# On Provenance in Model Fusion*

**Annie Amorim[1], João Vitor Moraes[1], Débora Pina[2], Aline Paes[1], Daniel de Oliveira[1]**

[1]Institute of Computing – Universidade Federal Fluminense (UFF)

[2]PESC/COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

annieamorim,joaovitormoraes@id.uff.br, dbpina@cos.ufrj.br,

alinepaes,danielcmo@ic.uff.br

***Abstract.*** *As the number of domain-specific language models grows at a fast pace, managing multiple specialized models becomes a challenge. Model fusion is an alternative that combines models to improve robustness. However, fusion workflows involve steps such as model selection, merging strategies, evaluation, and repeated iterations. Selecting the best combination of source models requires understanding the full data derivation path. In this paper, we present* `FusionProv`*, an approach compliant with the W3C PROV recommendation that captures provenance from fusion workflows to support traceability and analytics. We evaluate* `FusionProv` *in a hate-speech detection application that leverages fused language models.*

## 1. Introduction

Language Models have become key components of applications that perform AI-based tasks, *e.g.*, text classification and code generation. Their adoption has led to the development of several fine-tuned, specialized models for downstream tasks [Aftan et al. 2023]. Today, users have to maintain collections of models specialized for different languages, datasets, and contexts. Although this specialization improves performance in most cases, it also creates a challenge: selecting, maintaining, and deploying many overlapping models is costly and difficult to scale [Howard et al. 2018]. Model Fusion [Matena et al. 2022] is an alternative to this scenario. Instead of managing several independent models, fusion techniques combine multiple trained models into a single model [Yadav et al. 2023]. Depending on the strategy used, the fused model may show different capabilities, improve robustness, and simplify deployment.

However, obtaining a useful fused model is not a one-shot process, since users have to explore several design decisions, *e.g.*, which source models to combine, in what order, and under which fusion strategy. They must also configure parameter values, validation protocols, *etc*. This way, model fusion is an iterative process that requires multiple executions to converge on a final model [Choshen et al. 2022, Matena et al. 2022]. One can note that even minor changes in the selected source models or fusion parameters can impact the outcome, ranging from performance gains to severe degradation [Amorim et al. 2025]. The iterative nature of the model fusion workflow introduces a data management challenge: answering questions such as "*Which source models influenced the selected result?*" and "*Which strategy consistently outperformed alternatives?*". Existing platforms can store metrics, parameters, and logs, but they usually provide limited support for reconstructing derivation paths across multiple fusion iterations. We argue that provenance [Herschel et al. 2017] can be a basis

*__Acknowledgments__: The authors thank CAPES, CNPq, and FAPERJ for their partial support in carrying out this work. Additionally, the authors declare that LLM-based tools were used exclusively for text revision.

for addressing this problem, as it records the origin, transformations, and dependencies of data products. In model fusion, provenance data represents how data and models are transformed throughout the model fusion workflow. Thus, provenance can capture dependencies among source models, fusion strategies, checkpoints, and the final deployed models. Such information enables analytical queries, comparative analyses, and informed model selection.

This paper presents `FusionProv`, an approach for capturing provenance in model fusion workflows. Unlike [Amorim et al. 2025], which evaluates fusion strategies and predictive performance, this paper focuses on the provenance layer, including fusion-specific calls, templates, and a seven-transformation schema for recording and querying fusion executions. `FusionProv` extends an existing provenance capturing library by introducing a set of fusion-specific provenance calls and templates that can be embedded into fusion scripts. These provenance calls capture predefined metadata and workflow events, enabling *a posteriori* auditing, traceability, and querying of fusion executions. `FusionProv` adopts the W3C PROV recommendation [Moreau et al. 2015] to represent the entities, activities, and agents involved in iterative fusion processes. The approach supports both metadata-oriented analysis and relationship-oriented provenance queries, allowing users to inspect the complete lifecycle of fused models. We evaluate `FusionProv` through a case study involving a model fusion workflow for online hate speech detection in Portuguese [Amorim et al. 2025]. The evaluation comprises two representative provenance queries designed to evaluate traceability and lineage reconstruction throughout the fusion lifecycle. This suggests that provenance-aware model fusion can support traceability, reproducibility, and decision support.

## 2. Background and Related Work

**Model Fusion.** It is an approach that aims to combine multiple fine-tuned models into a single model to improve robustness and reduce the complexity of managing multiple specialized models. For BERT-based [Devlin et al. 2019] language models, fusion can be performed directly in the parameter space, comprising the weights and biases of the model's layers. There are multiple merging strategies that can be performed. Simple Merging [Choshen et al. 2022] computes an arithmetic mean of the parameters of models with the same architecture, as it assumes that all parameters contribute equally to the final model. Fisher Merging [Matena et al. 2022] weights each parameter according to its estimated importance, as measured by the Fisher Information Matrix, assigning greater influence to parameters that are more sensitive to the loss function. TIES-MERGING [Yadav et al. 2023] performs fusion using the difference between the fine-tuned parameters and the pre-trained model's parameters, *i.e.*, task vectors. It retains the top $k\%$ highest-magnitude updates from task vectors, resolves parameter conflicts through sign consensus across models, averages aligned updates, and optionally scales the resulting task vector by a factor $\lambda$ before applying it to the pre-trained model.

**Related Work.** Several studies have addressed provenance capture in machine learning (ML) workflows. However, to the best of our knowledge, none specifically target the model fusion context. [Souza et al. 2019] propose PROV-ML, a provenance model that tracks data, transformations, parameters, and results throughout the ML lifecycle. [Schelter et al. 2017] propose an approach for automatically capturing metadata and provenance data from experiments executed in ML frameworks. [Kerzel et al. 2021] introduce MLProvLab, a system for capturing, comparing, and visualizing provenance in ML notebooks. [Schlegel and Sattler 2023] propose MLflow2PROV, which extracts provenance data from MLflow projects. [Padovani et al. 2025] present yProv4ML, an extensible library for

provenance collection in large-scale ML systems, with additional emphasis on resource consumption and energy efficiency. [Pina et al. 2022] propose Keras-Prov for provenance capture in deep learning applications, while DLProv [Pina et al. 2025] enables users to trace provenance throughout deep learning workflows, including data acquisition, preprocessing, training, model selection, and deployment.

## 3. Proposed Approach: `FusionProv`

`FusionProv` is an approach for capturing, storing, and querying provenance data from model fusion workflows. Provenance data makes explicit the relationships among datasets, base models, checkpoints, fusion strategies, intermediate artifacts, and final fused models. Rather than focusing only on final performance values, `FusionProv` records how each fused model was produced, which artifacts contributed to it, and which configuration choices affected its generation. We discuss its architecture and provenance schema next.

**Architecture.** `FusionProv` extends an existing provenance capture library, and captures both prospective (*p-prov*) and retrospective (*r-prov*) provenance [Freire et al. 2008]. While *p-prov* describes the workflow structure, *i.e.*, transformations and parameters, *r-prov*, in turn, captures metadata about data produced during execution, *e.g.*, parameter values. Figure 1 presents the architecture of `FusionProv`, which has five modules: (i) Pre-built Templates, (ii) Client, (iii) Server, (iv) Provenance Database, and (v) Provenance Exporter.

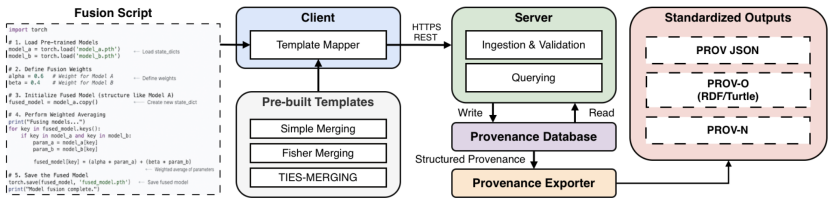


**Figure 1. Architecture of `FusionProv`.**

Execution begins with the submission of the script that implements the fusion workflow. `FusionProv` does not impose any specific format or programming language, allowing users to implement the script in their preferred language. The script contains all data transformations, and the *Client* receives it as input, injecting provenance-capturing calls (*p-prov* and *r-prov*) into selected parts of the script to collect provenance data. Determining which provenance data to capture is nontrivial, given the large number of parameters and hyperparameters involved in fusion workflows. To address this challenge, `FusionProv` provides a set of *Pre-built Templates* associated with each transformation. These templates define a superset of parameters and hyperparameters that may be captured during workflow execution.

These templates follow a general specification of a fusion workflow with seven transformations (Figure 2): (i) *Data Preparation*, (ii) *Data Split*, (iii) *Data Balancing*, (iv) *Data Tokenization*, (v) *Apply Model*, (vi) *Fuse Checkpoint*, and (vii) *Generalize*. The *Data Split* receives labeled datasets and partitions them into training, validation, and test sets. *Data Balancing* balances the data when necessary, while *Data Tokenization* tokenizes the datasets according to the selected model. The *Apply Model* transformation fine-tunes the model for the specific task. The *Fuse Checkpoint* transformation is the core step of the workflow, as it consumes fine-tuned models (*i.e.*, checkpoints) and produces fused models. Finally, *Generalize* consolidates the final set of fused model candidates by considering both individual checkpoints and fused models. For each candidate, `FusionProv` records its provenance

data, *e.g.*, checkpoints, fusion techniques, parameters, and evaluation metrics. The user then selects, from this predefined list in the templates, the specific parameters and hyperparameters to be recorded for each transformation. In `FusionProv` we assume that before and after each transformation, a relation (*i.e.*, a table in the *Provenance Database*) records the parameters consumed and produced during execution, thereby registering the data derivation path. For example, in Figure 2, the *Data Preparation* receives the paths to the input files as parameters, while the *Generalize* registers the metrics of the final fused model.

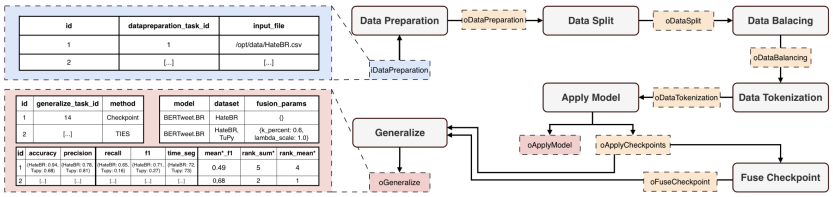


**Figure 2. The general fusion workflow and the intermediate relations in `FusionProv`.**

Once the *Fusion Script* is provided and the *Pre-built Templates* are configured by the user, the *Template Mapper* component within the *Client* acts as an intermediary between the fusion script and the provenance-capturing mechanism (*i.e.*, Server). It identifies events generated during the execution of the fusion script, matches them to the corresponding template, and forwards their metadata to the injected provenance calls. Each provenance call transmits metadata to the *Server* through a REST API. These calls capture specific provenance data, such as the start of a transformation, consumed parameter values, evaluation metrics, and generated artifacts. Upon receiving the provenance data, the *Server* validates it against the selected template to ensure that all expected fields are present and consistent. Once validated, the provenance data is stored in the *Provenance Database*. This database can be queried to analyze relationships among activities, entities, artifacts, parameters, and results throughout the workflow lifecycle. Finally, the *Provenance Exporter* enables provenance data to be exported in standard formats, *e.g.*, PROV-JSON.

**Provenance Schema.** The provenance schema adopted by `FusionProv` comprises the relations illustrated in Figure 2, which represent the data and parameters consumed and produced at each transformation. Each relation is implemented as a table in MonetDB DBMS. Tables representing parameter values exclusively consumed by a transformation are prefixed with 'i' followed by the name of the corresponding transformation, *e.g.*, `iDataPreparation`. Similarly, tables containing metrics and values produced by a transformation are prefixed with 'o' followed by the name of the transformation, *e.g.*, `oGeneralize`. In addition, the provenance database contains a table that records the execution of transformations (`Task`). Each consumed and produced relation maintains foreign keys referencing the tasks that consume and produce it, thereby establishing the data derivation path throughout the workflow. It is important to highlight that the provenance database is adaptable to evolving provenance requirements, *i.e.*, if a user decides to capture an additional metric, a new parameter can be added to the provenance call, and the corresponding attribute will be automatically created in the provenance database. The full provenance schema is available at this link[1].

## 4. Experimental Evaluation

**Case Study and Experimental Setup.** The selected case study consists of a model fusion workflow for online hate speech detection in Brazilian Portuguese. We used five datasets: (i)

[1]`https://osf.io/q2g9u/overview?view_only=97045fbcd3074badb688f44bd5d5f512`

HateBR, (ii) OLID-BR, (iii) ToLD-Br, (iv) HSPT, and (v) Tupy. This workflow combines multiple datasets, models, and fusion strategies, highlighting the importance of traceability throughout the fusion process. The datasets were partitioned into training and test sets using a 2/3–1/3 split ratio, with 10% of the training data further reserved for validation. Regarding models, we employed four BERT-based architectures: (i) Bernice, (ii) TwHIN-BERT, (iii) BERTweet.BR, and (iv) BERTimbau. Training was configured for a maximum of 10 epochs, using a learning rate of $5 \times 10^{-5}$, batch size of 16, weight decay of 0.01, and early stopping after two consecutive epochs without improvement in validation loss. The recorded evaluation metrics included accuracy, precision, recall, and F1-score, enabling each checkpoint to be associated with its corresponding model, dataset, epoch, and derived fused models. Although 72 checkpoints were generated during training, only the best checkpoint for each model–dataset pair was selected for the generalization phase. The evaluation considered three fusion strategies: (i) Simple Merging, (ii) Fisher Merging, and (iii) TIES-MERGING. For TIES-MERGING, all combinations of the parameters $k\%$ and $\lambda$ were evaluated using the values 0.2, 0.6, and 1.0. In total, 44 fused models were generated, enabling provenance-based comparisons across fusion techniques and parameter configurations.

**Provenance Queries.** The evaluation of `FusionProv` was conducted through SQL queries executed over the provenance database. These queries were designed to evaluate whether the provenance data can answer two key questions in model fusion workflows: (i) which model candidates achieved the best generalization performance after fusion, and (ii) which artifacts, configurations, and decisions contributed to those candidates. Figure 3 presents query `Q1`, which identifies the best-performing candidates on the test datasets. To achieve this, `Q1` computes, for each candidate, the sum of its rankings by F1-score across all datasets. In the event of ties, candidates are ordered by their mean F1-score. This ranking criterion favors models with more consistent performance across datasets, rather than prioritizing only the highest overall average. Query `Q1` enables a comparison between individual models and fused models. The results shown in Figure 3 indicate that the ten highest-ranked candidates were all generated using TIES, differing only in the selected base model and fusion parameters.

```sql
select
    g.id,
    g.candidate,
    g.variant,
    g.model,
    g.rank_sum_test_f1 as rank_sum,
    g.mean_test_f1 as mean_f1
from ds_ogeneralize g
where g.mean_test_f1 is not null
order by
    g.rank_sum_test_f1 asc,
    g.mean_test_f1 desc
limit 10;
```

| id | candidate | variant | model | rank_sum | mean_f1 |
|---|---|---|---|---|---|
| 31 | ties_bernice_hatebr-hspt-olidbr-t | ties-k0.2-l1 | Bernice | 39 | 0.504 |
| 58 | ties_twhinbert_hatebr-hspt-olidbr | ties-k0.2-l1 | TwHIN-BERT | 56 | 0.479 |
| 39 | ties_bertimbau_hatebr-hspt-olidbr | ties-k0.2-l0.6 | BERTimbau | 57 | 0.485 |
| 34 | ties_bernice_hatebr-hspt-olidbr-t | ties-k0.6-l1 | Bernice | 58 | 0.48 |
| 55 | ties_bertweetbr_hatebr-hspt-olidb | ties-k1-l1 | BERTweet.br | 58 | 0.48 |
| 43 | ties_bertimbau_hatebr-hspt-olidbr | ties-k0.6-l1 | BERTimbau | 60 | 0.49 |
| 46 | ties_bertimbau_hatebr-hspt-olidbr | ties-k1-l1 | BERTimbau | 60 | 0.485 |
| 49 | ties_bertweetbr_hatebr-hspt-olidb | ties-k0.2-l1 | BERTweet.br | 60 | 0.48 |
| 52 | ties_bertweetbr_hatebr-hspt-olidb | ties-k0.6-l1 | BERTweet.br | 63 | 0.473 |
| 40 | ties_bertimbau_hatebr-hspt-olidbr | ties-k0.2-l1 | BERTimbau | 65 | 0.488 |

**Figure 3. SQL query used to identify the highest-ranked final model candidates.**

Figure 4 presents query `Q2`, which reconstructs the data-derivation path for the five highest-ranked fused models. Starting from each final model candidate ID, the query retrieves the fusion strategy, the base model, the checkpoints and their epochs, the datasets, the tokenizer, the vocabulary size, the maximum input length, and the intermediate metrics. This query enables relating each candidate's final performance to the artifacts and configurations used to generate it. The best candidate, identified as candidate 31, corresponds to Bernice with the `ties-k0.2-l1` strategy. Candidate 58 uses the same fusion strategy, `ties-k0.2-l1`, but with TwHIN-BERT. We can also compare two fused models. Candidates 31 and 34 share the same Bernice checkpoints and the same TIES-MERGING strategy, but differ in their fusion parameters. Since candidate 31 achieved a better ranking, this indicates that changing the $k$ parameter, while keeping the other parameters fixed, affected

the final performance. Among the highest-ranked candidates, the top-three candidates use `k=0.2`, and all use $\lambda \geq 0.6$, with $\lambda = 1$ being predominant. This result shows that provenance can support the analysis of fusion parameters by revealing recurring configurations among high-performing candidates.

```
select
    g.id,
    f.variant,
    f.model,
    f.checkpoint_ids,
    f.datasets
from ds_ogeneralize g
left join ds_ofusecheckpoints f
    on f.id = g.fusion_id
where g.id in (31, 58, 39, 34, 55)
order by g.id;
```

| id | variant | model | checkpoints_ids | datasets |
|---|---|---|---|---|
| 31 | ties-k0.2-l1 | Bernice | [9, 23, 38, 54, 67] | hatebr,hspt,olidbr,toldbr,tupy |
| 58 | ties-k0.2-l1 | TwHIN-BERT | [14, 28, 43, 57, 70] | hatebr,hspt,olidbr,toldbr,tupy |
| 39 | ties-k0.2-l0.6 | BERTimbau | [4, 20, 35, 51, 64] | hatebr,hspt,olidbr,toldbr,tupy |
| 34 | ties-k0.6-l1 | Bernice | [9, 23, 38, 54, 67] | hatebr,hspt,olidbr,toldbr,tupy |
| 55 | ties-k1-l1 | BERTweet.br | [1, 17, 32, 46, 61] | hatebr,hspt,olidbr,toldbr,tupy |

```
select
    g.id,
    c.id as ckpt,
    c.dataset as data,
    c.model,
    t.tokenizer,
    t.vocab_size as vocab,
    t.max_length as max,
    c.epoch,
    c.eval_f1 as eval,
    c.test_f1 as test
from ds_ogeneralize g
left join ds_ocheckpoints c
    on c.id in g.checkpoint_ids  ← Representative: checkpoint id occurs in the textual list
left join ds_otokenization t
    on t.dataset = c.dataset
    and t.model = c.model
    and t.dataflow_version = c.dataflow_version
where g.id in (31, 58, 39, 34, 55)
order by
    g.id,
    c.dataset,
    c.id;
```

| id | ckpt | data | model | tokenizer | vocab | max | epoch | eval | test |
|---|---|---|---|---|---|---|---|---|---|
| 31 | 9 | hatebr | Bernice | XLMRobertaToken | 250000 | 128 | 3 | 0.711 | 0.695 |
| 31 | 23 | hspt | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.547 | 0.561 |
| 31 | 38 | olidbr | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.554 | 0.531 |
| 31 | 54 | toldbr | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.198 | 0.247 |
| 31 | 67 | tupy | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.485 | 0.495 |
| 58 | 14 | hatebr | TwHIN-BERT | XLMRobertaToken | 250002 | 512 | 3 | 0.587 | 0.651 |
| 58 | 28 | hspt | TwHIN-BERT | XLMRobertaToken | 250002 | 512 | 3 | 0.647 | 0.618 |
| 58 | 43 | olidbr | TwHIN-BERT | XLMRobertaToken | 250002 | 512 | 3 | 0.665 | 0.653 |
| 58 | 57 | toldbr | TwHIN-BERT | XLMRobertaToken | 250002 | 512 | 1 | 0.244 | 0.221 |
| 58 | 70 | tupy | TwHIN-BERT | XLMRobertaToken | 250002 | 512 | 1 | 0.302 | 0.37 |
| 39 | 4 | hatebr | BERTimbau | BertTokenizerFa | 29794 | 512 | 1 | 0.625 | 0.659 |
| 39 | 20 | hspt | BERTimbau | BertTokenizerFa | 29794 | 512 | 1 | 0.557 | 0.531 |
| 39 | 35 | olidbr | BERTimbau | BertTokenizerFa | 29794 | 512 | 1 | 0.681 | 0.67 |
| 39 | 51 | toldbr | BERTimbau | BertTokenizerFa | 29794 | 512 | 3 | 0.295 | 0.282 |
| 39 | 64 | tupy | BERTimbau | BertTokenizerFa | 29794 | 512 | 1 | 0.228 | 0.238 |
| 34 | 9 | hatebr | Bernice | XLMRobertaToken | 250000 | 128 | 3 | 0.711 | 0.695 |
| 34 | 23 | hspt | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.547 | 0.561 |
| 34 | 38 | olidbr | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.554 | 0.531 |
| 34 | 54 | toldbr | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.198 | 0.247 |
| 34 | 67 | tupy | Bernice | XLMRobertaToken | 250000 | 128 | 1 | 0.485 | 0.495 |
| 55 | 1 | hatebr | BERTweet.br | BertweetTokeniz | 64005 | 128 | 1 | 0.597 | 0.606 |
| 55 | 17 | hspt | BERTweet.br | BertweetTokeniz | 64005 | 128 | 1 | 0.675 | 0.648 |
| 55 | 32 | olidbr | BERTweet.br | BertweetTokeniz | 64005 | 128 | 2 | 0.544 | 0.547 |
| 55 | 46 | toldbr | BERTweet.br | BertweetTokeniz | 64005 | 128 | 1 | 0.201 | 0.218 |
| 55 | 61 | tupy | BERTweet.br | BertweetTokeniz | 64005 | 128 | 2 | 0.474 | 0.511 |

**Figure 4. SQL queries for reconstructing the lineage of the top-ranked candidates.**

In addition to fusion artifacts, the query `Q2` retrieves metadata that are relevant for reproducibility. Bernice uses an `XLM-RoBERTa` tokenizer, a vocabulary of 250,000 tokens, and a maximum input length of 128. TwHIN-BERT uses the same tokenizer but with a maximum input length of 512. BERTimbau uses `BertTokenizerFast`, with a vocabulary of 29,794 tokens and a maximum input length of 512, while BERTweet.BR uses a `BERTweetTokenizer`, with a vocabulary of 64,005 tokens and a maximum input length of 128. These metadata provide context for the final results and show that `FusionProv` can connect performance, preprocessing choices, and model configuration. The intermediate metrics further help explain the candidates' behavior. Although the individual Bernice checkpoints do not achieve the best scores on all datasets, their fusion through TIES-MERGING produced the candidate with the best aggregated generalization performance.

## 5. Conclusion

`FusionProv` is a provenance-based approach that supports traceability, reproducibility, and analytics in model fusion workflows. The proposed approach organizes and records information about datasets, base models, checkpoints, fusion strategies, parameters, intermediate artifacts, and metrics, enabling the reconstruction of the derivation paths of fused models. By adopting the W3C PROV model and building on the fusion workflow evaluated in [Amorim et al. 2025], `FusionProv` provides a structured, queryable representation of the model-fusion lifecycle. The evaluation was conducted through a case study on hate speech detection involving multiple datasets, BERT-based models, and three fusion strategies. The results show that provenance queries enable identifying the candidates with the best aggregated performance, reconstructing the lineage of the highest-ranked models, and relating their results to the configurations, checkpoints, and parameters used. These results indicate that provenance capture can go beyond merely storing final metrics, providing effective support for comparative analysis and model selection in fusion scenarios.

## References

Aftan, S. et al. (2023). A survey on bert and its applications. In *2023 20th Learning and Technology Conference (L&T)*, pages 161–166. IEEE.

Amorim, A. et al. (2025). Exploring language model fusion to improve generalization in portuguese hate speech detection. In *2025 28th FUSION*, pages 1–8.

Choshen, L. et al. (2022). Fusing finetuned models for better pretraining. *CoRR*, abs/2204.03044.

Devlin, J. et al. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of the 2019 NAACL*, pages 4171–4186, Minneapolis. ACL.

Freire, J. et al. (2008). Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21.

Herschel, M. et al. (2017). A survey on provenance: What for? what form? what from? *The VLDB Journal*, 26(6):881–906.

Howard, J. et al. (2018). Universal language model fine-tuning for text classification. In *Proc. of the 56th Annual Meeting of the ACL, Melbourne*, pages 328–339. ACL.

Kerzel, D. et al. (2021). Towards tracking provenance from machine learning notebooks. In *International Conference on Knowledge Discovery and Information Retrieval*.

Matena, M. et al. (2022). Merging models with fisher-weighted averaging.

Moreau, L. et al. (2015). The rationale of prov. *Journal of Web Semantics*, 35:235–257.

Padovani, G. et al. (2025). Provenance tracking in large-scale machine learning systems. In *Workshop Proceedings of the 54th International Conference on Parallel Processing*, ICPP Workshops '25, page 167–174, New York, NY, USA. Association for Computing Machinery.

Pina, D. et al. (2022). Capturing provenance from deep learning applications using keras-prov and colab: a practical approach. *Journal of Information and Data Management*, 13(5).

Pina, D. et al. (2025). Breadcrumbs for your deep learning model: Following provenance traces with dlprov. *Software Impacts*, 23:100730.

Schelter, S. et al. (2017). Automatically tracking metadata and provenance of machine learning experiments.

Schlegel, M. and Sattler, K.-U. (2023). Mlflow2prov: Extracting provenance from machine learning experiments. In *Proceedings of the Seventh DEEM*. ACM.

Souza, R. et al. (2019). Provenance data in the machine learning lifecycle in computational science and engineering. In *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 1–10.

Yadav, P. et al. (2023). Ties-merging: Resolving interference when merging models. In *Neural Information Processing Systems*.