A multi-level approach for hierarchical Ticket Classification

Matteo Marcuzzo [*]
Alessandro Zangari [*]
Digital Strategy Innovation Srl
30175, Venice, Italy
{name.surname}@unive.it

Lorenzo Giudice
Andrea Gasparetto
Ca' Foscari University of Venice
Department of Management
30121, Venice, Italy
{name.surname}@unive.it

Michele Schiavinato
Andrea Albarelli
Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics
30172, Mestre (VE), Italy
{michele.schiavinato,albarelli}@unive.it

Abstract

The automatic categorization of support tickets is a fundamental tool for modern businesses. Such requests are most commonly composed of concise textual descriptions that are noisy and filled with technical jargon. In this paper, we test the effectiveness of pre-trained LMs for the classification of issues related to software bugs. First, we test several strategies to produce single, ticket-wise representations starting from their BERT-generated word embeddings. Then, we showcase a simple yet effective way to build a multi-level classifier for the categorization of documents with two hierarchically dependent labels. We experiment on a public bugs dataset and compare our results with standard BERT-based and traditional SVM classifiers. Our findings suggest that both embedding strategies and hierarchical label dependencies considerably impact classification accuracy.

1 Introduction

Support tickets and incident reports are a valuable point of contact between customers and service providers (Al-Hawari and Barham, 2021). They are fundamental tools in the management of the relationship between businesses and users, allowing for the swift resolution of issues, thus leading to improved customer satisfaction, productivity, and compliance which Service-Level Agreements (SLAs) (Gupta and Sengupta, 2012). Tickets can be derived from multiple communication channels, most commonly emails, specialized web forms, phone calls, live chats, and social media platforms (Zicari et al., 2021). Help requests are therefore logged as text, which represents the most important source of information to be used for automatic ticket management. Being conversational by nature, tickets describe the issue or request in an often noisy and concise format (Cristian et al., 2019).

As a response to the increasingly high volume of these requests by customers, researchers have proposed the automation of various steps of the ticket resolution pipeline (Fuchs et al., 2022; Ali Zaidi et al., 2022). These include the classification of tickets into broad topic categories (*ticket classification*) (Zicari et al., 2021; Revina et al., 2020), the direct assignment of the issue to an expert capable of resolving it (*expert finding*) (Husain et al., 2019), as well as the direct resolution of tickets in a completely autonomous way (*ticket resolution*) (Zhou et al., 2017). Among these tasks, the accurate classification of incoming tickets within a pre-defined hierarchy of labels is among the most prevalent, as well as one of particular importance to ensure that these requests are dealt with swiftly. Indeed, it is common for support tickets to be framed within a multi-level hierarchy such as the one just mentioned: each level of the hierarchy describes the issue at different levels of specificity.

Contributions This work will explore Ticket Classification (TiC), a sub-task of Text Classification (TC), with the following objectives:

- Verifying the effectiveness of contextualized Language Models (LMs) (Radford et al., 2018; Marcuzzo et al., 2022) on noisy pieces of text from this particular domain;
- Exploring the impact of document embedding strategies on downstream task performance;
- Establishing how much a LM can benefit from the injection of hierarchy information for topical classification within a two-level hierarchy.

Our experiments show that both document summarization strategies and hierarchical information injection can contribute in a major way to TiC accuracy. The code and datasets utilized in this work's experiments are made publicly available online[1].

[*] Authors contributed equally.

[1] https://gitlab.com/distration/dsi-nlp-publib/-/tree/main/WNUT22

2 Related work

The task of topical TiC has been explored in recent works, which we briefly outline. Relatedly, we also discuss recent advancements in the broader TC environment, as well as a short mention of dedicated hierarchical TC methods.

Ticket Classification (TiC) In the particular context of TiC, much work has been done towards the application of traditional methods, a popular example being that of Support Vector Machines (SVMs) (Boser et al., 1992) applied on simple word-count-based text representation techniques such as TF-IDF (Jones, 1972). Recent works such as Yang (2021); Revina et al. (2020) have argued for the efficacy of traditional methods, often introducing more advanced text representation techniques such as Word2Vec (Mikolov et al., 2013). There has also been recent interest in the application of Deep Neural Networks, such as Multilayer Perceptrons (Kallis et al., 2019), Convolutional Neural Networks (Zicari et al., 2021; Pistellato et al., 2018), and Recurrent Neural Networks (Mani et al., 2019; Lyubinets et al., 2018).

Text Classification (TC) In the broader environment of Natural Language Processing (NLP), all downstream tasks — including TC — have been recently revolutionized by the introduction of the Transformer architecture (Vaswani et al., 2017). This approach to text representation has allowed for much more meaningful vectorial representations for words, crucially able to discern context. Contextualized LMs based on this architecture are now the staple NLP transfer learning approach, and have showcased massive performance boosts in TC benchmarks. Among others, we focus on the influential Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019), which we utilize in this work. We refer to Gasparetto et al. (2022a,b) for a thorough description of BERT's architecture. Though succeeded by more refined LMs in recent years, it is still widely studied and utilized. BERT-based LMs have been extensively applied to NLP tasks on user-generated content, such as tweets (Polignano et al., 2019) and user reviews (Lu et al., 2020). Interestingly, recent findings suggest that BERT is quite sensible to the presence of noise in text (*e.g.*, spelling mistakes) (Kumar et al., 2020). On the other hand, other works suggest that BERT is quite resistant to label-noise, and the application of common noise-handling methods can in fact deteriorate BERT's performance (Zhu et al., 2022). With this work, we aim to give practitioners some insights into the usage of BERT for classification in the support ticket classification domain. While an excellent source of noisy user-generated data, we find this context to be understudied at present.

Hierarchical Text Classification As we are discussing datasets whose labels are hierarchical by nature, it would be reasonable to utilize Hierarchical TC approaches. The architecture we propose in this work is partly inspired by these approaches, in particular by the distinction between *flattened classifiers* (Koller and Sahami, 1997), which simplifies the hierarchy by flattening it to a single multiclass or multilabel problem, and *global classifiers*, which build upon these classifiers but integrate hierarchy information within their framework (Labrou and Finin, 1999). Still, we point out that the amount of overlap with hierarchical TC literature is somewhat limited by the fact that the hierarchy in ticketing systems is usually very shallow (two to three levels), while HTC systems usually operate in multilabel environments with very complex hierarchies.

3 The Linux Bugs dataset

To evaluate a TiC scenario, we experiment on a dataset of bugs crawled from the publicly available Linux kernel bug-tracker [2], as inspired by Lyubinets et al. (2018). The resulting *Linux Bugs dataset* contains tickets organized through the hierarchical dependent labels of "product" (*e.g.*, Network, Drivers, etc.) and "component" (*e.g.*, BIOS, scheduler, etc.). Therefore, we utilize the former as main labels and the latter as sub-labels. To be precise, to avoid redundancies, we utilize the flattened labels as sub-labels, such as to differentiate sub-labels that share their name across main categories (*e.g.*, `Network_Other` vs `Drivers_Other`). Moreover, to reduce class imbalance, we discard all labels and sub-labels that appear less than 100 times. More details on the dataset, including an exemplary subset of the resulting hierarchy, reports on the labels' frequency and an example of the content of a ticket can be found in Appendix B.

4 Methods

The aforementioned BERT LM has been one of the most popular contextualized LMs since its inception. Conceptually, BERT is a bidirectional

[2] `https://bugzilla.kernel.org`

Transformer-based neural network, made by stacking multiple encoder blocks. These blocks are entirely based on the self-attention mechanism (Bahdanau et al., 2015), eliminating the sequential bottleneck of previous recurrent models.

BERT models are pre-trained on two specifically devised language modeling tasks that allow the networks to learn semantically and contextually meaningful representations of text. These models can then be fine-tuned on specific tasks quite easily (in the case of classification, by adding a simple linear layer as a classifier) to obtain state-of-the-art performances. In our case, we utilize the models available on HuggingFace (Wolf et al., 2020), which are pre-trained on BookCorpus (Zhu et al., 2015) and English Wikipedia, and have a vocabulary of around 30,500 word segments.

4.1 Document summarization strategies

In terms of text interpretation, BERT first transforms raw text into tokens through the WordPiece sub-word tokenization algorithm (Schuster and Nakajima, 2012). The output is then passed through the stacked encoder blocks, with each layer producing an embedding for each token.

A common approach to classification using BERT is to utilize the `[CLS]` token as a document summary. This special symbol is pre-pended to each sequence of words and is utilized during pre-training on the Next Sentence Prediction binary classification task (NSP). Despite its popularity, several authors suggest that other "summarization strategies" for documents may be preferable (Reimers and Gurevych, 2019). For instance, Tanaka et al. (2019) experiment with the averaging of the individual word embeddings composing the sentences rather than the single `[CLS]` token. They utilize the output of one or more encoder blocks and combine them, highlighting classification improvements when concatenating the output of several layers. Researchers have further hinted at the phenomenon of "layer specialization" within BERT, arguing that each encoding block may focus on the extraction of linguistic features at different levels: syntactic features are mostly extracted in the first blocks, while deeper layers progressively focus on semantic features. They also discuss how information from each layer can be beneficial to downstream tasks, like classification (de Vries et al., 2020; Jawahar et al., 2019; Torsello et al., 2014).

In this work, we explore the impact of different approaches to the creation of a condensed document representation starting from BERT's word vectors. In particular, we test several strategies, including the usage of the `[CLS]` token from the last layer (referred to as *cls last*), but also the concatenation and average of the `[CLS]` tokens of the last h hidden layers, respectively indicated with *cls concat*$_h$ and *cls avg*$_h$. We also experiment with approaches that do not use such token, such as averaging the embeddings of all words in a document (*avg*), taking the maximum (*max*), or using their normalized sum (*sum nor*). A description of each strategy is given in Appendix A.

4.2 Multi-level classifiers

Our second aim in this work is to design an effective multi-level architecture able to exploit the hierarchical dependency information between labels. The following paragraphs detail the two proposed approaches. Notably, our multi-level classifiers are based on BERT LMs, but may be used with any model capable of producing contextualized word embeddings. In both frameworks, we utilize two separate LMs trained on the categorization of macro-labels (task T1) and on the categorization of flattened sub-labels (T2). A visualization of the approaches is provided in the Appendix (Fig. 1).

ML-BERT The Multi-Level BERT (`ML-BERT`) classifier is a combination of two distinct pre-trained BERT LMs, previously fine-tuned on the prediction of the T1 and T2 tasks, respectively (LM_1, LM_2). In the `ML-BERT` model, the weights of the two base LMs are kept frozen during the fine-tuning procedure — the output of the pre-trained classifiers is discarded. Only the word embeddings produced by each model are utilized; document representations are obtained by using the best-performing summarization strategy among the ones mentioned in Section 4.1. Embeddings from both models are then concatenated into a global ticket representation and passed through a single linear layer with a softmax activation function. Therefore, fine-tuning only requires the learning of the last layer's weights, reducing the computational cost.

Supported-BERT The `Supported-BERT` classifier similarly utilizes a LM previously trained on T1 (LM_1). However, LM_2 is not trained in isolation but instead utilizes the fine-tuned LM_1 as support during its own fine-tuning. As before, the ticket embeddings from the two LMs are concatenated and passed to the output layer. Thus, the difference from the previous setup is that LM_2 is trained directly with the output layer and with external influence, instead of being trained beforehand.

5 Experiments

We report in this section the experiments we conducted to select the most suitable summarization strategy and to determine the effectiveness of the multi-level classifiers. While the core of our experiments was performed on BERT's base pre-trained model (*i.e.*, "bert-base-uncased"), we also report results using a larger BERT LM (*i.e.*, "bert-large-uncased").

5.1 Experimental setup

We describe our experimental settings in this section, adding details on the metrics we choose to use and on how we select the model hyper-parameters.

Metrics We evaluate the models in a multiclass setting, where the ground truth label is the concatenation of the parent and child categories. Therefore, the models must predict a single class for each ticket (*e.g.*, `Networking_IPV4` is the target label for a ticket that belongs to the `Networking` category and `IPV4` sub-category). This approach is widely utilized in the evaluation of global HTC methods, which our approaches can be seen as (Silla and Freitas, 2011). We use standard classification metrics, *i.e.*, accuracy and F_1-score, to measure the performance. Briefly, *accuracy* measures the ratio of correct predictions over the total of number predictions, but can give a skewed representation of imbalanced datasets. F-score is a combination of *precision* and *recall*, which measure a model's correctness and completeness, respectively (Gasparetto et al., 2022a, 2018). We report the F_1-score — the harmonic mean of precision and recall — in its *macro-averaged* version, *i.e.*, considering all class contributions equally.

Hyper-parameter tuning We use a stratified 3-fold CV to split the dataset into training and testing subsets. Before testing BERT's performance with different summarization strategies on the testing split, we tune the learning rate and the number of training epochs on the training split, reserving 20% of it as a validation set. We use the BERT-base model on task T2 using the standard *avg last* strategy with early stopping set on the loss function to determine the optimal number of epochs.

Table 1: Effect of additional processing procedures on the performance of a BERT model on the validation set.

Model	Clean	Weigh	Acc	F$_1$
BERT (base)	✗	✗	**0.533** [± 0.003]	**0.396** [± 0.005]
	✓	✗	0.503 [± 0.003]	0.374 [± 0.006]
	✗	✓	0.471 [± 0.005]	0.382 [± 0.004]
	✓	✓	0.464 [± 0.007]	0.371 [± 0.008]

[*] Standard deviation over 3 runs is reported in brackets.

After validation, the BERT-base models are trained for 3 epochs with learning rate set to $2e^{-5}$ and batch size set to 8. The BERT-large models were similarly validated and trained with a learning rate of $1e^{-5}$ for 3 epochs, with batch size set to 8.

Following Lyubinets et al. (2018), we test the impact of a more comprehensive text cleaning procedure that removes most of the stack traces and memory addresses, which are quite frequent in this dataset. Listing 2 in the Appendix showcases an example of a bug report treated with the more aggressive cleaning procedure. Furthermore, because our dataset is imbalanced class-wise, we experiment with weighting classes' contribution to the loss value based on their support. We find that neither the additional preprocessing step nor the weighting scheme improved the performance in terms of F$_1$ and accuracy scores using the default *avg last* strategy, as can be seen in Table 1. We hypothesize that, even though the representations of pieces of text such as the hexadecimal codes of Listing 1 have low syntactic and semantic value, they still provide discriminative power in the downstream classification task.

To train the multi-level models, we separately train LM_1 and LM_2 on T1 and T2 tasks respectively, and use the same hyperparameters selected for the previous tests. Moreover, we select the best learning rate and number of epochs for the final classifier using the same procedure as described above, obtaining the values of $2e^{-5}$ (2 epochs) and $1e^{-5}$ (3 epochs) for the base and large versions of BERT, respectively.

5.2 Results

In this section, we report test set results obtained with the best hyper-parameters as just described.

Document summarization Results with the different summarization strategies introduced in Section 4.1 using BERT-base on task T2 are reported in Table 2. First off, there is a considerable difference in performance between the pooled and "raw" ver-

Table 2: Test set results* with BERT classifier on T2 comparing summarization strategies on Linux Bugs.

Basis	Strategy	Acc	F_1
cls	*last* (p)†	0.518 [± 0.006]	0.354 [± 0.009]
	last	0.566 [± 0.012]	0.446 [± 0.018]
	avg$_2$	0.531 [± 0.010]	0.393 [± 0.012]
	concat$_2$	0.535 [± 0.010]	0.400 [± 0.014]
	concat$_3$	**0.571** [± 0.008]	0.456 [± 0.013]
	concat$_4$	0.568 [± 0.009]	**0.457** [± 0.014]
	concat$_5$	0.565 [± 0.012]	0.456 [± 0.013]
avg	*last*	0.525 [± 0.008]	0.387 [± 0.010]
	avg$_2$	0.522 [± 0.005]	0.383 [± 0.013]
	concat$_2$	0.523 [± 0.007]	0.390 [± 0.009]
max	*last*	0.522 [± 0.011]	0.385 [± 0.014]
	avg$_2$	0.519 [± 0.007]	0.375 [± 0.013]
	concat$_2$	0.518 [± 0.006]	0.373 [± 0.015]
max_min	*last*	0.522 [± 0.010]	0.377 [± 0.011]
	avg$_2$	0.522 [± 0.009]	0.374 [± 0.010]
max_avg	*last*	0.516 [± 0.007]	0.381 [± 0.012]
	avg$_2$	0.519 [± 0.006]	0.379 [± 0.007]
sum_nor	*last*	0.406 [± 0.018]	0.171 [± 0.017]
	concat$_2$	0.379 [± 0.013]	0.135 [± 0.019]
	concat$_5$	0.388 [± 0.015]	0.135 [± 0.015]

* Standard deviation over 6 runs is reported in brackets.
† Pooled, using *cls pooled* strategy.

Table 3: Test set results* for all models on the T2 task. BERT models utilize the *cls concat*$_3$ strategy.

Model	Acc	F_1
SVM	0.551 [± 0.004]	0.473 [± 0.006]
ML-BERT (base)	0.602 [± 0.010]	**0.500** [± 0.014]
Supp-BERT (base)	**0.611** [± 0.007]	0.485 [± 0.013]
BERT (base)	0.571 [± 0.008]	0.456 [± 0.013]
ML-BERT (large)	0.577 [± 0.008]	0.461 [± 0.006]
Supp-BERT (large)	0.597 [± 0.007]	0.480 [± 0.011]
BERT (large)	0.559 [± 0.008]	0.438 [± 0.011]

* Standard deviation over 6 runs is reported in brackets.

sions of the *cls last* strategy, with the raw `[CLS]` token without pooling achieving considerably better results. Stacking multiple layers further improved the results; tests with *cls concat*$_3$ achieved the best overall performance in terms of accuracy, precision, and recall (though macro-averaging favors *cls concat*$_4$ in terms of F_1 score), confirming that features extracted in other BERT hidden layers can be beneficial to the classification task (see Table 5 in the Appendix).

Multi-level classifiers Table 3 reports the classification performance of both `ML-BERT` and `Supported-BERT` (shortened as `Supp-BERT`), utilizing the previously determined best document summarization strategy (in our case, *cls concat*$_3$).

The smaller `ML-BERT` achieves an improvement of 9.7% macro F_1-score and 5.4% accuracy as compared to the BERT model trained on the flattened hierarchy of labels. `Supported-BERT`'s improvement amounts instead to 6.4% and 7.0% on the same metrics. While `ML-BERT` performed better in terms of macro F_1-score, `Supported-BERT` resulted in the highest accuracy. The larger pre-trained BERT model showcases overall a similar trend, though with lower performance than with the base model in all experiments. Nevertheless, the multi-level models still improved results over the flat T2 classifier: `ML-BERT` (large) achieved 3.1% and 5.2% improvement in macro F_1-score and accuracy respectively, while the performance of `Supported-BERT` improved by 6.8% and 9.5%. In practice, we observed that the larger model did not converge as well as the base one. This is likely to be a consequence of the limited size of our highly skewed dataset, as well as the limited semantic significance of its composing documents (that contain many technical bits of text, like stack traces).

The SVM classifier was trained with a simple one-vs-rest strategy and also performed very well, surprisingly achieving better macro F_1 than the smaller BERT model in the flattened setting. However, all base multi-level models perform better on both metrics. As is also discussed in the literature, BoW features with TF-IDF weighting are suitable representations for noisy text, effectively able to filter out many unimportant words (Das et al., 2021). On the other hand, contextualized LMs such as BERT are meant to exploit sentence structure and word context, which might be insufficiently informative in such environments.

Error analysis Because of time and space limitations, we do not perform an in-depth error analysis of our models in this work. However, a discussion in this regard can be found in Appendix D, in which we also discuss how we would like to address this analysis in future work.

6 Conclusion

In this article, we experimented with contextualized LMs for TiC, and found that different document embedding summarization strategies are a major factor in classification performance. Moreover, we devised two multi-level classification approaches based on LMs, and found further improvement by injecting information from the label hierarchy within the architecture. We hope our work can provide useful insights into the usage of BERT models for classification in a previously understudied domain.

References

Feras Al-Hawari and Hala Barham. 2021. A machine learning based help desk system for it service management. *Journal of King Saud University - Computer and Information Sciences*, 33(6):702–718.

Syed S. Ali Zaidi, Muhammad Moazam Fraz, Muhammad Shahzad, and Sharifullah Khan. 2022. A multiapproach generalized framework for automated solution suggestion of support tickets. *International Journal of Intelligent Systems*, 37(6):3654–3681.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *arXiv.org*, abs/1409.0473.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. Association for Computing Machinery.

Matei Cristian, Săcărea Christian, and Tolciu Dumitru-Tudor. 2019. A study in the automation of service ticket recognition using natural language processing. In *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6.

Mamata Das, Selvakumar Kamalanathan, and PJA Alphonse. 2021. A comparative study on tf-idf feature weighting method and its analysis using unstructured dataset. In *COLINS*, pages 98–107.

Wietse de Vries, Andreas van Cranenburgh, and Malvina Nissim. 2020. What's so special about BERT's layers? a closer look at the NLP pipeline in monolingual and multilingual models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4339–4350, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Simon Fuchs, Clemens Drieschner, and Holger Wittges. 2022. Improving support ticket systems using machine learning: A literature review. In *Proceedings of the 55th Hawaii International Conference on System Sciences*, pages 1893–1902, Honolulu, HI 96822. ScholarSpace.

Andrea Gasparetto, Matteo Marcuzzo, Alessandro Zangari, and Andrea Albarelli. 2022a. A survey on text classification algorithms: From text to predictions. *Information*, 13(2).

Andrea Gasparetto, Dalila Ressi, Filippo Bergamasco, Mara Pistellato, Luca Cosmo, Marco Boschetti, Enrico Ursella, and Andrea Albarelli. 2018. Cross-dataset data augmentation for convolutional neural networks training. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 910–915.

Andrea Gasparetto, Alessandro Zangari, Matteo Marcuzzo, and Andrea Albarelli. 2022b. A survey on text classification: Practical perspectives on the italian language. *PLOS ONE*, 17(7):1–46.

Hari S. Gupta and Bikram Sengupta. 2012. Scheduling service tickets in shared delivery. In *Service-Oriented Computing*, pages 79–95, Berlin, Heidelberg. Springer Berlin Heidelberg.

Omayma Husain, Naomie Salim, Rose Alinda Alias, Samah Abdelsalam, and Alzubair Hassan. 2019. Expert finding systems: A systematic review. *Applied Sciences*, 9(20).

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *J. Doc.*, 28(1):11–21.

Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket tagger: Machine learning driven issue classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 406–409.

Daphne Koller and Mehran Sahami. 1997. Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, page 170–178, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ankit Kumar, Piyush Makhija, and Anuj Gupta. 2020. Noisy text data: Achilles' heel of BERT. In *Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020)*, pages 16–21, Online. Association for Computational Linguistics.

Yannis Labrou and Tim Finin. 1999. Yahoo! as an ontology: Using yahoo! categories to describe documents. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, page 180–187, New York, NY, USA. Association for Computing Machinery.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*, New Orleans, Louisiana, USA.

Zhibin Lu, Pan Du, and Jian-Yun Nie. 2020. Vgcn-bert: Augmenting bert with graph embedding for text classification. In *Advances in Information Retrieval*, pages 369–382, Cham. Springer International Publishing.

Volodymyr Lyubinets, Taras Boiko, and Deon Nicholas. 2018. Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, pages 271–275.

Senthil Mani, Anush Sankaran, and Rahul Aralikatte. 2019. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, CoDS-COMAD '19, pages 171–179, New York, NY, USA. Association for Computing Machinery.

Matteo Marcuzzo, Alessandro Zangari, Andrea Albarelli, and Andrea Gasparetto. 2022. Recommendation systems: an insight into current development and future research challenges. *IEEE Access*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013*, volume abs/1301.3781.

Mara Pistellato, Luca Cosmo, Filippo Bergamasco, Andrea Gasparetto, and Andrea Albarelli. 2018. Adaptive albedo compensation for accurate phase-shift coding. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2450–2455.

Marco Polignano, Pierpaolo Basile, Marco de Gemmis, Giovanni Semeraro, and Valerio Basile. 2019. AlBERTo: Italian BERT language understanding model for NLP challenging tasks based on tweets. In *Proceedings of the Sixth Italian Conference on Computational Linguistics, CLiC-it 2019*, volume 2481 of *CEUR Workshop Proceedings*, Bari, Italy.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Aleksandra Revina, Krisztian Buza, and Vera G. Meister. 2020. It ticket classification: The simpler, the better. *IEEE Access*, 8:193380–193395.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Carlos N. Silla and Alex A. Freitas. 2011. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1):31–72.

Hirotaka Tanaka, Hiroyuki Shinnou, Rui Cao, Jing Bai, and Wen Ma. 2019. Document classification by word embeddings of BERT. In *16th International Conference of the Pacific Association for Computational Linguistics, PACLING 2019*, pages 145–154, Hanoi, Vietnam. Springer Singapore.

Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, and Ann Yuan. 2020. The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 107–118, Online. Association for Computational Linguistics.

A. Torsello, A. Gasparetto, L. Rossi, L. Bai, and E.R. Hancock. 2014. Transitive state alignment for the quantum jensen-shannon kernel. *Lect. Notes Comput. Sci.*, 8621:22–31.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, Florence, Italy. Association for Computational Linguistics.

Libo Yang. 2021. Fuzzy output support vector machine based incident ticket classification. *IEICE Transactions on Information and Systems*, E104.D(1):146–151.

Jun Yuan, Jesse Vig, and Nazneen Rajani. 2022. Isea: An interactive pipeline for semantic error analysis of nlp models. In *27th International Conference on Intelligent User Interfaces*, IUI '22, pages 878–888, New York, NY, USA. Association for Computing Machinery.

Wubai Zhou, Wei Xue, Ramesh Baral, Qing Wang, Chunqiu Zeng, Tao Li, Jian Xu, Zheng Liu, Larisa Shwartz, and Genady Ya. Grabarnik. 2017. Star: A system for ticket analysis and resolution. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 2181–2190, New York, NY, USA. Association for Computing Machinery.

Dawei Zhu, Michael A. Hedderich, Fangzhou Zhai, David Adelani, and Dietrich Klakow. 2022. Is BERT robust to label noise? a study on learning with noisy labels in text classification. In *Proceedings of the Third Workshop on Insights from Negative Results in NLP*, pages 62–67, Dublin, Ireland. Association for Computational Linguistics.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

Paolo Zicari, Gianluigi Folino, Massimo Guarascio, and Luigi Pontieri. 2021. Discovering accurate deep learning based predictive models for automatic customer support ticket classification. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, pages 1098–1101, New York, NY, USA. Association for Computing Machinery.

A Embedding summarization strategies

Table 4 describes all the summarization strategies tested in this work. The "*cls last* (p)" strategy refers to the one adopted in the original BERT paper, using the `[CLS]` token embedding passed through the NSP prediction layer, with a `tanh` activation, commonly referred to as "pooled" embedding. In contrast, all other *cls* strategies use the un-pooled embedding, meaning that it is used directly as provided by the encoder without additional processing.

Table 5 reports the complete set of performance metrics measured to test the effectiveness of the summarization strategies. In the main article, only the accuracy and F_1-score are reported. All metrics besides accuracy refer to the macro-averaged metrics; metrics are computed separately for each label and then averaged, irrespective of the labels' frequency.

B Details on the Linux Bugs dataset

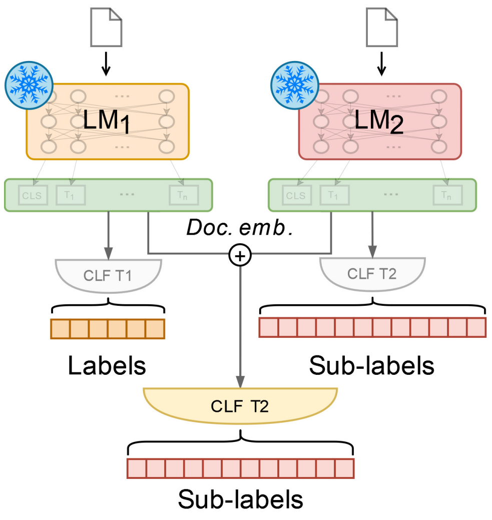
The preprocessing procedure applied to the Linux Bugs dataset discards bug reports without a valid message text, applies lowercasing to all text, and concatenates the issue title with the message body. The final dataset, after preprocessing, contains 35,050 bug descriptions, 17 first-level labels, and 73 sub-labels. The average number of characters per-ticket is 2,026 and each ticket is labeled with exactly one label and one sub-label. The label hierarchy for a subset of 3 macro-labels is shown in Fig. 2, and histograms with the frequency of labels and sub-labels are shown in Figs. 3 and 4, respectively.

One example of a pre-processed and lowercased bug report is displayed in Listing 1, and the effect of the text cleaning procedure described in Section 5.1 on the same body of text is showcased in Listing 2. As can be seen, these tickets are rich in technical information, like stack traces and error messages, mixed with text written in natural language. Misspellings are also quite frequent, since bugs are often filed by non-native English speakers.

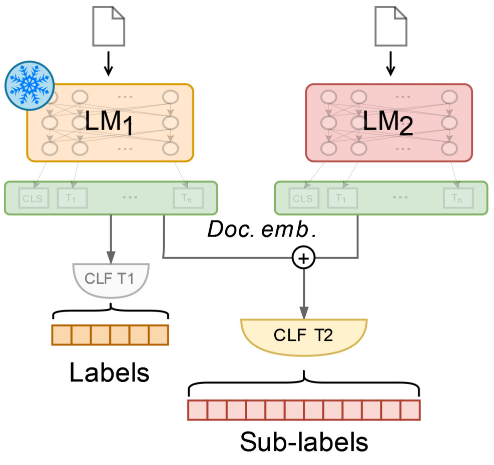
C Other experimental details

All tests are run using PyTorch 1.11.0 and Python 3.10 using an NVIDIA RTX 2080 Ti. We use the AdamW optimizer (Loshchilov and Hutter, 2019) during training.

BERT-large validation The BERT-large models are validated on the T2 task to find the most suitable learning rate (choosing between $1e^{-5}$, $2e^{-\{5,6\}}$)



(a) ML-LM.



(b) Supported-LM.

Figure 1: Two-level classification models.

and number of epochs. In this case, we use gradient accumulation to emulate this batch size value, due to the larger model size and computational limitations.

Multi-level validation We search for the best learning rate and number of epochs for the multi-level models separately, as `Supported-LM` contains a trainable LM, while `ML-LM` does not. In the first case, we validate with learning rates $1e^{-5}$ and $2e^{-\{6,5,4\}}$ with both base and large BERT, and use $2e^{-5}$ (2 epochs) and $1e^{-5}$ (3 epochs), respectively, during tests. The classification layer of the `ML-LM` models is validated with learning rates set to $1e^{-\{5,3\}}$ and $2e^{-\{5,4\}}$, and final tests are run with $2e^{-4}$ (1 epoch) and $1e^{-3}$ (1 epoch) respectively for the smaller and larger BERT models.

D Error analysis

We share a brief analysis of the per-class performance of our models (ML-BERT and Supported-BERT) in Table 6. In particular, the table reports per-class metrics of three of the top-performing labels, as well as of three of the worst-performing labels. The average length of tickets in that class (number of characters) and the number of samples of that class present in the test and train splits, respectively, are also displayed. The average ticket length is, in general, a good representative of actual length, as the outliers are few in this dataset, and are usually very long tickets (which will be truncated by the tokenizer in any case). Classes with a higher number of samples usually perform better, though this is not always the case (as exemplified by the Networking_Wireless category). Finally, the worst values of the F_1 score seem to be mostly dominated by low recall, which indicates a high number of false negatives. In this regard, it would be interesting to test different over- and under-sampling techniques, such as to verify whether this can help in the classification of these classes.

An analysis performed through specialized tools could reveal whether these classes are hard to classify because of linguistically-relevant factors, such as the lack of discriminative terms. For instance, it could be argued that certain labels are semantically similar (*e.g.*, Drivers_Network, Drivers_network-wireless, and Networking_Wireless), and might therefore contain semantically similar tickets. We plan to expand this analysis in future work, looking into more refined tools aimed at interpreting the inner workings of LMs, such as Errudite (Wu et al., 2019), the Language Interpretability Tool (LIT) (Tenney et al., 2020) and iSEA (Yuan et al., 2022). For example, the LIT would allow to directly examine individual examples that the model performs poorly upon as well as performing an investigation of the reasoning behind the model's decisions.

Table 4: Summarization strategies for document embeddings.

Basis	**Strategy**	**Emb. size**	**Description**
cls	*last* (p)	d	`[CLS]` embedding from last layer (default strategy)
	last		`[CLS]` embedding from last layer without pooling
	avg_h		Average of the `[CLS]` embeddings (no pooling) from the last h layers
	$concat_h$	$d * h$	Concatenation of the `[CLS]` embeddings from the last h layers
avg	*last*	d	Average of all embeddings* from the last layer
	avg_h		Average of the average of embeddings from the last h layers
	$concat_h$	$d * h$	Concatenation of the average of embeddings from the last h layers
max	*last*	d	Column-wise maximum of all embeddings* from the last layer
	avg_h		Average of the max of embeddings from the last h layers
	$concat_h$	$d * h$	Concatenation of the max of embeddings from the last h layers
max_min	*last*	$d * 2$	Concatenation of the max and min of embeddings from the last layer
	avg_h		As above, but averaging vectors from the last h layers
max_avg	*last*	$d * 2$	Concatenation of the max and avg of embeddings from the last layer
	avg_h		As above, but averaging vectors from the last h layers
sum_nor	*last*	d	Sum of token embeddings divided by its norm (*i.e.*, normalized sum)
	$concat_h$	$d * h$	Like *last* but concatenating the last h layers

* Excluding special symbols (*e.g.* `[CLS]` and padding).

Table 5: Test set results* with BERT classifier on T2 comparing summarization strategies on the Linux Bugs dataset. Best results are outlined in bold.

Basis	**Strategy**	**Acc**	**F$_1$**	**Prec**	**Rec**
cls	*last* (p)†	0.518 [± 0.006]	0.354 [± 0.009]	0.386 [± 0.009]	0.365 [± 0.006]
	last	0.566 [± 0.012]	0.446 [± 0.018]	0.479 [± 0.027]	0.452 [± 0.017]
	avg_2	0.531 [± 0.010]	0.393 [± 0.012]	0.420 [± 0.018]	0.398 [± 0.012]
	$concat_2$	0.535 [± 0.010]	0.400 [± 0.014]	0.426 [± 0.015]	0.401 [± 0.012]
	$concat_3$	**0.571** [± 0.008]	0.456 [± 0.013]	**0.498** [± 0.013]	**0.458** [± 0.015]
	$concat_4$	0.568 [± 0.009]	**0.457** [± 0.014]	0.490 [± 0.013]	0.458 [± 0.017]
	$concat_5$	0.565 [± 0.012]	0.456 [± 0.013]	0.486 [± 0.011]	0.458 [± 0.018]
avg	*last*	0.525 [± 0.008]	0.387 [± 0.010]	0.420 [± 0.015]	0.388 [± 0.010]
	avg_2	0.522 [± 0.005]	0.383 [± 0.013]	0.409 [± 0.021]	0.387 [± 0.010]
	$concat_2$	0.523 [± 0.007]	0.390 [± 0.009]	0.411 [± 0.015]	0.394 [± 0.011]
max	*last*	0.522 [± 0.011]	0.385 [± 0.014]	0.415 [± 0.011]	0.391 [± 0.014]
	avg_2	0.519 [± 0.007]	0.375 [± 0.013]	0.395 [± 0.021]	0.387 [± 0.014]
	$concat_2$	0.518 [± 0.006]	0.373 [± 0.015]	0.401 [± 0.021]	0.383 [± 0.012]
max_min	*last*	0.522 [± 0.010]	0.377 [± 0.011]	0.395 [± 0.019]	0.395 [± 0.010]
	avg_2	0.522 [± 0.009]	0.374 [± 0.010]	0.395 [± 0.019]	0.385 [± 0.011]
max_avg	*last*	0.516 [± 0.007]	0.381 [± 0.012]	0.406 [± 0.017]	0.390 [± 0.012]
	avg_2	0.519 [± 0.006]	0.379 [± 0.007]	0.402 [± 0.011]	0.392 [± 0.007]
sum_nor	*last*	0.406 [± 0.018]	0.171 [± 0.017]	0.192 [± 0.023]	0.206 [± 0.016]
	$concat_2$	0.379 [± 0.013]	0.135 [± 0.019]	0.149 [± 0.022]	0.179 [± 0.021]
	$concat_5$	0.388 [± 0.015]	0.135 [± 0.015]	0.149 [± 0.015]	0.180 [± 0.014]

* Standard deviation over 6 runs is reported in brackets.

† Pooled, using *cls pooled* strategy.

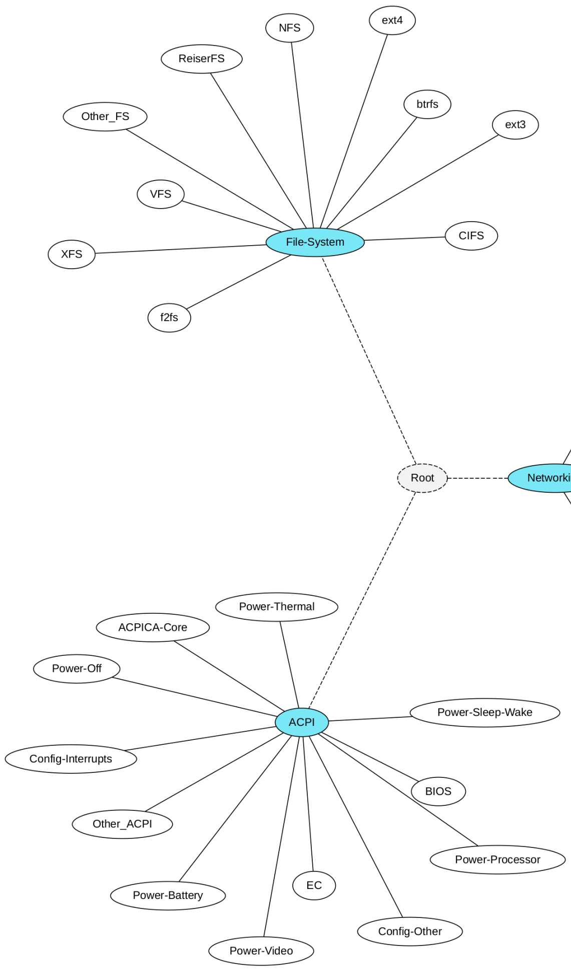


Figure 2: Example of 3 macro-categories (in blue) and their children from the Linux Bugs dataset.

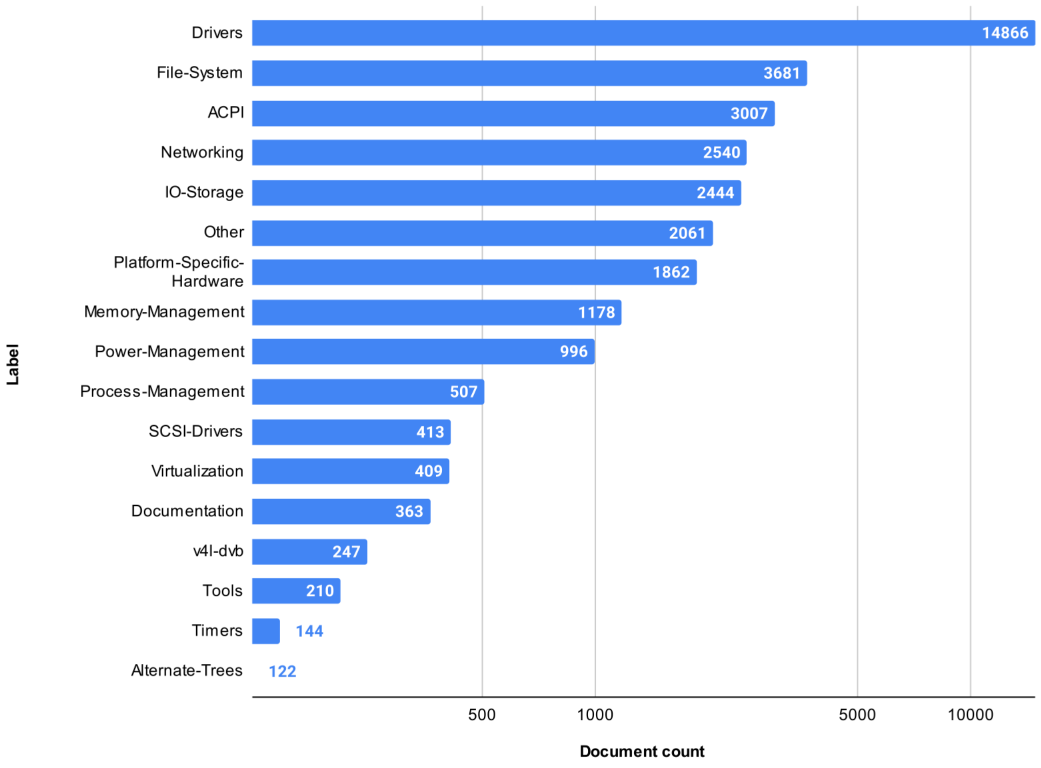


Figure 3: Frequency count of first-level labels in the Linux Bugs dataset.

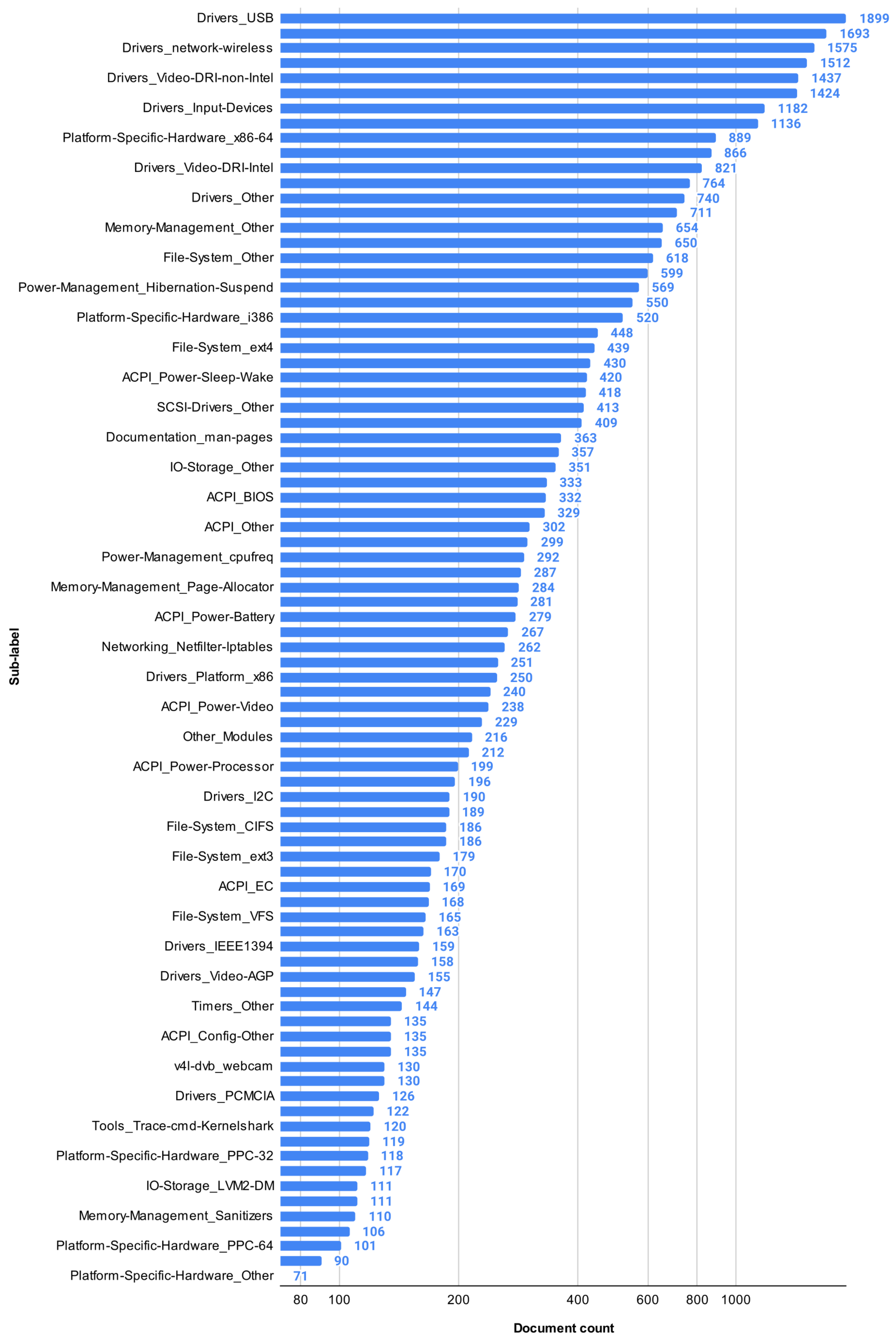


Figure 4: Frequency count of second-level labels in the Linux Bugs dataset, obtained by flattening labels and sub-labels.

Product: Drivers

Component: Network

Title: `kernel crashes after stop network and remove e100.`

Description:

```
exact kernel version:linux-2.5.51 distribution:redhat 8.0+linux2.5.51
hardware environment:intel stl2 mother boar d problem description:  compile
e100 as kernel module, insmod e100 and start the network.  then stop network and
remove e100 together.  then kernel crashes in random places.  for example:  use
command :  insmod e100 /etc/init.d/network start /etc/init.d/network stop; rmmod
e100 then the kernel crashes.  eflags:  00010887 eip is at cascade+0x25/0x60
eax:  defd02b8 ebx:  00000001 ecx:  00000000 edx:  c150a4c0 esi:  c150acd4
edi:  c150acd4 ebp:  c150acd4 esp:  c0559f1c ds:  0068 es:  0068 ss:  0068
process swapper (pid:  0, threadinfo=c0558000 task=c0497f60) stack:  00000000
c0559f94 00000001 c150a4c0 fffffffd 00000000 c012abb2 c150a4c0 c150acd4
c010baf5 00000000 00000000 c0559f94 00000001 c0553d68 fffffffd 00000000 c0125bb5
c150a4c0 00000046 00000000 00000000 c0559f94 c01070c0 call trace:  [<c012abb2>]
__run_timers+0x1b2/0x202 [<c010baf5>] handle_irq_event+0x45/0x70 [<c0125bb5>]
do_softirq+0xc5/0xd0 [<c01070c0>] default_idle+0x0/0x50 [<c0116f3d>]
smp_apic_timer_interrupt+0xcd/0x130 [<c01070c0>] default_idle+0x0/0x50
[<c010a3a6>] apic_timer_interrupt+0x1a/0x20 [<c01070c0>] default_idle+0x0/0x50
[<c01070c0>] default_idle+0x0/0x50 [<c01070ea>] default_idle+0x2a/0x50
[<c010718a>] cpu_idle+0x3a/0x50 [<c0105000>] rest_init+0x0/0x30 code:  0f 0b
6f 01 ba a7 41 c0 8b 18 89 44 24 04 89 3c 24 e8 a5 0c
```

Listing 1: A lowercased ticket from the Linux Bugs dataset; Fields "Product" and "Component" are used as labels and sub-labels respectively.

Product: Drivers

Component: Network

Title: `kernel crashes after stop network and remove e100.`

Description:

```
      exact kernel distribution:redhat hardware environment:intel stl2
mother boar d problem description:  compile e100 as kernel module, insmod e100
and start the network.  then stop network and remove e100 together.  then
kernel crashes in random places.  for example:  use command :  insmod e100
/etc/init.d/network start /etc/init.d/network stop; rmmod e100 then the kernel
crashes.  eflags:  eip is at eax:  ebx:  ecx:  edx:  esi:  edi:  ebp:  esp:  ds:
0068 es:  0068 ss:  0068 process swapper (pid:  0, stack:  fffffffd fffffffd
call trace:  code:  0f 0b 6f 01 ba a7 41 c0 8b 18 89 44 24 04 89 3c 24 e8 a5 0c
```

Listing 2: The same ticket from Listing 1, preprocessed with the more aggressive strategy.

Table 6: Label-specific performances and statistics for three of the best performing and three of the worse performing classes. The values are calculated and averaged over the usual 3-fold CV.

Model	Label	F_1	Prec	Recall	Avg ticket len	# in test	# in train
	Drivers_Network	0.958	0.953	0.963	2825.49	379	1008
	Drivers_Hardware-Monitoring	0.891	0.861	0.925	1365.25	62	74
`ML-BERT`	File-System_VFS	0.842	0.789	0.905	2494.28	143	110
(base)	...						
	Tools_Trace-cmd-Kernelshark	0.328	0.436	0.268	1429.63	24	80
	Documentation_man-pages	0.196	0.284	0.163	907.67	41	242
	Networking_Wireless	0.064	0.107	0.074	2406.04	45	434
	Drivers_Network	0.963	0.961	0.965	2825.49	379	1008
	Drivers_Hardware-Monitoring	0.888	0.887	0.892	1365.25	62	74
`Supp-BERT`	File-System_VFS	0.829	0.767	0.902	2494.28	143	110
(base)	...						
	Tools_Trace-cmd-Kernelshark	0.104	0.667	0.057	1429.63	24	80
	Documentation_man-pages	0.225	0.472	0.163	907.67	41	242
	Networking_Wireless	0.079	0.192	0.052	2406.04	45	434