Code requirements
tested on Python3.10/Ubuntu 22.04

Data split strategy
To avoid data contamination and replicate real world conditions data is split based on chronological ordering. Following [1], I used 70% for training (and HPO validation); 30% for testing.

Metric
Chronological based data splitting strategy resulted in unbalanceness of the data, therefore I used F1 as the metric.

Features/data prep
Text was normalization:

- Removed accents
- Removed all characters except latin letters and periods
- Lower case
- Lemmatization

Additional feature:
After inspecting the data I noticed that spam comments usually contain URL links. I decided to add the presence of the links as an additional feature by adding ‘http’ token to the text of the comment. Since URL links are not usually repeated, they will be ignored in TF-IDF vectorization with min_df (Minimum document frequency) of 2 and higher.

Models:

1. In [1] the authors achieved around 95% F1 score using the classical approach of combining TF-IDF features with Random Forest/Naive Bayes/SVM/etc. I reproduced their approach and performance. For the sake of brevity I only used Random Forest from sklearn.
   Based on hyperparameter optimization the best model is:
   Test F1: 95.57%
   RandomForest
   lemmatize: False
   use_url_feat: True}

BoW__max_features: 500
BoW__min_df: 5
BoW__ngram_range: (1, 3)
BoW__stop_words: None
RF__n_estimators: 100
TFIDF__use_idf: True

2. As a second model I applied FastText [2]. It uses subword information and is lightweight. Based on hyperparameter optimization the best model is:
   Test F1: 93.02%
   FastText
   lemmatize: False
   use_url_feat: True
   dim: 100
   epoch: 5
   lr: 0.1
   lrUpdateRate: 100
   maxn: 0
   minCount: 1
   minCountLabel: 0
   minn: 0
   neg: 5
   wordNgrams: 1
   ws: 5

3. I also fine tuned BERT (base, cased) using the huggingface library. Hyperparameter optimization in this case is costly, so I manually tuned the parameters (batch size, learning rate and number of training epochs). The model is:
   Test F1: 95.59%
   Parameters:
   model: 'bert-base-cased',
   learning_rate: 5e-5
   'num_train_epochs: 5

batch_size: 32
weight_decay: 0.01

Overall best model: fine tuned transformer Test F1: 95.59%. However, given only marginal performance improvement and much higher train and inference cost I would choose RandomForest w/ TF-IDF with Test F1: 95.57%

Model selection

Note: If compute budget is ignored I would use std error of the F1 score from cross validation to statistically test for the difference in performance between models.
Since my compute budget is constrained I had to rely on simply ordering the models according to the mean of the test F1 score.

Failure case analysis

To further improve the model I would analyze failure cases and try to spot patterns that can give clues on how to improve features/model architecture.

References:

[1] Alberto, Túlio C., Johannes V. Lochter, and Tiago A. Almeida. "Tubespam: Comment spam filtering on youtube." *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 2015.
[2] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.