Predicting Through Generation: Why Generation Is Better for Prediction

Md Kowsher[1], Nusrat Jahan Prottasha[1], Prakash Bhat[2], Chun-Nam Yu[3], Mojtaba Soltanalian[4], Ivan Garibay[1], Ozlem Garibay[1], Chen Chen[1], Niloofar Yousefi[1],
[1]University of Central Florida, USA [2]DotStar Inc, USA
[3]Nokia Bell Labs, USA [4]University of Illinois Chicago, USA
github.com/Kowsher/PredGen

Abstract

This paper argues that generating output tokens is more effective than using pooled representations for prediction tasks because token-level generation retains more mutual information. Since LLMs are trained on massive text corpora using next-token prediction, generation aligns naturally with their learned behavior. Using the *Data Processing Inequality (DPI)*, we provide both theoretical and empirical evidence supporting this claim. However, autoregressive models face two key challenges when used for prediction: (1) *exposure bias*, where the model sees ground-truth tokens during training but relies on its own predictions during inference, leading to errors, and (2) *format mismatch*, where discrete tokens do not always align with the task's required output structure. To address these challenges, we introduce **PredGen (Predicting Through Generating)**, an end-to-end framework that (i) uses *scheduled sampling* to reduce exposure bias, and (ii) introduces a *task adapter* to convert the generated tokens into structured outputs. Additionally, we introduce *Writer-Director Alignment Loss (WDAL)*, which ensures consistency between token generation and final task predictions, improving both text coherence and numerical accuracy. We evaluate **PredGen** on multiple classification and regression benchmarks. Our results show that **PredGen** consistently outperforms standard baselines, demonstrating its effectiveness in structured prediction tasks.

1 Introduction

Large Language Models (LLMs) have significantly advanced natural language processing (NLP), demonstrating strong performance across various tasks such as text completion (Kenton and Toutanova, 2019), machine translation (Vaswani, 2017; Wu, 2016), summarization (Lewis, 2019; Zhang et al., 2020), and question answering (Rajpurkar, 2016; Yang, 2019). By training on massive text corpora, these models learn contextual embeddings that capture rich semantic information, enabling them to generalize across a wide range of applications (Bommasani et al., 2021). Beyond traditional NLP tasks, LLMs are increasingly used for predictive tasks, such as classification (Liu, 2019), regression (Raffel et al., 2020), and reasoning (Wei et al., 2022), where they map input sequences to structured outputs.

A key strength of LLMs is their ability to perform tasks in a *zero-shot* or *few-shot* setting (Brown et al., 2020). By conditioning on a few examples or carefully crafted prompts, LLMs can generalize to new tasks without explicit fine-tuning (Kojima et al., 2022). However, while prompting is flexible, it lacks precision, particularly for tasks requiring structured outputs, such as numerical reasoning (Lewkowycz et al., 2022). To improve accuracy, fine-tuning is often performed by training a prediction head on pooled representations (e.g., [CLS] tokens or mean-pooled embeddings) (Kenton and Toutanova, 2019) (Figure 1-Left & Middle). However, pooling discards positional and sequential information, limiting the model's ability to capture fine-grained dependencies (Huang et al., 2024; Oh et al., 2022).

We argue that generation-based training, where an LLM is fine-tuned to produce task outputs as token sequences, preserves richer information than classification-based approaches (Figure 1-Right). Since LLMs are originally trained using next-token prediction on large corpora, generation aligns naturally with their learning paradigm. Switching to pooled classification may not fully use their pre-training knowledge, leading to weaker transfer learning. Our experiments show that token-level generation retains more mutual information than traditional prediction methods. Using the Data Processing Inequality (DPI)(Beaudry and Renner, 2011), we theoretically prove that generating tokens preserves strictly more information with the

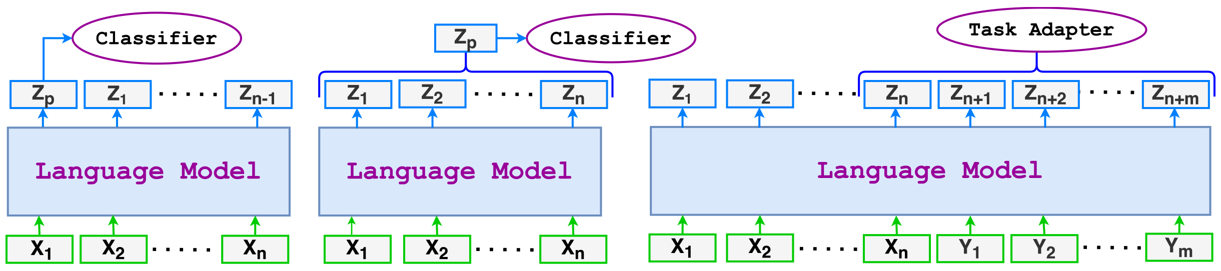

Figure 1: Comparison of different prediction methods using a language model. (Left) The traditional approach where a pooled representation $\mathbf{Z_p}$ is passed to a classifier for prediction. (Middle) A similar method where $\mathbf{Z_p}$ is extracted from the hidden states and used for classification. (Right) The generative approach, where the model generates additional tokens $\mathbf{Y_1}, \mathbf{Y_2}, ..., \mathbf{Y_m}$, and their hidden states are processed by a task adapter for prediction. This method retains more task-relevant information by using token-level generation.

target output than pooling-based representations, preventing irreversible information loss.

Despite the benefits of generation, two key challenges arise. First, many tasks require structured outputs, but generative models produce discrete tokens. For example, in a Semantic Textual Similarity (STS-B) task, the model must output a similarity score like 0.75. Representing this as separate tokens $[0, ., 7, 5]$ can introduce ambiguities—generating "0.76" or "1.75" may lead to similar token-level losses, even though 0.76 is numerically much closer to 0.75 than 1.75. This makes fine-grained numerical learning difficult. Second, exposure bias occurs because, during training, the model always conditions on ground-truth tokens, but during inference, it must rely on its own previously generated outputs. Small errors in the early steps can accumulate, leading to compounding inaccuracies.

To address these issues, we introduce PredGen (Predicting Through Generating), an end-to-end framework that fine-tunes LLMs for supervised prediction tasks. PredGen treats the target output as a sequence of tokens and incorporates scheduled sampling (Bengio et al., 2015) to mitigate exposure bias. Additionally, a task adapter transforms discrete token outputs into structured predictions, ensuring both numerical precision and task-specific formatting.

Furthermore, we introduce a specialized loss function, termed *Writer-Director Alignment Loss* (WDAL), designed to align token generation with task-specific predictions. In this framework, the *writer* generates output tokens (analogous to drafting a film script), while the *director* transforms these tokens into the required task format (comparable to producing a film from a script). WDAL ensures that the generated sequence maintains both textual coherence and numerical accuracy by effectively balancing token-level generation with task-specific objectives.

We evaluate PredGen on multiple regression and classification benchmarks, covering both numerical reasoning tasks (e.g., mathematical problem solving, similarity scoring) and high-level reasoning tasks (e.g., commonsense understanding). PredGen consistently outperforms baselines that use pooled representations or a standard generative approach, demonstrating its effectiveness across a wide range of structured prediction tasks.

Our contributions are summarized as follows:

- We argue that generation is superior to traditional classifier-based prediction and provide both theoretical and empirical evidence to support this claim.
- We introduce **PredGen**, a framework designed to address the key challenges in generative prediction.
- We propose a novel loss function, WDAL, that aligns token generation with final task predictions to ensure consistency between the generated sequences and structured outputs.

2 Problem Formulation

Prediction Using Language Models. Let $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n]$ be an input sequence, and suppose we wish to predict a structured output $\mathbf{P}$. For instance, if $\mathbf{P} = 13.4$, we may represent it as a discrete token sequence $\mathbf{Y} = [\text{'1'}, \text{'3'}, \text{'.'}, \text{'4'}]$. A pre-trained language model $\mathcal{M}$ encodes $\mathbf{X}$ into hidden states

$$\mathbf{Z} = [\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_n],$$

where $\mathbf{Z}_i \in \mathbb{R}^d$ is the contextual embedding for token x_i with d dimension. In standard prediction settings, we often *pool* these hidden states into a single vector representation $\mathbf{Z}_p \in \mathbb{R}^d$. A classifier function $f_{\text{cls}}(\mathbf{Z}_p)$ then transforms this pooled representation into the final prediction $\hat{\mathbf{P}}$, typically returning probabilities (for classification) or a real-valued score (for regression).

Reformulating Prediction as Token Generation: We redefine classification as a sequence generation task. Given an input sequence $\mathbf{X}$ and target sequence $\mathbf{Y}$, the model is trained to generate $\mathbf{Y}$ autoregressively, predicting one token at a time. The probability distribution of the target sequence is given by:

$$P(\mathbf{Y}|\mathbf{X};\theta) = \prod_{t=1}^{m} P(Y_t|\mathbf{X}, \mathbf{Y}_1, \ldots, \mathbf{Y}_{t-1};\theta),$$

where θ represents the learnable parameters of the model. This formulation allows the model to generate structured outputs while preserving sequential dependencies in the data.

3 Why Generation is More Effective for Prediction

One main reason to treat prediction as a token-by-token generation process is that LLMs are originally trained on massive text corpora, which gives them strong generative skills. By generating each output token step by step, the model uses all the contextual clues it learned during pre-training and preserves more details that matter for the task. This property also explains why LLMs often succeed in zero-shot or few-shot scenarios.

In contrast, when we fine-tune an LLM by pooling all hidden states into a single representation, we rely on a *deterministic* procedure. According to the Data Processing Inequality and **Theorem 1 (Mutual Information Decreases Under Deterministic Compression)**, this inevitably discards important information. Because token-level generation keeps more mutual information at each step, it can produce predictions that are both more precise and more faithful to the original context.

Theorem 1. *Let $\mathbf{X}$ be an input random variable, and let $\mathbf{Z} \in \mathcal{Z}$ be the final hidden representation produced by a model (e.g., an LLM). Suppose $\mathbf{Z_p} = g(\mathbf{Z})$ for some deterministic function $g : \mathcal{Z} \to \mathcal{W}$ (e.g., first-token pooling or mean pooling). Let $\mathbf{Y}$ be the target random variable. Then the following holds:*

$$I(\mathbf{Y}\ ;\ \mathbf{Z}) \ \geq\ I(\mathbf{Y}\ ;\ \mathbf{Z_p}).$$

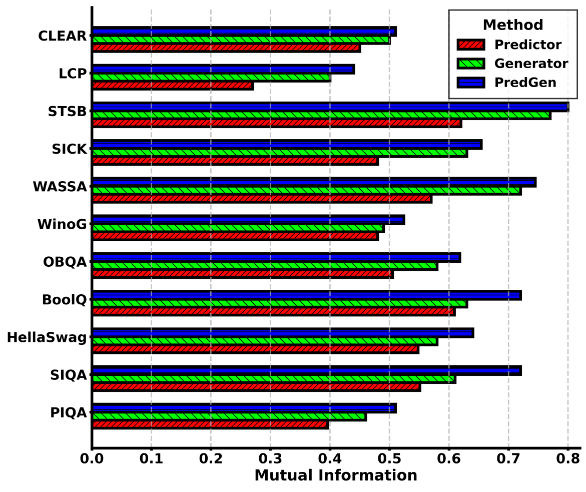

Figure 2: Comparison of mutual information estimates for Predictor, Generator, and PredGen across multiple datasets. PredGen consistently retains higher mutual information, supporting the theoretical claim that token-level generation preserves richer task-relevant information than pooled representations.

Proof Sketch: The core idea behind the theorem relies on the *DPI*, which states that applying a deterministic function to a random variable cannot increase the mutual information between the original variable and another variable. Since $\mathbf{Z}_p$ is derived from $\mathbf{Z}$ through a deterministic process, it follows that $\mathbf{Z}_p$ contains less information than $\mathbf{Z}$ (or at most, the same amount), meaning that conditioning on $\mathbf{Z}_p$ cannot reduce uncertainty about $\mathbf{Y}$ more than conditioning on $\mathbf{Z}$. This leads to the following relationship between the conditional entropies:

$$H(\mathbf{Y} \mid \mathbf{Z}) \leq H(\mathbf{Y} \mid \mathbf{Z}_p).$$

which leads to the following inequality in the mutual information:

$$I(\mathbf{Y};\mathbf{Z}_p) \leq I(\mathbf{Y};\mathbf{Z}),$$

A detailed proof is provided in Appendix B.3.

Empirical Evidence: To empirically validate Theorem 1, we estimate mutual information using the MINE method proposed by Belghazi et al. (2018). We train separate models for the Predictor, Generator, and PredGen approaches and extract the pooled representation $\mathbf{Z_p}$ for the Predictor and $\mathbf{Z}$

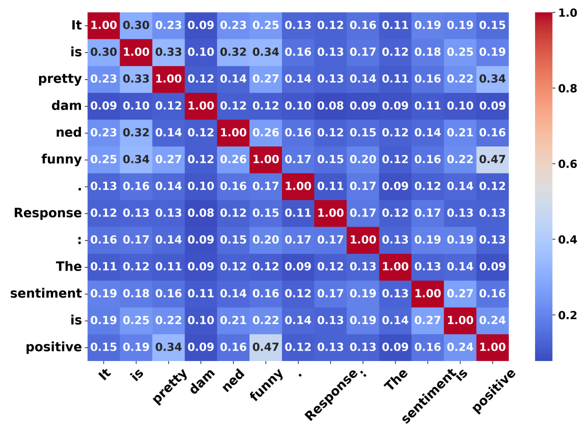

Figure 3: Token-wise mutual information on SST-2 (Socher et al., 2013). The predicted token *"positive"* shows high MI with sentiment-related tokens like *"funny"* (0.47) and *"pretty"* (0.34), highlighting strong contextual dependencies.

for the Generator and PredGen across all training and testing data.

To simplify the computational cost of using all token representations in $\mathbf{Z}$, we apply Principal Component Analysis (PCA) (Maćkiewicz and Ratajczak, 1993) to reduce the original $n \times d$ representation down to a $2 \times d$ space. Let $\mathbf{Z_r}$ be the reduced representation $\mathbf{Z_r} = \mathrm{PCA}(\mathbf{Z})$.

Next, we estimate mutual information using a two-layer neural network with MINE (Belghazi et al., 2018), which uses a neural variational method to learn a lower bound on mutual information. When evaluating the *predictor*, we feed $\mathbf{Z_p}$ as input; when evaluating *generation*, we use the reduced representation $\mathbf{Z_r}$. The estimation function is:

$$\mathcal{I}(\mathbf{Y};\mathbf{Z}) = \sup_{\theta\in\Theta} \mathbb{E}_{p(\mathbf{YZ})}[T_\theta(\mathbf{Y},\mathbf{Z})] \\ - \log \mathbb{E}_{p(\mathbf{Y})p(\mathbf{Z})}[e^{T_\theta(\mathbf{Y},\mathbf{Z})}]$$

where T_θ is a trainable function parameterized by θ.

Figure 2 compares mutual information estimates on the test set, showing that PredGen consistently retains more information than both the Predictor and Generator, supporting our theoretical claims.

Additionally, Figure 3 provides a detailed token-wise mutual information analysis. We train the model on the SST-2 dataset and compute MI between token pairs. The predicted sentiment token *"positive"* shows high MI with sentiment-relevant words such as *"funny"* (0.47) and *"pretty"* (0.34), indicating that PredGen effectively captures contextual dependencies. For additional details on mutual information in regression and classification, refer to Appendix Section H. The experimental setup is provided in Section 5.

4 PredGen

When using generation to perform prediction, one primary challenge is *exposure bias*. In a typical autoregressive setup, the model is trained to predict the next token based on all previous *ground-truth* tokens. Specifically, at time step t, the model receives $[\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n, \mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_{t-1}]$ and produces y_t. However, during inference, the model must rely on its own previously *generated* tokens $\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2, \ldots, \tilde{\mathbf{Y}}_{t-1}$—not the ground-truth sequence. This mismatch means the model never learns to correct its own mistakes, since it always conditions on true tokens during training but must condition on its own (potentially flawed) outputs at test time. Consequently, small errors can accumulate and lead to compounding inaccuracies.

To address exposure bias during autoregressive training, we apply sequence level *scheduled sampling* (Bengio et al., 2015).

$$\tilde{\mathbf{Y}} = \begin{cases} \mathbf{Y} & \text{with probability } (1-p), \\ \tilde{\mathbf{Y}} & \text{with probability } p, \end{cases}$$

where $\mathbf{Y}$ is the ground-truth tokens and $\tilde{\mathbf{Y}}_{t-1}$ is the generated predictions' tokens. The parameter p gradually increases during training, shifting from ground truth to self-conditioning (predictions). This helps the model learn to correct errors arising from its own outputs, thus reducing exposure bias.

Another challenge is that generative models produce *discrete* tokens, whereas certain tasks (e.g., regression) require *continuous* values. To address this, we introduce a transformation step (*Task Adapter*) that maps the generated token sequence into the final prediction form. Concretely, let

$$\mathbf{Z}[n : n+m] = [\mathbf{Z}_n, \mathbf{Z}_{n+1}, \cdots \mathbf{Z}_{n+m}]$$

Here, $\mathbf{Z}[n : n+m]$ denotes the hidden representations for the generated tokens, where n is the length of the input $\mathbf{X}$ and m is the number of generated tokens. Now We define a *task adapter* $\mathcal{T}$ that transforms $\tilde{\mathbf{Y}}$ into the desired output:

$$\hat{\mathbf{P}} = \mathcal{T}(\mathbf{Z}[n : n+m]).$$

For example, $\mathcal{T}$ could convert a sequence of digit tokens into a real-valued number for regression or map generated tokens to a categorical label (e.g., 0, 1, 2, ...) for classification. This ensures that discrete outputs from the generator can accommodate both continuous and structured predictions.

Writer-Director Alignment Loss (WDAL): In generative prediction, ensuring that the generated token sequence aligns with the final structured output is crucial. The *writer* (generator) is responsible for generating tokens, while the *director* (task adapter) transforms them into the required task format. If these two components are not well-coordinated, errors in generation can propagate to the final prediction, reducing accuracy. To address this issue, we introduce a novel loss function, **WDAL**, which optimizes both components together to improve prediction quality.

WDAL consists of two primary loss terms: the *writer loss* L_W, which measures the generation error using cross-entropy loss, and the *director loss* L_D, which quantifies the error in the final prediction. A natural way to combine these losses is through their product, $L_{\text{WDAL}} = L_W \cdot L_D$, ensuring that both the generation and task-specific transformation contribute to the optimization. However, this formulation can lead to numerical instability when the losses differ significantly in scale. To address this, we apply a log-sum-exp trick, leading to the final formulation:

$$L_{\text{WDAL}} = \max\big(L_W^2, L_D^2\big) \exp\Big(-\Big|\log L_W - \log L_D\Big|\Big).$$

The first term, $\max(L_W^2, L_D^2)$, serves as an *authority component*, prioritizing the larger loss so that optimization focuses on the component with higher error. The second term, $\exp\big(-|\log L_W - \log L_D|\big)$, acts as an *alignment penalty*, ensuring that both losses remain balanced. When one loss is significantly higher than the other, the penalty reduces, keeping the overall loss high and encouraging better coordination between the writer and director. A complete derivation of WDAL is provided in Appendix A.

5 Experiments

We compare PredGen with two baselines: the traditional **Predictor**, which uses pooled hidden representations followed by classification, and the standard **Generator**, which directly generates output tokens without additional transformations.

To efficiently fine-tune large language models, we employ several PEFT techniques, including **LoRA** (Hu et al., 2021), **AdaLoRA** (Zhang et al., 2023b), **RoCoFT** (Kowsher et al., 2024a), and **DoRA** (Liu et al., 2024). These methods allow us to adapt large models with fewer parameters, reducing computational costs while maintaining strong performance.

For classification, we evaluate on the following datasets: `BoolQ` (Clark et al., 2019), `PIQA` (Bisk et al., 2020), `SIQA` (Sap et al., 2019), `HellaSwag` (Zellers et al., 2019), `WinoGrande` (Sakaguchi et al., 2021), `ARC-e` (Clark et al., 2018), `ARC-c` (Clark et al., 2018), and `OBQA` (Mihaylov et al., 2018). The reported metric for classification tasks is accuracy.

For regression, we use the following datasets: `WASSA` (Vinayakumar et al., 2017), `SICK` (Marelli et al., 2014), `STSB` (Cer et al., 2017), `LCP` (Shardlow et al., 2020), `CLEAR` (Crossley et al., 2023), and `Humicroedit` (Hossain et al., 2019). The reported metrics for regression tasks are Mean Squared Error (MSE) and Mean Absolute Error (MAE).

Details about the datasets, implementation details, and hyper-parameters are provided in Appendix E, D, and Table 3 respectively.

Main Results: Table 1 presents the classification performance of Llama models using different PEFT methods. PredGen consistently outperforms both the Predictor and Generator models across all tasks. For Llama2-7B, PredGen achieves an average accuracy of 79.67%, surpassing both the Predictor (73.49%) and Generator (76.63%). Similarly, for Llama2-13B, PredGen reaches an average accuracy of 82.71%, outperforming the other methods (76.20% for Predictor and 80.40% for Generator). Finally, for Llama2-8B, PredGen achieves an average accuracy of 80.92%, again showing superior performance compared to the other models.

Table 2 presents the regression performance where PredGen consistently outperforms both the Predictor and Generator models in most tasks. For Llama2-7B, PredGen achieves an average score of 0.338, outperforming the Predictor (0.928) and Generator (0.509). Similarly, for Llama2-13B, PredGen shows an average score of 0.308, better than the Predictor (0.867) and Generator (0.474). Finally, for Llama2-8B, PredGen reaches an average of 0.319, surpassing both the Predictor (0.888) and Generator (0.493). These results indicate that

Model	PEFT	Method	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
Llama2-7B	LoRA	Predictor	66.29	81.11	78.95	88.53	70.49	75.27	53.9	73.39	73.49
		Generator	68.09	80.37	77.15	90.86	77.54	79.54	60.55	78.93	76.63
		PredGen	**73.82**	82.76	79.87	93.14	83.21	84.79	**60.86**	78.93	79.67
	AdaLoRA	Predictor	65.22	80.91	78.88	89.33	70.13	75.39	54.29	73.81	73.50
		Generator	70.03	80.69	77.06	90.85	76.47	79.50	59.30	74.22	76.02
		PredGen	72.45	84.54	**80.42**	93.19	82.26	84.80	59.53	79.11	79.54
	RoCoFT	Predictor	66.48	81.53	79.85	89.24	68.84	76.85	54.38	73.38	73.82
		Generator	69.36	80.08	77.99	89.46	77.41	79.46	59.09	76.90	76.22
		PredGen	73.62	84.32	79.65	92.64	**83.83**	84.67	60.81	**80.20**	79.97
	DoRA	Predictor	66.23	82.24	78.83	88.21	71.36	73.78	54.22	75.48	73.79
		Generator	69.56	80.33	77.09	90.10	76.69	79.66	59.05	76.96	76.18
		PredGen	73.45	**84.73**	80.11	**93.28**	83.80	**84.99**	60.19	79.89	**80.06**
Llama2-13B	LoRA	Predictor	68.37	83.42	81.34	91.54	72.32	79.24	55.12	78.23	76.20
		Generator	71.19	83.99	81.15	92.86	83.24	83.35	66.05	81.37	80.40
		PredGen	73.43	85.32	**82.45**	94.25	85.82	**86.79**	68.24	**85.41**	82.71
	AdaLoRA	Predictor	69.83	84.38	80.27	90.19	72.22	78.77	53.75	79.56	76.12
		Generator	71.71	82.55	81.88	92.61	83.01	83.04	67.33	81.76	80.49
		PredGen	74.21	85.99	82.16	94.51	86.09	86.42	**69.73**	84.98	**83.01**
	RoCoFT	Predictor	68.22	82.90	79.99	91.28	71.60	79.21	57.26	78.56	76.13
		Generator	71.44	83.52	79.50	91.84	83.20	83.39	68.06	81.73	80.33
		PredGen	**74.27**	**86.13**	81.71	**94.58**	86.16	85.79	69.22	85.29	82.89
	DoRA	Predictor	69.18	83.20	80.84	90.38	72.43	75.17	57.68	80.35	76.15
		Generator	71.36	83.73	79.54	91.27	83.62	83.61	66.32	81.54	80.12
		PredGen	74.18	85.88	81.41	93.62	**86.76**	86.25	69.58	84.77	82.81
Llama2-8B	LoRA	Predictor	68.44	82.93	79.84	91.47	71.58	77.97	56.02	74.49	75.34
		Generator	71.31	81.45	79.05	90.65	82.46	82.83	62.33	76.64	78.34
		PredGen	72.57	83.63	**81.72**	92.98	84.76	**84.78**	**64.64**	80.54	80.70
	AdaLoRA	Predictor	68.11	81.50	79.88	89.49	71.37	78.97	54.72	75.63	74.96
		Generator	70.62	82.48	79.15	91.17	83.13	82.62	61.77	78.53	78.68
		PredGen	73.10	**84.88**	80.61	**93.22**	85.23	84.72	62.81	81.56	80.77
	RoCoFT	Predictor	67.96	76.59	79.92	89.63	72.02	76.39	54.92	74.41	73.98
		Generator	71.79	83.23	79.37	90.84	82.74	82.67	62.03	77.73	78.80
		PredGen	**74.76**	84.81	80.86	92.44	**85.87**	84.49	62.97	81.14	**80.92**
	DoRA	Predictor	67.88	82.12	80.26	91.68	71.68	76.36	54.42	77.57	75.25
		Generator	72.32	82.38	80.01	90.85	83.38	82.48	61.03	78.40	78.86
		PredGen	74.21	83.59	81.24	93.17	84.99	84.72	62.26	**81.68**	80.73

Table 1: Performance of Classification with Different PEFT Methods Across Benchmarks. The best results are highlighted in bold for each model.

PredGen outperforms standard approaches in classification and regression tasks because generating predictions as token sequences carries more mutual information, leading to higher accuracy.

6 Ablation Study

Scheduled Sampling vs. Performance We analyze how different scheduled sampling strategies affect accuracy on the MultiArith dataset (Roy and Roth, 2016). Instead of always using ground-truth tokens as input, the model gradually shifts from using reference tokens to using its own generated tokens. Figure 4 shows how varying `max_steps_for_sampling` impacts the performance.

If the transition happens too quickly (e.g., `max_steps_for_sampling` = 50 or 100), the model receives too few reference tokens, leading to unstable predictions. On the other hand, if the transition is too slow (e.g., `max_steps_for_sampling` = 7000), the model remains overly dependent on reference tokens and struggles during inference.

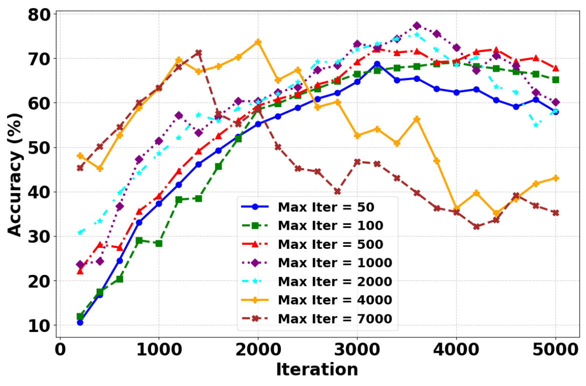

Figure 4: Effect of `max_steps_for_sampling` on performance. A gradual transition (`max_steps_for_sampling` = 1000) achieves the best performance, balancing reference-based and self-generated predictions.

The best results occur when `max_steps_for_sampling` = 1000, striking a balance between guidance from reference tokens and adaptation using self-generated tokens. This suggests that carefully tuning the transition period

Model	PEFT	Method	WASSA	SICK	STSB	LCP	CRP	Humicroedit	Avg.
Llama2-7B	LoRA	Predictor	0.454/0.151	0.860/0.280	0.965/0.950	0.930/0.105	1.014/0.784	1.348/1.046	0.928/0.553
		Generator	0.090/0.023	0.340/0.195	0.610/0.630	0.900/0.105	0.465/0.349	0.650/0.505	0.509/0.301
		PredGen	0.088/0.022	0.320/0.190	0.576/**0.569**	0.062/0.008	0.420/0.280	0.550/0.455	0.338/0.257
	AdaLoRA	Predictor	0.424/0.148	0.845/0.270	0.950/0.935	0.918/0.100	1.020/0.790	1.360/1.050	0.920/0.549
		Generator	0.087/0.022	0.325/0.185	0.600/0.620	0.890/0.097	0.455/0.335	0.630/0.490	0.498/0.291
		PredGen	**0.080/0.020**	0.305/0.185	**0.575**/0.570	**0.058/0.006**	**0.405/0.270**	**0.535/0.440**	0.326/0.248
	RoCoFT	Predictor	0.424/0.148	0.854/0.274	0.958/0.942	0.924/0.102	0.990/0.770	1.340/1.040	0.915/0.546
		Generator	0.085/0.021	0.332/0.191	0.605/0.623	0.895/0.099	0.460/0.337	0.641/0.497	0.503/0.295
		PredGen	0.084/0.021	0.311/0.187	0.583/0.580	0.06/0.007	**0.405**/0.274	0.543/0.448	0.332/0.253
	DoRA	Predictor	0.511/0.150	0.850/0.275	0.960/0.945	0.922/0.104	0.980/0.780	1.355/1.048	0.930/0.550
		Generator	0.086/0.022	0.330/0.190	0.607/0.625	0.885/0.100	0.462/0.338	0.645/0.500	0.503/0.296
		PredGen	0.085/0.021	**0.301/0.184**	0.580/0.578	0.061/0.007	0.415/0.275	0.540/0.445	0.333/0.252
Llama2-13B	LoRA	Predictor	0.370/0.130	0.800/0.250	0.920/0.910	0.880/0.090	0.950/0.720	1.280/1.000	0.867/0.517
		Generator	0.075/0.018	0.310/0.175	0.580/0.590	0.850/0.090	0.430/0.310	0.600/0.460	0.474/0.274
		PredGen	0.074/0.018	**0.287/0.169**	0.550/0.540	0.052/0.006	0.380/0.250	0.500/**0.400**	0.308/0.231
	AdaLoRA	Predictor	0.360/0.125	0.810/0.255	0.930/0.920	0.890/0.095	0.960/0.730	1.300/1.010	0.875/0.522
		Generator	0.078/0.019	0.315/0.178	0.585/0.600	0.860/0.093	0.440/0.320	0.610/0.470	0.481/0.280
		PredGen	0.078/0.019	0.300/0.175	**0.530/0.530**	0.054/0.006	0.390/0.255	0.510/0.410	0.315/0.236
	RoCoFT	Predictor	0.380/0.135	0.790/0.245	0.910/0.900	0.870/0.088	0.940/0.710	1.270/0.990	0.860/0.511
		Generator	0.072/0.017	0.305/0.172	0.575/0.580	0.845/0.088	0.425/0.305	0.590/0.450	0.860/0.511
		PredGen	**0.070**/0.017	0.288/**0.169**	0.545/0.538	**0.053/0.007**	**0.375/0.248**	**0.495**/0.401	0.307/0.232
	DoRA	Predictor	0.365/0.128	0.805/0.252	0.925/0.915	0.924/0.102	0.955/0.725	1.290/1.005	0.877/0.521
		Generator	0.076/0.018	0.312/0.176	0.590/0.605	0.855/0.092	0.435/0.315	0.605/0.465	0.479/0.279
		PredGen	**0.070/0.016**	0.295/0.172	0.555/0.548	0.053/0.006	0.385/0.252	0.505/0.405	0.311/0.233
Llama2-8B	LoRA	Predictor	0.380/0.140	0.820/0.260	0.940/0.925	0.910/0.098	0.970/0.740	1.310/1.020	0.888/0.531
		Generator	0.081/0.019	0.320/0.180	0.595/0.610	0.870/0.095	0.440/0.325	0.620/0.480	0.488/0.285
		PredGen	0.077/0.019	0.298/0.173	0.565/0.555	**0.055/0.006**	0.395/0.260	0.520/0.420	0.318/0.239
	AdaLoRA	Predictor	0.375/0.135	0.830/0.265	0.945/0.930	0.910/0.098	0.980/0.750	1.320/1.030	0.893/0.535
		Generator	0.080/0.020	0.325/0.183	0.600/0.615	0.875/0.097	0.450/0.330	0.630/0.485	0.493/0.288
		PredGen	0.078/0.019	0.303/0.177	0.570/0.560	0.057/0.007	0.400/0.265	**0.509/0.410**	0.323/0.243
	RoCoFT	Predictor	0.390/0.145	0.810/0.255	0.935/0.920	0.910/0.098	0.960/0.730	1.300/1.015	0.884/0.527
		Generator	0.082/0.020	0.315/0.177	0.585/0.605	0.865/0.092	0.435/0.320	0.610/0.475	0.482/0.282
		PredGen	0.079/0.020	**0.288/0.169**	0.565/**0.558**	0.058/0.007	**0.385/0.255**	0.530/0.425	0.317/0.238
	DoRA	Predictor	0.385/0.138	0.825/0.261	0.950/0.935	0.905/0.096	0.975/0.745	1.315/1.025	0.893/0.533
		Generator	0.078/0.019	0.322/0.179	0.592/0.608	0.880/0.096	0.445/0.328	0.625/0.482	0.490/0.285
		PredGen	**0.073/0.018**	0.300/0.175	**0.562/0.558**	0.066/0.007	0.390/0.262	0.525/0.425	0.319/0.241

Table 2: Performance of regression with Different PEFT Methods Across Benchmarks.

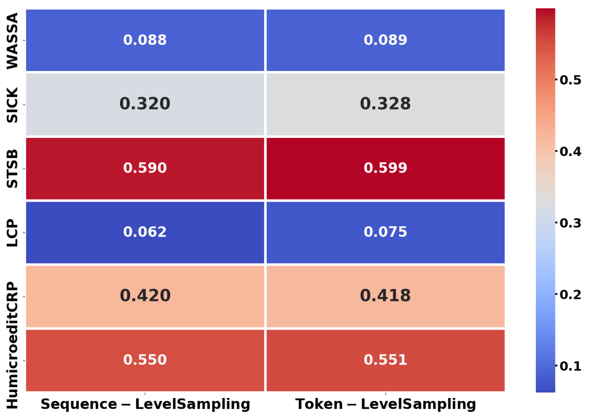

Figure 5: MSE loss comparison between Sequence-Level and Token-Level Sampling across datasets.

is key to improving the accuracy of generative prediction tasks.

Token-Level vs. Sequence-Level Scheduled Sampling We analyze the impact of two different approaches in scheduled sampling: **Sequence-Level Sampling** and **Token-Level Sampling**. In *Sequence-Level Sampling*, the entire output sequence $\mathbf{Y}$ is either fully generated by the model or fully replaced with ground-truth tokens during training. In contrast, *Token-Level Sampling* selectively decides for each token $\mathbf{Y_t}$ whether to use the generated output or ground truth, allowing a more gradual transition.

Figure 5 presents the MSE loss comparison between these two schedule sampling methods across six datasets. We observe that Token-Level Sampling consistently achieves slightly lower MSE across most datasets, indicating better alignment between generated tokens and the true outputs. For example, on STSB, the MSE decreases from 0.590 to 0.599, and on LCP, it improves from 0.062 to 0.075. This suggests that allowing the model to mix the generated and ground-truth tokens at a finer level helps it adapt more smoothly, reducing the sharp transitions between training and inference.

Loss Function Comparison: We compare **WDAL** with three other loss functions: **Adaptive Loss**, **Multiplicative Loss**, and **Only Task Adapter Loss**. The *Only Task Adapter Loss* trains the model using only the task adapter's objective,

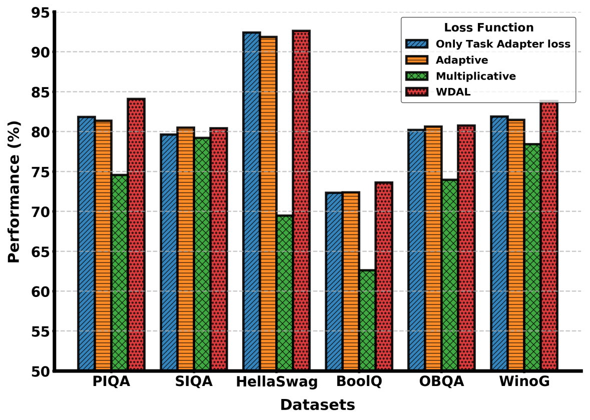

Figure 6: Comparison of WDAL, Adaptive, and Multiplicative loss functions across multiple datasets.

without the generator's loss. This means that the model focuses only on final predictions and does not align with the token-level generation process. Figure 6 shows the accuracy scores across the six datasets.

Our results show that **WDAL** consistently performs the best, achieving the highest accuracy across all datasets, including `HellaSwag` (92.64%), `WinoGrande` (83.85%), and `BoolQ` (73.62%). The *Only Task Adapter Loss* performs the worst, confirming that ignoring the generator's loss weakens the overall performance. This suggests that learning from both token generation and final predictions is important. The *Multiplicative Loss* is unstable, especially on `HellaSwag` (69.48%) and `BoolQ` (62.63%), likely due to an imbalance between the two objectives. The *Adaptive Loss* is more stable but still lags behind WDAL, as it does not properly align token-level generation with task predictions.

7 Related Work

Token-Level Generation: Generating output tokens using LLMs has been widely explored in tasks such as question answering (Rajpurkar, 2016; Yang, 2019), summarization (Lewis, 2019; Zhang et al., 2020), and machine translation (Vaswani, 2017; Wu, 2016). However, using token-level generation for supervised learning tasks like structured prediction remains underexplored. Recent studies (Chen et al., 2023; Yu et al., 2024) show that token-level generation can be more effective than pooled representations by aligning with the LLMs' pre-training objective, leading to better efficiency and robustness against errors.

Mutual Information: Mutual information (MI) helps measure dependencies between features in deep learning (Cover, 1999; Covert et al., 2023). In language models, Chen et al. (2024) used MI for Chain-of-Thought Distillation, while the MIST framework (Kamthawee et al., 2024) applied it to short-text clustering. Unlike these works, we use MI to show that token-level generation retains more information than pooled representations.

Mitigating Exposure Bias and Format Mismatch: Exposure bias occurs when an autoregressive model is trained with ground-truth tokens but must rely on its own predictions during inference. Scheduled sampling (Bengio et al., 2015) helps reduce this gap. Format mismatch arises when generated tokens do not align with the required structured output. Wang et al. (2022) improved coherence by extracting structured information from LLMs, while Liu et al. (2022) modeled structured outputs as action sequences to preserve dependencies.

Unlike previous works, we extend token-level generation to both regression and classification and provide theoretical and empirical proof of its advantages. We also integrate scheduled sampling with a task adapter to ensure generated tokens match numerical or categorical outputs, addressing exposure bias and format mismatch.

8 Conclusion

In this work, we explored the advantages of generation-based prediction over traditional classifier-based approaches. We provided both theoretical and empirical evidence showing that token-level generation retains more task-relevant information than pooled representations. Using the *DPI*, we proved that generating tokens preserves strictly more mutual information with the target output, addressing the limitations of classification-based fine-tuning. To tackle key challenges in generative prediction, we introduced **PredGen**, an end-to-end framework that mitigates *exposure bias* using scheduled sampling and ensures structured outputs through a task adapter. Furthermore, we proposed the **WDAL**, which aligns token generation with task-specific objectives, leading to more coherent and numerically accurate predictions. Extensive experiments on classification and regression benchmarks demonstrated that PredGen consistently outperforms standard baselines.

9 Limitations

While PREDGEN demonstrates strong performance in structured prediction tasks, it has several limitations that warrant further investigation:

- **Inference Latency:** Generation-based prediction introduces additional computational overhead due to the sequential nature of autoregressive decoding. Unlike classification-based methods that produce outputs in a single forward pass, PREDGEN generates outputs token by token, leading to increased inference time, especially for long outputs.
- **Exposure Bias:** Although scheduled sampling helps mitigate exposure bias, it does not fully eliminate the issue. During training, the model sees ground-truth tokens, but during inference, it must rely on its own generated outputs. This transition can still cause compounding errors, particularly in long-horizon predictions.
- **Increased Training Time:** Scheduled sampling increases training time since it requires generating tokens to obtain the final representation. As token generation is inherently sequential and non-parallelizable, this process slows down training compared to traditional classification-based fine-tuning.
- **Task-Specific Adaptation:** PREDGEN depends on carefully designed task adapters to map generated tokens into structured outputs. Developing effective adapters for different tasks may require task-specific tuning, limiting the framework's adaptability to new domains without additional engineering effort.

Addressing these limitations in future work could further improve the efficiency and generalizability of PREDGEN.

References

Normand J Beaudry and Renato Renner. 2011. An intuitive proof of the data processing inequality. *arXiv preprint arXiv:1107.0740*.

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. 2018. Mine: mutual information neural estimation. *arXiv preprint arXiv:1801.04062*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

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

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Xin Chen, Hanxian Huang, Yanjun Gao, Yi Wang, Jishen Zhao, and Ke Ding. 2024. Learning to maximize mutual information for chain-of-thought distillation. *arXiv preprint arXiv:2403.03348*.

Yutian Chen, Hao Kang, Vivian Zhai, Liangze Li, Rita Singh, and Bhiksha Raj. 2023. Token prediction as implicit classification to identify llm-generated text. *arXiv preprint arXiv:2311.08723*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Thomas M Cover. 1999. *Elements of information theory*. John Wiley & Sons.

Ian Connick Covert, Wei Qiu, Mingyu Lu, Na Yoon Kim, Nathan J White, and Su-In Lee. 2023. Learning to maximize mutual information for dynamic feature selection. In *International Conference on Machine Learning*, pages 6424–6447. PMLR.

Scott Crossley, Aron Heintz, Joon Suh Choi, Jordan Batchelor, Mehrnoush Karimi, and Agnes Malatinszky. 2023. A large-scaled corpus for assessing text readability. *Behavior Research Methods*, 55(2):491–507.

Sigmund Freud. 1997. *Dora: An Analysis of a Case of Hysteria*. Simon and Schuster.

Nabil Hossain, John Krumm, and Michael Gamon. 2019. " president vows to cut< taxes> hair": Dataset and analysis of creative text editing for humorous headlines. *arXiv preprint arXiv:1906.00274*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Suyuan Huang, Chao Zhang, Yuanyuan Wu, Haoxin Zhang, Yuan Wang, Maolin Wang, Shaosheng Cao, Tong Xu, Xiangyu Zhao, Zengchang Qin, et al. 2024. Scalingnote: Scaling up retrievers with large language models for real-world dense retrieval. *arXiv preprint arXiv:2411.15766*.

Krissanee Kamthawee, Can Udomcharoenchaikit, and Sarana Nutanong. 2024. Mist: mutual information maximization for short text clustering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11309–11324.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1. Minneapolis, Minnesota.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Md Kowsher, Tara Esmaeilbeig, Chun-Nam Yu, Mojtaba Soltanalian, and Niloofar Yousefi. 2024a. Rocoft: Efficient finetuning of large language models with row-column updates. *arXiv preprint arXiv:2410.10075*.

Md Kowsher, Nusrat Jahan Prottasha, and Prakash Bhat. 2024b. Propulsion: Steering llm with tiny finetuning. *arXiv preprint arXiv:2409.10927*.

Mike Lewis. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Tianyu Liu, Yuchen Jiang, Nicholas Monath, Ryan Cotterell, and Mrinmaya Sachan. 2022. Autoregressive structured prediction with language models. *arXiv preprint arXiv:2210.14698*.

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364.

Andrzej Maćkiewicz and Waldemar Ratajczak. 1993. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Dongsuk Oh, Yejin Kim, Hodong Lee, H Howie Huang, and Heuiseok Lim. 2022. Don't judge a language model by its last layer: Contrastive learning with layer-wise attention pooling. *arXiv preprint arXiv:2209.05972*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

P Rajpurkar. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten Rijke, Zhumin Chen, and Jiahuan Pei. 2024. Melora: mini-ensemble low-rank adapters for parameter-efficient fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3052–3064.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3:1–13.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Matthew Shardlow, Michael Cooper, and Marcos Zampieri. 2020. Complex: A new corpus for lexical complexity prediction from likert scale data. *arXiv preprint arXiv:2003.07008*.

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

MTCAJ Thomas and A Thomas Joy. 2006. *Elements of information theory*. Wiley-Interscience.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

R Vinayakumar, B Premjith, Sachin Kumar, Soman Kp, and Prabaharan Poornachandran. 2017. deep-cybernet at emoint-2017: Deep emotion intensities in tweets. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 259–263.

Xingyao Wang, Sha Li, and Heng Ji. 2022. Code4struct: Code generation for few-shot event structure prediction. *arXiv preprint arXiv:2210.12810*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024. Mixture-of-subspaces in low-rank adaptation. *arXiv preprint arXiv:2406.11909*.

Yonghui Wu. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Zhilin Yang. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Yao-Ching Yu, Chun-Chih Kuo, Ziqi Ye, Yu-Cheng Chang, and Yueh-Se Li. 2024. Breaking the ceiling of the llm community by treating token generation as a classification for ensembling. *arXiv preprint arXiv:2406.12585*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International conference on machine learning*, pages 11328–11339. PMLR.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023a. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Contents

A Writer-Director Alignment Loss

A.1 Full Derivation of WDAL

Here, we present the step-by-step derivation of the WDAL for completeness. The WDAL is defined

$$L_{\text{WDAL}} = \max\bigl(L_W^2, L_D^2\bigr) \exp\Bigl(-\Bigl|\log L_W - \log L_D\Bigr|\Bigr).$$

Multiplicative Form: We begin by defining the overall WDAL as the product of the writer's loss L_W and the director's loss L_D:

$$L_{\text{WDAL}} = L_W \cdot L_D.$$

The key rationale is twofold:

1. If the writer fails (L_W is large), the director (conditioned on the writer) must also fail; hence the product remains large.
2. If the writer succeeds (L_W is small) but the director fails (L_D is large), the product remains large, enforcing joint success.

Log-Sum-Exp Trick: To ensure numerical stability, we consider the logarithms:

$$\log L_{\text{WDAL}} = \log L_W + \log L_D.$$

However, adding $\log L_W$ and $\log L_D$ directly can be numerically unstable if L_W or L_D has a large disparity.

To further stabilize computations, we introduce

$$M = \log\bigl(\max(L_W, L_D)\bigr),$$

and rewrite:

$$\log L_{\text{WDAL}} = (\log L_W - M) + (\log L_D - M) + 2M.$$

Exponentiating both sides gives:

$$L_{\text{WDAL}} = e^{(\log L_W - M)} \cdot e^{(\log L_D - M)} \cdot e^{2M}.$$

Substituting $M = \log(\max(L_W, L_D))$, we can factor out the largest term squared:

$$L_{\text{WDAL}} = \max\bigl(L_W^2, L_D^2\bigr) \cdot e^{\log L_W - \log\max(L_W, L_D)} \cdot e^{\log L_D - \log\max(L_W, L_D)}.$$

Case Analysis: We now consider two cases based on which of L_W or L_D is larger.

Case I: $L_W > L_D$. Then

$$\max(L_W, L_D) = L_W \implies M = \log L_W,$$

and

$$L_{\text{WDAL}} = L_W^2\, e^{\log L_D - \log L_W}.$$

Case II: $L_D > L_W$. Then

$$\max(L_W, L_D) = L_D \implies M = \log L_D,$$

and

$$L_{\text{WDAL}} = L_D^2\, e^{\log L_W - \log L_D}.$$

Generalized Form: Both cases can be combined by noting the exponent is $\exp(-|\log L_W - \log L_D|)$. Thus, we obtain:

$$L_{\text{WDAL}} = \max\bigl(L_W^2, L_D^2\bigr) \cdot \exp\Bigl(-\Bigl|\log L_W - \log L_D\Bigr|\Bigr).$$

A.2 Interpretation

- **Authority Component $\max(L_W^2, L_D^2)$:** Focuses on whichever loss is larger, ensuring the training signals address the most critical error source.
- **Alignment Penalty $\exp(-|\log L_W - \log L_D|)$:** Penalizes mismatches between writer and director, pushing their losses to be consistent in log-space.

This completes the derivation and motivational breakdown of the Writer-Director Alignment Loss.

A.3 Differentiability and Subdifferentiability of $\max(L_W^2, L_D^2)$

The function $\max(L_W^2, L_D^2)$ is defined as:

$$\max(L_W^2, L_D^2) = \begin{cases} L_W^2, & \text{if } L_W^2 > L_D^2, \\ L_D^2, & \text{if } L_D^2 > L_W^2. \end{cases} \tag{1}$$

This function is continuous everywhere but introduces a non-differentiable point at $L_W^2 = L_D^2$. To analyze its differentiability and subdifferentiability, we compute the partial derivatives (with respect to L_W and L_D) for $L_W^2 \neq L_D^2$:

$$\frac{\partial}{\partial L_W} \max(L_W^2, L_D^2) = \begin{cases} 2L_W, & \text{if } L_W^2 > L_D^2, \\ 0, & \text{if } L_W^2 < L_D^2, \end{cases}$$

$$\frac{\partial}{\partial L_D} \max(L_W^2, L_D^2) = \begin{cases} 0, & \text{if } L_W^2 > L_D^2, \\ 2L_D, & \text{if } L_W^2 < L_D^2. \end{cases} \tag{2}$$

At the point $L_W^2 = L_D^2$, the derivative is undefined due to the non-smooth transition between the two cases. However, $\max(L_W^2, L_D^2)$ is *subdifferentiable* at $L_W^2 = L_D^2$, and its subdifferential can be expressed as:

$$\partial \max(L_W^2, L_D^2) = \Big\{\big(a \cdot 2L_W,\ (1-a) \cdot 2L_D\big). \tag{3}$$

This subgradient property ensures that for any $a \in [0, 1]$,

$$\frac{\partial}{\partial L_W} \max(L_W^2, L_D^2) + \frac{\partial}{\partial L_D} \max(L_W^2, L_D^2) = 2L_W + 2L_D.$$

Notably, $\max(L_W^2, L_D^2)$ retains the convexity of the $\max$ operator, ensuring its suitability in optimization methods even with the non-differentiability at $L_W^2 = L_D^2$. Standard gradient-based approaches can use any valid subgradient from $\partial \max(L_W^2, L_D^2)$ at that point, thereby preserving convergence guarantees in convex optimization frameworks.

Differentiability of $\exp\big(-|\log L_W - \log L_D|\big)$

The term $\big|\log L_W - \log L_D\big|$ is piecewise differentiable and can be written as:

$$|\log L_W - \log L_D| = \begin{cases} \log L_W - \log L_D, & \text{if } \log L_W \geq \log L_D, \\ \log L_D - \log L_W, & \text{if } \log L_D > \log L_W. \end{cases}$$

Hence, its partial derivatives with respect to L_W and L_D are:

$$\frac{\partial}{\partial L_W}\big|\log L_W - \log L_D\big| = \begin{cases} \frac{1}{L_W}, & \text{if } \log L_W \geq \log L_D, \\ -\frac{1}{L_W}, & \text{if } \log L_D > \log L_W, \end{cases}$$

$$\frac{\partial}{\partial L_D}\big|\log L_W - \log L_D\big| = \begin{cases} -\frac{1}{L_D}, & \text{if } \log L_W \geq \log L_D, \\ \frac{1}{L_D}, & \text{if } \log L_D > \log L_W. \end{cases}$$

By the chain rule, the exponential term $\exp\big(-|\log L_W - \log L_D|\big)$ is smooth and differentiable everywhere. Its partial derivatives become:

$$\frac{\partial}{\partial L_W} \exp\big(-|\log L_W - \log L_D|\big) = -\exp\big(-|\log L_W - \log L_D|\big) \times \frac{\partial}{\partial L_W}\big|\log L_W - \log L_D\big|,$$

$$\frac{\partial}{\partial L_D} \exp\big(-|\log L_W - \log L_D|\big) = -\exp\big(-|\log L_W - \log L_D|\big) \times \frac{\partial}{\partial L_D}\big|\log L_W - \log L_D\big|.$$

At the boundary case $L_W = L_D$, we have $|\log L_W - \log L_D| = 0$, which implies:

$$\exp\big(-|\log L_W - \log L_D|\big) = 1,$$
$$\frac{\partial}{\partial L_W} \exp\big(-|\log L_W - \log L_D|\big) = 0,$$
$$\frac{\partial}{\partial L_D} \exp\big(-|\log L_W - \log L_D|\big) = 0.$$

Hence, there is no discontinuity in the exponential term at $L_W = L_D$. It remains differentiable and can be seamlessly incorporated into gradient-based optimization routines.

A.4 Theoretical Bounds of the WDAL

To gain insight into the possible range of the Writer-Director Alignment Loss (WDAL), recall its final form:

$$L_{\text{WDAL}} = \max\big(L_W^2, L_D^2\big)\ \exp\Big(-\big|\log L_W - \log L_D\big|\Big).$$

For $L_W, L_D > 0$, we can show that this simplifies to $L_W\, L_D$ except at the boundary $L_W = L_D$ (where it still equals $L_W\, L_D$). Consequently:

$$L_{\text{WDAL}} = L_W\, L_D.$$

provided $L_W \neq 0$ and $L_D \neq 0$. Below, we outline the key boundary behaviors.

Lower Bound. If either $L_W \to 0$ or $L_D \to 0$, then $L_W\, L_D \to 0$. Hence,

$$\lim_{L_W \to 0 \text{ or } L_D \to 0} L_{\text{WDAL}} = 0.$$

However, note that $\log(0)$ is undefined numerically; in practice, one ensures L_W and L_D stay positive or uses a small ϵ (e.g., 10^{-8}) to avoid taking the log of zero. Still, from a theoretical standpoint, $\boxed{\min L_{\text{WDAL}} = 0}$.

Upper Bound. As either $L_W \to \infty$ or $L_D \to \infty$, the product $L_W\, L_D \to \infty$. Thus, there is no finite upper bound:

$$\lim_{L_W \to \infty \text{ or } L_D \to \infty} L_{\text{WDAL}} = \infty.$$

Hence, $\boxed{L_{\text{WDAL}} \text{ is unbounded above.}}$

Overall Range. Combining these observations, for $L_W, L_D \geq 0$, the possible values of L_{WDAL} lie in the interval $[0, \infty)$. In most practical NLP applications, neither loss would *exactly* be zero nor unbounded, so L_{WDAL} typically occupies a finite, positive range. Nonetheless, the flexibility to approach 0 or grow arbitrarily large is crucial for reflecting both *complete success* (very small losses) and *catastrophic failure* (very large losses).

B Theoretical Justification

B.1 Conditioning Reduces Entropy

Theorem 2. *Let X and Y be continuous random variables with joint density $f_{X,Y}(x, y)$, marginal densities $f_X(x)$, $f_Y(y)$, and conditional density $f_{X|Y}(x|y)$. The differential entropy satisfies:*

$$H(X) \geq H(X|Y),$$

where $H(X)$ and $H(X|Y)$ denote the marginal and conditional differential entropy, respectively. (Thomas and Joy, 2006)

Proof. For continuous random variables, differential entropy is defined as:

$$H(X) = -\int f_X(x) \log f_X(x) dx,$$
$$H(X|Y) = -\iint f_{X,Y}(x, y) \log f_{X|Y}(x|y) dx dy.$$

Substituting $f_{X|Y}(x|y) = \frac{f_{X,Y}(x,y)}{f_Y(y)}$ into $H(X|Y)$, we derive:

$$H(X|Y) = -\iint f_{X,Y}(x, y) \log \frac{f_{X,Y}(x, y)}{f_Y(y)} dx dy$$

Expanding the logarithm:

$$H(X|Y) = \underbrace{-\iint f_{X,Y}(x, y) \log f_{X,Y}(x, y)\, dx dy}_{H(X,Y)} + \iint f_{X,Y}(x, y) \log f_Y(y)\, dx dy.$$

The second term simplifies using the marginal $\int f_{X,Y}(x, y) dx = f_Y(y)$:

$$\iint f_{X,Y}(x, y) \log f_Y(y) dx dy = \int f_Y(y) \log f_Y(y) dy = -H(Y).$$

Thus,

$$H(X|Y) = H(X, Y) - H(Y).$$

To show $H(X) \geq H(X|Y)$, we invoke the non-negativity of the Kullback-Leibler (KL) divergence:

$$D_{\text{KL}}(f_{X,Y} \| f_X f_Y) = \iint f_{X,Y}(x, y) \log \frac{f_{X,Y}(x, y)}{f_X(x) f_Y(y)} dx dy \geq 0.$$

Expanding the integrand:

$$D_{\text{KL}} = \iint f_{X,Y}(x, y) \log f_{X,Y}(x, y) dx dy - \iint f_{X,Y}(x, y) \log f_X(x) dx dy - \iint f_{X,Y}(x, y) \log f_Y(y) dx dy.$$

Recognizing the entropy terms:

$$D_{\text{KL}} = -H(X, Y) + H(X) + H(Y) \geq 0 \implies H(X) + H(Y) \geq H(X, Y).$$

Substituting $H(X, Y) = H(X|Y) + H(Y)$ into the inequality:

$$H(X) \geq H(X|Y).$$

□

B.2 Monotonicity of Conditional Entropy

.

Theorem 3. *Let X, Y, Z be continuous random variables. Differential entropy satisfies:*

$$H(X|Y) \geq H(X|Y, Z),$$

with equality if $X \perp Z|Y$. This generalizes to

$$H(X|Y_1) \geq H(X|Y_1, Y_2) \geq \cdots \geq H(X|Y_1, \ldots, Y_n).$$

Proof. The conditional differential entropies are defined as:

$$H(X|Y) = -\iint f_Y(y) f_{X|Y}(x|y) \log f_{X|Y}(x|y)\, dxdy,$$

$$H(X|Y,Z) = -\iiint f_{Y,Z}(y,z) f_{X|Y,Z}(x|y,z) \log f_{X|Y,Z}(x|y,z)\, dxdydz.$$

The marginal conditional density relates to the joint via:

$$f_{X|Y}(x|y) = \int f_{X|Y,Z}(x|y,z) f_{Z|Y}(z|y)\, dz.$$

Substituting this into $H(X|Y)$:

$$H(X|Y) = -\iint f_Y(y) \cdot \underbrace{\left[\int f_{X|Y,Z}(x|y,z) f_{Z|Y}(z|y)\, dz\right]}_{f_{X|Y}(x|y)} \log\left(\int f_{X|Y,Z}(x|y,z) f_{Z|Y}(z|y)\, dz\right) dxdy.$$

Apply Jensen's inequality

Using the convexity of $-\log(\cdot)$ and Jensen's inequality:

$$-\log\left(\int f_{X|Y,Z}(x|y,z) f_{Z|Y}(z|y)\, dz\right) \leq -\int f_{Z|Y}(z|y) \log f_{X|Y,Z}(x|y,z)\, dz.$$

Substituting this bound:

$$H(X|Y) \geq -\iiint f_Y(y) f_{Z|Y}(z|y) f_{X|Y,Z}(x|y,z) \log f_{X|Y,Z}(x|y,z)\, dxdydz.$$

Simplifying via $f_Y(y) f_{Z|Y}(z|y) = f_{Y,Z}(y,z)$:

$$H(X|Y) \geq -\iiint f_{Y,Z}(y,z) f_{X|Y,Z}(x|y,z) \log f_{X|Y,Z}(x|y,z)\, dxdydz = H(X|Y,Z).$$

Generalization. Iteratively applying this result gives:

$$H(X|Y_1) \geq H(X|Y_1,Y_2) \geq \cdots \geq H(X|Y_1,\ldots,Y_n).$$

□

B.3 Mutual Information Decreases Under Deterministic Compression

Let $\mathbf{X} \in \mathbb{R}^{n\times d}$ denote the input with n tokens and d-dimensional embeddings, and let $\mathbf{Z} \in \mathbb{R}^{n\times d}$ represent the final hidden representation. Here $\mathbf{Z_p} \in \mathbb{R}^{d}$ as a pooled representation. $\mathbf{Y}$ is output.

We aim to show that the generator provides better predictions than the predictor, i.e.,

$$I(\mathbf{Y};\mathbf{Z}) \geq I(\mathbf{Y};\mathbf{Z_p}). \tag{4}$$

According to the Data Processing Inequality (DPI) (Beaudry and Renner, 2011), this process of predictor can be expressed as

$$\mathbf{X} \to \mathbf{Z} \to \mathbf{Z_p} \to \mathbf{Y}. \tag{5}$$

Similarly for the generator

$$\mathbf{X} \to \mathbf{Z} \to \mathbf{Y}. \tag{6}$$

Now, from the definition of mutual information in the Information Bottleneck (IB) framework, we know

$$I(\mathbf{Y};\mathbf{Z_p}) = H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{Z_p}), \tag{7}$$

and similarly

$$I(\mathbf{Y};\mathbf{Z}) = H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{Z}). \tag{8}$$

To prove that the the generator is better than the predictor in predicting $\mathbf{Y}$, we need to show

$$I(\mathbf{Y};\mathbf{Z}) \geq I(\mathbf{Y};\mathbf{Z_p}). \tag{9}$$

From (7) and (8), this reduces to proving

$$H(\mathbf{Y}|\mathbf{Z}) \leq H(\mathbf{Y}|\mathbf{Z_p}).$$

Now in order to proof this, we can consider two real scenario how we are getting $\mathbf{Z_p}$. One common practice is pooling where output $\mathbf{Z_p}$ is considered the first special token representation of model and another case is $\mathbf{Z_p}$ which will be considered as the means of all tokens representation.

When $\mathbf{Z}_p$ is the representation of the first special token:

In this case, we can define

$$\mathbf{Z} = \big(\mathbf{Z}_p, \mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_{n-1}\big) \in \mathbb{R}^{n\times d}.$$

Then we have

$$H(\mathbf{Y} \,|\, \mathbf{Z}) = H\big(\mathbf{Y} \,|\, \mathbf{Z}_p, \mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_{n-1}\big).$$

Now following the principle of conditional entropy and the previous theorem (B.1 and B.2):

$$H(X|Y_1, Y_2, \dots, Y_n) \le H(X|Y_1, Y_2, \dots, Y_{n-1}) \le \cdots \le H(X|Y_1),$$

it follows that:

$$H(\mathbf{Y}|\mathbf{Z_p}, \mathbf{Z_1}, \mathbf{Z_2}, \cdots \mathbf{Z_{n-1}}) \le H(\mathbf{Y}|\mathbf{Z_p}).$$

Therefore:

$$H(\mathbf{Y}|\mathbf{Z}) \le H(\mathbf{Y}|\mathbf{Z_p}).$$

Combining this result with the mutual information definitions, we conclude:

$$I(\mathbf{Y}; \mathbf{Z}) \ge I(\mathbf{Y}; \mathbf{Z_p}).$$

When $\mathbf{Z_p}$ is the mean of $\mathbf{Z}$: From the definition of conditional entropy

$$H(\mathbf{Y} \mid \mathbf{Z}) = \mathbb{E}_{p(\mathbf{Z})}\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}) \log p(\mathbf{Y} \mid \mathbf{Z})\, d\mathbf{Y}\Big].$$

Similarly,

$$H(\mathbf{Y} \mid \mathbf{Z}_p) = \mathbb{E}_{p(\mathbf{Z}_p)}\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}_p) \log p(\mathbf{Y} \mid \mathbf{Z}_p) d\mathbf{Y}\Big].$$

Hence, we need to show

$$\mathbb{E}_{p(\mathbf{Z}_p)}\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}_p) \log p(\mathbf{Y} \mid \mathbf{Z}_p)\, d\mathbf{Y}\Big] \ge \mathbb{E}_{p(\mathbf{Z})}\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}) \log p(\mathbf{Y} \mid \mathbf{Z})\, d\mathbf{Y}\Big].$$

By the law of total probability (marginalization), for each fixed value $\mathbf{z}_p$ of $\mathbf{Z}_p$:

$$p(\mathbf{Y} \mid \mathbf{Z}_p = \mathbf{z}_p) = \int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p = \mathbf{z}_p)\, d\mathbf{Z}.$$

Hence

$$p(\mathbf{Y} \mid \mathbf{Z}_p) = \int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p)\, d\mathbf{Z}.$$

Substitute the above mixture form into the conditional entropy expression:

$$H(\mathbf{Y} \mid \mathbf{Z}_p) = \mathbb{E}_{p(\mathbf{Z}_p)}\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}_p) \log p(\mathbf{Y} \mid \mathbf{Z}_p)\, d\mathbf{Y}\Big]$$
$$= \mathbb{E}_{p(\mathbf{Z}_p)}\Big[-\int \Big(\int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p)\, d\mathbf{Z}\Big) \log\Big(\int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p)\, d\mathbf{Z}\Big) d\mathbf{Y}\Big].$$

Note that the function $-x \log x$ is *concave* for $x > 0$. Equivalently, the Shannon entropy

$$H(p) = -\int p(\mathbf{y}) \log p(\mathbf{y})\, d\mathbf{y}$$

is a *concave functional* in p. Therefore, for the mixture $\int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p)\, d\mathbf{Z}$, we have:

$$-\int \Big(\int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p)\, d\mathbf{Z}\Big) \log\Big(\int p(\mathbf{Y} \mid \mathbf{Z})\, p(\mathbf{Z} \mid \mathbf{Z}_p)\, d\mathbf{Z}\Big)\, d\mathbf{Y}$$
$$\ge \iint p(\mathbf{Z} \mid \mathbf{Z}_p)\, p(\mathbf{Y} \mid \mathbf{Z}) \Big[-\log p(\mathbf{Y} \mid \mathbf{Z})\Big]\, d\mathbf{Y}\, d\mathbf{Z}$$
$$= \int p(\mathbf{Z} \mid \mathbf{Z}_p)\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}) \log p(\mathbf{Y} \mid \mathbf{Z})\, d\mathbf{Y}\Big] d\mathbf{Z}.$$

Thus, inside the expectation over $\mathbf{Z}_p$, we get using Jensen's inequality:

$$-\int p(\mathbf{Y} \mid \mathbf{Z}_p) \log p(\mathbf{Y} \mid \mathbf{Z}_p)\, d\mathbf{Y} \ge \int p(\mathbf{Z} \mid \mathbf{Z}_p)\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}) \log p(\mathbf{Y} \mid \mathbf{Z})\, d\mathbf{Y}\Big] d\mathbf{Z}.$$

Taking the expectation w.r.t. $p(\mathbf{Z}_p)$ then yields

$$H(\mathbf{Y} \mid \mathbf{Z}_p) = \mathbb{E}_{p(\mathbf{Z}_p)}\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}_p) \log p(\mathbf{Y} \mid \mathbf{Z}_p)\, d\mathbf{Y}\Big]$$
$$\ge \mathbb{E}_{p(\mathbf{Z}_p)}\Big[\int p(\mathbf{Z} \mid \mathbf{Z}_p)\Big(-\int p(\mathbf{Y} \mid \mathbf{Z}) \log p(\mathbf{Y} \mid \mathbf{Z})\, d\mathbf{Y}\Big) d\mathbf{Z}\Big].$$

Using the law of total expectation, we get

$$\int p(\mathbf{Z} \mid \mathbf{Z}_p)\Big[-\int p(\mathbf{Y} \mid \mathbf{Z}) \log p(\mathbf{Y} \mid \mathbf{Z})\, d\mathbf{Y}\Big] d\mathbf{Z} = \mathbb{E}_{p(\mathbf{Z}|\mathbf{Z}_p)}\Big[H(\mathbf{Y} \mid \mathbf{Z})\Big].$$

Hence

$$H(\mathbf{Y} \mid \mathbf{Z}_p) \ge \mathbb{E}_{p(\mathbf{Z}_p)}\Big[\mathbb{E}_{p(\mathbf{Z}|\mathbf{Z}_p)}\big[H(\mathbf{Y} \mid \mathbf{Z})\big]\Big] = \mathbb{E}_{p(\mathbf{Z})}\big[H(\mathbf{Y} \mid \mathbf{Z})\big].$$

where the last equality uses the law of total expectation ($\mathbb{E}_{\mathbf{Z}_p}[\mathbb{E}_{\mathbf{Z}|\mathbf{Z}_p}(\cdot)] = \mathbb{E}_{\mathbf{Z}}[\cdot]$) and the fact that $\mathbf{Z}_p$ is a deterministic function of $\mathbf{Z}$. Thus,

$$H(\mathbf{Y} \mid \mathbf{Z}_p) \ge H(\mathbf{Y} \mid \mathbf{Z}).$$

Combining this result with the mutual information definitions, we conclude:

$$I(\mathbf{Y}; \mathbf{Z}) \ge I(\mathbf{Y}; \mathbf{Z_p}).$$

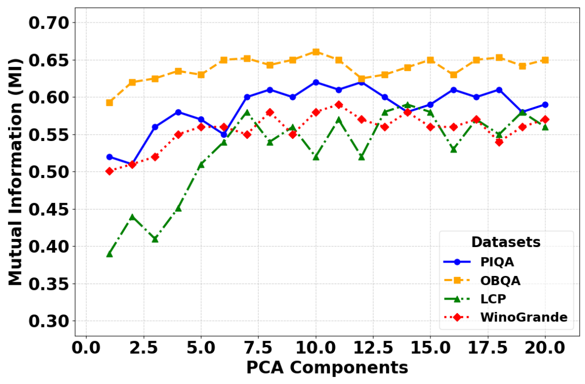

Figure 7: Effect of PCA-based dimensionality reduction on mutual information.

C More Ablation Studies

C.1 PCA Components vs. Mutual Information:

To estimate the mutual information for generation (as discussed in Section 3), we apply Principal Component Analysis (PCA) to reduce the dimensionality of the token representations. Instead of using the full representation of $\mathbf{Z}$ (which has dimensions $n \times d$), we reduce it to k PCA components. This helps lower the computational cost while still retaining the most important information.

Figure 7 shows how the number of PCA components affects mutual information estimation. Initially, as we increase the number of components, the mutual information improves because more task-relevant details are preserved. However, after about 10 to 12 components, the mutual information levels off and adding more components doesn't significantly improve the results. For example, in the OBQA dataset, mutual information increases from 0.593 with just 1 component to 0.661 with 10 components, but stabilizes beyond that point. We observe similar patterns across other datasets.

D Implementation and Hyperparameter Details

We design the *task adapter* $\mathcal{T}$ in PREDGEN to map the model-generated token sequence $\tilde{\mathbf{Y}} = [\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2, \ldots, \tilde{\mathbf{Y}}_m]$ into the final prediction $\tilde{\mathbf{P}}$. Below, we distinguish classification and regression setups.

D.1 Classification Task Adapter.

For classification, let $\tilde{\mathbf{Y}}$ be the autoregressively generated tokens. We then define

$$\tilde{\mathbf{P}} \;=\; \mathcal{T}(\tilde{\mathbf{Y}}) \;=\; \mathrm{softmax}\Big(W\,\big[\mathrm{CLS}(\tilde{\mathbf{Y}})\big]\Big),$$

where W is a trainable parameter matrix and $\mathrm{CLS}(\cdot)$ denotes extracting a "classification" embedding. We use the standard cross-entropy loss $\mathcal{L}_{\mathrm{CE}}$ for both the generator (token-level) and the classifier (label-level). Since both losses live in the same space (cross-entropy), our Writer-Director Alignment Loss (WDAL) remains stable.

D.2 Regression Task Adapter with Ordered Penalty.

When the final output must be a continuous value (e.g., 0.75 in a Semantic Textual Similarity task), representing it as tokens `[0, ., 7, 5]` can introduce ambiguity: cross-entropy penalizes each token without directly encoding how numerically close $\tilde{\mathbf{Y}}$ is to $\mathbf{Y}$. To address this, we introduce an *ordered penalty* that magnifies errors in more significant token positions.

Formally, let $\mathbf{Y} = [\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_m]$ be the ground-truth token sequence (e.g., digits of a decimal number) and $\tilde{\mathbf{Y}} = [\tilde{\mathbf{Y}}_1, \tilde{\mathbf{Y}}_2, \ldots, \tilde{\mathbf{Y}}_m]$ be the predicted tokens. Define a monotonically decreasing penalty vector

$$\boldsymbol{\alpha} \;=\; [\alpha_1, \alpha_2, \ldots, \alpha_m]$$

$$\text{where } \alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m > 0.$$

Each position i is then weighted by α_i to reflect its significance. We modify the token-level cross-entropy $\mathcal{L}_{\mathrm{CE}}(\tilde{y}_i, y_i)$ to:

$$L_D = \mathcal{L}_{\mathrm{ord}}\big(\tilde{\mathbf{Y}}, \mathbf{Y}\big) \;=\; \sum_{i=1}^{m} \alpha_i\, \mathcal{L}_{\mathrm{CE}}\big(\tilde{\mathbf{Y}}_i, \mathbf{Y}_i\big).$$

In practice, we choose $\boldsymbol{\alpha}$ so that early-token mismatches (e.g., the integer or first decimal digit) incur substantially larger penalties compared to later tokens. For example, if $\mathbf{Y}$ has length 4, we might set $\alpha_1 = 1.67, \alpha_2 = 1.33, \alpha_3 = 1.01, \alpha_4 = 1.00$, ensuring that predicting `0.76` (only off by one digit in the third position) has lower cost than predicting `1.75` (off in the first digit).

D.3 WDAL Consistency.

Both the "writer" (generative cross-entropy) and the "director" (classification or regression output) losses must be consistently derived from the same token sequence to maintain stability in WDAL. For classification, this alignment is direct since both losses are cross-entropy. For regression, the ordered penalty $\mathcal{L}_{\mathrm{ord}}$—which is still a sum of token-level cross-entropies with position-dependent scaling—remains compatible with the generative objective, preventing conflicts that arise from mixing cross-entropy (token-level) and MSE/MAE (numerical-value-level).

In all cases, hyperparameters (e.g., learning rate, penalty coefficients α_i, and scheduling parameters for training) are tuned to optimize final validation performance while preserving the coherence of generation and prediction. The hyperparameters details of datasets are given in Table 3.

E Dataset Description

The details of the used datasets in this work have been described in Table 4.

F Additional Experiments

Table 5 presents the results of our math reasoning evaluation. Following Hu et al. (2023), we first train on the *math-10k* dataset and then evaluate performance on various math reasoning benchmarks. The reported results use an exact match criterion, where a prediction is considered correct if the difference from the ground truth is less than 0.0001.

We observe that traditional pooling-based classification performs poorly, primarily because it relies on a classifier layer to predict numerical values. Since classification models are not optimized for generating precise numerical outputs, achieving an exact match is extremely rare. In contrast, PredGen consistently achieves higher accuracy across all benchmarks, demonstrating that token-level generation better preserves numerical precision and reasoning ability.

G PEFT Methods

Parameter-Efficient Fine-Tuning (PEFT) methods have gained significant attention for their ability to adapt large pre-trained models with minimal computational overhead, In this work we have used a whole range of PEFT methods. A key aspect of these methods lies in their reparameterization of the delta weights ($\Delta\mathbf{W}$), which represent the updates to the base model weights. Table 6 illustrates the diverse strategies employed by various PEFT methods, such as LoRA (Hu et al., 2021), AdaLoRA (Zhang et al., 2023b), RoCoFT (Kowsher et al., 2024a), DoRA (Freud, 1997), MELoRA (Ren et al., 2024), LoRA-FA (Zhang et al., 2023a), MoSLoRA (Wu et al., 2024), and Propulsion (Kowsher et al., 2024b). For instance, LoRA employs low-rank decomposition with $\mathbf{W}_{\text{down}}$ and $\mathbf{W}_{\text{up}}$ matrices, while AdaLoRA leverages singular value decomposition (SVD) for adaptive rank updates. RoCoFT introduces restricted row or column updates, and Propulsion focuses on updating only a mask $\mathbf{Z}$ while freezing the base weights. These approaches highlight the trade-offs between efficiency, flexibility, and performance in fine-tuning large models.

H Token Laval Mutual Information

H.1 MI for Classification

To further analyze mutual information (MI), we present token-label MI heatmaps in Figures 8 and 9 using the PIQA dataset. In Figure 8, the correct answer is "Solution 1," where the input text has higher mutual information with "Solution 1" than with "Solution 2." Similarly, in Figure 9, the correct answer is "Solution 2," and the input shows stronger MI with "Solution 2" than with "Solution 1."

H.2 MI for Regression

In Figures 10, 13, and 12, we present MI heatmaps showing the relationship between input tokens and regression values from the LCP dataset.

Dataset	Learning Rate	Batch Size	Grad. Accum. Steps	Epochs	Warmup Steps	LR Scheduler Type	Max Steps for Sampling
CLEAR	2e-3	3	4	25	200	cosine	500
Humicroedit	2e-3	6	4	10	200	cosine	500
LCP	2e-3	1	5	10	200	cosine	500
SICK	2e-3	6	4	10	200	cosine	500
stsb	2e-3	5	4	20	200	cosine	500
WASSA	2e-3	5	4	10	200	cosine	500
BoolQ	2e-3	10	3	10	500	cosine	500
PIQA	2e-3	10	3	10	500	cosine	1000
SIQA	2e-3	10	3	10	500	cosine	500
HellaSwag	2e-3	10	3	10	500	cosine	1000
WinoGrande	2e-3	10	3	10	500	cosine	1000
ARC-e	2e-3	10	3	10	500	cosine	500
ARC-c	2e-3	10	3	10	500	cosine	500
OBQA	2e-3	10	3	10	500	cosine	1000

Table 3: Hyperparameters for Different Datasets

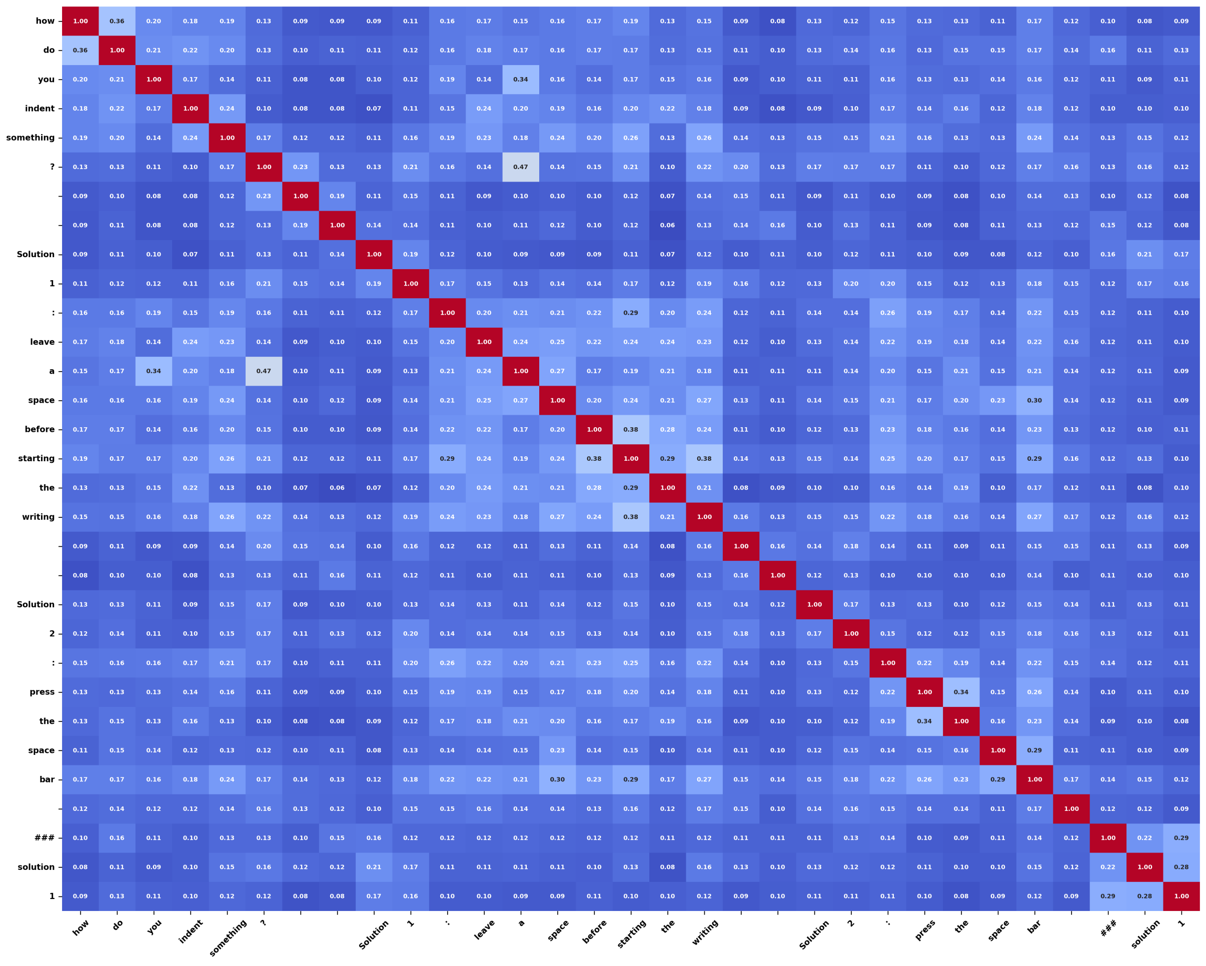

Figure 8: Mutual information (MI) heatmap for classification on the PIQA dataset. The correct answer is "Solution 1," which has higher MI with the input text compared to "Solution 2," indicating stronger alignment with the correct label.

Dataset	**Task Type**	**Domain**	**Description**
BoolQ (Clark et al., 2019)	Classification	Reading Comprehension	A binary question-answering dataset where each naturally occurring question requires a yes/no response based on a Wikipedia passage.
PIQA (Bisk et al., 2020)	Classification	Physical Commonsense	Evaluates models' ability to perform physical reasoning by selecting the most plausible solution to a given problem scenario.
SIQA (Sap et al., 2019)	Classification	Social Intelligence	Tests reasoning about human intentions, emotions, and social interactions by predicting the most likely response to a given situation.
HellaSwag (Zellers et al., 2019)	Classification	Commonsense Reasoning	Challenges models to predict the most logically coherent continuation of a given textual scenario.
WinoGrande (Sakaguchi et al., 2021)	Classification	Coreference Resolution	A large-scale dataset designed to assess pronoun resolution by leveraging commonsense knowledge.
ARC-e (Clark et al., 2018)	Classification	Science QA	The *easy* subset of the AI2 Reasoning Challenge (ARC), featuring relatively straightforward multiple-choice science questions.
ARC-c (Clark et al., 2018)	Classification	Science QA	The *challenge* subset of ARC, containing more difficult questions requiring complex reasoning and external knowledge.
OBQA (Mihaylov et al., 2018)	Classification	Open-Book QA	Evaluates the ability to answer multiple-choice science questions using a predefined set of knowledge facts and external world knowledge.
WASSA (Vinayakumar et al., 2017)	Regression	Sentiment Analysis	A dataset for emotion intensity and sentiment prediction, focusing on fine-grained sentiment classification.
SICK (Marelli et al., 2014)	Regression	Semantic Similarity	The Sentences Involving Compositional Knowledge (SICK) dataset, used for sentence similarity assessment and textual entailment tasks.
STSB (Cer et al., 2017)	Regression	Sentence Similarity	A benchmark for measuring semantic textual similarity, where sentences are scored based on their degree of similarity.
LCP (Shardlow et al., 2020)	Regression	Lexical Complexity	A dataset designed to predict the perceived complexity of words in context.
CLEAR (Crossley et al., 2023)	Regression	Reasoning Complexity	Evaluates the difficulty level of reasoning tasks, quantifying cognitive complexity in natural language understanding.
Humicroedit (Hossain et al., 2019)	Regression	Humor Perception	A dataset from SemEval that assesses humor perception by analyzing minor text modifications (micro-edits).
MultiArith (Roy et al., 2015)	Arithmetic	Mathematics	A dataset focusing on multi-step arithmetic word problems requiring reasoning across multiple operations.
GSM8K (Cobbe et al., 2021)	Arithmetic	Mathematics	A high-quality dataset of challenging grade school math word problems for benchmarking reasoning capabilities.
AddSub (Hosseini et al., 2014)	Arithmetic	Mathematics	A dataset designed for evaluating models' ability to solve arithmetic word problems involving addition and subtraction.
SingleEq (Koncel-Kedziorski et al., 2015)	Arithmetic	Mathematics	Contains single-equation arithmetic problems where the goal is to predict the correct numerical result.
SVAMP (Patel et al., 2021)	Arithmetic	Mathematics	A dataset that introduces variations in arithmetic problems to test the robustness of mathematical reasoning models.

Table 4: Overview of benchmark datasets used for classification, regression, and arithmetic reasoning tasks.

Model	PEFT	Method	MultiArith	GSM8K	AddSub	SingleEq	SVAMP	Avg.
Mistral-7B	MELoRA	Predictor	5.430	0.032	2.890	3.710	0.986	2.610
		Generator	75.22	40.22	71.64	71.73	57.12	63.19
		PredGen	76.47	42.59	73.43	72.48	58.92	64.78
	LoRA-FA	Predictor	4.530	0.030	2.170	3.890	1.030	2.330
		Generator	75.83	40.11	73.43	70.37	56.84	63.32
		PredGen	77.53	42.48	75.14	72.48	57.38	65.00
	MoSLoRA	Predictor	4.470	0.082	3.280	4.890	0.894	2.720
		Generator	74.28	41.48	70.34	71.89	58.06	63.21
		PredGen	75.45	43.68	72.68	71.11	57.92	64.17
	Propulsion	Predictor	5.840	0.103	2.790	88.21	0.837	19.56
		Generator	74.52	41.78	72.07	72.18	56.39	63.39
		PredGen	75.83	43.22	74.38	72.90	57.72	64.81
Gemma2-9B	MELoRA	Predictor	4.790	1.064	2.960	2.850	1.157	2.560
		Generator	76.75	40.96	73.78	72.49	57.68	64.33
		PredGen	78.32	43.63	74.65	74.49	59.32	66.08
	LoRA-FA	Predictor	3.640	0.945	3.230	2.740	1.570	2.420
		Generator	74.79	42.56	74.24	73.81	57.93	64.67
		PredGen	77.27	43.86	75.93	74.63	59.11	66.16
	MoSLoRA	Predictor	3.470	0.976	2.860	2.430	1.670	2.280
		Generator	76.23	43.73	73.98	72.68	56.84	64.69
		PredGen	77.36	46.11	75.85	72.82	58.77	66.18
	Propulsion	Predictor	2.750	1.190	2.660	2.590	1.280	2.09
		Generator	75.23	43.61	73.72	73.18	57.52	64.65
		PredGen	78.18	44.83	75.92	72.87	58.20	66.00
DeepSeek-R1-8B	MELoRA	Predictor	6.276	1.785	8.099	5.374	2.407	4.791
		Generator	78.80	45.25	73.99	71.89	58.70	65.95
		PredGen	80.99	47.40	74.89	73.13	59.15	66.82
	LoRA-FA	Predictor	6.285	2.764	9.564	6.644	2.364	5.529
		Generator	77.92	44.83	73.70	72.00	59.67	65.46
		PredGen	78.68	47.46	73.82	73.55	60.12	66.71
	MoSLoRA	Predictor	5.490	1.882	11.29	5.005	2.727	5.286
		Generator	78.75	45.71	73.91	72.73	60.24	66.93
		PredGen	80.00	46.94	76.01	73.42	61.15	67.65
	Propulsion	Predictor	5.333	2.260	8.837	4.743	1.930	5.024
		Generator	76.80	43.85	73.75	71.52	60.03	65.71
		PredGen	79.05	45.37	76.61	73.68	60.95	68.34

Table 5: Performance of LLMs with Different PEFT Methods Across the Commonsense Benchmarks. Here we used DeepSeek-R1-Distill-Llama-8B.

Method	ΔW Reparameterization	Notes
LoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	$W_{\text{down}} \in \mathbb{R}^{d \times r}$, $W_{\text{up}} \in \mathbb{R}^{r \times d}$, and $r \ll \{k, d\}$.
AdaLoRA	$\Delta W = PAQ$	$PP^\top = P^\top P \neq I = QQ^\top = Q^\top Q$, $\Lambda = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$.
RoCoFT	$\Delta W = W_0 + R$ or $\Delta W = W_0 + C$	R and C are restricted weight matrices such that only at most r of the rows or columns are non-zero.
DoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	Similar to LoRA but with dynamic rank adaptation during training.
MELoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	Multi-expert LoRA, where multiple low-rank updates are combined.
LoRA-FA	$\Delta W = W_{\text{down}} W_{\text{up}} = QRW_{\text{up}}$	W_{down} is frozen, and only W_{up} is updated.
MoSLoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	Mixture of sparse LoRA, combining sparse and low-rank updates.
Propulsion	$\Delta W = W \odot Z$	W is frozen, and only Z is updated.

Table 6: Comparison of delta weight reparameterization across various PEFT methods.

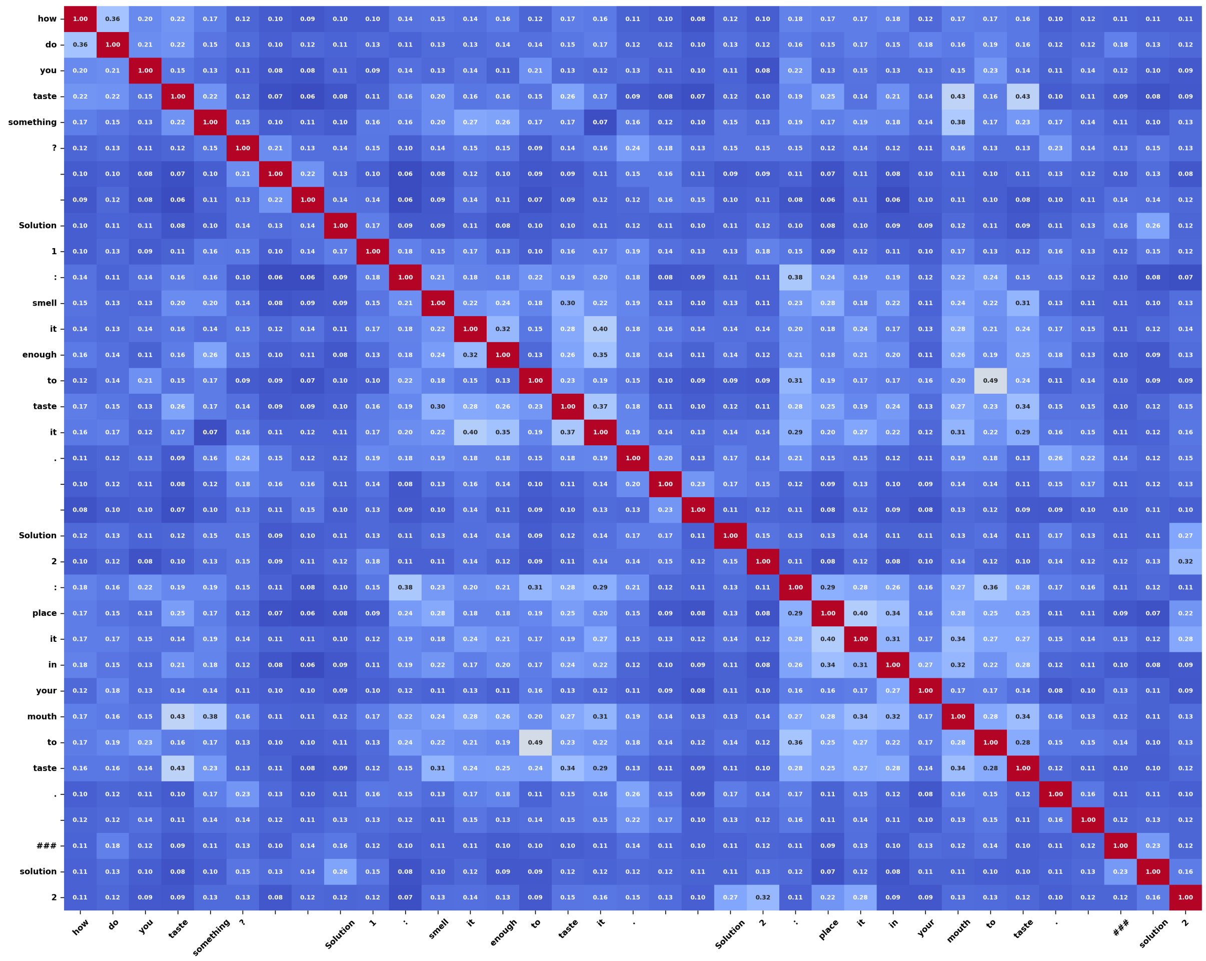

Figure 9: Mutual information (MI) heatmap for classification on the PIQA dataset. The correct answer is "Solution 2," which has higher MI with the input text compared to "Solution 1," demonstrating effective label association.

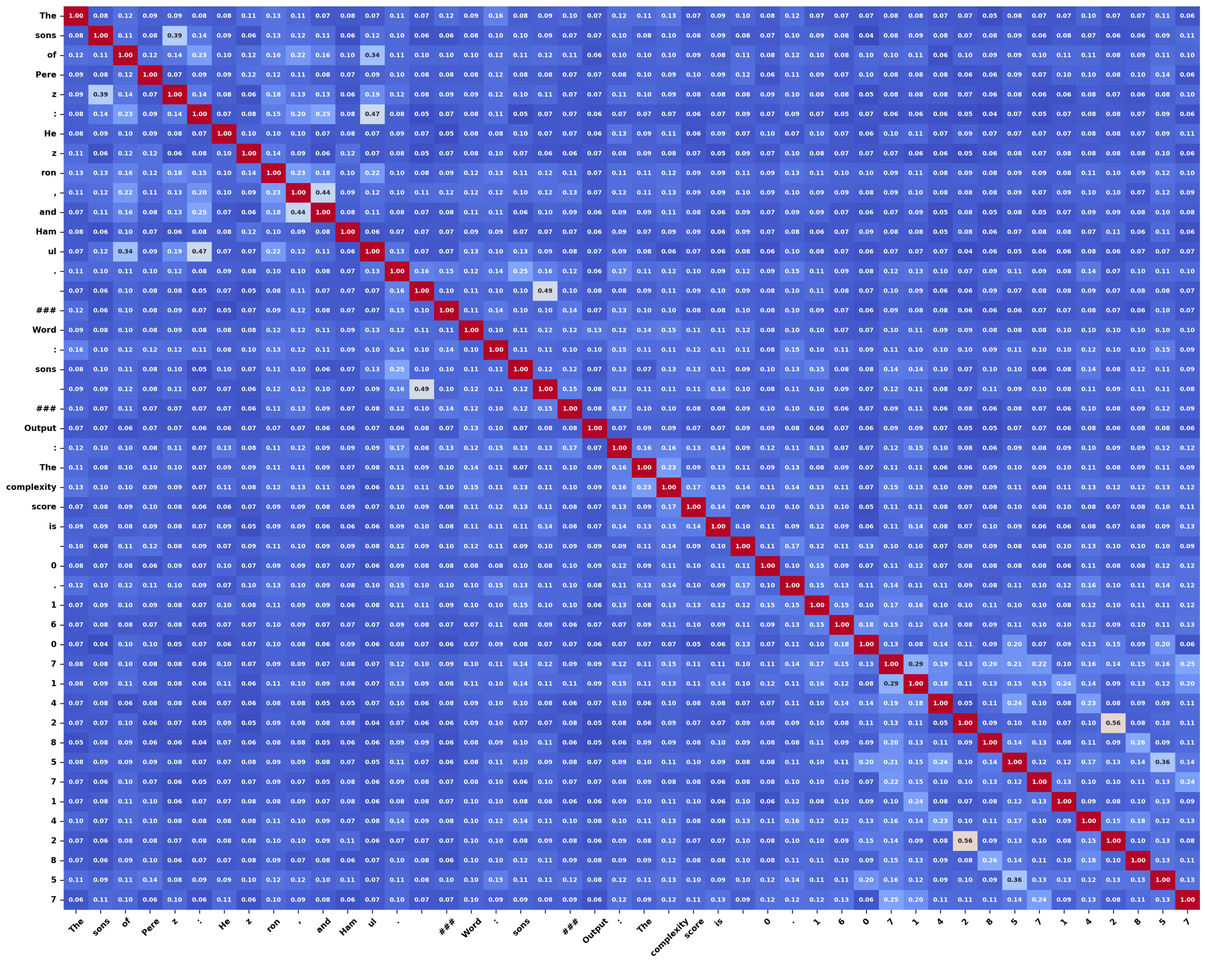

Figure 10: Mutual information (MI) heatmap for regression on the LCP dataset. The visualization shows how input tokens contribute to the predicted regression values, highlighting key dependencies.

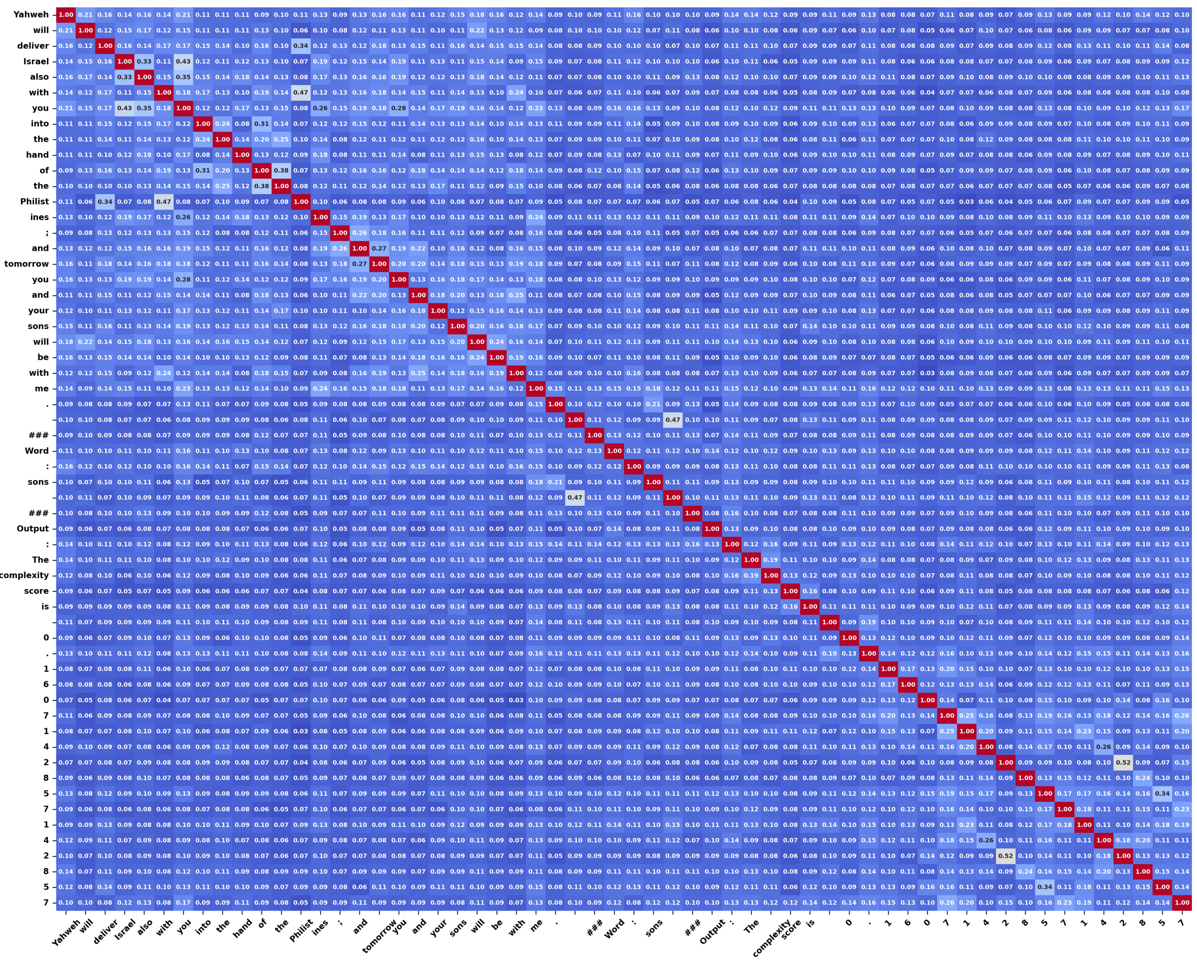
Figure 11: Mutual information (MI) heatmap for regression on the LCP dataset. The visualization shows how input tokens contribute to the predicted regression values, highlighting key dependencies.

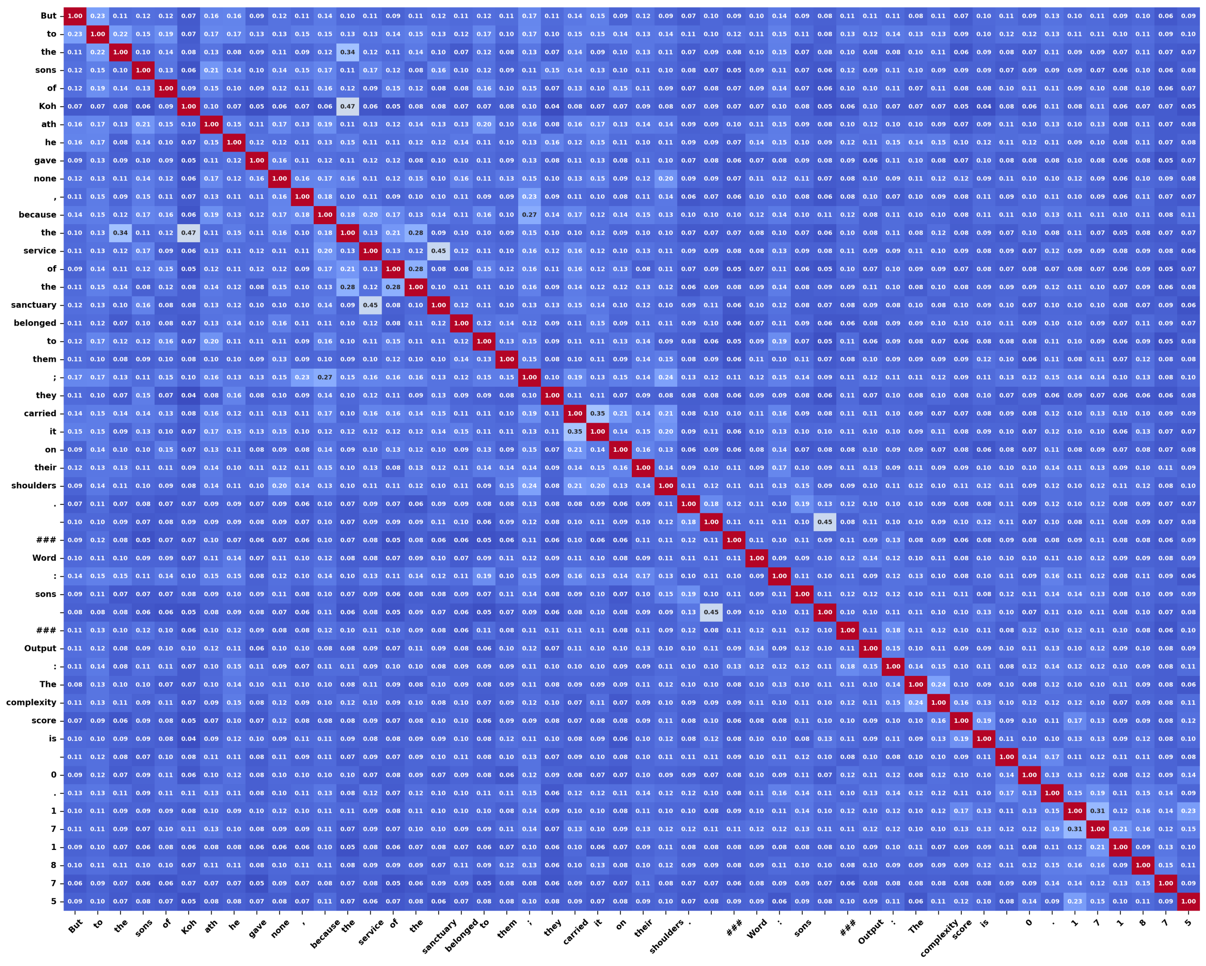

Figure 12: Mutual information (MI) heatmap for regression on the LCP dataset. The model learns fine-grained relationships between input tokens and regression outputs, preserving important task-specific information.

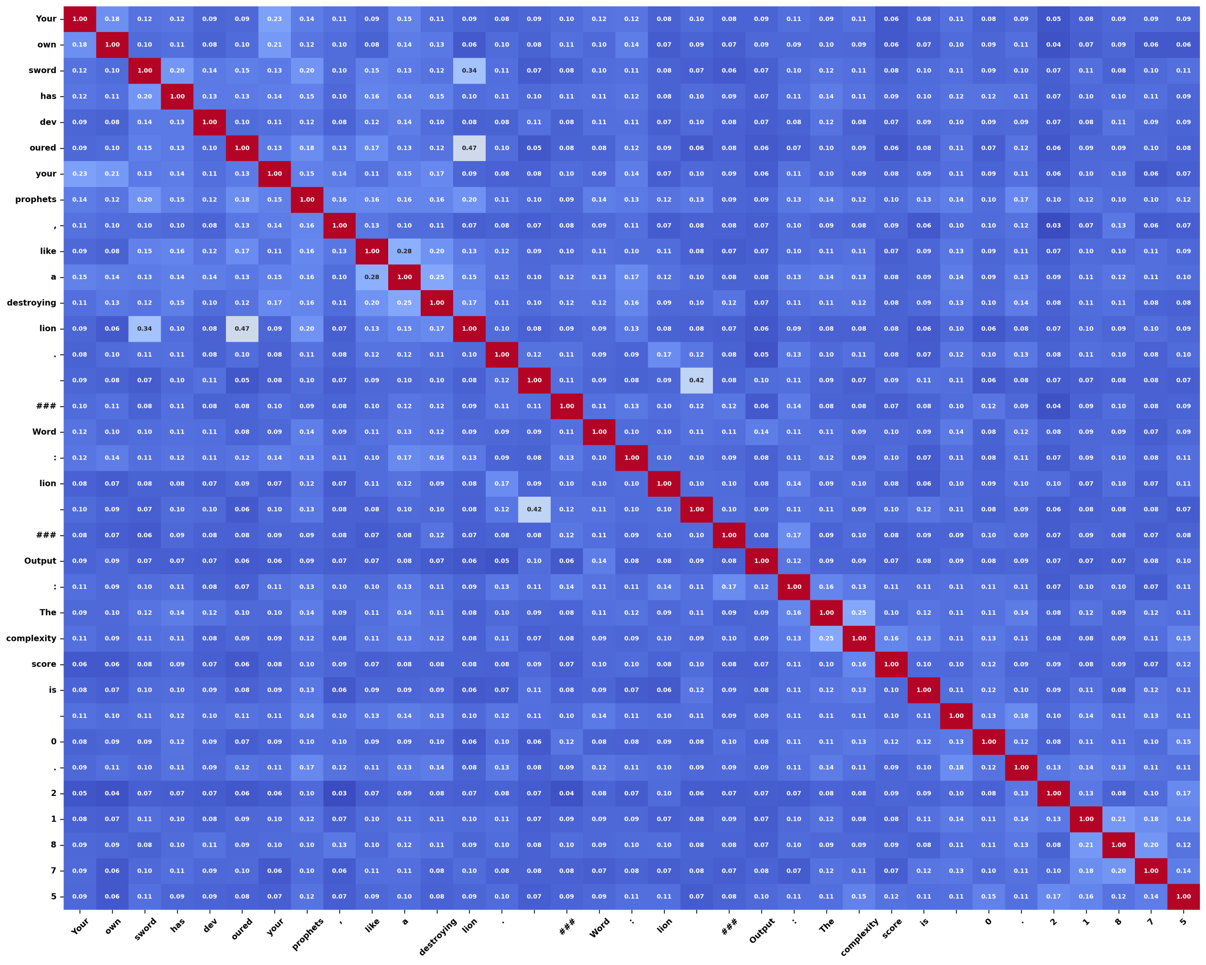

Figure 13: Mutual information (MI) heatmap for regression on the LCP dataset. The results demonstrate that token-level generation retains more mutual information with regression values compared to pooled representations.