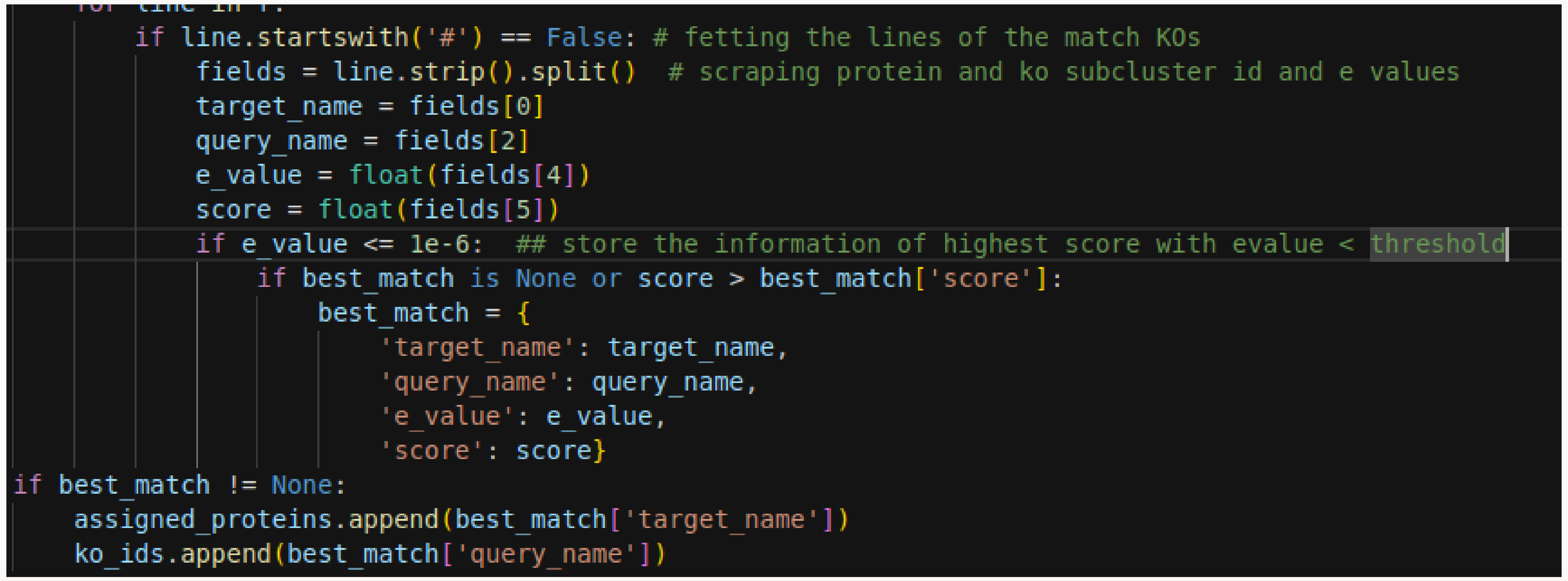
Shall I compare thee to a summer's day.
John runs in the park .
Slow and steady wins the race.

Genomic corpus

K00001.1 K00002.2 Hypo.1 K00001.2.
K00042.2 K00084.6 K00002.2.
Hypo.12 Hypo.5 Hypo.7.

STEPS

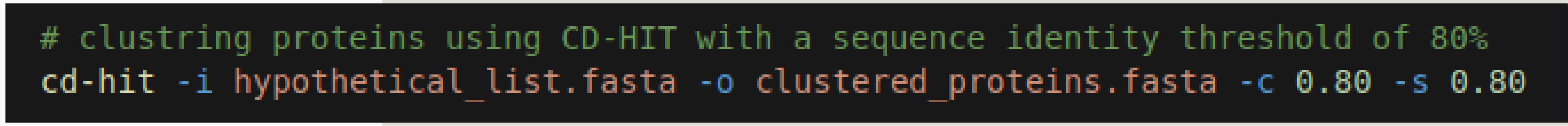
1- proteins significantly matching a KO HMM (E-value threshold of 10−6) assigned to the best scoring subcluster



```
        if line.startswith('#') == False: # fetting the lines of the match KOs
            fields = line.strip().split()  # scraping protein and ko subcluster id and e values
            target_name = fields[0]
            query_name = fields[2]
            e_value = float(fields[4])
            score = float(fields[5])
            if e_value <= 1e-6:  ## store the information of highest score with evalue < threshold
                if best_match is None or score > best_match['score']:
                    best_match = {
                        'target_name': target_name,
                        'query_name': query_name,
                        'e_value': e_value,
                        'score': score}
if best_match != None:
    assigned_proteins.append(best_match['target_name'])
    ko_ids.append(best_match['query_name'])
```

KO HMM matching

2- unassigned proteins and their sequences add in one fasta file then clustered using cd hit

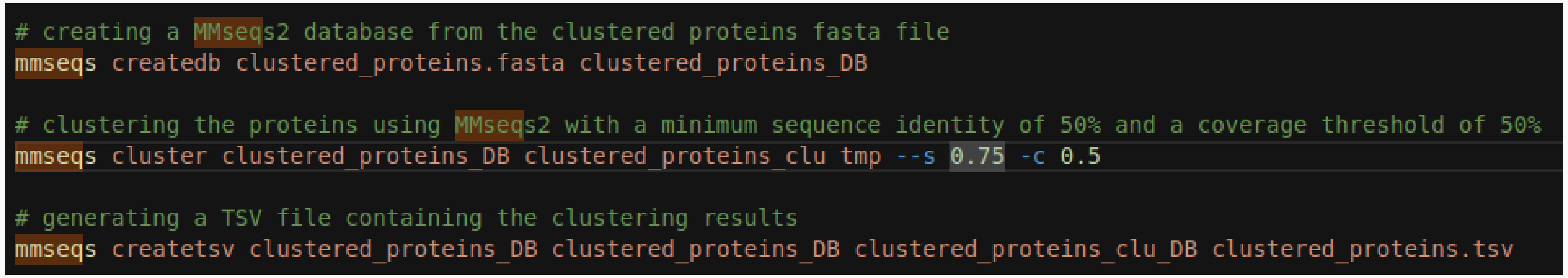


```
# clustring proteins using CD-HIT with a sequence identity threshold of 80%
cd-hit -i hypothetical_list.fasta -o clustered_proteins.fasta -c 0.80 -s 0.80
```

cd-hit clustering

STEPS

3- Reecluster with mmseq2 and assigning to unmatched proteins to their subclusters



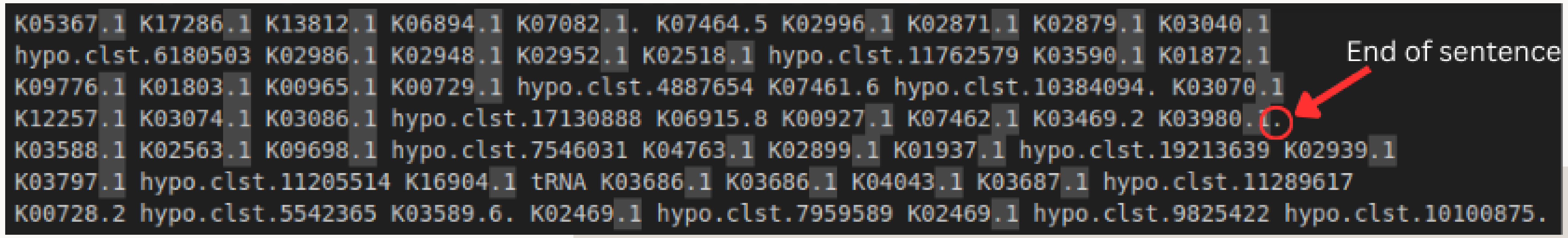
```
# creating a MMseqs2 database from the clustered proteins fasta file
mmseqs createdb clustered_proteins.fasta clustered_proteins_DB

# clustering the proteins using MMseqs2 with a minimum sequence identity of 50% and a coverage threshold of 50%
mmseqs cluster clustered_proteins_DB clustered_proteins_clu tmp --s 0.75 -c 0.5

# generating a TSV file containing the clustering results
mmseqs createtsv clustered_proteins_DB clustered_proteins_DB clustered_proteins_clu_DB clustered_proteins.tsv
```

mmseq clustering

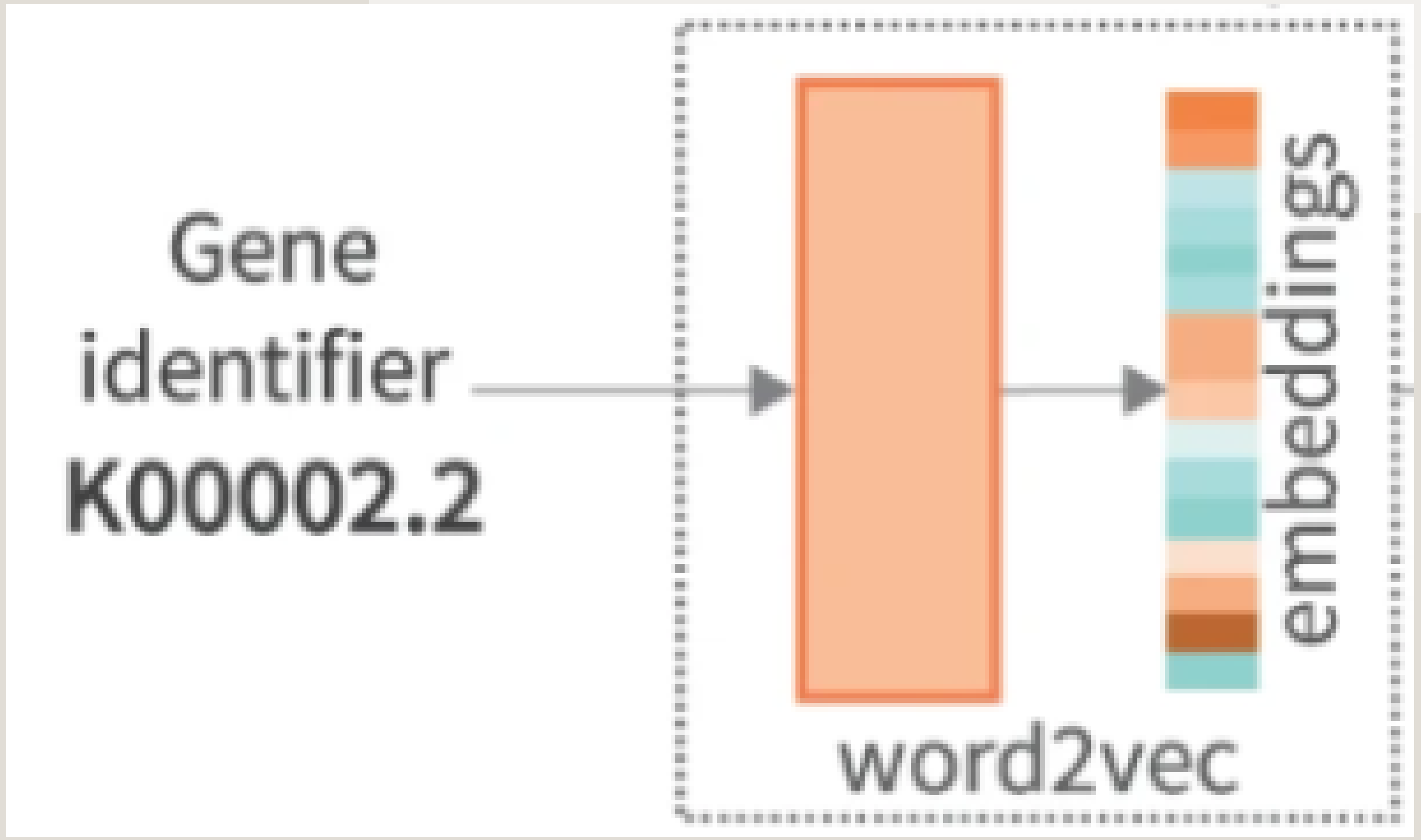
4 - for each biosample, a contig is a sentence and proteins encoded in it are words (their subclusters).



```
K05367.1 K17286.1 K13812.1 K06894.1 K07082.1. K07464.5 K02996.1 K02871.1 K02879.1 K03040.1
hypo.clst.6180503 K02986.1 K02948.1 K02952.1 K02518.1 hypo.clst.11762579 K03590.1 K01872.1
K09776.1 K01803.1 K00965.1 K00729.1 hypo.clst.4887654 K07461.6 hypo.clst.10384094. K03070.1
K12257.1 K03074.1 K03086.1 hypo.clst.17130888 K06915.8 K00927.1 K07462.1 K03469.2 K03980.1.
K03588.1 K02563.1 K09698.1 hypo.clst.7546031 K04763.1 K02899.1 K01937.1 hypo.clst.19213639 K02939.1
K03797.1 hypo.clst.11205514 K16904.1 tRNA K03686.1 K03686.1 K04043.1 K03687.1 hypo.clst.11289617
K00728.2 hypo.clst.5542365 K03589.6. K02469.1 hypo.clst.7959589 K02469.1 hypo.clst.9825422 hypo.clst.10100875.
```

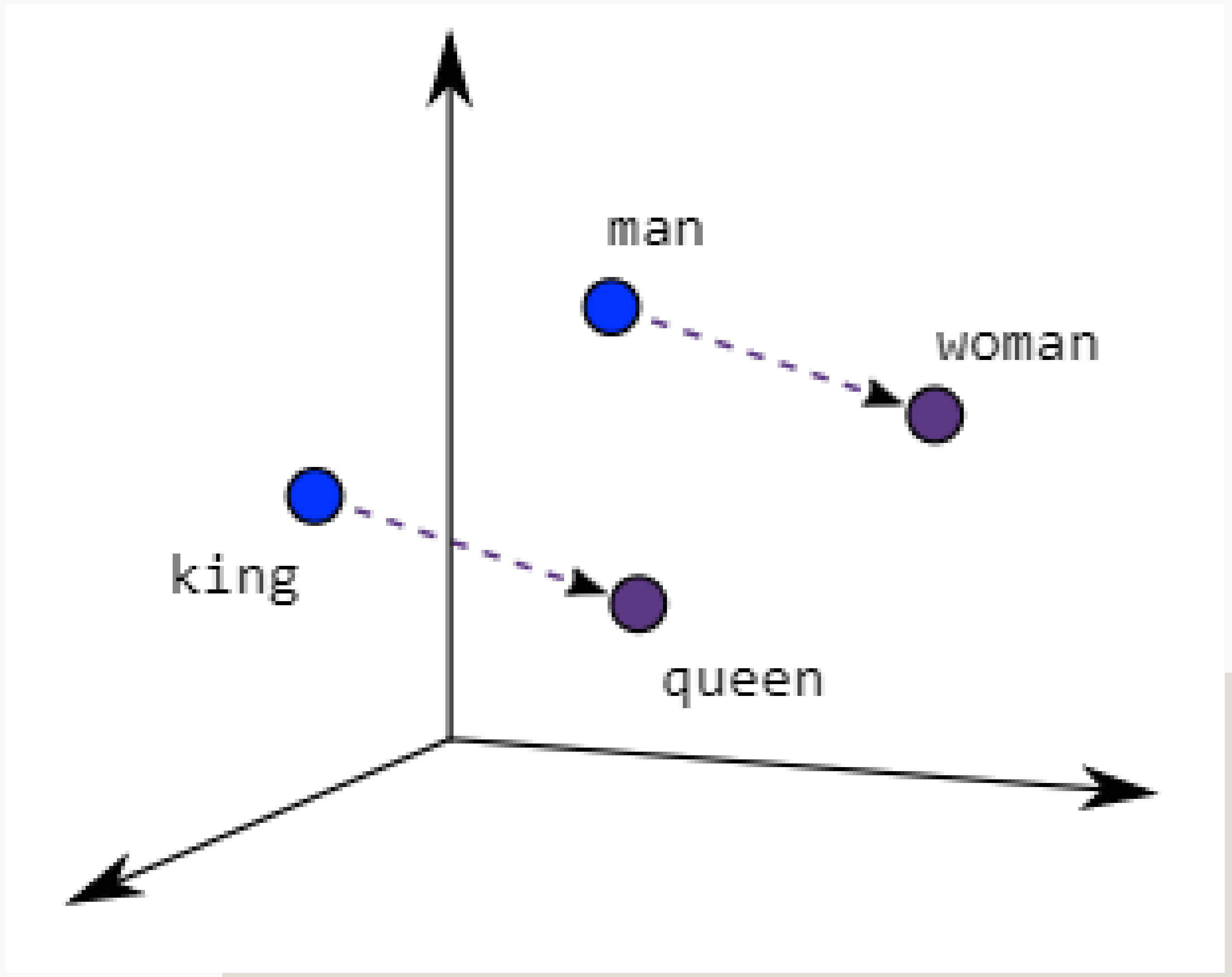
Corpus shape

4- WORD EMBEDDINGS



WORD2VEC

- vector representations of words
- detect contextual meaning

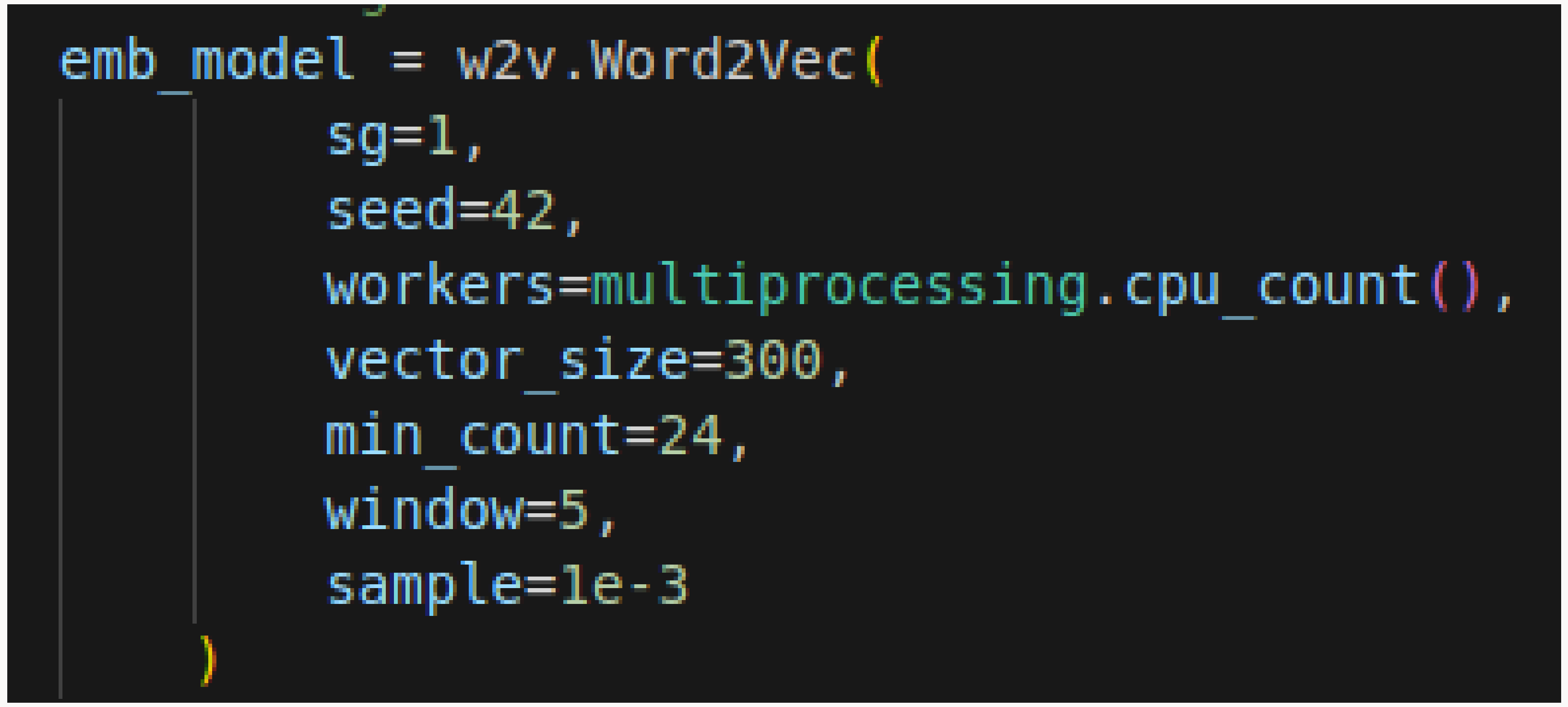


Word2vec vectors: https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/

STEPS

1- words with low frequency frequency are filtered

2- word2vec is trained on the remaining word

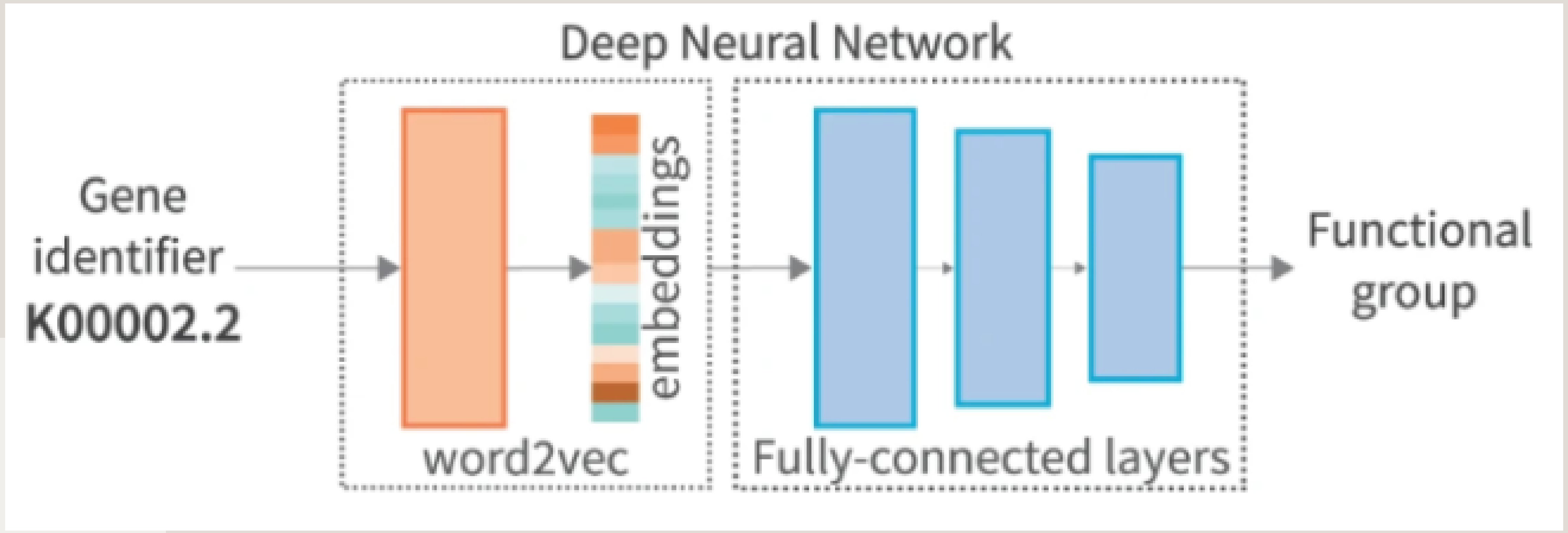


```
emb_model = w2v.Word2Vec(
            sg=1,
            seed=42,
            workers=multiprocessing.cpu_count(),
            vector_size=300,
            min_count=24,
            window=5,
            sample=1e-3
      )
```

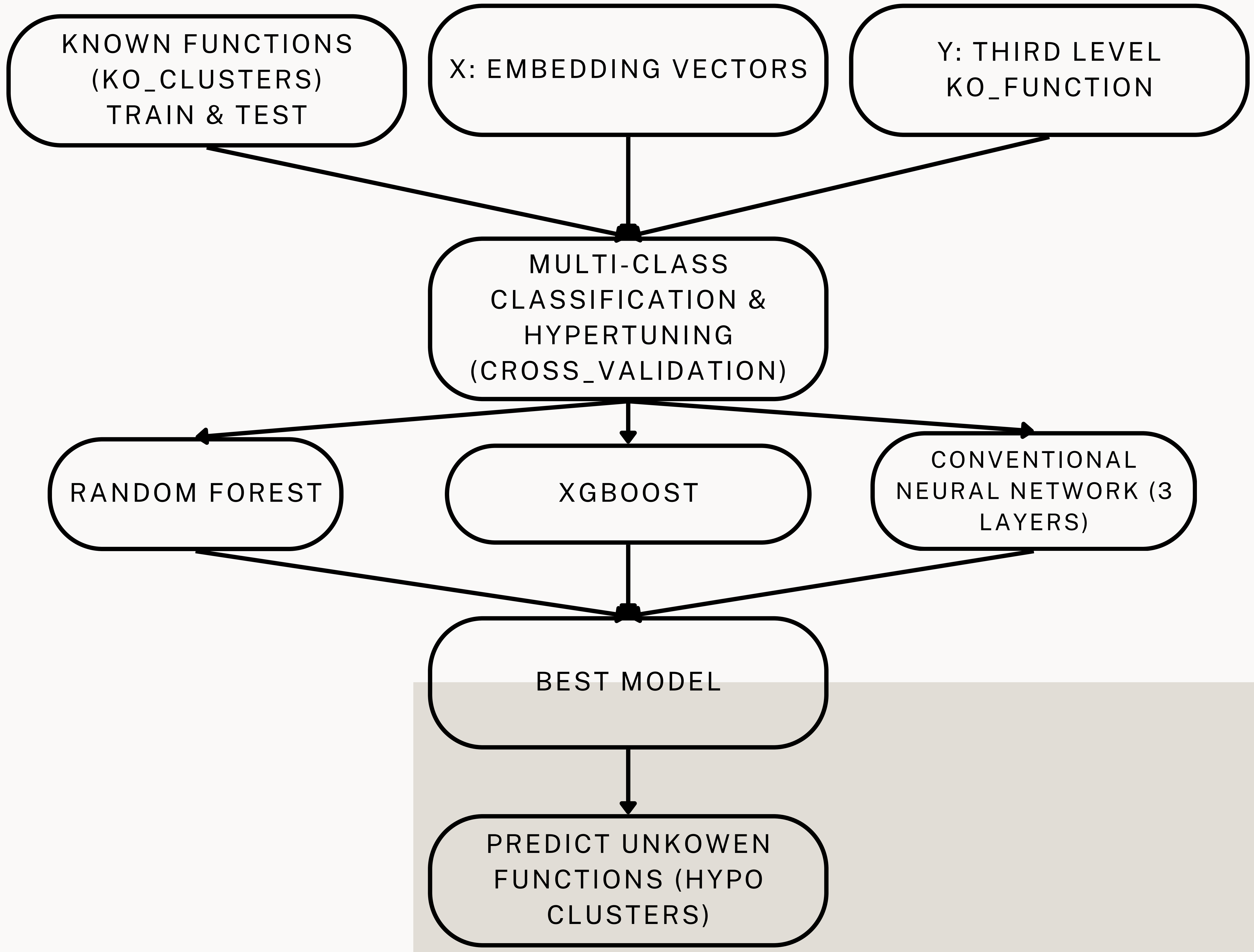
Word2vec model

VOCABULARY SIZE: 563841

5- CLASSIFICATION



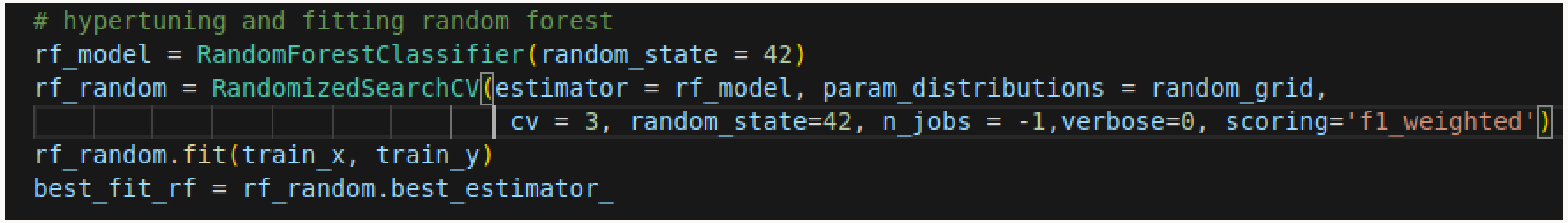
WORKFLOW


KNOWN FUNCTIONS (KO_CLUSTERS) TRAIN & TEST
X: EMBEDDING VECTORS
Y: THIRD LEVEL KO_FUNCTION
MULTI-CLASS CLASSIFICATION & HYPERTUNING (CROSS_VALIDATION)
RANDOM FOREST
XGBOOST
CONVENTIONAL NEURAL NETWORK (3 LAYERS)
BEST MODEL
PREDICT UNKOWEN FUNCTIONS (HYPO CLUSTERS)

Workflow of the classification pipeline

MODELS

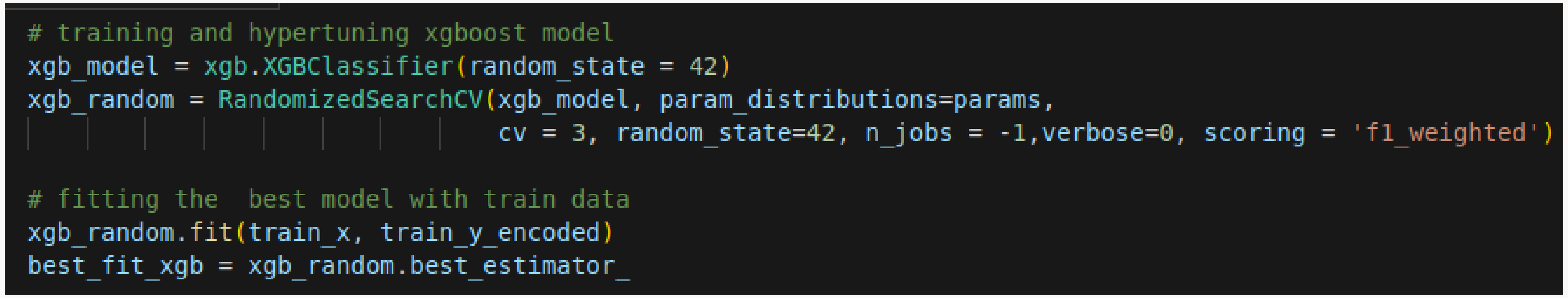
1- Random forest



```
# hypertuning and fitting random forest
rf_model = RandomForestClassifier(random_state = 42)
rf_random = RandomizedSearchCV(estimator = rf_model, param_distributions = random_grid,
                                cv = 3, random_state=42, n_jobs = -1,verbose=0, scoring='f1_weighted')
rf_random.fit(train_x, train_y)
best_fit_rf = rf_random.best_estimator_
```

hypertuning and fitting random forest

2- XGBoost

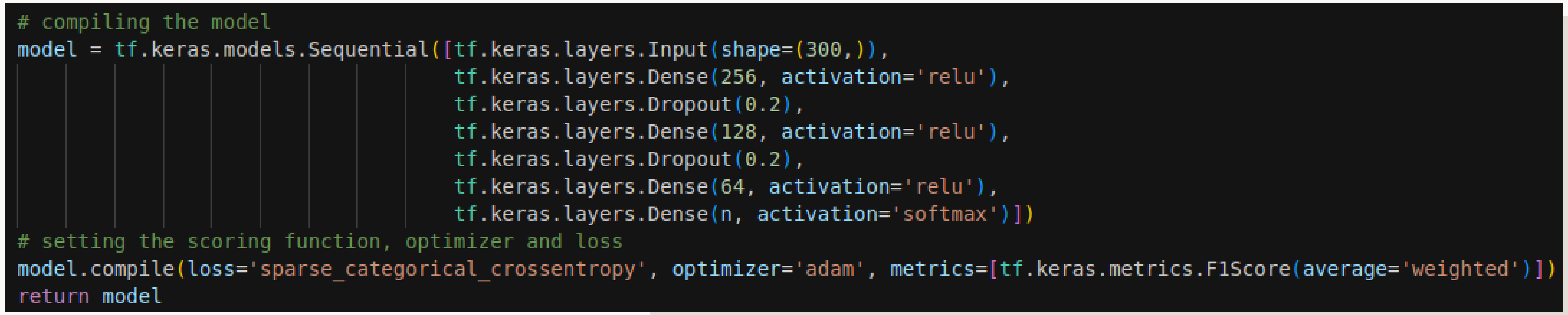


```
# training and hypertuning xgboost model
xgb_model = xgb.XGBClassifier(random_state = 42)
xgb_random = RandomizedSearchCV(xgb_model, param_distributions=params,
                                cv = 3, random_state=42, n_jobs = -1,verbose=0, scoring = 'f1_weighted')

# fitting the  best model with train data
xgb_random.fit(train_x, train_y_encoded)
best_fit_xgb = xgb_random.best_estimator_
```

hypertuning and fitting xgboostt

3- CNN



```
# compiling the model
model = tf.keras.models.Sequential([tf.keras.layers.Input(shape=(300,)),
                                    tf.keras.layers.Dense(256, activation='relu'),
                                    tf.keras.layers.Dropout(0.2),
                                    tf.keras.layers.Dense(128, activation='relu'),
                                    tf.keras.layers.Dropout(0.2),
                                    tf.keras.layers.Dense(64, activation='relu'),
                                    tf.keras.layers.Dense(n, activation='softmax')])
# setting the scoring function, optimizer and loss
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=[tf.keras.metrics.F1Score(average='weighted')])
return model
```

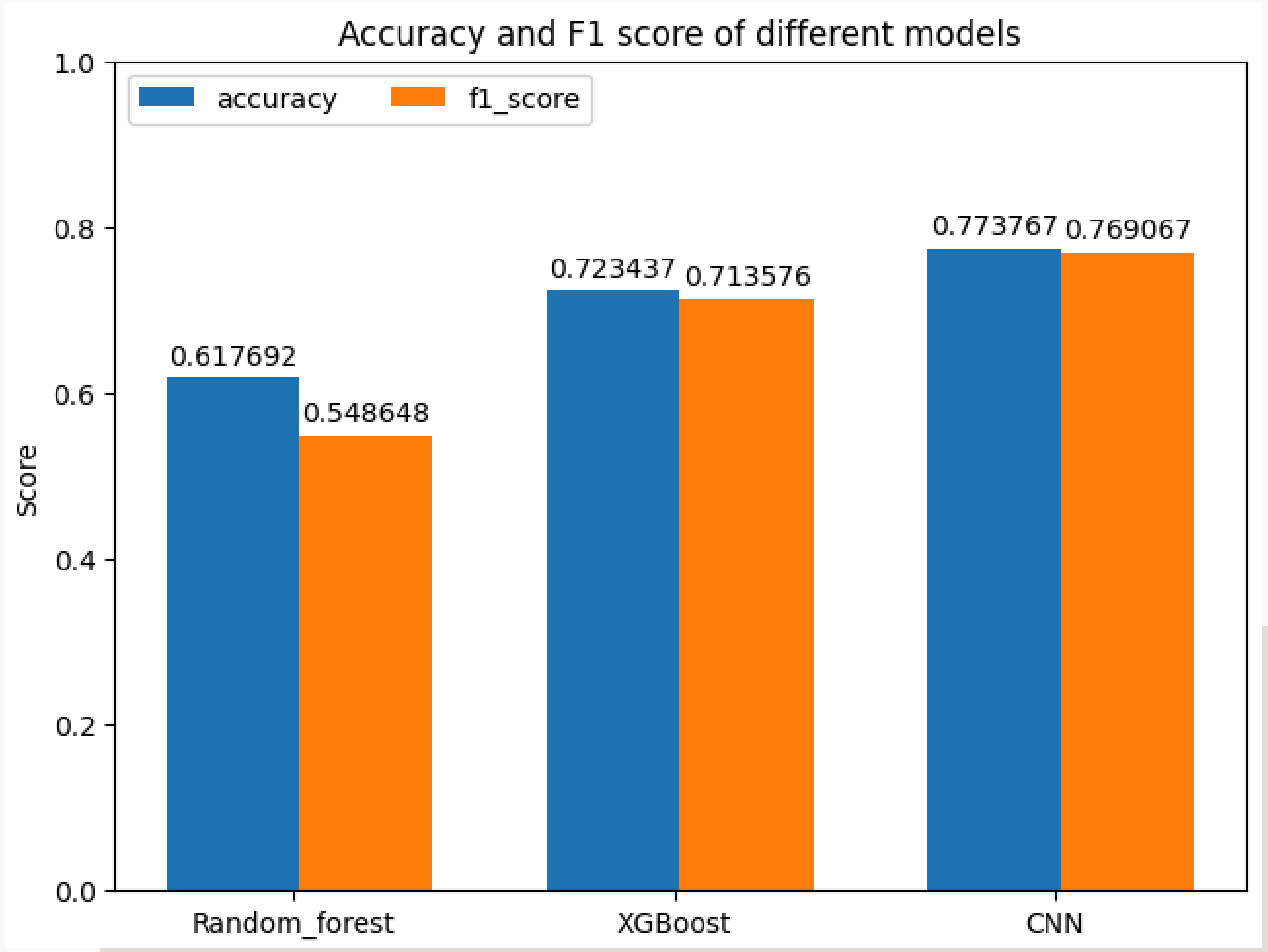
hypertuning and fitting cnn

RESULTS

BEST MODEL

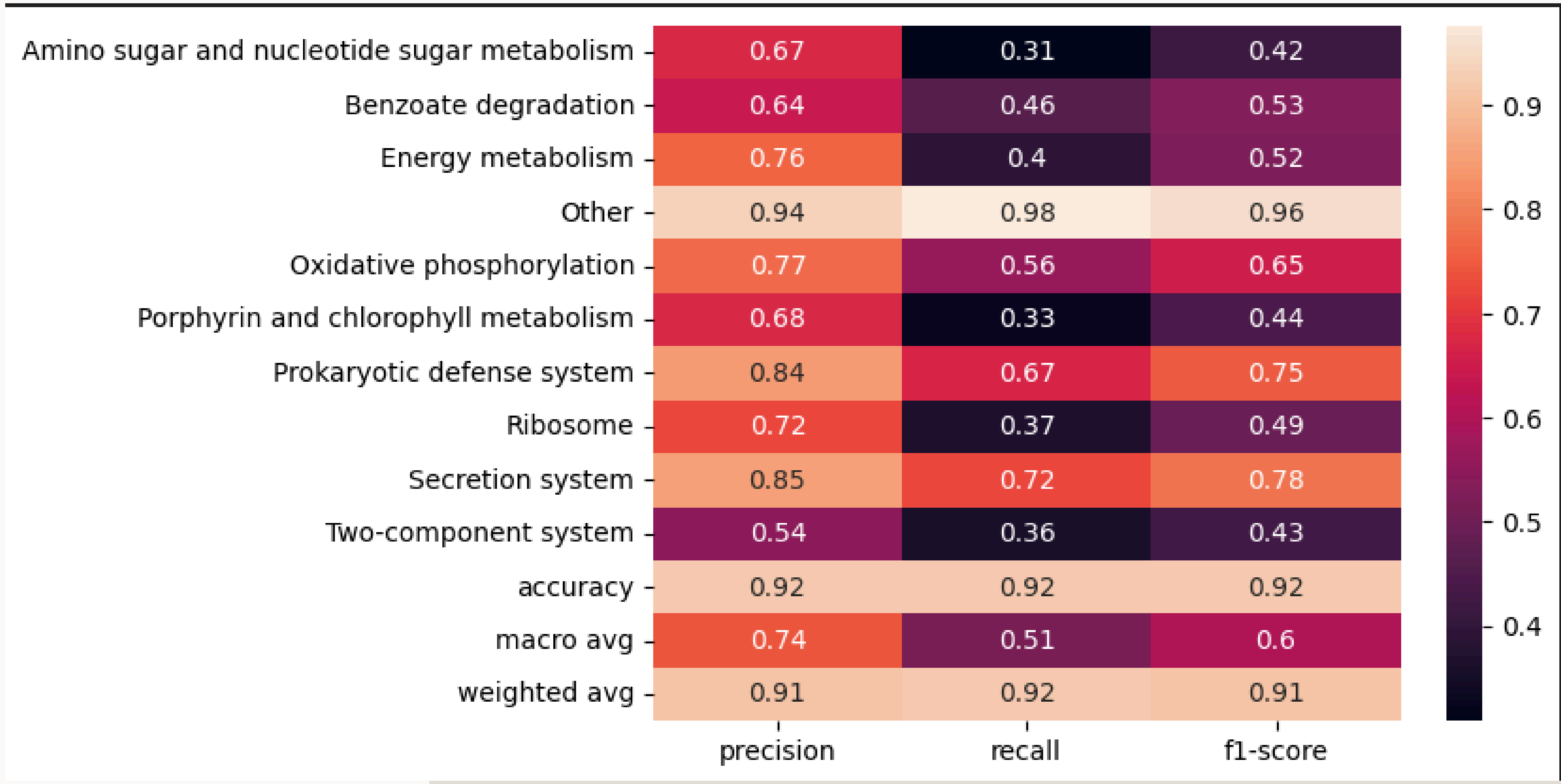
Best model: CNN
Accuracy: 0.77%
Scoring function: F1
Loss: sparse_categorical_crossentropy
Structure: 3 layers, relu activation, softmax for output

high F1 score: balanced performance across classes



Accuracy and F1 score of different models

BEST MODEL



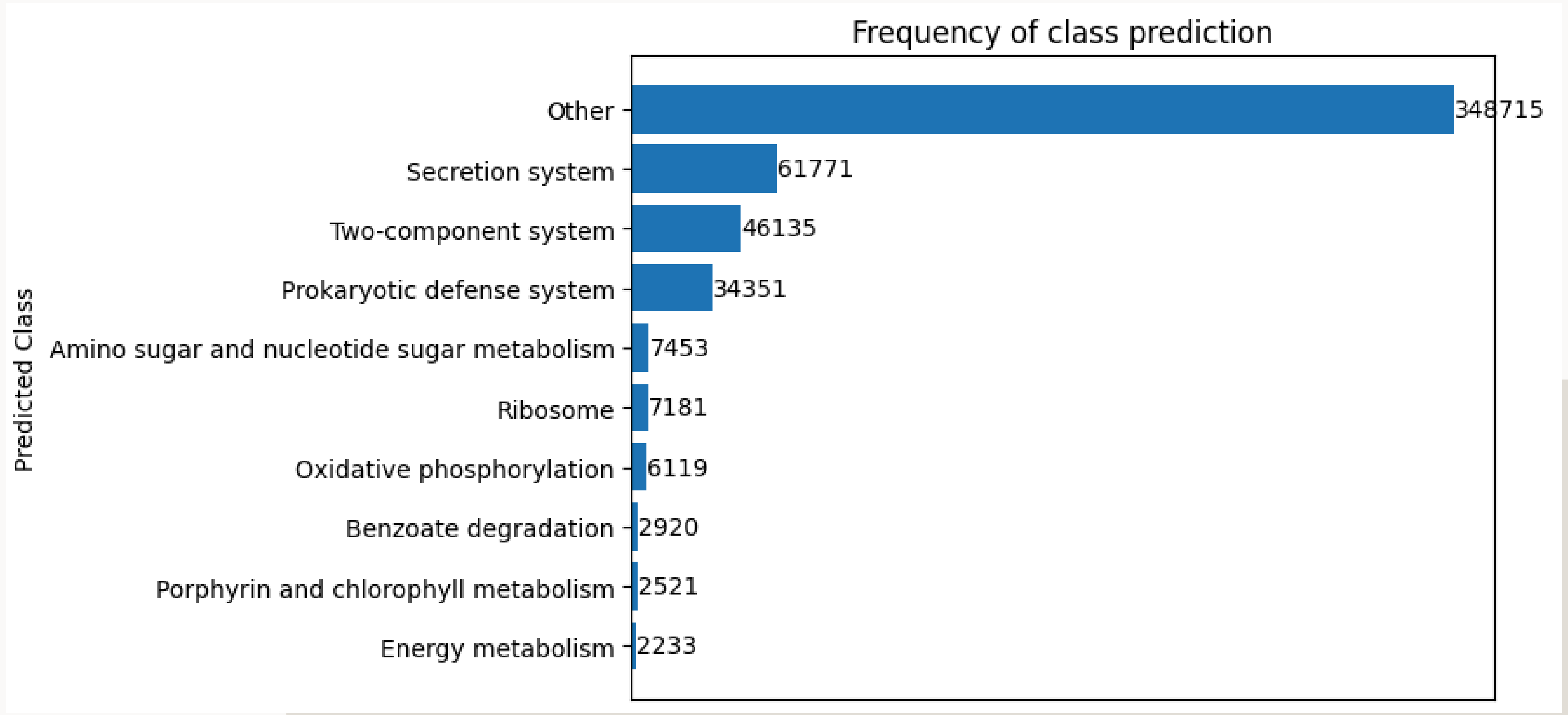
	precision	recall	f1-score
Amino sugar and nucleotide sugar metabolism	0.67	0.31	0.42
Benzoate degradation	0.64	0.46	0.53
Energy metabolism	0.76	0.4	0.52
Other	0.94	0.98	0.96
Oxidative phosphorylation	0.77	0.56	0.65
Porphyrin and chlorophyll metabolism	0.68	0.33	0.44
Prokaryotic defense system	0.84	0.67	0.75
Ribosome	0.72	0.37	0.49
Secretion system	0.85	0.72	0.78
Two-component system	0.54	0.36	0.43
accuracy	0.92	0.92	0.92
macro avg	0.74	0.51	0.6
weighted avg	0.91	0.92	0.91

Best model metrics across classes

Best metrics: Prokaryotic defense system and secretion system

PREDICTION OF HYPOTHETICAL CLUSTERS

Majority: others
Secretion systems: > 61K
Two component system: >46k
Prokaryotic defense systeme: >34k



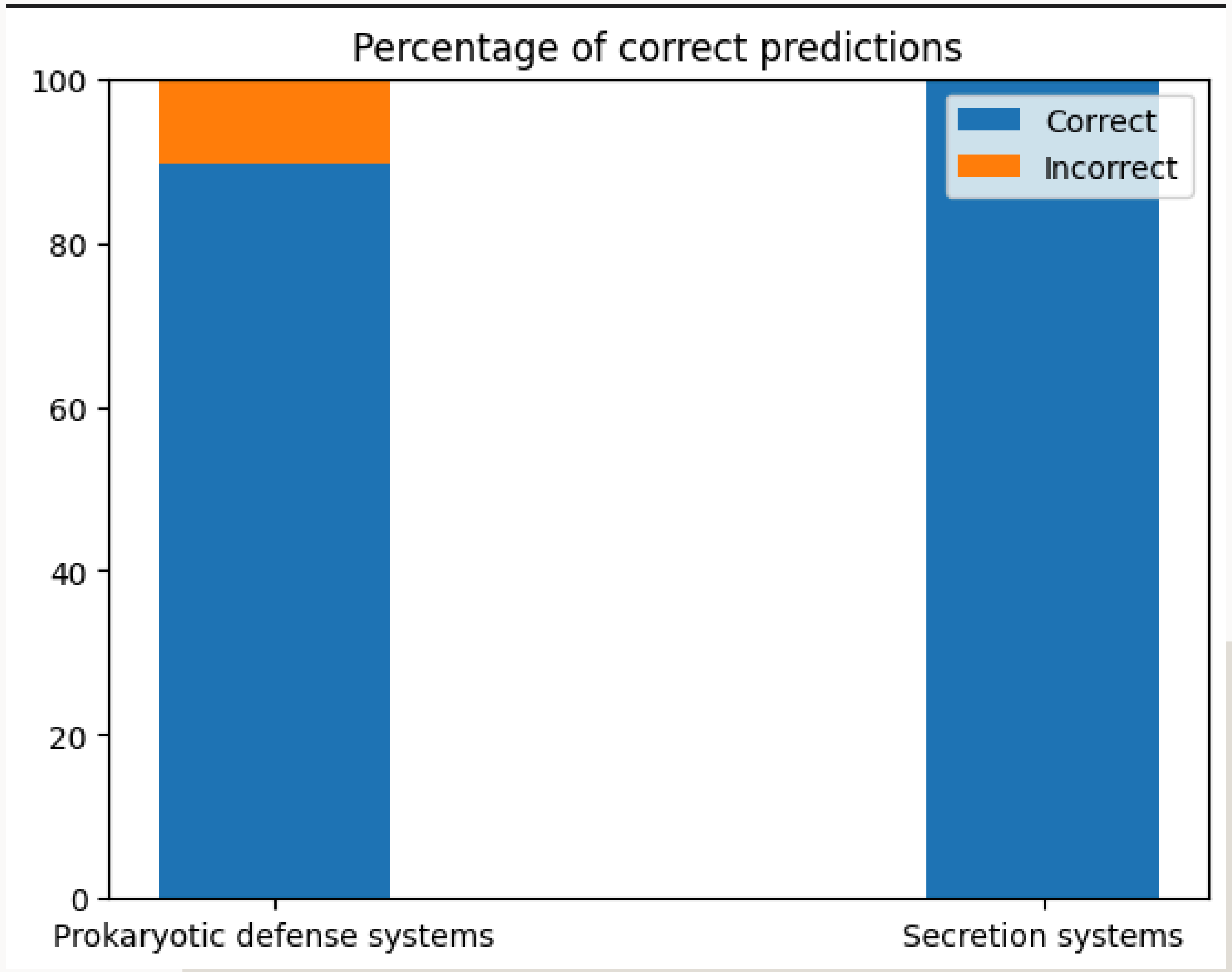
Frequency of hypoyhetical class prediction

TESTING THE PREDICTION

- genes with newly discovere function not yet annotated in databases.

1- Prokaryotic defense systems: 413 genes; 90% correctly predicted.

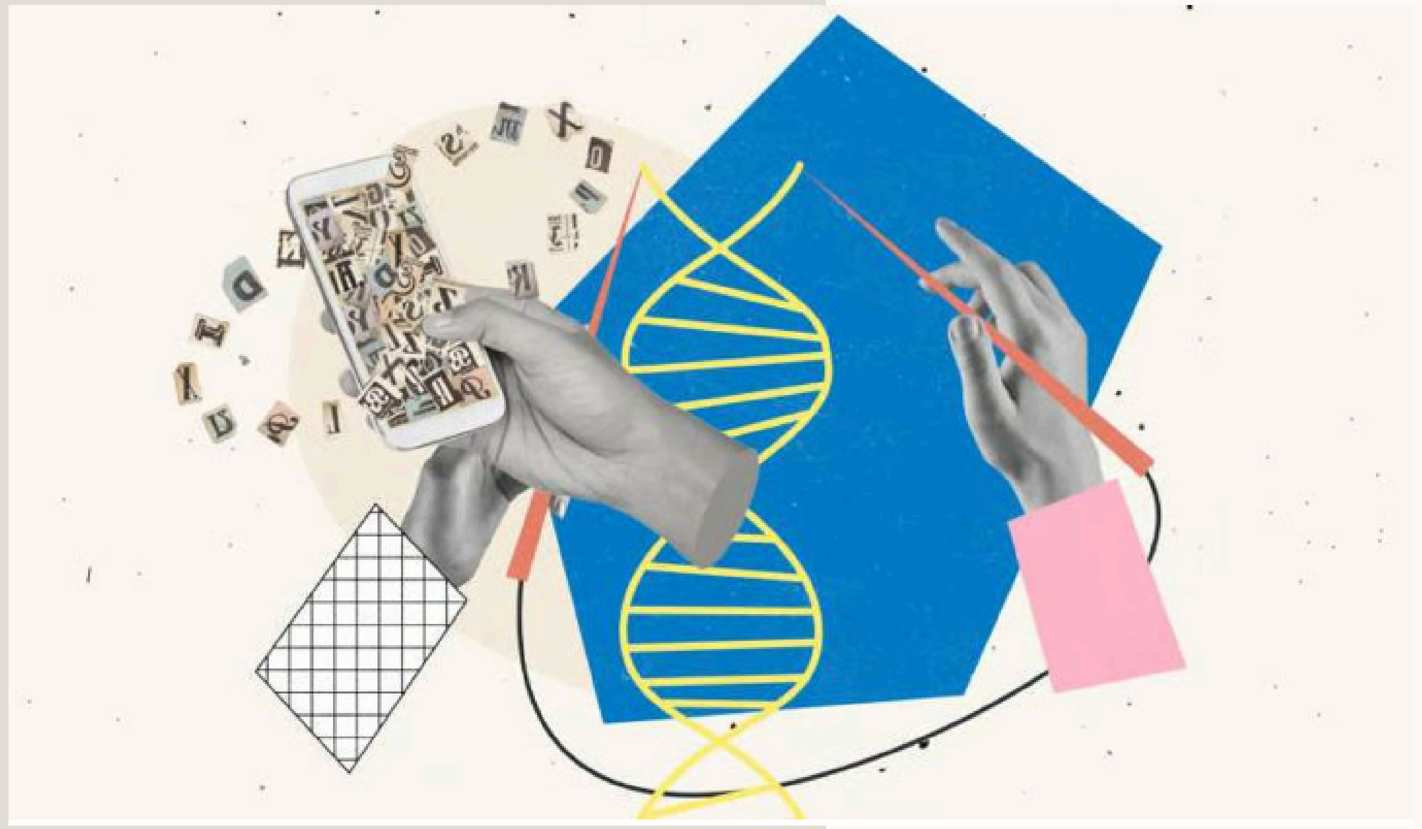
2- Secretion systems: 3201 genes, 100% correctly predicted.



Percentage of correctly predicted finctions for newly discovered genes

DISCUSSION

- Using NLP techniques, it was possible to build a classification model to predict gene function based on it context.
- Systems whose genes are known to co-occur, abundant, and has its genes clusteres in one to KOs were better predicted.


https://machinelearninginterview.com/topics/natural-language-processing/what-is-the-difference-between-word2vec-and-glove/

PERSPECTIVES

- Testing on more data for various classes
- Trying different techniques like Glove for embedding since it studies ‘words’ in the context of the whole corpus.
- revsiting preprocessing steps, especially filtering and threshold, and clusturing

THANK YOU FOR LISTENING!