

The GRADIEND Python Package: An End-to-End System for Gradient-Based Feature Learning

Jonathan Drechsel and Steffen Herbold

Faculty of Computer Science and Mathematics

University of Passau

{jonathan.drechsel, steffen.herbold}@uni-passau.de

Abstract

We present *gradiend*, an open-source Python package that operationalizes the GRADIEND method for learning feature directions from factual-counterfactual MLM and CLM gradients in language models. The package provides a unified workflow for feature-related data creation, training, evaluation, visualization, persistent model rewriting via controlled weight updates, and multi-feature comparison. We demonstrate *gradiend* through an English pronoun running example, a semantic sentiment use case that evaluates lexical generalization to held-out target words, and a large-scale feature comparison.

1 Introduction

Interpretability is essential for responsible deployment of modern language models, which are highly capable yet opaque (Molnar, 2025). Accordingly, many interpretability methods have emerged, including input attributions (Lundberg and Lee, 2017; Sundararajan et al., 2017), counterfactual analyses (Wachter et al., 2017), and activation-based interventions (Zhang and Nanda, 2024). However, these tools largely operate at analysis time and rarely yield reusable, feature-level objects that support cross-model comparison or persistent behavioral changes. Conversely, weight-space steering methods such as task vectors (Ilharco et al., 2023) and weight arithmetic (Fierro and Roger, 2026) produce persistent updates but infer directions indirectly from fine-tuning deltas (effectively accumulated gradient updates (Zhou et al., 2025)).

GRADIEND (Drechsel and Herbold, 2026) offers an interpretability-driven alternative: it learns reusable *feature directions* directly from feature-specific gradient signals and instantiates them as controlled parameter-space interventions. In this system demonstration, we present *gradiend*, an open-source Python package implementing GRADIEND for **learning, comparing, and manipulating**

feature directions in language models. The library provides a unified, end-to-end workflow covering feature-specific data construction, feature direction learning from language model gradients, evaluation, scalable multi-feature comparison, and controlled weight updates to create *persistent* rewritten model variants. It exposes this workflow through a Hugging Face Trainer-style API, making the GRADIEND method readily usable in practice.

We demonstrate the package through (i) an English pronoun running example covering the full *gradiend* workflow, (ii) a semantic sentiment feature that evaluates generalization to unseen target words, (iii) large-scale feature comparisons using weight overlap and cross-encoding, and (iv) a pruning ablation quantifying efficiency gains.

This work introduces *gradiend* as the first reusable Python package for the GRADIEND workflow and extends prior work (Drechsel and Herbold, 2026; Drechsel et al., 2026) with:

- new feature settings beyond the original experiments, showing that *gradiend* can be readily applied beyond the original experiments, including the learning of semantic features and cross-feature analysis;
- pruning utilities that reduce memory, runtime, and storage costs for large-scale analyses by improving the efficiency while still allowing effective feature learning;
- standardized multi-feature comparison tools, including cross-encoding, which turns comparisons between pairwise GRADIEND models into feature-level comparisons to support reproducible and comparable research;
- a flexible, end-to-end Trainer-style interface for the full GRADIEND workflow that minimizes the feature-specific implementation effort;

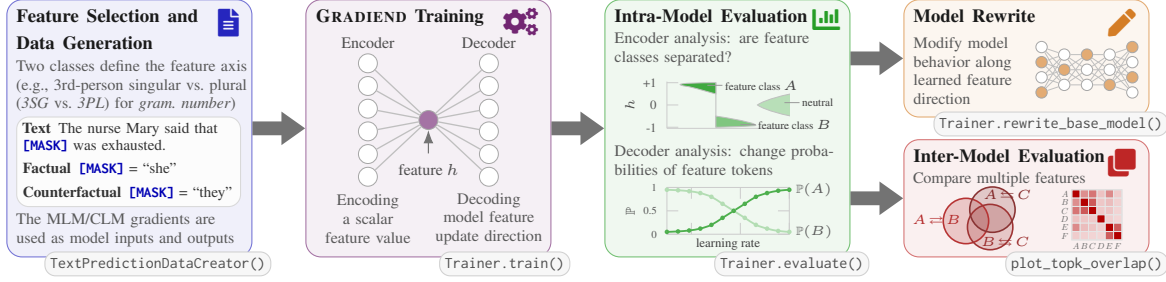


Figure 1: GRADIEND workflow and package overview.

System	Runtime activation intervention	Model-weight rewrite	Learns feature representation	Targeted feature learning	Feature/edit diagnostics	Multi-feature comparison
TransformerLens (Nanda and Bloom, 2022)	✓	✗	✗	✗	✗	✗
pyvene (Wu et al., 2024)	✓	✗	✓	✓	✓	✗
NNsight (Fiotto-Kaufman et al., 2025)	✓	✗	✗	✗	✗	✗
EasyEdit (Wang et al., 2024)	✗	✓	✗	✗	✓	✗
EasyEdit2 (Xu et al., 2025)	✓	✗	✓	✓	✓	✗
steering-vectors (Chanin and Tan, 2024)	✓	✗	✓	✓	✗	✗
SAELens (Bloom et al., 2024)	✓	✗	✓	✗	✓	✗
gradiend (ours)	✗	✓	✓	✓	✓	✓

Table 1: Capability/API comparison of gradiend with neighboring reusable systems. ✓ indicates direct support as a primary package workflow, and ✗ indicates no primary package support under the criterion.

- documentation and examples that demonstrate the package on multiple feature types, including morphosyntactic and semantic features.

The gradiend is published via pypi at <https://pypi.org/project/gradiend>, its source code is released under the Apache 2.0 license at <https://github.com/aieng-lab/gradiend>, with documentation at <https://aieng-lab.github.io/gradiend/>.

2 Related Work

Interpretability and model-modification systems differ in the workflows they support and the artifacts they produce. Input-attribution libraries such as SHAP (Lundberg and Lee, 2017), Captum (Kokhlikyan et al., 2020), and Inseq (Sarti et al., 2023) typically provide example or token-level explanations for specific predictions or generations. Interactive analysis systems like LIT (Tenney et al., 2020) support dataset-level slicing and counterfactual inspection. Mechanistic interpretability toolkits such as TransformerLens (Nanda and Bloom, 2022), pyvene (Wu et al., 2024), and NNsight (Fiotto-Kaufman et al., 2025) expose activations and internal components, and support activation-space interventions.

Sparse autoencoders (SAEs; Bricken et al. 2023) extract sparse components from activation spaces and have been used to uncover interpretable features. Reusable SAE tooling such as SAELEns

(Bloom et al., 2024) supports training and visualization of SAE features. These features are discovered unsupervised and commonly require post-hoc interpretation, and interventions are most often applied at runtime. Model editing methods such as ROME (Meng et al., 2022) and MEMIT (Meng et al., 2023), often accessed via EasyEdit (Wang et al., 2024), instead produce persistent model weight changes via targeted knowledge updates. Activation-steering systems such as EasyEdit2 (Xu et al., 2025) and steering-vectors (Chanin and Tan, 2024) learn feature-like steering artifacts that are applied at runtime rather than as persistent model rewrites. Recent weight-space steering methods, such as task vectors (Ilharco et al., 2023) and weight arithmetic (Fierro and Roger, 2026), can produce persistent parameter updates, but they infer directions indirectly from fine-tuning deltas (Zhou et al., 2025).

GRADIEND complements these lines by supporting a feature-based workflow: learning feature directions directly from contrastive gradient signals, enabling comparison of many features at scale, and applying controlled weight updates that yield persistent rewritten models. Table 1 summarizes this positioning for the gradiend Python package, which implements GRADIEND as an end-to-end API for feature learning, rewriting, and comparison. We include close neighboring systems for model-internal intervention, editing, steering, and feature discovery.

3 The GRADIEND Python Package

The GRADIEND method (Drechsel and Herbold, 2026) learns a reusable *feature direction* from factual-counterfactual gradients: an encoder compresses each gradient into a single class-separating scalar, and a decoder maps this scalar to a parameter-update direction which, when applied to the base model, yields a controlled and *persistent* behavior shift along the learned feature. The gradiend package turns this workflow into a reusable interface that standardizes data creation, training, evaluation, rewriting, and feature comparison. The package further extends the workflow, including pruning utilities and a feature-class comparison tool (cross-encoding).

Figure 1 summarizes the end-to-end workflow implemented by gradiend. The key design choice is to treat a *feature direction* as a persistent, reusable artifact learned from factual-counterfactual model gradients: once trained, it can be evaluated, used to derive a rewritten checkpoint, and compared against other learned directions without rerunning training. A typical run follows five steps:

 **Feature Selection and Data Creation:** specify feature classes and construct class-specific text-prediction instances.

 **GRADIEND Training:** train a GRADIEND model on MLM or CLM gradients to obtain a feature direction.

 **Intra-Model Evaluation:** validate class separation and select rewrite hyperparameters.

 **Model Rewrite:** export a controlled modified base model with changed feature behavior.

 **Inter-Model Evaluation:** quantify similarity between multiple learned features.

In the remainder of this section, we illustrate these steps with a simple English *grammatical-number* feature contrasting third-person singular pronouns (3SG; *he/she/it*) with plural pronouns (3PL; *they*). This running example focuses on the core workflow. Sections 4 and 5 evaluate the same interface on more complex semantic and large-scale feature settings, while additional package features are explained in the documentation.

3.1 Feature Selection and Data Generation

A feature is defined by a set of *feature classes* (i.e., discrete variants of the feature, e.g., 3SG and 3PL). A single GRADIEND run selects two classes and learns a direction along the induced axis between them. The library supports feature learning via **text prediction** objectives such as Masked Language Modeling (MLM; Devlin et al. 2018) and Causal Language Modeling (CLM; Radford et al. 2018). Concretely, it constructs prediction instances in which a feature-specific target is to be predicted.

The gradiend data module centers around TextPredictionDataCreator and lightweight feature class specifications via TextFilterConfig. Users provide a base dataset (e.g., as a Hugging Face identifier) and one TextFilterConfig per feature class, defined through string-based word matches and optionally spaCy tags.

For the running example (3SG vs. 3PL), simple word matching is sufficient because the selected pronouns are not syncretic. In addition to the class-specific datasets, gradiend can create feature-neutral evaluation data by excluding texts that contain any feature-specific target words.

```
from gradiend import TextPredictionDataCreator,
    TextFilterConfig
creator = TextPredictionDataCreator(
    base_data='wikimedia/wikipedia',
    hf_config='20231101.en',
    feature_targets=[
        TextFilterConfig(["he", "she", "it"], id="3SG"),
        TextFilterConfig(["they"], id="3PL"),
    ],
    min_left_context_words=10,
)
training_data = creator.generate_training_data(
    max_size_per_class=2500)
neutral_data = creator.generate_neutral_data(
    additional_excluded_words=["i", "we", "you"],
    max_size=1000,
)
```

3.2 GRADIEND Training

Training is exposed via a Hugging Face-Trainer-style interface: users specify a base model, the data, standard training parameters (e.g., batch size and learning rate) and GRADIEND-specific settings (e.g., which base-model parameters are used for gradient extraction, and which gradient variants to use as source and target signals). The framework supports automatic multi-seed training and best-model selection to ensure stable model convergence. A training run produces a structured experiment directory with checkpoints, metrics, and plots for evaluation, rewriting, and comparison.

To make large-scale studies feasible, we extend the GRADIEND workflow with pruning. Pre- and

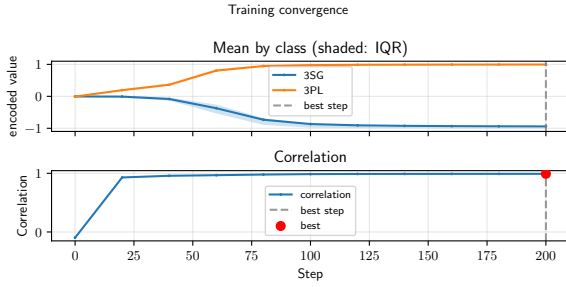


Figure 2: Training convergence example plot.

post-pruning shrink the effective input and output dimensionality by removing low-importance dimensions (measured by absolute value). Pre-pruning estimates importance from averaged gradients computed on a small random sample of training instances, while post-pruning ranks dimensions by the learned GRADIEND parameters. Appendix C evaluates the resulting runtime, storage, and feature-quality trade-offs. The snippet below illustrates GRADIEND training yielding the convergence plot shown in Figure 2.

```
from gradiend import TrainingArguments, TextPredictionTrainer,
    PrePruneConfig, PostPruneConfig
args = TrainingArguments(
    train_batch_size=8,
    max_steps=200,
    eval_steps=20,
    learning_rate=1e-5,
    experiment_dir="runs/demonstration",
    pre_prune_config=PrePruneConfig(n_samples=8, topk=0.1), #
    keep top 10% dimensions
    post_prune_config=PostPruneConfig(topk=0.01), # prune
    again after training; keep 1% of remaining dimensions
)
trainer = TextPredictionTrainer(
    model="bert-base-cased",
    data=training_data,
    eval_neutral_data=neutral_data,
    args=args,
)
trainer.train()
trainer.plot_training_convergence(class_spread="iqr") # Fig. 2
```

3.3 Intra-Model Evaluation

Intra-model evaluation validates whether a *single* GRADIEND model yields a class-separating feature direction that is usable for controlled rewriting.

Encoder evaluation checks whether the learned scalar feature separates the two feature classes and whether feature-neutral examples remain close to zero. It reports separation metrics (e.g., Pearson correlation) and can generate distribution plots such as Figure 3, showing a clear separation for the running example.

```
enc = trainer.evaluate_encoder(plot=True) # Fig. 3
print("Correlation:", enc["correlation"]) # 0.9688
print("Means:", enc["mean_by_feature_class"]) # {'3PL':
    0.9656, '3SG': -0.8577}
```

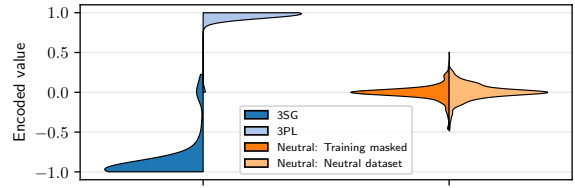


Figure 3: Encoder analysis: distribution of learned feature values across the two target classes and neutral data.

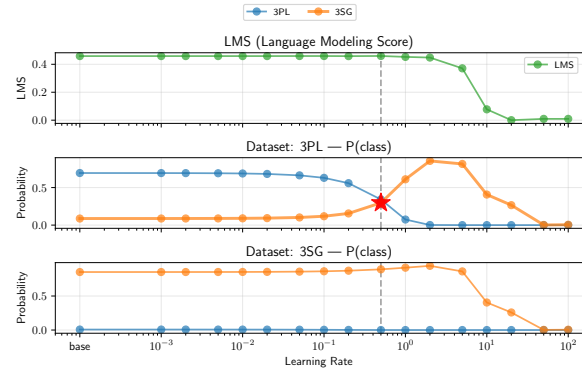


Figure 4: Decoder analysis: shift of target token probabilities; selected settings increase the target class while maintaining near-base-model LMS.

Decoder evaluation assesses whether the decoded parameter update can be applied to the base model in a controlled way. Concretely, gradiend performs a small grid search over a learning rate multiplier, applies the decoded update at each scale to the base weights, and measures the induced shift in target-token probabilities. Overly large learning rates can degrade general language modeling behavior (Drechsel and Herbold, 2026). Therefore, by default, gradiend selects the largest learning rate that satisfies a language-modeling constraint named Language Modeling Score (LMS), yielding the strongest increase in the desired target-class probabilities while keeping overall model behavior close to the base model. Figure 4 illustrates this trade-off for the 3SG target class.

```
dec = trainer.evaluate_decoder(plot=True, target_class="3SG")
# Fig. 4
print(dec["3SG"]["learning_rate"]) # 1.0
```

3.4 Model Rewrite

GRADIEND can export a persistently modified checkpoint whose behavior is shifted along the learned feature direction (e.g., increasing token probabilities of one target class). The rewrite can increase (default) or decrease a selected feature class, with learning rate chosen based on the previ-

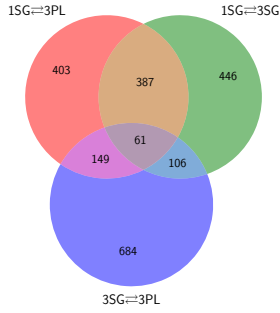


Figure 5: Venn diagram example plot.

ous decoder evaluation.

```
changed_model = trainer.rewrite_base_model(decoder_results=dec
, target_class="3SG")
```

3.5 Inter-Model Evaluation

Inter-model evaluation compares *multiple* trained GRADIEND runs for the same base model to quantify how similarly different features are encoded in that model’s parameters. Because *gradiend* learns *persistent* feature directions, runs can be analyzed and compared after training using only their stored artifacts, enabling scalable studies across many features.

The library provides several tools for inter-model evaluation, including Top-*k* weight overlap, cosine similarity, and cross-encoding. As a simple example, Figure 5 visualizes the Top-*k* overlap between three trained GRADIEND runs as a Venn diagram. The same comparison tools also scale to larger feature collections: Section 5 applies Top-*k* overlap as a pairwise heatmap of GRADIEND models, and uses cross-encoding to obtain a feature-class-level view.

```
# assuming trainers is a list of Trainer instances over the
same base model
models = [trainer.get_model() for trainer in trainers]
plot_topk_overlap(models, topk=1000) # Fig. 5
```

4 Sentiment as a Semantic Feature

The running example in Section 3 demonstrates *gradiend* on a *morphosyntactic* pronoun feature. In prior work, we have also shown that this approach works for more complex morphosyntactic features, i.e., how German articles are determined based on noun case and gender (Drechsel et al., 2026). In this section, we use sentiment to demonstrate a package-level extension of the established workflow: *gradiend* can construct, split, train, evaluate, and visualize features whose classes

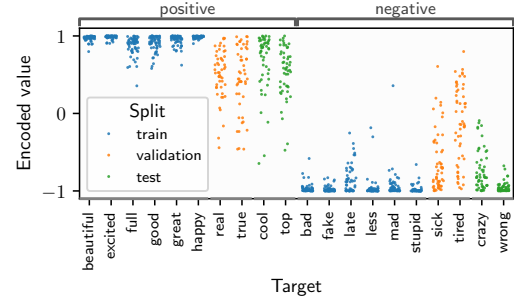


Figure 6: Sentiment encoder values grouped by target for bert-base-cased.

are defined by multiple semantic target words. We train a sentiment GRADIEND contrasting various positive and negative target words. Conceptually, this follows the same masked prediction setup: contexts like “*The movie was [MASK]*” induce gradients for positive (e.g., *great*, *exciting*) and negative (e.g., *bad*, *sad*) targets, from which GRADIEND learns a positive-negative feature.

Unlike the pronoun example, sentiment provides a larger target vocabulary, i.e., various positive and negative targets. We split the data by target word, so validation and test examples contain sentiment words not observed during training. This tests to what extent the learned feature captures sentiment beyond individual lexical targets, while still using the same package workflow: *gradiend* groups multiple target words into each feature class, filters and masks their occurrences, trains the positive-negative GRADIEND, and produces the target-grouped visualization in Figure 6. Implementation details are in Appendix B.

Figure 6 shows clear separation for training targets and weaker but visible generalization to the small set of held-out target words. The target-level view also reveals lexical differences within the sentiment classes: targets such as *late*, *sick*, and *tired* are encoded less negatively than other negative targets in their split, potentially indicating weaker negative sentiment associations in the model. This illustrates that *gradiend* supports semantic feature workflows beyond morphosyntactic examples, while its by-target plots can reveal lexical variation within a class.

5 Large-Scale Analysis and System Validation

Beyond applying GRADIEND to individual features, this section demonstrates *gradiend*’s package-level support for large-scale, multi-feature compari-

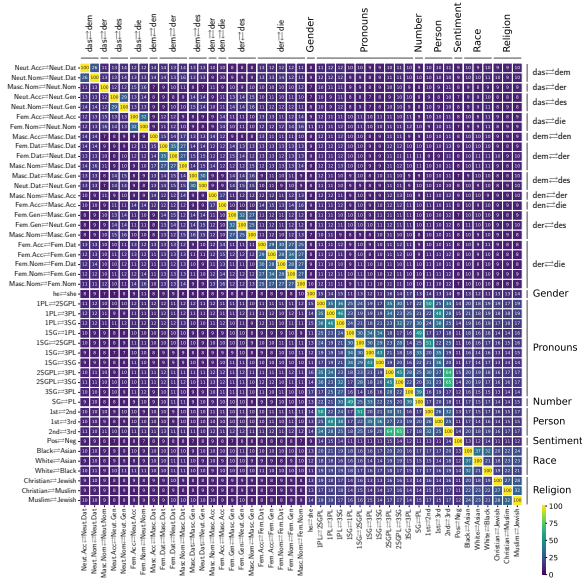


Figure 7: Proportion of Top-1000 weight overlap of affected parameters.

son by quantifying how similarly different features are encoded in parameters of the base model. We evaluate this functionality on a heterogeneous set of features using the multilingual model google-bert/bert-base-multilingual-cased (Devlin et al., 2018). The analysis includes reproduced use-cases from prior work, namely English gender (*he/she*), race (*Asian/Black/White*), religion (*Christian/Jewish/Muslim*) (Drechsel and Herbold, 2026), and German gender-case articles (Drechsel et al., 2026). We further include the English pronoun paradigm from Section 3, extended to all five pronoun classes: *1SG* (*I*), *1PL* (*we*), *2SGPL* (*you*), *3SG* (*he/she/it*), and *3PL* (*they*). From these classes, the framework can not only train each pair of these, but also combine features together to derive more general grammatical number and person features. Using the `class_merge_map` of the Trainer, feature classes can be merged together such as the singular instances *1SG* and *3SG* as well as plural classes *1PL* and *3PL* to derive a general *grammatical number* feature. Finally, we include the sentiment feature from Section 4 (*Pos/Neg*), yielding a collection of morphosyntactic, semantic, and social features.

The analysis uses gradiend’s multi-seed mode, which aggregates results over convergent random seeds to obtain more stable estimates. The subsequent large-scale analyses report means over three convergent seeds.

Figure 7 reports proportional overlap of the Top-

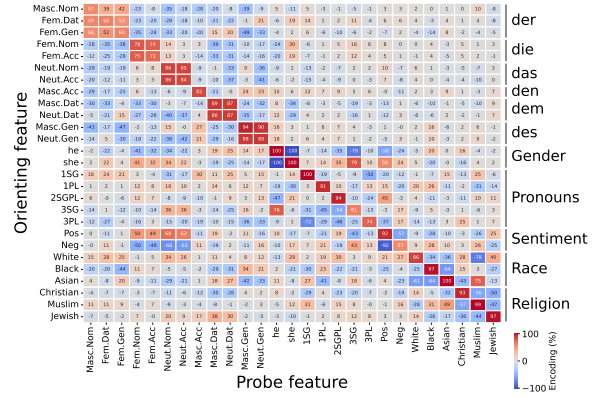


Figure 8: Cross-encoding matrix. Rows are orienting features, i.e., the feature classes used to align pairwise GRADIEND models. Columns are probe features whose examples are encoded.

k most affected parameters ($k=1000$) between 45 pairwise GRADIEND runs trained over pairs of feature classes described above. This package-level visualization scales the small-set Venn diagrams used in prior case studies (Drechsel et al., 2026) to large feature collections. The block structure around the diagonal indicates that related features often share a non-trivial intersection of affected parameters. In particular, German article transitions that share the same surface form exhibit higher overlap, consistent with the syncretism-driven analysis by Drechsel et al. (2026). Cross-group overlap is generally smaller, although race and religion show comparatively larger overlap than most other cross-family pairs.

As an additional package-level comparison tool, we complement this weight-space view by introducing an oriented cross-encoding matrix, formalized in Appendix D. For each feature class, cross-encoding takes the GRADIEND models that contain this class, aligns their signs toward it, and measures how strongly they encode probes from the other feature classes. The two analyses are complementary: Top- k overlap compares pairwise GRADIEND features by their affected parameters, whereas cross-encoding asks how GRADIEND features encode probes from another. Thus, the former is a direct and inexpensive comparison of trained models, while the latter requires additional encoder evaluations but yields a feature-class-level view that is easier to interpret.

The feature-level matrix in Figure 8 reveals cross-domain associations that are less direct to read from the pairwise overlap matrix. For example, *Muslim* is positively associated with *Asian*

and *Black*, negatively associated with *White*, and is the only religion feature with a clearly negative (*Neg*) sentiment association. In contrast, *Christian* and *Jewish* are more positively aligned with *White*, while only the race feature *Asian* is positively aligned with *Muslim*. These patterns suggest that the model may encode *Muslim* in a racialized and comparatively negative way. Separately, English pronoun gender (*he/she*) is associated with German grammatical gender in some cases (*Masc./Fem.*), indicating that social and grammatical gender are not fully separated in the representation. To scope these interpretations, we report seed-level standard deviations in Figure 11. Since no null controls are included, we interpret the patterns as hypotheses surfaced by the package rather than as conclusive findings.

Overall, the analysis illustrates a central strength of gradiend as a package: feature directions can be trained, compared, and analyzed at scale through a unified interface, enabling both replication of prior findings and rapid exploration of new hypotheses.

6 Conclusions and Future Work

We presented gradiend, an open-source Python package that makes the GRADIEND method available as a reusable, end-to-end workflow for learning, evaluating, rewriting, and comparing feature directions in language models. By treating learned feature directions as persistent artifacts and providing standardized utilities for intra-model evaluation and inter-model comparison, the toolkit supports reproducible analyses that scale from a single feature experiment to large, multi-feature studies. In this work, we further introduced pruning for efficient large-scale training and feature-comparison tools such as cross-encoding, which moves beyond pairwise GRADIEND comparisons toward feature-level analysis, and demonstrated their utility on new use cases.

References

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa-Anke, and Leonardo Neves. 2020. [TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification](#). In *Proceedings of Findings of EMNLP*.

Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. 2024. Saelens. <https://github.com/decoderresearch/SAELens>.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermy, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, and 6 others. 2023. [Towards monosemanticity: Decomposing language models with dictionary learning](#). *Transformer Circuits Thread*.

David Chanin and Daniel Tan. 2024. steering-vectors: Steering vectors for transformer language models in PyTorch / Hugging Face. <https://github.com/steering-vectors/steering-vectors>. Python package.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. [BERT: pre-training of deep bidirectional transformers for language understanding](#). *CoRR*, abs/1810.04805.

Jonathan Drechsel, Erisa Bytyqi, and Steffen Herbold. 2026. [Understanding or memorizing? A case study of German definite articles in language models](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9626–9652, San Diego, California, United States. Association for Computational Linguistics.

Jonathan Drechsel and Steffen Herbold. 2026. [GRADIEND: Feature learning within neural networks exemplified through biases](#). In *The Fourteenth International Conference on Learning Representations*.

Constanza Fierro and Fabien Roger. 2026. [Steering language models with weight arithmetic](#). In *The Fourteenth International Conference on Learning Representations*.

Jaden Fried Fiotto-Kaufman, Alexander Russell Loftus, Eric Todd, Jannik Brinkmann, Koyena Pal, Dmitrii Troitskii, Michael Ripa, Adam Belfki, Can Rager, Caden Juang, Aaron Mueller, Samuel Marks, Arnab Sen Sharma, Francesca Lucchetti, Nikhil Prakash, Carla E. Brodley, Arjun Guha, Jonathan Bell, Byron C Wallace, and David Bau. 2025. [NNSight and NDIF: Democratizing access to open-weight foundation model internals](#). In *The Thirteenth International Conference on Learning Representations*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. [Editing models with task arithmetic](#). In *The Eleventh International Conference on Learning Representations*.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. [Captum: A unified and generic model interpretability library for pytorch](#). *Preprint*, arXiv:2009.07896.

- Scott M Lundberg and Su-In Lee. 2017. [A unified approach to interpreting model predictions](#). In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. [Locating and editing factual associations in gpt](#). In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22*, Red Hook, NY, USA. Curran Associates Inc.
- Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. 2023. [Mass-editing memory in a transformer](#). In *The Eleventh International Conference on Learning Representations*.
- Saif M. Mohammad and Peter D. Turney. 2013. [Crowdsourcing a word-emotion association lexicon](#). *Computational Intelligence*, 29(3):436–465.
- Christoph Molnar. 2025. [Interpretable Machine Learning: A Guide for Making Black Box Models Explainable](#), 3 edition. Molnar Interpretable ML.
- Neel Nanda and Joseph Bloom. 2022. Transformerlens. <https://github.com/TransformerLensOrg/TransformerLens>.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. [Improving language understanding by generative pre-training](#). Technical report, OpenAI.
- Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. [Semeval-2017 task 4: Sentiment analysis in twitter](#). In *Proceedings of the 11th international workshop on semantic evaluation (SemEval-2017)*, pages 502–518.
- Gabriele Sarti, Nils Feldhus, Ludwig Sickert, and Oscar van der Wal. 2023. [Inseq: An interpretability toolkit for sequence generation models](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 421–435, Toronto, Canada. Association for Computational Linguistics.
- Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. [Axiomatic attribution for deep networks](#). In *Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17*, page 3319–3328. JMLR.org.
- Qwen Team. 2024. [Qwen2.5: A party of foundation models](#).
- 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.
- Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. 2017. [Counterfactual explanations without opening the black box: Automated decisions and the GDPR](#). *CoRR*, abs/1711.00399.
- Peng Wang, Ningyu Zhang, Bozhong Tian, Zekun Xi, Yunzhi Yao, Ziwen Xu, Mengru Wang, Shengyu Mao, Xiaohan Wang, Siyuan Cheng, Kangwei Liu, Yuan-sheng Ni, Guozhou Zheng, and Huajun Chen. 2024. [EasyEdit: An easy-to-use knowledge editing framework for large language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 82–93, Bangkok, Thailand. Association for Computational Linguistics.
- Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah Goodman, Christopher Manning, and Christopher Potts. 2024. [pyvene: A library for understanding and improving PyTorch models via interventions](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*, pages 158–165, Mexico City, Mexico. Association for Computational Linguistics.
- Ziwen Xu, Shuxun Wang, Kewei Xu, Haoming Xu, Mengru Wang, Xinle Deng, Yunzhi Yao, Guozhou Zheng, Huajun Chen, and Ningyu Zhang. 2025. [EasyEdit2: An easy-to-use steering framework for editing large language models](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 522–535, Suzhou, China. Association for Computational Linguistics.
- An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. [Qwen2 technical report](#). *arXiv preprint arXiv:2407.10671*.
- Fred Zhang and Neel Nanda. 2024. [Towards best practices of activation patching in language models: Metrics and methods](#). In *The Twelfth International Conference on Learning Representations*.
- Luca Zhou, Daniele Solombrino, Donato Crisostomi, Maria Sofia Bucarelli, Giuseppe Alessio D’Inverno, Fabrizio Silvestri, and Emanuele Rodolà. 2025. [On task vectors and gradients](#). In *UniReps: 3rd Edition of the Workshop on Unifying Representations in Neural Models*.

A Package Implementation Details

The codebase is organized into five main modules: data, model, trainer, evaluator, and visualizer. Most workflows are accessible through the trainer abstraction, which exposes training, evaluation, and visualization while keeping low-level model internals optional. For custom analyses, GRADIEND model objects can be accessed via `Trainer.get_model()`.

The implementation separates task-specific data and training logic from the core components via task-specific abstractions such as `TextPredictionTrainer`. This design makes it straightforward to add new gradient-producing objectives and modalities with minimal code extensions, e.g., vision models.

The default package installation via `pip install gradiend` builds on core libraries such as `torch` and `transformers` while keeping additional dependencies minimal to reduce version conflicts. This core setup supports data generation, training, evaluation, and model rewriting. Additional functionality can be enabled using:

- `gradiend[data]`: installs libraries for advanced dataset access and generation, including datasets and spacy.
- `gradiend[plot]`: installs visualization libraries such as matplotlib and Venn diagram utilities.
- `gradiend[recommended]`: installs both `[data]` and `[plot]`, and adds `safetensors`, which is automatically used for safer checkpoint serialization when available.

B Sentiment Implementation Details

For the sentiment experiment in Section 4, we construct masked sentences from `tweet_eval` sentiment tweets (Rosenthal et al., 2017; Barbieri et al., 2020) and the NRC Emotion Lexicon (Mohammad and Turney, 2013). We use the positive and negative lexicon categories, restrict candidate targets to adjectives occurring in the tweet corpus, canonicalize them by lowercasing and lemmatization, and retain the ten most frequent words per class, defining the feature classes *Pos* and *Neg*. For each selected target occurrence, `gradiend` replaces the target word with the mask token and uses the original word as the prediction label. To evaluate lexical generalization, we

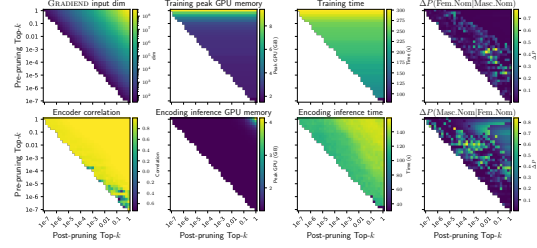


Figure 9: Pruning ablation including cost, encoder quality, and decoder shifts across pre-/post-pruning values. $\Delta P(x|y)$ denotes probability of target x on data y .

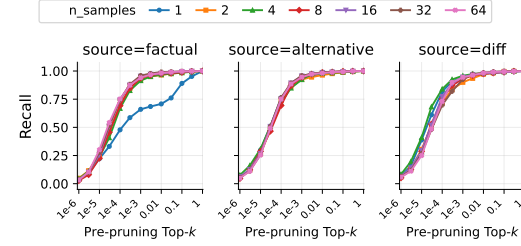


Figure 10: Pre-pruning ablation: recall of the unpruned Top-1000 affected parameters.

split the data by canonical target word (by setting `split_col="heldout"`) with a 60/20/20 train/validation/test split. Hence, all examples of a given target word occur in exactly one split, so validation and test targets are unseen during training. The target-grouped visualization in Figure 6 is produced from the encoder evaluation output using `trainer.plot_encoder_by_target()`.

C Pruning Ablation

Using the *Fem.Nom.* vs. *Masc.Nom.* feature as an example, we evaluate how pruning affects resource usage, encoder quality, decoder-induced probability shifts, and weight-overlap stability. Figure 9 shows that pre-pruning, which reduces the effective GRADIEND dimensionality before training, yields most training-memory savings already at `Top-k=0.1`, since gradient computation still requires the full base model (unless late layers are fully pruned). Post-pruning only affects post-training analysis such as encoding. Memory savings can be substantial (about a factor of three) while training time improves less strongly. Encoder correlations and decoder shifts remain stable except under very aggressive pruning. Decoder shifts vary non-smoothly due to grid-based learning rate selection, but non-aggressive pruned settings often reach at least the unpruned probability shift.

Figure 10 complements this analysis by asking whether the resource savings from pre-pruning

come at the cost of less stable weight-based comparisons. It measures how well different pre-pruning settings recover the unpruned Top-1000 affected parameters. Recall remains high for moderate pre-pruning: Top- $k=0.1$ is close to the unpruned reference across all gradient sources and sample counts, and Top- $k=0.01$ retains at least 90% recall in most settings. Lower sample counts can reduce recall, while using more than eight samples yields little additional benefit for Top- $k \geq 0.01$. The gradient source usually has only a small effect on recall. Although this ablation is limited to one feature and seed, it suggests that source=alternative, topk=0.1, and n_samples=8 are reasonable provisional package defaults: source=alternative performs similarly to source=diff here, but avoids computing both factual and alternative gradients.

More generally, this ablation suggests that moderate pre-pruning may be appropriate when weight-overlap analyses should remain close to the unpruned reference (Figure 10), whereas more aggressive pruning may still be acceptable when the analysis only relies on encoder evaluation or rewriting (Figure 9).

D Cross-Encoding Matrix

Let $\mathcal{F} := \{f_1, \dots, f_K\}$ be feature classes (e.g. *3SG*, *Fem.Nom.*) and $\mathcal{G} := \{G_1, \dots, G_M\}$ a set of pairwise GRADIEND models (e.g., *3SG*↔*3PL*), each trained on an ordered pair $(a_G, b_G) \in \mathcal{F}^2$. We evaluate each G on a shared test pool $\mathcal{D}$. Each probe $x \in \mathcal{D}$ is associated with a factual class $\text{fac}(x) \in \mathcal{F}$ and a counterfactual class $\text{cnf}(x) \in \mathcal{F}$. For a probe feature f_j , let $\mu_G(f_j)$ be the mean encoded value of G over all probes whose GRADIEND source-side class is f_j , i.e., $\text{fac}(x) = f_j$ if G uses factual gradients as input, and $\text{cnf}(x) = f_j$ if G uses counterfactual gradients as input.

For each entry $M_{i,j}$ of the cross-encoding matrix M , f_i is the *orienting feature* and f_j is the *probe feature*: we ask whether GRADIEND models that separate f_i from other features assign high encoder values to f_j probes. Let $\mathcal{G}_{f_i} = \{G : f_i \in \{a_G, b_G\}\}$ and define $\text{sign}_G(f_i) = -1$ if $f_i = a_G$ and $+1$ if $f_i = b_G$. The cross-encoding matrix is then defined as

$$M_{i,j} := \frac{1}{|\mathcal{G}_{f_i}|} \sum_{G \in \mathcal{G}_{f_i}} \text{sign}_G(f_i) \mu_G(f_j).$$

The matrix is generally not symmetric, and hence not a classical similarity matrix: rows aggregate

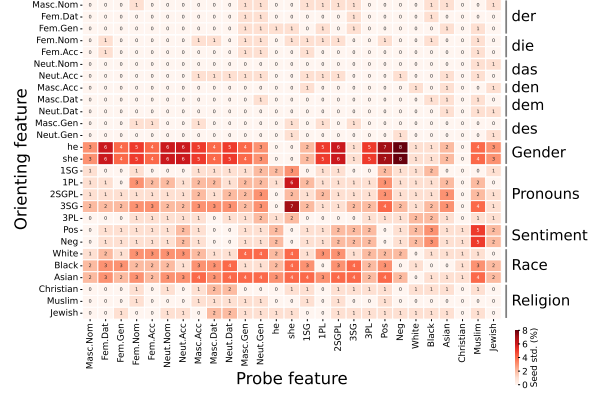


Figure 11: Seed-level standard deviation of the cross-encoding matrix in Figure 8.

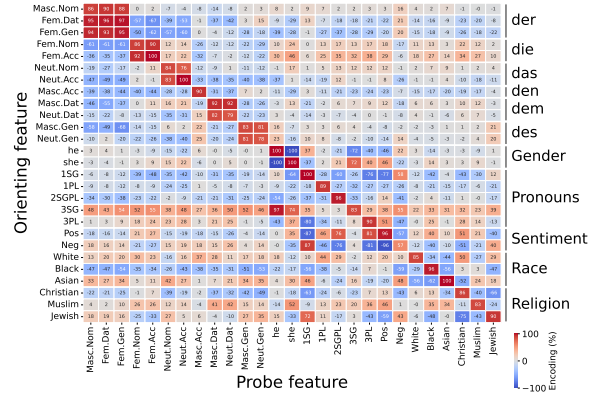


Figure 12: Cross-encoding matrix for Qwen-2.5.

models oriented toward a feature, whereas columns select the probes being encoded. Diagonal entries reflect self-encoding rather than perfect self-similarity and are therefore typically below 1.

E Supplementary Cross-Encoding Analyses

Figure 11 reports seed-level standard deviations for the cross-encoding matrix in Figure 8. The values are generally small relative to the main cross-encoding scale, suggesting that the visualization is not dominated by seed variation. However, this stability check does not replace null controls, so the cross-domain associations in Section 5 remain exploratory.

To evaluate whether the large-scale comparison workflow also applies beyond encoder-only masked language models, we repeat the cross-encoding analysis from Section 5 with Qwen-2.5 0.5B (Yang et al., 2024; Team, 2024) using Causal Language Modeling (CLM) gradients (Radford et al., 2018). Figure 12 shows the resulting heatmap.