TOWARD AI-FRIENDLY DATA QUALITY METRICS: A FEATURE DENSITY PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

In Large Language Models (LLMs) fine-tuning stage, input sequences differ in the proportion of task relevant tokens that help the model predict target sequences, and quantifying this variation can help clarify the relationship between data and fine-tuning performance. Therefore, we introduce Feature Density: the fraction of input tokens whose deletion increases a pretrained model's loss on the target sequence. We validate the tokens it identifies by comparing them to known feature tokens in synthetic data and human-annotated feature tokens in real world data. In synthetic data, Feature Density increases with the proportions of task relevant tokens. Across twelve real-world datasets using Qwen LLMs, Feature Density varies by task, with higher densities in translation and math reasoning. In both settings, Adam, AdamW, Muon, and SGD achieve lower test loss when fine tune on higher density data within the same tasks. These results support Feature Density as a metric of task relevant token proportions and its relation with model fine-tuning performance across different optimizers.

1 INTRODUCTION

LLMs trained on massive datasets can achieve strong performance across a wide range of tasks, including language understanding, code generation, and mathematical reasoning (Brown et al., 2020). A substantial body of research has focused on improving model performance through the effective use of training data and optimization methods such as SGD (Robbins & Monro, 1951), Adam (Kingma & Ba, 2014), AdamW (Loshchilov & Hutter, 2017), and Muon (Jordan et al., 2024). For example, Chinchilla scaling law investigated the relationship between model size and training data volume (Hoffmann et al., 2022), while DeepSeek-V3 (Liu et al., 2024a) and Qwen2.5 (Yang et al., 2024) provided strategies for constructing, organizing, and using training data, highlighting the role of data in LLMs training.

However, training LLMs faces two challenges: smaller performance gains in scaling up and potential data shortages. While LLMs have significantly benefited from scaling model size, computational resources and data size (Kaplan et al., 2020; Hoffmann et al., 2022), enlarging text corpora yields progressively smaller performance gains (Tirumala et al., 2023; Arora & Zanette, 2026; Zhou et al., 2023). This scaling bottleneck is also compounded by the scarcity of high-quality human data. If current trends persist, frontier training datasets may soon approach the available stock of human-generated text (Villalobos et al., 2024). And the proliferation of LLMs-generated content online raises the concern that future training corpora will increasingly consist of LLMs-generated data (Gillman et al., 2024). Consequently, the quality evaluation based on the internal structure of training data have become more critical than its sheer scale.

To address the need for the quality evaluation of data, some works have begun to investigate the structure of data. TSDS (Liu et al., 2024c) shows that selecting only 1% of candidate instruction fine-tuning data, guided by a small representative set of task relevant data samples, outperforms the baselines on the full dataset. Moreover, a mathematical problem can contain both facts needed for its solution and background that leaves the answer unchanged (Yang et al., 2025b; Liu et al., 2024b; Shi et al., 2023). Two inputs of the same length can therefore differ in the proportions of task relevant tokens (Maekawa et al., 2025), namely the tokens related to the target sequences. And two equal sized subsets can lead to different generalization performance (Swayamdipta et al., 2020). Context-use studies show why this distinction matters: BABILong embeds supporting facts in background

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

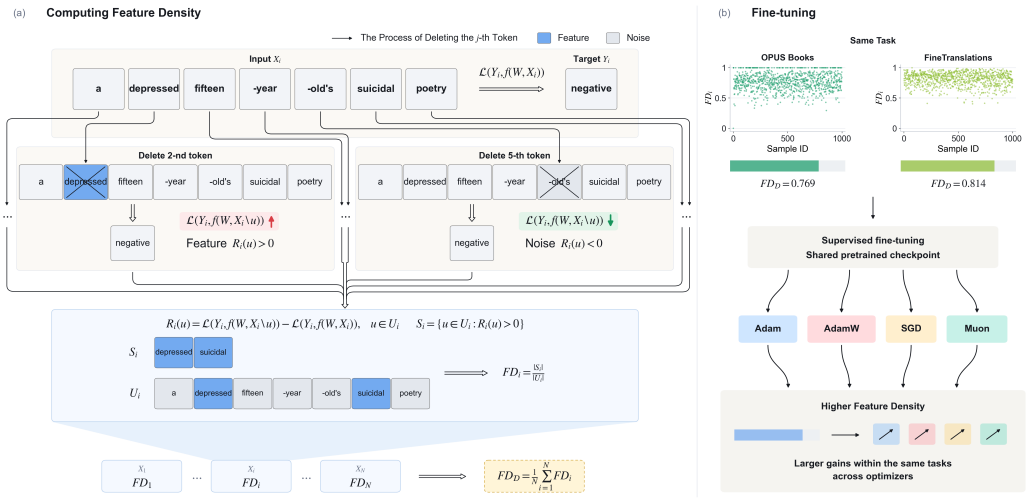


Figure 1: Overview of Feature Density. (a) We introduce Feature Density in Definition 1, a metric quantifying the proportions of task relevant tokens in input sequences. (b) Fine-tuning experiments show that higher Feature Density is associated with better fine-tuning performance within the same task across four optimizers.

text (Kuratov et al., 2024), while GSM-Symbolic finds that logically unnecessary information can reduce reasoning accuracy at inference time (Mirzadeh et al., 2025). Recent work makes the role of task relevant content in training data more specific: question-answering fine-tuning depends on which training knowledge the pretrained model already memorizes (Jiang et al., 2026; Ye et al., 2024), and data diversity on tasks is associated with instruction-tuning performance (Yang et al., 2025c). These works expose sensitivity of fine-tuning performance to input sequence, but do not establish a quantitative metric of the proportions of task relevant tokens in training data, and how it affects fine-tuning performance across different optimizers.

These findings motivate us to quantify the proportions of task relevant tokens in training data. Toward AI-Friendly data quality metrics, we proceed by the following question:

How does the proportions of task relevant tokens in a sequence affect the model's fine-tuning performance across different optimizers?

The main contributions of this work are summarized as follows.

1. We propose a computable data quality metric, i.e., Feature Density (Definition 1 in Section 2), to characterize the proportions of task relevant tokens within input sequences, complementing traditional descriptions based on dataset size and sequence length. It provides a basis for examining how this data property relates to model fine-tuning performance across optimizers.
2. We evaluate Feature Density on synthetic and real-world data to assess its validity. It captures the intended task relevant tokens in synthetic sequences (Section 3), and the feature tokens it identifies in real-world data align with human annotations (Section 4).
3. We investigate the influence of dataset Feature Density on model fine-tuning performance. Supervised fine-tuning experiments (Sections 3.3 and 4.5) show improved loss curves on datasets with higher Feature Density across four optimizers: Adam, AdamW, SGD and Muon within the same tasks.

2 Feature Density

We define Feature Density by measuring which units consisted by tokens in input sequences help a pretrained model predict the target. Deletion scores defined in equation 1 identify these units, and

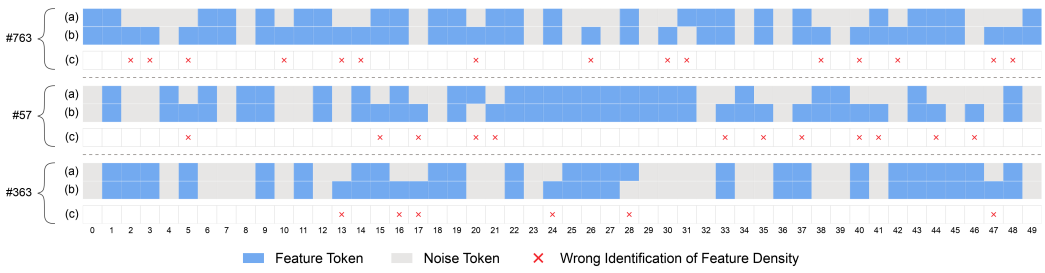


Figure 2: Evaluation of whether Feature Density correctly identifies the feature tokens on synthetic data, where #763 means the 763-th generated synthetic sequence. Row (a) shows the feature tokens(blue squares) and noise tokens(white squares) of generated sequence, row (b) shows the task relevant tokens identified by Feature Density, and row (c) marks their disagreements with red crosses. The generation methods of synthetic data are summarized in Section 3.1.

their fraction summarizes the proportions of task relevant tokens used in the experiments. Figure 1 presents the intuition and example of Feature Density.

Input units and deletion scores. For input sequences $i = 1, \ldots, N$, write $X_i = (x_{i,1}, \ldots, x_{i,n})$ and $Y_i = (y_{i,1}, \ldots, y_{i,m})$ for the input and target sequences; their lengths may differ across examples. Let U_i be a nonempty set of indexed input units. Each unit $u \in U_i$ is a nonempty set of tokens of the form $u = \{x_{i,j} \mid j \in J\}$, where J is a nonempty index set $\{1, \ldots, n\}$. For a fixed model $f(W, X_i)$ with parameters W, define the deletion score

$$R_i(u) = \mathcal{L}\left(Y_i, f(W, X_i \setminus u)\right) - \mathcal{L}\left(Y_i, f(W, X_i)\right), \tag{1}$$

where $\mathcal{L}$ is the loss function and $X_i \setminus u$ removes only unit u, retaining the order of the remaining units. The target, model parameters, and loss function remain fixed for both evaluations.

A positive score means that removing u makes the target harder for this model to predict. We call u a *feature unit* when $R_i(u) > 0$, namely the task relevant tokens, and a *noise unit* when $R_i(u) \leq 0$, namely the task irrelevant content.

Definition 1 (Feature Density)**.** *The identified feature set and Feature Density of input sequence X_i are*

$$S_i = \{u \in U_i : R_i(u) > 0\}, \quad FD_i = \frac{|S_i|}{|U_i|}. \tag{2}$$

For a dataset $\mathcal{D}$ of N input–target pairs, the dataset-level Feature Density is the mean over input sequences,

$$FD_{\mathcal{D}} = \frac{1}{N} \sum_{i=1}^{N} FD_i. \tag{3}$$

Sequence-level and dataset-level Feature Density both lie in $[0, 1]$. Each input sequence receives equal weight in $FD_{\mathcal{D}}$, regardless of its number of units. Thus this mean can differ from pooling all feature units and dividing by the total number of input units.

What Feature Density characterizes. Feature Density measures the proportions of input units that support target prediction under the specified deletion rule. For example, $FD_i = 3/5$ means that deleting three of five units individually increases the target loss. A larger Feature Density indicates that a larger fraction of units is identified as task relevant in a dataset. Figure 1 shows how to compute Feature Density on a sequence data sample. Figure 2 compares these identified units with generated feature tokens. Figure 4 illustrates the same rule for translation, sentiment classification, and mathematical reasoning.

The following experiments examine these labels against generated features and human annotations, verifying the validity of Feature Density in estimating task relevant feature tokens and task irrelevant noise tokens.

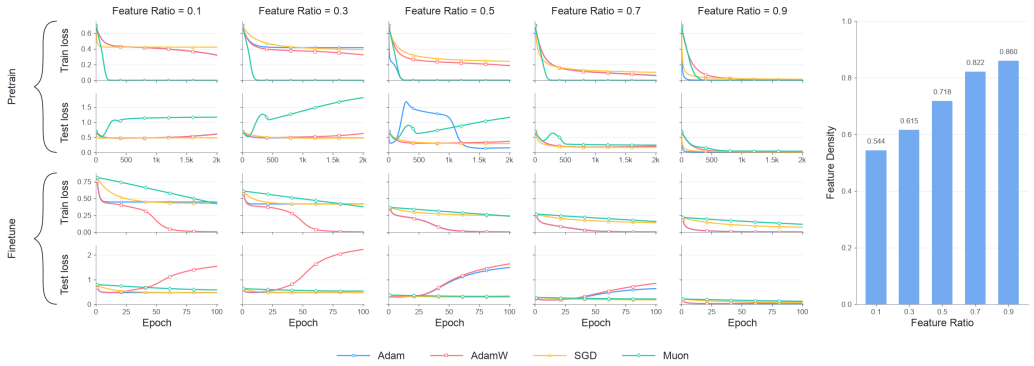


Figure 3: Optimizer comparison on synthetic data with varying feature ratios. pretraining losses over 2000 epochs for Adam, AdamW, SGD and Muon (upper row). Fine-tuning losses over 100 epochs from the same Adam-pretrained Transformer checkpoint (lower row). The Feature Density computed on synthetic data with varying feature ratios ρ, where ρ is defined in Section 3.1 (right column). Hyperparameter details are in Appendix A.1.

3 Experiments on Synthetic Data

Synthetic data let us vary the proportions of feature tokens and compare their known locations with those identified by Feature Density. We then examine how pretraining and fine-tuning losses change across these settings for four optimizers. The main experiments randomize feature and noise token positions. Appendix B presents the fixed-position comparison and the theory for token-position invariance of Feature Density.

3.1 Data Generation

We generate N sequences X_i $(i = 1, \ldots, N)$ with a sequence length of n. The j-th token of X_i is denoted by $x_{i,j}$ $(j = 1, \ldots, n)$. The target is denoted by $Y_i = (y_{i,1})$ where $y_{i,1} \in \{-1, +1\}$ is a scalar. Each input sequence X_i is composed of k task relevant feature tokens and $n - k$ task irrelevant noise tokens. For a prescribed feature ratio ρ, we set $k = \lfloor n\rho + 1/2 \rfloor$ (where $n \geq 2$ and $1 \leq k \leq n - 1$).

The feature tokens and noise tokens of X_i are generated as follows[1]:

$$x_{i,j} = \begin{cases} y_{i,1} \dfrac{\alpha}{\sqrt{k}} g_j, & 1 \leq j \leq k, \ \textit{feature tokens}, \\ \xi_{i,j-k}, & k+1 \leq j \leq n, \ \textit{noise tokens}, \end{cases} \tag{4}$$

Here, $g_j \in \mathbb{R}^D$ is a normalized random vector drawn from $\mathcal{N}(0, I_D)$, and $\xi_{i,j-k} \sim \mathcal{N}(0, \dfrac{\sigma^2}{D} I_D)$. To simulate varying token positions, we introduce a random permutation matrix $Q_i \in \{0, 1\}^{n \times n}$. And the final generated sequence is defined by:

$$X_i = Q_i[x_{i,1}, \ldots, x_{i,n}]. \tag{5}$$

For each sequence X_i, the randomly sampled permutation matrix Q_i randomizes the positions of feature and noise tokens. If Q_i is the identity matrix, the token positions are fixed by the index j in equation 4. Considering this, we experimentally compare the randomized-position setting with this fixed-position setting to examine the effect of feature and noise token positions on Feature Density. We present the comparison experiments of two settings in Appendix B.1. And Appendix B.2 provides a theoretical analysis of the invariance of Feature Density under token-position permutations.

[1] $1/\sqrt{k}$ normalizes the variance of feature magnitude to α^2 such that increasing feature ratio ρ does not affect this value. $\alpha > 0$ and $\sigma \geq 0$ control feature strength and noise variance level, respectively.

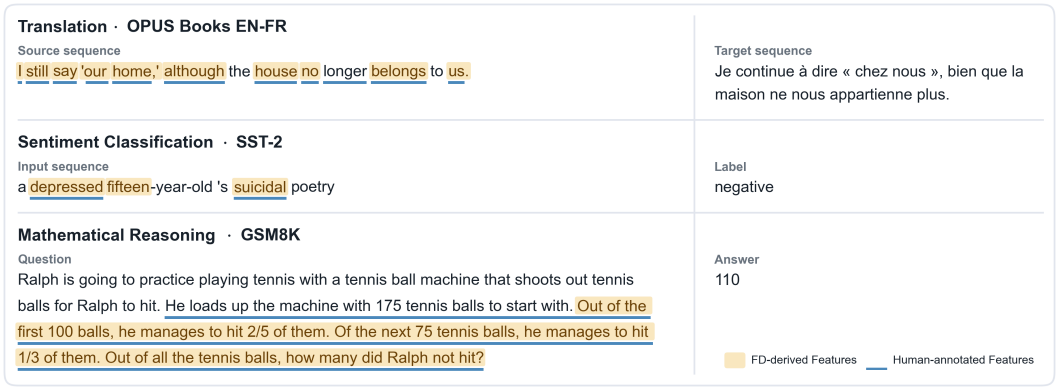


Figure 4: With Qwen3-0.6B, it presents the agreement and differences between features identified by Feature Density and human annotations across translation, sentiment classification, and mathematical reasoning. Gold highlights denote feature units identified by Feature Density; blue underlines denote human-annotated features.

3.2 Verifying the Feature Token Identified by Feature Density

To assess the validity of Feature Density, we conduct experiments with a Adam-pretrained Transformer model on the synthetic data. Results of Figure 2 compare the generated feature token set F_i with the feature token set S_i identified by Feature Density for three synthetic sequences. Each contains 50 tokens, of which 25 are generated features. The identified sets recover most generated feature tokens, but also include some generated noise tokens. The red crosses mark these disagreements. These examples show that deletion scores can recover generated features while also selecting noise used by the model.

3.3 Pretraining and Fine-Tuning Results

Figure 3 compares Adam, AdamW, SGD and Muon across different level of feature ratios $\rho \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The upper two rows show training and test loss over 2,000 pretraining epochs, and the lower two rows show the corresponding losses over 100 supervised fine-tuning epochs. The rightmost panel shows that measured Feature Density increases from approximately 0.54 to 0.86 as ρ increases from 0.1 to 0.9, linking higher feature ratios to higher measured Feature Density (see also Appendix B.1).

During pretraining, at $\rho \in \{0.1, 0.3\}$, Muon rapidly reduces training loss to nearly zero while its test loss rises, whereas Adam and SGD retain higher training loss and more stable test loss. At $\rho = 0.5$, Adam shows a delayed reduction in test loss after an early rise. At $\rho = 0.9$, all four attain low training loss and much lower test loss than in the low Feature Density settings.

Moreover, to examine whether Feature Density also matters when learning starts from a pretrained model, we compare the four fine-tuning optimizers using the same Adam-pretrained checkpoints. This comparison thus evaluates four choices of fine-tuning optimizer after Adam pretraining. All four optimizers achieve lower fine-tuning test loss at $\rho = 0.9$ than at $\rho = 0.1$, consistent with the higher measured Feature Density of this setting, but their responses at intermediate ratios differ. At $\rho \in \{0.1, 0.3\}$, AdamW reduces training loss to nearly zero while test loss rises substantially; Adam retains a higher training-loss plateau. At $\rho \in \{0.5, 0.7\}$, both Adam and AdamW fit the training data closely but show increasing test loss later in fine-tuning. At $\rho = 0.9$, both maintain low training and test losses. SGD and Muon reduce training loss more gradually and have comparatively stable test trajectories, with lower test loss at higher feature ratios.

All four optimizers benefit in the highest-density setting, but their responses across intermediate ratios are not uniformly monotonic. Based on the results of synthetic data, the subsequent focus will be on examining the Feature Density in real-world applications.

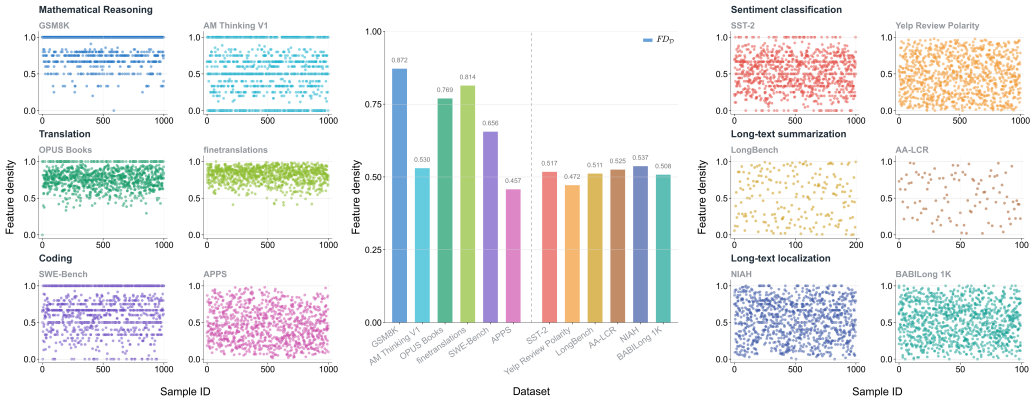


Figure 5: Feature Density across twelve real-world datasets with Qwen3-0.6B. Each point in a scatter panel gives one input sequence's FD_i; Sample ID is the index of each input sequence. The central bars show $FD_{\mathcal{D}}$, i.e., the mean Feature Density on dataset. Translation task and mathematical reasoning task show a higher Feature Density.

4 Experiments on Real-World Data

Unlike synthetic data, real-world inputs do not come with feature labels specified by the data-generation process. So the first step is to compare the task relevant token identified by Feature Density with human-annotated feature tokens. Next, we examine how Feature Density varies on real-world across datasets, and how many data samples how many input sequences are needed to obtain a stable estimates of Feature Density of a dataset, and its relation to fine-tuning performance.

Evaluation protocol. We use the archived Qwen3-0.6B (Yang et al., 2025a) evaluations with the deletion score $R_i(u) > 0$ defined in Definition 1. The target loss is the sum of negative log probabilities over target tokens, with the same target scored before and after deletion. We further examine whether our findings hold for a different base model through additional experiments with Qwen3.5-0.8B, reported in Appendix C.

4.1 Verifying the Feature Token Identified by Feature Density

To assess the validity of Feature Denisty on real-world data, we compare task relevant tokens identified by Feature Density with human annotations on OPUS Books English–French translation (Tiedemann, 2012), SST-2 (Socher et al., 2013), and GSM8K (Cobbe et al., 2021) in Figure 4. The feature units identified by Feature Density are text tokens for translation and sentiment classification, and sentences for mathematical reasoning. The figure maps these units back to the source text sequence, showing the expected alignment with human annotations.

Specifically, in Figure 4, the OPUS Books translation example contains identified features throughout the source, consistent with the broad content needed to predict its translation. In SST-2, both annotations and deletion scores identify *"depressed"* and *"suicidal"*, while the model also selects *"fifteen"*. In GSM8K, both Feature Density and human annotations identify the sentences specifying the group sizes and hit fractions, together with the question asking how many balls Ralph did not hit. The sentence *"He loads up...with 175 tennis balls..."* is human-annotated but not identified by Feature Density. Its numerical information is redundant for computing the final answer: the subsequent sentences*"Out of the first 100 balls...Of the next 75 tennis balls..."*, from which the total of 175 can be recovered. The omission of this sentence is therefore consistent with the redundancy of its numerical information. And the feature units identified by Feature Density align with human-annotated feature tokens of input sequence.



Figure 6: Feature Density across relative input positions with Qwen3-0.6B. Each tile gives the feature fraction in one of five equal position bands, averaged equally across input sequences. Position is defined over text tokens or sentences as indicated below each panel. Panel headers give $FD_{\mathcal{D}}$ and the number of input sequences.

4.2 Feature Density Across Datasets

The twelve datasets cover mathematics, translation, code, sentiment, and long-context tasks. The evaluated collections contain 1,000 input sequences each, except LongBench with 200 and Artificial Analysis Long Context Reasoning (AA-LCR) with 100. Units are text tokens for OPUS Books, finetranslations, APPS, SST-2, and Yelp Review Polarity, and sentences for the other datasets. Details of unit separation methods are provided in Appendix D.

Variation across and within datasets. Figure 5 shows high mean Feature Density for GSM8K (0.872), finetranslations (0.814) (Penedo et al., 2026), and OPUS Books (0.769). SWE-Bench (0.656) (Jimenez et al., 2024) also exceeds APPS (0.457) (Hendrycks et al., 2021). The mathematical subset of AM-Thinking-v1-Distilled (AM Thinking V1 in the figures; 0.530) (Tian et al., 2025), SST-2 (0.517), and Yelp Review Polarity (0.472) (Zhang et al., 2015) lie closer to one half. The long-context evaluations have means of 0.511 for LongBench (Bai et al., 2024), 0.525 for AA-LCR (Team, 2025), 0.537 for Needle in a Haystack (NIAH) (Kamradt, 2023), and 0.508 for BABILong 1K (Kuratov et al., 2024). LongBench uses the QASPER subset, and NIAH uses the English 4K setting.

The higher Feature Density in GSM8K and translation datasets (OPUS Books, finetranslations) aligns with the expectation that a larger portion of their input is task relevant, driven by GSM8K's inherently relevant problem descriptions (Shi et al., 2023) and the necessity of covering all source material in translation (Tu et al., 2016). In contrast, the lower values for long-context localization (NIAH, BABILong 1K) and sentiment classification (SST-2, Yelp Review Polarity) indicate that task relevance is concentrated in smaller components—specifically, selected supporting facts within distracting contexts or localized sentiment cues (Hsieh et al., 2024; Kuratov et al., 2024; Lei et al., 2016), as the SST-2 example in Figure 4 shows.

4.3 Feature Density Across Input Positions

Dataset-level Feature Density quantifies the overall proportions of input units that support target prediction. We next examine whether this proportion is similar across different regions of an input sequence or varies with relative position. Figure 6 divides each sequence into five equal bands. Each unit occupies an interval of relative length $1/|U_i|$ and contributes to every band it overlaps in proportion to that overlap. The resulting within-band feature fractions are averaged equally across input sequences.



Figure 7: Feature Density estimation under different sample counts with Qwen3-0.6B. Curves accumulate input sequences in a fixed shuffled order, with the horizontal axis on a logarithmic scale. Dashed lines give the means of the complete evaluated collections. Shading gives 95% ranges from random sampled subsets without replacement.

Finetranslations also has a nearly uniform profile, from 0.809 to 0.823, while GSM8K peaks in the middle band at 0.941 and falls to 0.777 in the last band. SST-2 and LongBench have similar dataset-level Feature Density (0.517 and 0.511), but their positional profiles differ. For example, SST-2 decreases from 0.602 in the first band to 0.452 in the last, whereas LongBench remains nearly uniform across bands, ranging from 0.503 to 0.520. These comparisons show that similar overall Feature Density can correspond to different distributions of identified features across input regions. The positional analysis therefore complements our characterization of task relevant tokens by describing its proportion in the input sequence and how that proportion varies along the sequence position.

4.4 Stability with the Number of Input Sequences

Figure 7 tracks mean Feature Density as input sequences are included in a fixed shuffled order. The estimates fluctuate at small sample counts and approach the mean of the complete evaluated collection as more examples are included. The random-subset variance shading narrow with sample count and collapse when the complete collection is included, by construction. For the collections evaluated with 1,000 input sequences, the running means exhibit only small changes at larger sample counts, even before the complete collection is included. This empirical stability supports our choice of 1,000 input sequences for most real-data experiments as a practical sample size for obtaining stable estimates of mean Feature Density. Appendix C complements these sign-based measurements with the magnitudes of the underlying deletion scores.

4.5 Fine-Tuning Results

In this section, we conduct fine-tuning experiments on the same twelve datasets used in Section 4.2 with Qwen3-0.6B and four optimizers: Adam, AdamW, SGD and Muon. All models are fine-tuned for one epoch. Figure 8 presents the training loss curves across the six task categories, with the corresponding dataset Feature Density values annotated in each panel. For mathematical reasoning, GSM8K has a higher Feature Density than AM Thinking V1 (0.872 versus 0.530) and exhibits a



Figure 8: Fine-tuning Qwen3-0.6B on real-world datasets . Training loss curves for Adam, AdamW, SGD and Muon across twelve datasets, with dataset feature density FD_D annotated in each panel.

lower initial training loss, maintaining this advantage throughout fine-tuning. This pattern is more pronounced in long-text localization: BABILong, which has a lower Feature Density than NIAH, exhibits a substantially higher loss at the beginning of training and larger loss fluctuations throughout fine-tuning. These results suggest that, within the same task category, higher Feature Density is associated with lower training loss across all four optimizers.

5 Conclusion

We proposed a computable metric called Feature Density to quantify the proportions of task relevant tokens within input sequences. We validated this metric by comparing identified features against generated tokens and human annotations across twelve real-world datasets. Under the same tasks, we found that higher Feature Density correlated with lower loss curves across Adam, AdamW, SGD and Muon in both synthetic and real-world fine-tuning settings. While synthetic experiments showed differences at intermediate feature ratios, these results collectively establish Feature Density as a computable metric for task relevant content in training data and demonstrate its link to fine-tuning performance across various optimizers.

Limitations. Although we evaluate Feature Density on synthetic and real-world data, its real-world measurements depend on task-specific unit separation methods, and a formalized approach to unit separation across tasks has yet to be established. Moreover, while our fine-tuning experiments under the same tasks examine the relationship between Feature Density and model performance, using this metric to directly compare fine-tuning performance across tasks remains to be explored. Finally, Feature Density captures only one dimension of data quality, namely task-relevant content, and toward a comprehensive data quality metric may require more comprehensive consideration such as data diversity, implying the need of future works.

Future Directions. Future works could investigate Feature Density's stability across different model architectures and datasets, and explore its applicability beyond sequential data (e.g., image data). More than that, based on Feature Density and other metrics, it is important to develop a quantitative framework for evaluating dataset quality across different tasks and models. A meaningful long-term goal is to connect measurable data properties with the learning process of LLMs and thereby help the joint design of benchmarks, model architectures and optimization methods.

AI USE STATEMENT

In this work, we used generative AI tools for polish writing and coding tasks. We have not used generative AI tools for retrieval and discovery, research ideation or execution, draft sections of the paper, generating synthetic datasets and proving mathematical claims. We have reviewed all AI-assisted work. All authors have verified the correctness and reproducibility of LLM-generated code. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

To support reproducibility, we describe the Feature Density computation in Section 2, synthetic data generation in Section 3.1, hyper-parameter configurations in Section A, and task-specific input separation in Appendix D. The assumptions and complete proof of the token-position invariance result are provided in Appendix B.2. Code is available at an anonymous repository `https://anonymous.4open.science/r/feature-density-of-dataset-3CC8`.

REFERENCES

Daman Arora and Andrea Zanette. Training language models to reason efficiently. *Advances in Neural Information Processing Systems*, 38:60770–60808, 2026.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL `https://arxiv.org/abs/2110.14168`.

Nate Gillman, Michael Freeman, Daksh Aggarwal, Chia-Hong Hsu, Calvin Luo, Yonglong Tian, and Chen Sun. Self-correcting self-consuming loops for generative model training. *arXiv preprint arXiv:2402.07087*, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi: 10.1080/01621459.1963.10500830.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. An empirical analysis of compute-optimal large language model training. *Advances in neural information processing systems*, 35:30016–30030, 2022.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Changhao Jiang, Ming Zhang, Yifei Cao, Junjie Ye, Xiaoran Fan, Shihan Dou, Zhiheng Xi, Jiajun Sun, Yi Dong, Yujiong Shen, et al. Beyond scaling: Measuring and predicting the upper bound of knowledge retention in language model pre-training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26377–26396, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Greg Kamradt. LLMTest: Needle in a haystack. GitHub repository, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 107–117, 2016.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024b.

Zifan Liu, Amin Karbasi, and Theodoros Rekatsinas. Tsds: Data selection for task-specific model finetuning. *Advances in Neural Information Processing Systems*, 37:10117–10147, 2024c.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Seiji Maekawa, Hayate Iso, and Nikita Bhutani. Holistic reasoning with long-context lms: A benchmark for database operations on massive textual data. In *International Conference on Learning Representations*, volume 2025, pp. 78468–78492, 2025.

Iman Mirzadeh, Keivan Alizadeh-Vahid, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, volume 2025, pp. 94743–94765, 2025.

Guilherme Penedo, Hynek Kydlíček, Amir Hossein Kargaran, and Leandro von Werra. Finetranslations. https://huggingface.co/datasets/HuggingFaceFW/finetranslations, 2026.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International conference on machine learning*, pp. 31210–31227. PMLR, 2023.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9275–9293, 2020.

Artificial Analysis Team. Artificial analysis long context reasoning benchmark(lcr), 2025.

Xiaoyu Tian, Yunjie Ji, Haotian Wang, Shuaiting Chen, Sitong Zhao, Yiping Peng, Han Zhao, and Xiangang Li. Not all correct answers are equal: Why your distillation source matters. *arXiv preprint arXiv:2505.14464*, 2025.

Jörg Tiedemann. Parallel data, tools and interfaces in OPUS. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pp. 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL https://aclanthology.org/L12-1246/.

Kushal Tirumala, Daniel Simig, Armen Aghajanyan, and Ari Morcos. D4: Improving llm pretraining via document de-duplication and diversification. *Advances in Neural Information Processing Systems*, 36:53983–53995, 2023.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 76–85, 2016.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Position: Will we run out of data? limits of llm scaling based on human-generated data. In *Forty-first International Conference on Machine Learning*, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Yang Wang, and Liangming Pan. How is llm reasoning distracted by irrelevant context? an analysis using a controlled benchmark. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13340–13358, 2025b.

Yuming Yang, Yang Nan, Junjie Ye, Shihan Dou, Xiao Wang, Shuo Li, Huijie Lv, Tao Gui, Qi Zhang, and Xuan-Jing Huang. Measuring data diversity for instruction tuning: A systematic analysis and a reliable metric. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18530–18549, 2025c.

Junjie Ye, Yuming Yang, Qi Zhang, Tao Gui, Xuanjing Huang, Peng Wang, Zhongchao Shi, and Jianping Fan. Empirical insights on fine-tuning large language models for question-answering. *arXiv preprint arXiv:2409.15825*, 2024.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, 2023.

A REPRODUCIBILITY

A.1 HYPER-PARAMETER CONFIGURATION FOR SYNTHETIC-DATA EXPERIMENTS

Pretraining. We use Transformer with four attention heads, query/key width $d_k = 40$, configured value width $d_v = 20$, softmax attention with unit scaling, linear activation, normally initialized value projections, and model seed 0. Each dataset contains 1,000 training and 1,000 test sequences with $n = 50$, $D = 100$, balanced labels, $\alpha = 1$, $\sigma = 2$, and $\rho \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Token positions are randomly permuted using seed 42. Models are trained from scratch for 2,000 epochs with training and evaluation batch sizes of 100. The learning-rate grid is $\{5\times10^{-5}, 10^{-4}, 1.5\times10^{-4}, 2\times10^{-4}, 3\times10^{-4}, 5\times10^{-4}, 10^{-3}, 3\times10^{-3}\}$ for Adam, AdamW, and Muon, and $\{0.005, 0.01, 0.02, 0.03, 0.05, 0.075, 0.1, 0.15\}$ for SGD. The weight-decay grid is $\{0, 10^{-5}, 10^{-2}\}$. Adam and AdamW use $(\beta_1, \beta_2) = (0.9, 0.99)$, $\epsilon = 10^{-15}$. SGD uses zero momentum, zero dampening, and no Nesterov acceleration. Muon uses momentum in $\{0.90, 0.95, 0.99\}$, Nesterov acceleration, Newton–Schulz coefficients $(3.4445, -4.775, 2.0315)$, five Newton–Schulz steps, and $\epsilon = 10^{-7}$. The pretraining experiments were conducted on NVIDIA L20 GPUs.

Fine-tuning. All runs are initialized from the same Transformer checkpoint pretrained with Adam on $\sigma = 2$, $\rho = 0.8$, learning rate 3×10^{-3}, and weight decay 10^{-2}; the checkpoint saved at epoch 5 is used. Fine-tuning datasets contain 1,000 training and 1,000 test sequences with $n = 50$, $D = 100$, balanced labels, $\alpha = 1$, $\sigma = 2$, and $\rho \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Fine-tuning is performed for 100 epochs with training and test batch sizes of 100, gradient accumulation of 1, and seed 42. The learning-rate grid is $\{5\times10^{-5}, 10^{-4}, 1.5\times10^{-4}, 2\times10^{-4}, 3\times10^{-4}, 5\times10^{-4}, 10^{-3}, 3\times10^{-3}\}$ for Adam, AdamW, and Muon, and $\{0.005, 0.01, 0.02, 0.03, 0.05, 0.075, 0.1, 0.15\}$ for SGD. The weight-decay grid is $\{0, 10^{-5}, 10^{-2}\}$. Adam and AdamW use $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$. SGD uses zero momentum, zero dampening, and no Nesterov acceleration. Muon uses momentum in $\{0.90, 0.95, 0.99\}$, Nesterov acceleration, Newton–Schulz coefficients $(3.4445, -4.775, 2.0315)$, five Newton–Schulz steps, and $\epsilon = 10^{-7}$. Its auxiliary AdamW optimizer uses $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, the Muon learning rate, and the corresponding weight decay. The fine-tuning experiments were conducted on NVIDIA L20 GPUs.

A.2 HYPER-PARAMETER CONFIGURATION FOR REAL-WORLD DATA EXPERIMENTS

Fine-tuning. For both Qwen3-0.6B and Qwen3.5-0.8B, we perform the same hyperparameter grid search, comprising 3,870 configurations per model and dataset across four optimizers. All optimizers share learning rates in $\{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and weight decay values in $\{5\times10^{-6}, 5\times10^{-5}, 5\times10^{-4}, 5\times10^{-3}, 5\times10^{-2}\}$. For Adam and AdamW, we independently vary $\beta_1, \beta_2 \in \{0.3, 0.6, 0.9\}$ and $\epsilon \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, yielding 1,620 configurations per optimizer. For SGD, we vary momentum over $\{0.3, 0.6, 0.9\}$, yielding 90 configurations. For Muon, we use the same momentum values and additionally vary ϵ over the six values above, yielding 540 configurations. Parameters not updated by Muon are optimized using an auxiliary AdamW optimizer whose hyperparameters are fixed to the best AdamW configuration for the corresponding model and dataset. The fine-tuning experiments were conducted on NVIDIA L20 GPUs.

Following the hyperparameter search, we selected the configurations used for fine-tuning on the 12 datasets. The configurations for Qwen3-0.6B and Qwen3.5-0.8B are summarized in Tables 1 and 2, respectively.

Context Truncation Scheme. We truncate sequences before all perturbations to respect the context limits of Qwen3 and Qwen3.5. For loss-based FD, let n and m denote the original lengths of the full prompt (including template tokens) and target, respectively. Let b_P and b_Y denote the numbers of prompt and target tokens retained after truncation, respectively. Given a total token budget B, when both parts are nonempty and $n + m > B$, we set

$$b_P = \max\left\{1, \left\lfloor B\frac{n}{n+m}\right\rfloor\right\}, \qquad b_Y = B - b_P.$$

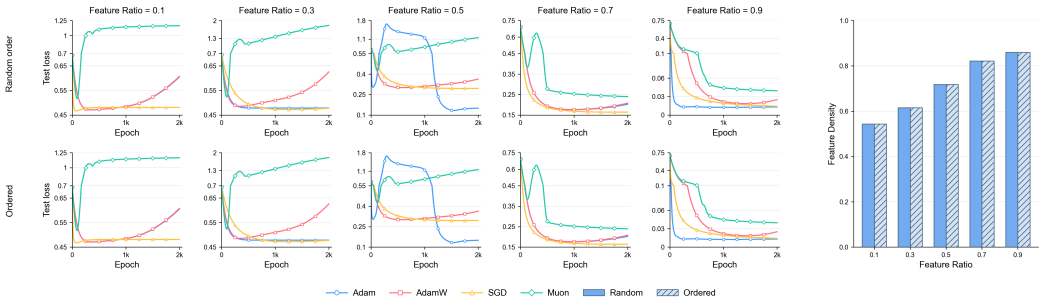


Figure 9: Fixed versus randomized token positions in synthetic data under the fixed random seed 42. The left panels show test-loss trajectories under random token position (upper row) and fixed token position (lower row) for Adam, AdamW, SGD, and Muon. Columns vary the feature ratio ρ. The right bars is to compare Feature Density in the random and ordered settings and verfiy Proposition 1.

We retain the last b_P prompt tokens and the first b_Y target tokens. Sequences within the budget remain unchanged. If either part is empty, only the nonempty part is truncated when necessary.

All perturbations use the same retained target. FD is computed over the retained input units, with sentence spans adjusted accordingly. For generation-based FD, the budget applies only to the prompt, which is left-truncated when necessary.

B Additional Results on Synthetic Data

In this section, we summarize theoretical analysis of token position permutation invariance of Feature Density and additional experiments on synthetic data.

B.1 Fixed and Randomized Token Positions

Figure 9 compares the sequences of random token positions with the sequences of fixed token position, where feature tokens in fixed token order sequence occupy the first k entries of X_i. The test-loss trajectories have similar overall shapes in the two settings, and the paired Feature Density bars agree closely at the plotted resolution. Across the five feature ratios, the displayed Feature Density values are approximately 0.54, 0.62, 0.72, 0.82, and 0.86. These experiments are conducted on a small scale Transformer. The detailed reproducibility is summarized in Section A.

The experiments on random token positions and fixed token positions motivate a theoretical analysis of Feature Density on token order. In the next section, we show that reordering tokens leaves Feature Density unchanged for fixed model parameters and an unchanged target. The proof establishes that token permutations preserve predictions and reorder the corresponding deletion scores, thereby preserving the fraction of tokens identified as features.

B.2 Feature Density Under Token Position Permutations

Assumption 1. *Each $X_i \in \mathbb{R}^{n\times D}$ stores tokens as rows, and $Q_i \in \{0,1\}^{n\times n}$ permutes those rows. The synthetic generated sequences (X_i, Y_i) are independent draws from a common distribution and are jointly independent of permutation matrices $(Q_1, \ldots, Q_N)$.*

Assumption 2. *Under the permutation matrix Q_i, assume the random token position sequence $(Q_i X_i, Y_i)$ is the same distribution of the fixed token position sequence (X_i, Y_i).*

Proposition 1 (Concentration of dataset-level Feature Density)**.** *Under Assumption 1 and 2, define the difference between the original and permuted dataset means by*

$$\Delta_N = \frac{1}{N}\sum_{i=1}^{N}\big[FD(Q_i X_i; Y_i) - FD(X_i; Y_i)\big]. \tag{6}$$

Table 1: Hyperparameters of the Qwen3-0.6B Fine-tuning.

Dataset	Optimizer	Learning rate	β_1	β_2	ϵ	Weight decay	Momentum
GSM8K	Adam	10^{-5}	0.9	0.3	10^{-6}	5×10^{-3}	–
	AdamW	10^{-4}	0.6	0.6	10^{-3}	5×10^{-5}	–
	SGD	10^{-3}	–	–	–	5×10^{-6}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-2}	0.3
AM Thinking V1	Adam	10^{-5}	0.3	0.9	10^{-5}	5×10^{-4}	–
	AdamW	10^{-5}	0.3	0.9	10^{-6}	5×10^{-2}	–
	SGD	10^{-4}	–	–	–	5×10^{-3}	0.6
	Muon	10^{-3}	–	–	10^{-4}	5×10^{-4}	0.3
OPUS Books	Adam	10^{-5}	0.9	0.3	10^{-6}	5×10^{-3}	–
	AdamW	10^{-4}	0.3	0.9	10^{-2}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.3
	Muon	10^{-3}	–	–	10^{-3}	5×10^{-4}	0.3
finetranslations	Adam	10^{-4}	0.9	0.9	10^{-3}	5×10^{-3}	–
	AdamW	10^{-4}	0.3	0.3	10^{-3}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-5}	0.6
SWE-Bench	Adam	10^{-4}	0.9	0.9	10^{-3}	5×10^{-3}	–
	AdamW	10^{-4}	0.9	0.9	10^{-3}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.9
	Muon	10^{-3}	–	–	10^{-3}	5×10^{-5}	0.3
APPS	Adam	10^{-5}	0.9	0.6	10^{-6}	5×10^{-6}	–
	AdamW	10^{-5}	0.9	0.9	10^{-5}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.3
	Muon	10^{-3}	–	–	0.1	5×10^{-5}	0.3
SST-2	Adam	10^{-5}	0.9	0.9	10^{-5}	5×10^{-4}	–
	AdamW	10^{-5}	0.9	0.9	10^{-5}	5×10^{-2}	–
	SGD	10^{-5}	–	–	–	5×10^{-3}	0.9
	Muon	10^{-4}	–	–	0.1	5×10^{-2}	0.9
Yelp Review Polarity	Adam	10^{-5}	0.9	0.6	10^{-6}	5×10^{-6}	–
	AdamW	10^{-5}	0.9	0.6	10^{-6}	5×10^{-5}	–
	SGD	10^{-6}	–	–	–	5×10^{-6}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-5}	0.6
LongBench	Adam	10^{-4}	0.9	0.3	10^{-5}	5×10^{-5}	–
	AdamW	10^{-4}	0.9	0.3	10^{-4}	5×10^{-5}	–
	SGD	10^{-7}	–	–	–	5×10^{-4}	0.3
	Muon	10^{-7}	–	–	0.1	5×10^{-5}	0.9
AA-LCR	Adam	10^{-4}	0.6	0.3	10^{-2}	5×10^{-4}	–
	AdamW	10^{-4}	0.3	0.3	10^{-2}	5×10^{-5}	–
	SGD	10^{-4}	–	–	–	5×10^{-5}	0.6
	Muon	10^{-3}	–	–	10^{-3}	5×10^{-5}	0.3
NIAH	Adam	10^{-5}	0.6	0.3	10^{-4}	5×10^{-6}	–
	AdamW	10^{-4}	0.3	0.3	10^{-3}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-5}	0.6
	Muon	10^{-3}	–	–	0.1	5×10^{-5}	0.3
BABILong	Adam	10^{-4}	0.6	0.3	10^{-2}	5×10^{-4}	–
	AdamW	10^{-4}	0.3	0.3	10^{-2}	5×10^{-5}	–
	SGD	10^{-4}	–	–	–	5×10^{-5}	0.6
	Muon	10^{-3}	–	–	10^{-6}	5×10^{-3}	0.3

For any $0 < \delta < 1$*, with probability at least* $1 - \delta$*,*

$$|\Delta_N - \mathbb{E}[\Delta_N]| \leq \sqrt{\frac{2\log(2/\delta)}{N}}. \tag{7}$$

Table 2: Hyperparameters used in the Qwen3.5-0.8B runs on 12 datasets.

Dataset	Optimizer	Learning rate	β_1	β_2	ϵ	Weight decay	Momentum
GSM8K	Adam	10^{-5}	0.9	0.3	10^{-6}	5×10^{-3}	–
	AdamW	10^{-4}	0.6	0.6	10^{-3}	5×10^{-5}	–
	SGD	10^{-3}	–	–	–	5×10^{-6}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-2}	0.3
AM Thinking V1	Adam	10^{-5}	0.3	0.9	10^{-5}	5×10^{-4}	–
	AdamW	10^{-5}	0.3	0.9	10^{-6}	5×10^{-2}	–
	SGD	10^{-4}	–	–	–	5×10^{-3}	0.6
	Muon	10^{-3}	–	–	10^{-4}	5×10^{-4}	0.3
OPUS Books	Adam	10^{-5}	0.9	0.3	10^{-6}	5×10^{-3}	–
	AdamW	10^{-4}	0.3	0.9	10^{-2}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.3
	Muon	10^{-3}	–	–	10^{-3}	5×10^{-4}	0.3
finetranslations	Adam	10^{-4}	0.9	0.9	10^{-3}	5×10^{-3}	–
	AdamW	10^{-4}	0.3	0.3	10^{-3}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-5}	0.6
SWE-Bench	Adam	10^{-4}	0.9	0.9	10^{-3}	5×10^{-3}	–
	AdamW	10^{-4}	0.9	0.9	10^{-3}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.9
	Muon	10^{-3}	–	–	10^{-3}	5×10^{-5}	0.3
APPS	Adam	10^{-5}	0.9	0.3	10^{-6}	5×10^{-3}	–
	AdamW	10^{-5}	0.3	0.9	10^{-6}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-2}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-2}	0.3
SST-2	Adam	10^{-5}	0.9	0.9	10^{-5}	5×10^{-4}	–
	AdamW	10^{-5}	0.9	0.9	10^{-5}	5×10^{-2}	–
	SGD	10^{-5}	–	–	–	5×10^{-3}	0.9
	Muon	10^{-4}	–	–	0.1	5×10^{-2}	0.9
Yelp Review Polarity	Adam	10^{-5}	0.9	0.6	10^{-6}	5×10^{-6}	–
	AdamW	10^{-5}	0.9	0.6	10^{-6}	5×10^{-5}	–
	SGD	10^{-6}	–	–	–	5×10^{-6}	0.9
	Muon	10^{-3}	–	–	10^{-2}	5×10^{-5}	0.6
LongBench	Adam	10^{-4}	0.6	0.9	10^{-2}	5×10^{-2}	–
	AdamW	10^{-4}	0.6	0.9	10^{-2}	5×10^{-4}	–
	SGD	10^{-3}	–	–	–	5×10^{-4}	0.3
	Muon	10^{-3}	–	–	0.1	5×10^{-4}	0.3
AA-LCR	Adam	10^{-4}	0.6	0.3	10^{-2}	5×10^{-4}	–
	AdamW	10^{-4}	0.3	0.3	10^{-2}	5×10^{-5}	–
	SGD	10^{-4}	–	–	–	5×10^{-5}	0.6
	Muon	10^{-3}	–	–	10^{-3}	5×10^{-5}	0.3
NIAH	Adam	10^{-5}	0.6	0.3	10^{-4}	5×10^{-6}	–
	AdamW	10^{-4}	0.3	0.3	10^{-3}	5×10^{-2}	–
	SGD	10^{-3}	–	–	–	5×10^{-5}	0.6
	Muon	10^{-3}	–	–	0.1	5×10^{-5}	0.3
BABILong	Adam	10^{-4}	0.6	0.3	10^{-2}	5×10^{-4}	–
	AdamW	10^{-4}	0.3	0.3	10^{-2}	5×10^{-5}	–
	SGD	10^{-4}	–	–	–	5×10^{-5}	0.6
	Muon	10^{-3}	–	–	10^{-6}	5×10^{-3}	0.3

Under Assumption 1, then $\mathbb{E}[\Delta_N] = 0$ *and* $\Delta_N \to 0$ *almost surely as* $N \to \infty$*, with repeated permutations allowed.*



Figure 10: Deletion-score distributions for twelve real-world datasets under Qwen3-0.6B. Blue and orange histograms show positive and negative scores, respectively, on a signed-logarithmic horizontal axis. Heights are sample-averaged unit fractions per bin, rather than probability densities per unit of the horizontal scale. The right column gives $FD_{\mathcal{D}}$.

Proof. The summands in equation 6 are independent and lie in $[-1, 1]$. Hoeffding's inequality (Hoeffding, 1963) gives, for every $\varepsilon > 0$,

$$\mathbb{P}(|\Delta_N - \mathbb{E}[\Delta_N]| > \varepsilon) \leq 2\exp(-N\varepsilon^2/2). \tag{8}$$

Taking $\varepsilon = \sqrt{2\log(2/\delta)/N}$ proves the stated bound. Under Assumption 2, $FD(Q_i X_i; Y_i)$ and $FD(X_i; Y_i)$ have the same distribution, so each difference has zero expectation. The displayed tail bound is summable in N for every $\varepsilon > 0$. Applying the Borel–Cantelli lemma to a countable sequence $\varepsilon \downarrow 0$ gives $\Delta_N \to 0$ almost surely. □



Figure 11: Feature Density across datasets and positions, and its stability with sample count, for Qwen3.5-0.8B. Bars show $FD_{\mathcal{D}}$, i.e., the mean Feature Density on each dataset (first row). For example, the GSM8K panel is composed by three parts: the upper-left scatter plot shows the FD_i of each input sequence; the upper-right plot shows the stability of Feature Density as the sample count increases; the lower-row subfigures shows the feature fraction across five equal position bands. Panel headers give $FD_{\mathcal{D}}$ and the number of input sequences (from the second row to the last row).

C Additional Results on Real-World Data

In this section, we present additional experiments on real-world data.

C.1 Deletion-Score Distributions

Figure 10 shows the deletion scores underlying the Feature Density measurements in Section 4.2. The signed-logarithmic horizontal axis preserves the sign of each score and compresses large magnitudes. Blue and orange bins show positive and negative scores, respectively. Each bin gives the fraction of an input sequence's units whose scores fall in that bin, averaged equally across sequences. The positive-bin masses sum to $FD_{\mathcal{D}}$.

GSM8K and the translation datasets have greater positive than negative mass, while sentiment and long-context datasets have more balanced masses. SST-2 and Yelp Review Polarity have more scores near zero than GSM8K, so small changes in those scores could change their feature and noise labels. Datasets with similar Feature Density can still differ in the magnitudes of their deletion scores and the histograms show these differences alongside the fraction of positive scores.

C.2 Additional Results on Real-World Data

In this section, we use Qwen3.5-0.8B model to conduct additional experiments on real-world benchmarks, keeping all other experimental settings identical to those in Section 4 of main paper.

Feature Density. We further examine whether the relationship between Feature Density and task relevance remains consistent for a different base model. Figure 11 presents the results.We observe that the translation datasets have a mean Feature Density of 0.884, with individual values of 0.879 for OPUS Books and 0.889 for finetranslations. The sentiment classification datasets have a lower mean Feature Density of 0.550, with values of 0.596 for SST-2 and 0.503 for Yelp Review Polarity. These results suggest a consistent relationship for a different base model, with higher Feature Density values associated with greater relevance to the target task.

Fine-Tuning Results. Similarly, we extend our analysis to examine whether the association between higher feature density and lower training loss within the same task category remains consistent for a different base model. As shown in Figure 12, for mathematical reasoning, GSM8K has a higher Feature Density than AM Thinking V1 (0.931 versus 0.457). Its training loss decreases more rapidly throughout fine-tuning and reaches a lower final value.This pattern is even more pronounced in long-text localization. Compared with NIAH, BABILong, which has a lower Feature Density, exhibits a substantially higher initial training loss and larger fluctuations during fine-tuning. NIAH, in comparison, maintains lower and more stable training loss. These results suggest that the association between higher Feature Density and lower fine-tuning loss within the same task category remains consistent for a different base model.

D Unit Separation Methods of Feature Density

Feature Density is defined over a nonempty set of indexed input units U_i. Different task types necessitate distinct partitioning schemes of unit U_i. In this section, we elaborate on the rationale behind the specific unit division methods chosen for each task type utilized in this paper.

We use sentence-level units for mathematical reasoning, long-context localization, and long-text summarization, and token-level units for code tasks, translation, and sentiment classification. These choices specify the resolution at which we assess the contribution of input content to target prediction. Each unit is deleted individually, while the target sequence Y_i, the pretrained model, and the loss function remain fixed.

Segmentation and example conventions. A token-level unit is one token produced by the evaluated model's tokenizer. A sentence-level unit comprises the tokens assigned to one sentence. In our implementation, sentence boundaries occur after a period, question mark, or exclamation mark followed by whitespace; any remaining nonempty text forms the final sentence. The sentence spans are mapped to tokenizer positions so that deleting a sentence removes its associated tokens together. Fixed prompt text outside the designated input region is not included in U_i.

The examples below are constructed to illustrate segmentation, where the ultra-long text examples are not fully presented for readability. Within each example, $u_1, u_2, \ldots$ identify units in input order, and ␣ denotes a space within a displayed token.

D.1 Mathematical Reasoning: Sentence-Level Units

We use sentences to assess the contributions of numerical facts, relations, and the question being asked. For example, the statement that one person gives another person five apples specifies a quantity together with the entities and their relationship. Deleting this statement as a unit removes the associated information together, rather than removing only part of the statement. This choice provides a statement-level interpretation of Feature Density, although a sentence need not correspond to a single reasoning step. The problem's question is also included in the segmented input.

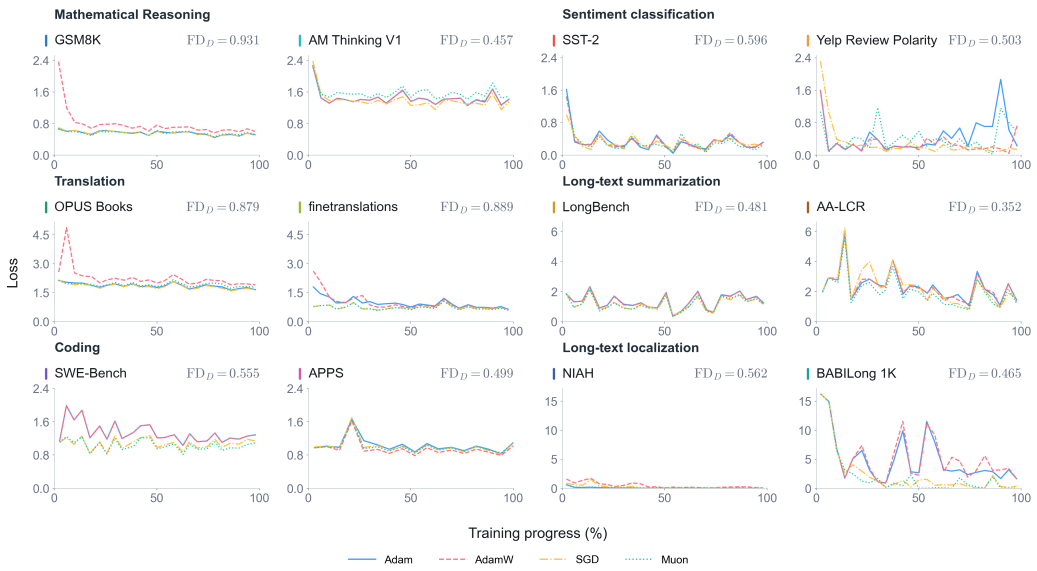


Figure 12: Fine-tuning Qwen3.5-0.8B on real-world datasets. Training loss curves for Adam, AdamW, SGD, and Muon across twelve datasets, with dataset feature density FD_D annotated in each panel. Models are fine-tuned for one epoch, using the dataset sizes specified in Section 4.4.

Mathematical Reasoning: Sentence Units

 # Input.

Maya has 12 apples. She gives 5 apples to Leo. The apples are in a blue basket. How many apples does Maya have left?

 # Target.

7 apples.

 # Units after segmentation.

u_1: Maya has 12 apples.
u_2: She gives 5 apples to Leo.
u_3: The apples are in a blue basket.
u_4: How many apples does Maya have left?

D.2 Code Tasks: Token-Level Units

We use token-level units because individual parts of a programming requirement can specify an operation, an argument, or a constraint on the target program. Sentence-level deletion would combine these contributions into a single intervention for a short requirement. When the input contains code, sentence punctuation also does not provide a suitable general rule for separating program content. Token-level units therefore provide finer resolution, using the model's tokenizer rather than a programming-language lexer. In the example below, the requirement is segmented, while the function signature is supplied by a fixed instruction.

Code Tasks: Token Units

 # Fixed instruction.

Write a Python function `add(a, b)`.

 # Input.

```
Return a + b.
```

 # Target.

```
def add(a, b):
    return a + b
```

 # Units after segmentation.

u_1: `Return`
u_2: `␣a`
u_3: `␣+`
u_4: `␣b`
u_5: `.`

D.3 Translation: Token-Level Units

We segment the source text into token-level units to examine contributions within a sentence. Source tokens can provide lexical or grammatical information relevant to predicting the target translation. Sentence-level segmentation would combine these contributions and, for a single-sentence input, provide only one unit to evaluate. Token-level segmentation provides finer resolution without assuming a one-to-one correspondence between source tokens and target words. The translation instruction and target translation remain fixed.

Translation: Token Units

 # Fixed instruction.

Translate the input from English into French.

 # Input.

```
The cat sleeps.
```

 # Target.

Le chat dort.

 # Units after segmentation.

u_1: `The`
u_2: `␣cat`
u_3: `␣sleeps`
u_4: `.`

D.4 Long-Context Localization: Sentence-Level Units

We use sentence-level units to assess which statements in a long context support answering a fixed question. A sentence can connect an answer value to its identifying description, as in the statement linking a delivery code to its value. Deleting the whole sentence tests the contribution of that statement. Sentence-level segmentation also reduces the number of deletion evaluations relative to token-level segmentation when each sentence contains multiple tokens. Only the context is segmented; the question remains fixed.

Long-Context Localization: Sentence Units

 # Fixed question.

What is the delivery code?

 # Input.

...

...

The museum opens at nine. The delivery code is K7. Visitors enter through the east gate. The cafe serves tea. The museum closes at six.

...

...

 # Target.

K7.

 # Units after segmentation.

u_1: `The museum opens at nine.`
u_2: `The delivery code is K7.`
u_3: `Visitors enter through the east gate.`
u_4: `The cafe serves tea.`
u_5: `The museum closes at six.`

D.5 Long-Text Summarization: Sentence-Level Units

We use sentences to assess the contributions of statements in the source document to predicting a fixed reference summary. Sentence-level deletion removes a statement together with its internal wording, which makes the intervention interpretable at the level of document content. It also reduces the number of deletion evaluations for long documents. Summary content may depend on several sentences jointly, and repeated information may remain available after one sentence is deleted. The resulting scores therefore describe each sentence's contribution in the presence of the remaining document, rather than its importance in isolation.

Long-Text Summarization: Sentence Units

 # Fixed instruction.

Summarize the document in one sentence.

 # Input.

...

...

The city opened a new library on Monday. The library provides free internet access. Students can use its study rooms in the evenings. The opening ceremony lasted one hour.

...

...

 # Target.

A new city library provides free internet and study rooms that students can use in the evenings.

 # Units after segmentation.

u_1: `The city opened a new library on Monday.`
u_2: `The library provides free internet access.`
u_3: `Students can use its study rooms in the evenings.`
u_4: `The opening ceremony lasted one hour.`

D.6 Sentiment Classification: Token-Level Units

We use token-level units because sentiment can depend on expressions within a sentence, including evaluative words and negation. In the example below, *not* changes the interpretation of *good*. Token-level deletion allows these parts of the review to be examined separately while retaining the rest of the input. Sentence-level deletion would provide only one unit for this short review. The sentiment label remains fixed during evaluation, and the contribution of each token is determined by its deletion score.

Sentiment Classification: Token Units

 # Fixed instruction.

Classify the review as positive or negative.

 # Input.

`The movie was not good.`

 # Target.

`negative`

 # Units after segmentation.

u_1: `The`
u_2: `␣movie`
u_3: `␣was`
u_4: `␣not`
u_5: `␣good`
u_6: `.`