

# StackExchange Prediction and Analysis

ADM stackexchange project

Mateusz Mazur, Tomasz Makowski, Tomasz Kawiak

December 2, 2025

# Table of contents



**1** EDA

**2** TAG Embeddings

**3** TAG 2

**4** Tag Prediction

# EDA

- As of now, we have collected a dataset containing 100,000 questions from StackExchange (specifically StackOverflow), along with their associated metadata.

- As stated earlier, the dataset has the following features:

#	Column	Dtype
0	title	object
1	has_accepted_answer	bool
2	accepted_answer_score	float64
3	time_to_accepted_answer_hours	float64
4	question_score	int64
5	question_text	object
6	num_tags	int64
7	tags	object
8	accepted_answer_id	float64
9	accepted_answer_length_chars	float64
10	accepted_answer_length_tokens	float64

## Deduplication

- Out of these questions, 8 were dropped for being duplicates.

## Acceptance Analysis

- From those remaining, 39938 questions have an accepted answer, while 60054 do not.
- Interestingly, out of those accepted answers, only 12000 have `time_to_accepted_answer_hours` defined (i.e., non-null values).

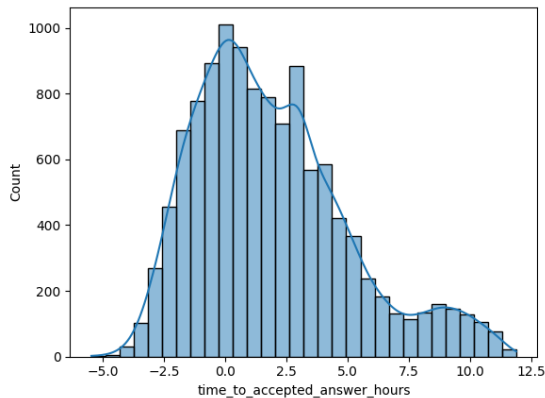


Figure 1: Histogram of log of time to accepted answer

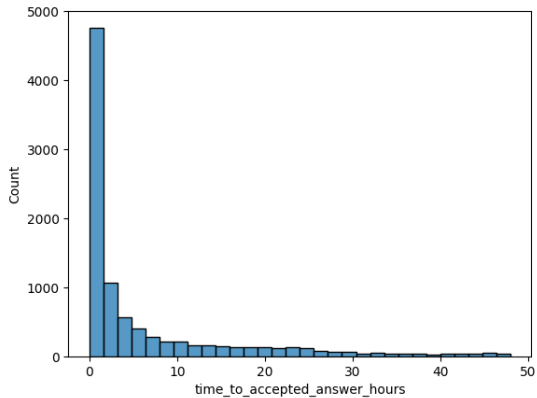


Figure 2: Histogram of time to accepted answer with time  $< 48$



- The dataset contains a total of 7684 unique tags. The most common being:

Tag	Count
python	1528
c#	746
javascript	703
c++	689
java	592

- Since we are dealing with such a vast number of different tags, we propose the following approaches to investigate:

## 1 Frequency-Based Filtering

- ▶ Aside from the top  $N$  most common tags, one approach could limit the scope to a subset of tags relevant to our analysis – e.g. **top  $N$  programming languages**

## 2 “Semantic Clustering”:

- ▶ Use pre-trained embeddings to represent question descriptions in a vector space.
- ▶ Apply clustering algorithms to group similar questions together based on their embeddings.

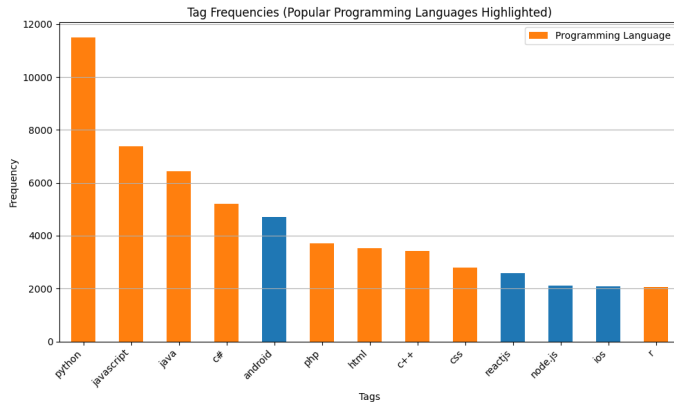


Figure 3: Tag Frequency

We filtered the tags using the list of programming languages (from Wikipedia) and selected the most frequent ones:

```
['python', 'javascript', 'java', 'c#', 'php', 'html', 'c++', 'css', 'r']
```

### Key observations about the resulting subset:

- **Subset size:** 42,037 questions
- **Class balance:** The distribution across these languages is relatively balanced (see last plot), and can be further balanced if needed.
- **Multi-label cases:** Some questions are tagged with multiple programming languages.

## Distribution of programming language tags per question:

# Tags	# Questions
1	38,547
2	3,015
3	464
4	11

- Finally, we analyzed the distribution of the scores of the questions in the dataset (already of the selected programming-language-tagged subset).

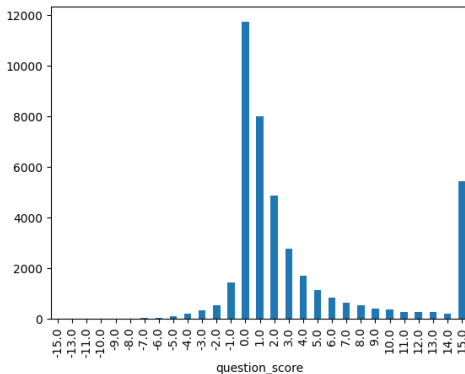


Figure 4: Question Score Histogram

Based on the hisogram, we proposed the following score classes:

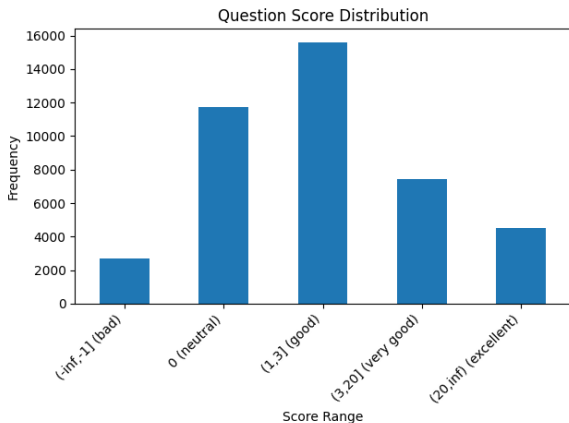


Figure 5: Question Score Distribution

### Proportion of Questions where Tag(s) Appears in Text

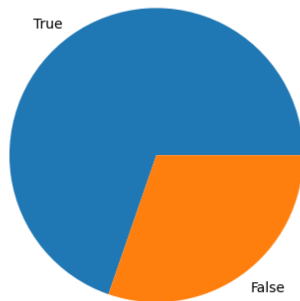


Figure 6: Proportion of Questions where Tag(s) appears in text



- the data about the accepted answer is provided for only small fraction of the data.
- there are too many tags to perform the classification using all of them
- however, we can choose a certain subset (e.g. most popular programming languages) and focus on it
- we can also try to predict the class of the score of the answer (integer score mapped to classes of uneven frequencies)

## TAG Embeddings

- Having 22753 unique tags poses a significant problem for our dataset,
- each questions can have multiple tags (for example `python`, `pandas`, ...)

- We've decided to embed tags, as well as question texts using `qwen3-embedding:8b` model, translating each tag into a 4096 vector.
- This resulted in a dataframe of shape 22753x4096 vector, which introduced more issues:
  - ▶ Lack of memory to process this data (the dtype is `f64`)
  - ▶ Too large dimensions

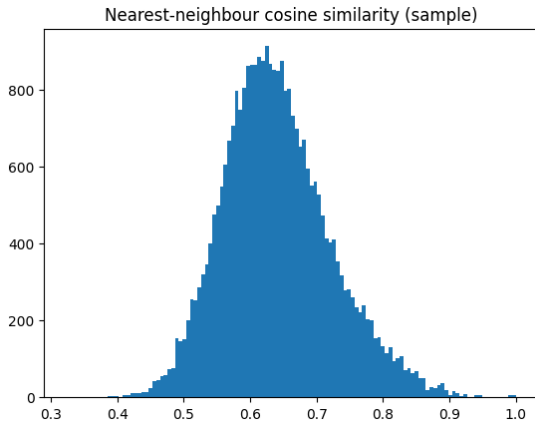


Figure 7: Question embeddings cosine similarity.

- Initial approach considered HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise), but due to memory required to perform this operation (over 64GB of RAM)
- Later on we've moved to a Birch algorithm, with HDBSCAN as well as AgglomerativeClustering which got unsatisfactory results

- After reading forums, we've decided to try UMAP (Uniform Manifold Approximation and Projection) which sounded ideal for our problem, since it can be used for general non-linear dimension reduction.

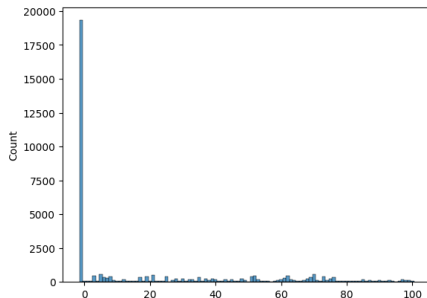


Figure 8: Recurring problem during UMAP and other clustering algorithms

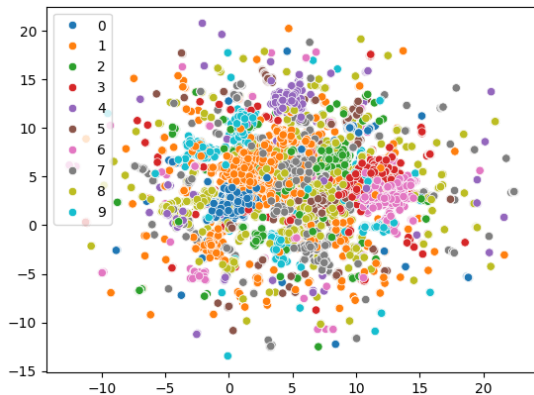


Figure 9: Naive UMAP reducing tag embeddings to 2 components.



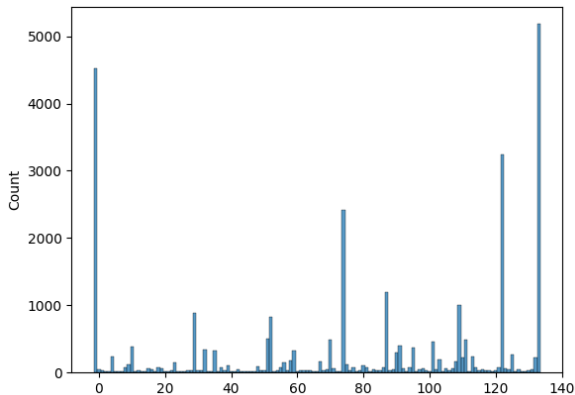


Figure 10: Histogram of labels resulting from UMAP reduction coupled with HDBSCAN

- A hyperparameter optimization framework which we hoped would find optimal parameters for our dimensionality reduction task.
- We need a score which we can maximize/minimize during dimension reduction:
  - ▶ Caliński-Harabasz index
    - ratio of the between-cluster separation to the within-cluster dispersion, normalized by their number of degrees of freedom.
  - ▶ Silhouette score
  - ▶ Cluster persistence
    - stability of each cluster, indicating well-defined grouping

- UMAP coupled with HDBSCAN
- Optimized hyperparameters:
  - ▶ `umap_n_components`
  - ▶ `umap_n_neighbors`
  - ▶ `umap_min_dist`
  - ▶ `umap_metric`
  - ▶ `hdb_min_cluster_size`
  - ▶ `hdb_min_samples`
  - ▶ `hdb_cluster_selection_method`
  - ▶ `hdb_cluster_selection_epsilon`

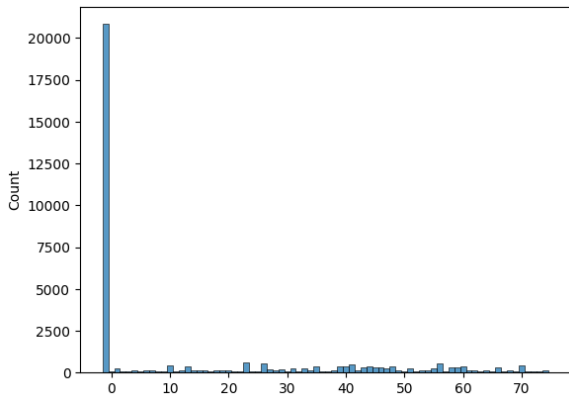


Figure 11: Results of optuna training

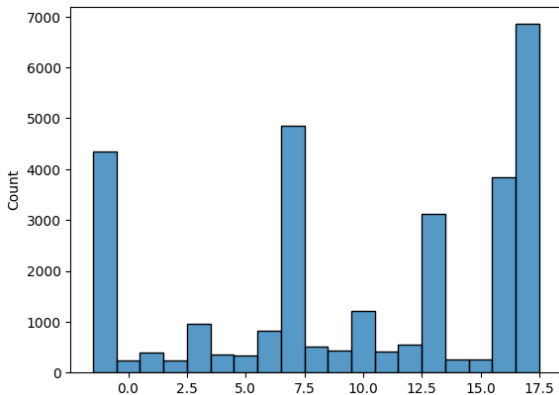


Figure 12: Results of Optuna optimization of only HDBSCAN hyperparameters search ran on UMAP reduction to 5 components

- We have to find better approach
- Possible culprits:
  - ▶ For UMAP to work as intended 'The data must be uniformly distributed on Riemannian manifold'
  - ▶ additionally 'The Riemannian metric is locally constant (or can be approximated as such)'
    - embeddings generally create spaces with varying local geometry depending on the semantic density of the training data.
  - ▶ Too few optimization runs.

# Recursive Embedding and Clustering

- Based on Spotify's blog.

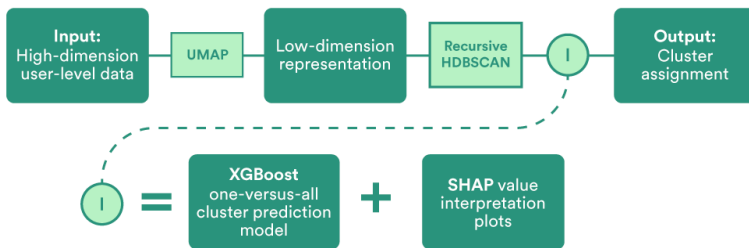


Figure 13: Diagram representing the complete recursive embedding and clustering process. Source

# Recursive KMEANS clustering



- Since properly implementing HDBSCAN in the context of our problem is still a standing question, we've decided to use KMEANS.
- Initially, we've specified the clustering hierarchy structure 100,100.
- After finding those centroids, we've taken their mean embeddings and selected 'most descriptive' tags for each cluster (based on cosine/euclidean metric).
- euclidean metric worked 'better' in the case of KMEANS.

	cluster_mean	top_5_popular_tags
facebook	[0.0052028694, -0.002167758, -0.017349796, -0....	[instagram-api, instagram, instagram-graph-api...
pyobject	[0.004231446, 0.0026400657, -0.0142704435, -0....	[python, pandas, jupyter-notebook, jupyter, py...
google-workspace-add-ons	[0.022996485, 0.0034478805, -0.014011028, -0.0...	[google-merchant-center, google-signin, google...
.net	[0.008871326, 0.00042595304, -0.003047627, -0....	[c#, wpf, .net, blazor, asp.net-core]
coverflow	[0.018253814, 0.0024098884, 0.0059977337, -0.0...	[background-color, css, tailwind-css, css-posi...
azure	[0.02082736, 0.0026177948, 0.0014911512, -0.00...	[azure-pipelines, azure, azureservicebus, azur...
importerror	[0.017333947, 0.00064369285, 0.0025193274, -0....	[segmentation-fault, try-catch, connection-ref...

Figure 14: Initial results



# Recursive Spherical KMEANS clustering



- Since spherical KMEANS is better suited for cosine similarity and for text data in general, we've decided to try it as well. Based on a Accelerating Spherical k-Means papaer [2].

	cluster_mean	top_5_popular_tags
facebook	[0.004879954, -0.0019050128, -0.016777342, -0....	[instagram-api, instagram, instagram-graph-api...
integer	[0.027635492, 0.0065359636, -0.0029253308, -0....	[intervals, differential-equations, math, maxi...
storage	[0.008282, -0.016636355, -0.0092419945, -0.004...	[dicomweb, remotestorage, storage, multipartfo...
tf-idf	[0.027263727, -0.007913517, 0.0023698097, -0.0...	[wpml, locale, dictionary, spacy, microsoft-tr...
qt	[0.01614129, 0.005696066, 0.016760174, -0.0084...	[qt, qml, pyqt, qasync, qt6]

Figure 15: Sample results

- To verify the quality of new clusters, we've used a very basic predictor based on the dot product of normalized question embeddings and tag cluster embeddings.
  - ▶ We've taken 2 most similar clusters for each question.

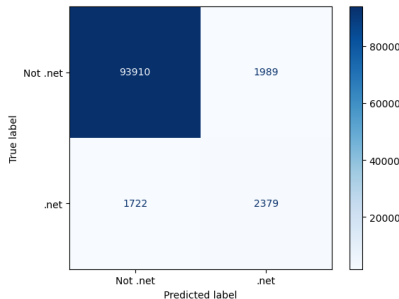


Figure 16: Results of top 2 results for .net

	tag1	tag2
count	780	780
unique	780	780
top	(mongodb, 0.5270306638492696) (hyperfilesq1, 0.4293950990405074)	
freq	1	1

Figure 17: Most popular results for mongodb

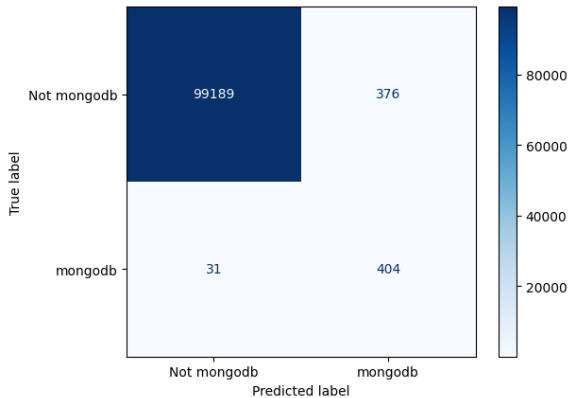


Figure 18: Results of top 2 results for mongodb

## TAG 2

- Problem: 22,753 unique tags is too large for direct multi-label classification.
- Solution: Reduce tags to 100 clusters using *Recursive Spherical K-Means* (with cosine similarity), then classify into these clusters.

- Our dataset has 100k records many of which have multiple tags.
- The idea is that we can remove tags with low frequency and remove records that become tag-less after this operation.
- Aforementioned reduction for the threshold of 100 resulted in 90,205 records and 411 unique tags.

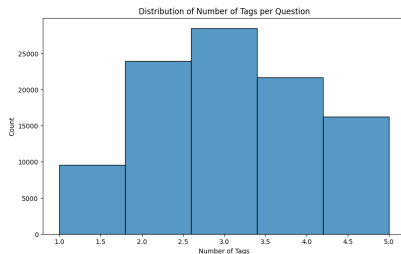


Figure 19: Distribution of number of tags per question

- With the resulting tags, we performed Recursive Spherical K-Means to reduce the 411 tags to 100 centroids.
- The outlier with 15 tags corresponds to a cluster with following tags: database, mongodb, sql, elasticsearch, firebase, postgresql, sqlite, mysql, jdbc, supabase, sql-update, redis, hibernate, sqlalchemy, snowflake-cloud-data-platform

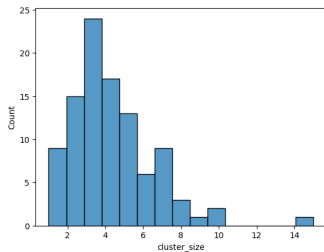


Figure 20: Distribution of number of clusters per centroid



- For the 9787 orphan tags (those with insufficient frequency), we assigned them to the nearest centroid based on cosine similarity of their embeddings.
- For example:

```
closest_centroid_tag(tags,centroid_tags, 'ntp')  
('datetime', np.float64(0.8098148848579392))
```

- This resulted in the final set of 100 centroids covering all 22,753 original tags, without losing any data.

# Why Neural Networks?



- 1 Multi-Label Classification:** NNs naturally handle multi-label outputs (e.g., using Sigmoid activations + BCE Loss).
- 2 Feature Interaction:** Deep learning models can learn complex non-linear interactions between Title and Body embeddings.
- 3 Custom Loss Functions:** Easier to implement losses like Asymmetric Loss to handle class imbalance.

## Tag Prediction

- XGBoost stands for “Extreme Gradient Boosting”, where the term “Gradient Boosting” originates from the paper *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman.
- It is an ensemble learning method that builds multiple weak learners (decision trees) sequentially, where each tree attempts to correct the errors of the previous ones.
- Task: **predict the question’s centroid tag based on question text embedding.**



- To combat the unbalanced distribution of centroid tags, we applied class weights inversely proportional to their frequencies during training, using `sklearn.utils.class_weight.compute_class_weight` and passing these weights to XGBoost's `scale_pos_weight` parameter.
- Learning process took 599 minutes.
- Accuracy: 0.6928123848138592
- F1 Weighted: 0.6882358640179885

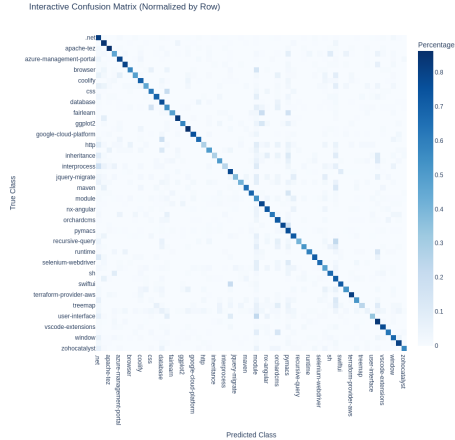


Figure 22: Confusion Matrix of XGBoost Classifier

# Limitations of XGBoost Approach





- For better modeling of the multi-label nature of the problem, we turned to Neural Networks using PyTorch.
- We can easily design architectures for multi-label classification and implement custom loss functions to handle class imbalance.
- We can leverage **Attention Mechanisms** and other fancy architectures for our predictive model.

# Baseline: Simple MLP



## ■ Architecture:

```
BaselineNetwork(  
    (fc1): Linear(in_features=4096, out_features=2048)  
    (relu): ReLU()  
    (dropout): Dropout(p=0.3)  
    (fc2): Linear(in_features=2048, out_features=1024)  
    (dropout2): Dropout(p=0.3)  
    (fc3): Linear(in_features=1024, out_features=100)  
)
```

## ■ Input: question\_text\_embedding

## ■ Loss: BCEWithLogitsLoss

- ▶ Since question can have a very popular, and very rare tag (e.g. .net, nvim), we can't just weigh all tags equally. Instead, we will penalize by BCEWithLogitsLoss for missing a rare tag much more heavily than for missing a popular tag.

- Training:
  - ▶ 50 epochs ~6 minutes
  - ▶ F1 (weighted):  $\approx 67.9\%$
- Properly models multi-label outputs (unlike a multiclass XGBoost mapping)
- Low compute and quick iteration for ablations
- Slightly underperforms XGBoost baseline on this dataset, but performs multi-label classification instead of basic classification.

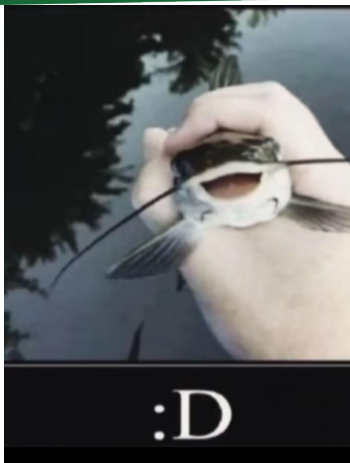


Figure 23: Per-Tag Performance of Baseline MLP

# Per-Tag Performance: Python

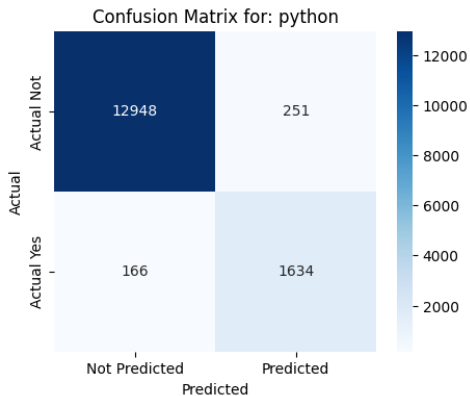


Figure 24: Confusion Matrix for Python only

- Inspired by 'Dual-stream fusion network with multi-head self-attention for multi-modal fake news detection' [3].
- Instead of processing text and images as in the paper above, we will map two streams:
  - question\_text embeddings,
  - title embeddings,
- and fuse them using multi-head self-attention to capture correlation between the two modalities.
- As a fusion, we will use the paper's proposed MHSA Fusion strategy to let the Title "attend" to the question body and vice versa.

# Dual Stream Fusion Network (DSF)



- Process Title and Body separately and fuse them.
- **Architecture:**
  - ▶ **Stream 1:** Title Embedding -> Projection -> LayerNorm
  - ▶ **Stream 2:** Body Embedding -> Projection -> LayerNorm
  - ▶ **Fusion:** Multi-Head Self-Attention (MHSA) on stacked features.
- Since the shape of both embeddings is the same, we have the same architecture for both streams:

```
(title_proj): Sequential(  
  (0): Linear(in_features=4096, out_features=1024, bias=True)  
  (1): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)  
  (2): GELU(approximate='none')  
  (3): Dropout(p=0.1, inplace=False)
```

- The initial results were disappointing- model overfitted very quickly and proved immediately useless on validation set.
- To mitigate overfitting, we:
  - ▶ Increased dropout rate from 0.1 to 0.5.
  - ▶ Changed optimizer from Adam to AdamW.
    - AdamW decouples weight decay from the gradient update, leading to better generalization.
  - ▶ Implemented **Asymmetric Loss** [1] to handle label imbalance.



- In multi-label classification with 100 classes, for any given question, most labels are **Negative**. The model can get high accuracy by just predicting “0” for everything.
- To counter this, ASL:
  - ▶ Separates the treatment of positive and negative samples.
    - Applies different focusing parameters to down-weight easy negatives more aggressively.
    - Keeps the weight high for positive samples to ensure they are learned well.
  - ▶ Focuses training on hard negatives and positive samples.
  - ▶ Essentially, mistakes on positives matter more than mistakes on easy negatives (based on the probability score output by the model).
- Parameters:  $\gamma_- = 4$ ,  $\gamma_+ = 1$ .

- Aside from the loss function, we've added:
  - ▶ ReduceLROnPlateau scheduler to reduce learning rate on validation F1 plateau.
  - ▶ As earlier mentioned, AdamW optimizer with weight decay of  $1e-2$ .
- Training for 50 epochs took ~20 minutes.
- F1 score (micro) on validation: 0.713
- F1 score (weighted) on validation: 0.711

# Hedgehog Anatomy

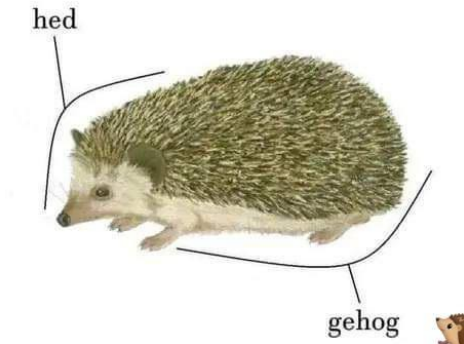


Figure 25: Per-Tag Performance & Multilabel Confusion Matrices of DSF\_MHSA\_Classifier

- Some sort of overfitting still occurred, so we tried replacing the MHSA fusion with Cross-Attention:
  - ▶ Instead of both streams attending to each other equally, we let the Title embedding “query” the Body embedding.
- Intuition:
  - ▶ *“Given this Title, which parts of the Body Embedding are relevant?”*
  - ▶ Title is short and dense with information.
  - ▶ Body is longer and more verbose.
  - ▶ Letting Title attend to Body helps focus on relevant parts of the Body text.
- Same as before, we used ASL loss to combat label imbalance. This time with  $\gamma_- = 2$ ,  $\gamma_+ = 1$ .
- To further reduce overfitting, we also applied Manifold Mixup on the embeddings:
  - ▶ It creates new training samples by interpolating between the embeddings (and tags) of random pairs.
  - ▶ Enforces smoother decision boundaries in the latent space, improving generalization.

- Training for 100 epochs took ~60 minutes.
- Used OneCycleLR scheduler.
- Implemented basic Curriculum Learning by starting with smaller dropout (0.25)
- F1 score (micro) on validation: 0.725291274763489
- F1 score (weighted) on validation: 0.7196345470456641

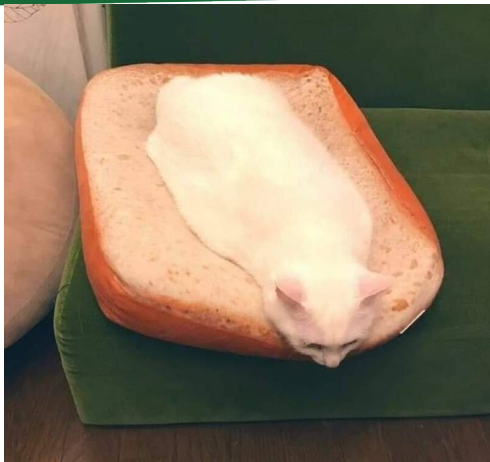


Figure 26: Per-Tag Performance & Multilabel Confusion Matrices of DSF\_CrossAttn\_Classifier

# Summary of Results



Model	F1 (weighted)
XGBoost	0.6882
Baseline MLP	0.6790
DSF with MHSA Fusion	0.7110
DSF with Cross-Attention Fusion	0.7196

Thank you for your attention



## Questions

## References

- [1] Ben-Baruch, E., Ridnik, T., Zamir, N., Noy, A., Friedman, I., Protter, M. and Zelnik-Manor, L. 2021. Asymmetric loss for multi-label classification.
- [2] Schubert, E., Lang, A. and Feher, G. 2021. Accelerating spherical k-means. *Similarity search and applications*. Springer International Publishing. 217–231.
- [3] Yang, Y., Liu, J., Yang, Y. and Cen, L. 2024. Dual-stream fusion network with multi-head self-attention for multi-modal fake news detection. *Applied Soft Computing*. 167, (2024), 112358. DOI:<https://doi.org/https://doi.org/10.1016/j.asoc.2024.112358>.