RELISH: LLM REgression with a Latent Iterative State Head

Yiheng Su
School of Information
The University of Texas at Austin
sam.su@utexas.edu

Matthew Lease
School of Information
The University of Texas at Austin
ml@utexas.edu

Abstract

We present **RELISH** (**RE**gression with a **L**atent **I**terative **S**tate **H**ead), a novel, lightweight architecture designed for text regression with large language models. Rather than decoding numeric targets as text or aggregating multiple generated outputs, RELISH predicts scalar values directly from frozen LLM representations by iteratively refining a learned latent state through cross-attention over token-level representations, and then mapping the final state to a point estimate with a linear regressor. Across six datasets, four LLM backbones, and two LLM training regimes, RELISH consistently outperforms prior baselines from all three major LLM regression families, including autoregressive decoding, regression-aware inference, and existing predictive head methods. Despite these gains, RELISH remains highly parameter-efficient, requiring only ∼3.4–3.7M trainable parameters across frozen LLM backbones (only 0.01–0.04% additional overhead), far less than LoRA-based alternatives that grow with model size (0.26–0.42%). Our code is available at https://github.com/SamSoup/RELISH.

1 Introduction

In the era of large language models (LLMs), most natural language processing (NLP) tasks are unified under the *text-to-text* paradigm, in which models consume text inputs and produce text outputs (Raffel et al., 2020; Anthropic, 2025; Singh et al., 2025; Yang et al., 2025; Google DeepMind, 2026). Theoretically, this paradigm is appealing because the same pre-trained model, linguistic interface, and decoding procedure can be applied across diverse applications that traditionally required different architectures. Empirically, this paradigm has been particularly successful for intrinsically *generative* tasks such as question answering, summarization, translation, and code generation (Clark et al., 2018; Hendrycks et al., 2020; Chen et al., 2021; Srivastava et al., 2023; Wang et al., 2024b; Kazemi et al., 2025; Li et al., 2025b; Center for AI Safety et al., 2026). Here, the need to produce coherent, contextually appropriate text aligns naturally with the language modeling objectives used to pre-train LLMs, such as autoregressive next-token prediction (Radford et al., 2019; Brown et al., 2020).

However, the *text-to-text* paradigm is less suited to *predictive* tasks such as classification and regression (Lukasik et al., 2025). Unlike generative tasks whose outputs are naturally text, predictive tasks seek discrete labels or continuous scalars. In regression, this mismatch makes text generation suboptimal because predictive quality depends on numerical error, whereas language modeling penalizes token mismatches. For example, if the target is 1.0, a regression objective like MSE prefers 0.9 to 0.1, whereas a language modeling objective like cross-entropy treats them as discrete tokens and remains agnostic to numerical proximity.

Recent work has explored how to adapt LLMs to regression. Three major families have emerged (see Table 1): *autoregressive decoding* (Vacareanu et al., 2024; Song et al., 2024; Song & Bahri, 2025; Akhauri et al., 2025), *regression-aware inference* (Lukasik et al., 2024; 2025; Chiang et al., 2025), and *predictive heads* (Xin et al., 2021; Wang et al., 2022; Zhuang et al., 2023; Fernandes et al., 2023; Zhang & Li, 2024; Tang et al., 2025; Nguyen et al., 2024). Autoregressive methods simply prompt the LLM. Regression-aware methods complement next-token generation with regression-aware decision rules that explicitly minimize numerical error.

Predictive heads avoid text generation entirely by finetuning a predictive head (e.g., a linear layer) attached to the LLM to directly output numerical estimates.

Autoregressive decoding is simplest and most faithful to how LLMs are pre-trained (Vacareanu et al., 2024). Consequently, it is least aligned with the numerical nature of regression targets, since decoding is based on token probabilities rather than numerical proximity. Regression-aware inference seeks to remedy this mismatch by forming predictions from the LLM's output distribution using Bayes-optimal decision rules to directly minimize the loss metric, such as the posterior mean under mean squared error (Lukasik et al., 2024; 2025). However, approximating this posterior often requires sampling multiple outputs or enumerating candidate targets, thereby increasing computation at inference time. Moreover, this approach only partially resolves the mismatch, since the posterior is still defined over textualized numbers (e.g., "3.5") rather than directly over the continuous output space.

Predictive head methods instead bypass text generation at inference time by mapping LLM representations directly to numeric predictions through a task-specific regressor. This enables direct optimization in the continuous output space, yielding efficient single-pass inference. However, existing approaches (Zhuang et al., 2023; Zhang & Li, 2024; Nguyen et al., 2024; Tang et al., 2025) are limited, typically relying on basic pooling heuristics (e.g., mean pooling) over frozen hidden states. As a result, they may fail to fully extract regression-relevant information from LLM representations and thus may underperform regression-aware inference. *Can we match the predictive performance of regression-aware approaches without sacrificing the efficiency of predictive heads?*

In this work, we propose **RE**gression with a **L**atent **I**terative **S**tate **H**ead (**RELISH**), a novel, lightweight yet expressive predictive head architecture for LLM-based regression. Rather than decoding numeric targets as text or aggregating across several generated outputs, RELISH predicts scalar values directly from frozen LLM representations through an iterative refinement mechanism. Concretely, RELISH repeatedly updates a learned latent state via cross-attention over the LLM's token representations, and then maps the final state to a scalar prediction with a linear regressor. Compared with existing predictive heads based on simple pooling heuristics, RELISH provides a more robust summarization mechanism for extracting regression-relevant information without sacrificing inference efficiency.

We evaluate RELISH on six datasets spanning semantic textual similarity (Marelli et al., 2014; Muennighoff et al., 2023), machine translation quality estimation (Specia et al., 2020), and code quality estimation (Song et al., 2026). While the first two are canonical regression tasks, evaluating LLM-generated code has gained prominence given the rise of autonomous agents. In AI4Science (Zheng et al., 2025), recent benchmarks valuably assess if generated code produces correct scientific outputs (Chen et al., 2025; Joseph et al., 2025), but estimating code quality at inference time, necessary for calibrated trust, remains underexplored.

Empirically, RELISH consistently outperforms state-of-the-art regression baselines across the six datasets, four LLM backbones (8–32B), and both frozen and LoRA (Hu et al., 2022) LLM training regimes. Relative to the second-best method, RAFT (Lukasik et al., 2025), RELISH improves Pearson correlation by 10.2%, Spearman correlation by 9.3%, and range-normalized RMSE by 19.7% (Table 3). RELISH is also highly parameter-efficient, requiring only $\sim$3.4–3.7M trainable parameters across frozen LLM backbones (only 0.01–0.04% more overhead), far less than LoRA-based alternatives (Table 4) that grow with model size (0.26–0.42%). **Overall, our key contributions are**:

1. A novel predictive head architecture designed for text regression (§ 3).
2. A unified conceptual (§ 2.2) and empirical (§ 4.2) comparison of autoregressive decoding, regression-aware inference, and predictive head methods across six datasets, four LLM backbones, and both frozen and LoRA fine-tuned regimes, showing that RELISH achieves the strongest overall performance with a substantially smaller trainable footprint than LoRA.

2 Background on LLM-based Regression

In this section, we begin with problem formulation (§ 2.1), then review existing methods for inference (§ 2.2) and training (§ 2.3) (see Table 1). Last, we differentiate RELISH in § 2.4.

Inference family	How to make predictions	Suitable training setups
Autoregressive decoding (Vacareanu et al., 2024)	*Parse from string:* $\hat{y}_{\text{ARD}}(x) = \text{float}(\arg\max_{y \in \mathcal{Y}} p(\text{str}(y) \mid x))$	**PEFT/SFT:** Minimize token-level CE on $\text{str}(y^*)$. $\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{\lvert str(y^*)\rvert} \log p(y_t \mid x, y_{<t})$
Regression-aware inference (RAIL) (Lukasik et al., 2024)	*Bayes-optimal from $p(\cdot \mid x)$:* $\hat{y}_{\text{RAIL}}(x) = \arg\min_{v \in \mathbb{R}} \mathbb{E}_{y \sim p}[\ell(\text{float}(y), v)]$ When $\ell = \mathcal{L}_{\text{MSE}}$, same as posterior mean: $\hat{y}_{\text{RAIL}}(x) \approx \sum_{y \in \mathcal{Y}_{\text{grid}}} y \cdot p(\text{str}(y) \mid x)$	**PEFT/SFT:** (i) Minimize $\mathcal{L}_{\text{CE}}$. (ii) *RAFT* (Lukasik et al., 2025). $\mathcal{L}_{\text{RAFT}} = \left(y^* - \mathbb{E}_{y \sim p}[\text{float}(y)]\right)^2$
Predictive Head (Tang et al., 2025)	*Extract representations, then regress:* $\hat{y}_{\text{HEAD}}(x) = f_{\text{REG}}(\phi(H(x)))$	**Head-only:** train f_{REG} with $\mathcal{L}_{\text{MSE}}$. **PEFT/SFT + head:** fine-tune backbone jointly with f_{REG}.
RELISH (ours)	*Iteratively refine a latent state, then regress:* $r^{(i)} = \text{Refine}\left(r^{(i-1)}, H(x)\right),\ i = 1, \ldots, L$ $\hat{y}_{\text{RELISH}}(x) = w^\top r^{(L)} + b$	**Head-only:** train $r^{(0)}$, w, b, and refinement modules with $\mathcal{L}_{\text{MSE}}$. **PEFT/SFT + head:** optionally fine-tune backbone jointly with RELISH modules.

Table 1: **Taxonomy of methods leveraging LLMs for regression.** We characterize LLM-based regression along two axes: (a) how the model produces a point estimate (*inference family*) and (b) which parameters are trained and how (*training setup*). Autoregressive decoding generates $\text{str}(y)$ and parses it. RAIL computes a Bayes-optimal estimate under $p(\cdot \mid x)$ for loss ℓ (e.g., the posterior mean for $\mathcal{L}_{\text{MSE}}$), where the posterior is estimated via sampling or candidate enumeration over a grid $\mathcal{Y}_{\text{grid}}$. Predictive heads regress from pooled hidden states, whereas RELISH replaces static pooling with iterative latent-state summarization over token-level representations. Typical objectives are $\mathcal{L}_{\text{CE}}$ for autoregressive decoding, $\mathcal{L}_{\text{CE}}$ or $\mathcal{L}_{\text{RAFT}}$ for RAIL, and $\mathcal{L}_{\text{MSE}}$ for predictive-head methods.

2.1 Problem Formulation

Following Lukasik et al. (2025), we study **natural language regression**. Given a dataset $\mathcal{D} = \{(x_i, y_i^*)\}_{i=1}^N$, where each x_i is a textual input and each $y_i^* \in \mathbb{R}$ is a continuous target, the goal is to learn a predictor $\hat{y}(x) \in \mathbb{R}$ that minimizes some regression loss $\mathcal{L}(y^*, \hat{y})$.

LLMs offer a flexible foundation for natural language regression. They can encode rich textual inputs without manual feature engineering, transfer broad semantic knowledge from pre-training, and support a common interface across diverse tasks (Song et al., 2024; Akhauri et al., 2025). However, *there is no consensus on how to best adapt LLMs for regression*.

We categorize prior work by two axes: (a) **how the model produces a numeric prediction**, and (b) **whether (and how) the model is trained**. For (a), three inference families emerge: autoregressive decoding, regression-aware inference, and predictive head methods. For (b), four training regimes emerge: no training (prompt-only), parameter-efficient fine-tuning (PEFT), full supervised fine-tuning (SFT), and training only the predictive heads while freezing the LLM backbone. **Table 1** summarizes this taxonomy, and **Table 2** compares the operational costs of standard LLM regression approaches.

Inference methods and training regimes are largely orthogonal. For instance, the same inference procedure (e.g., predictive heads) can be applied to either a frozen or fine-tuned LLM, while the same training regime (e.g., PEFT) can support different inference procedures. However, some pairings are more natural than others, since certain training objectives (e.g., regression-aware objectives) better support particular inference procedures.

Method	Family	Training	Inference
Zero / Many Shot (Vacareanu et al., 2024)	Autoregressive	None	Medium–High
Linear / MLP (Tang et al., 2025)	Predictive head	Low	Low
RAIL (Lukasik et al., 2024)	Regression-aware	None	High
RAFT (Lukasik et al., 2025)	Regression-aware	High	High
RELISH (ours)	**Predictive head**	**Low**	**Low**

Table 2: **Qualitative operational costs of common LLM regression approaches.** Training costs reflect optimization overhead. Autoregressive methods and RAIL do not update parameters (None), predictive heads and RELISH freeze the LLM backbone and only train auxiliary modules (Low), while RAFT fine-tunes the LLM backbone (High). Inference costs correspond to runtime latency. Predictive heads and RELISH are lightweight (Low), whereas autoregressive methods sequentially decode tokens, so latency scales with the number of in-context demonstrations (Medium–High). RAIL and RAFT approximate model posterior via multiple samples or scoring grid candidates (High).

2.2 Inference Paradigms

Autoregressive decoding (ARD). Autoregressive decoding (Vacareanu et al., 2024; Song et al., 2024; Song & Bahri, 2025; Akhauri et al., 2025) treats scalar targets as text. Given an input x, a decoder LLM produces a point prediction by sequentially generating a numeric string from vocabulary tokens $\mathcal{Y}$ and then parsing it into a real value:

$$\hat{y}_{\text{ARD}}(x) = \text{float}(\arg\max_{y \in \mathcal{Y}} p(\text{str}(y) \mid x))$$

In practice, the most common approach is direct prompting, either zero-shot or with many-shot demonstrations. Thus, ARD requires no task-specific training, although inference overhead scales with the number of in-context demonstrations (Table 2). Additionally, ARD is sensitive to prompt formatting and insensitive to numerical proximity among candidate predictions (Lukasik et al., 2025).

Regression-aware inference (RAIL). RAIL (Lukasik et al., 2024) addresses the representational mismatch by augmenting autoregressive decoding with a loss-aware decision rule over the LLM's predictive distribution. Formally, given an input x, RAIL predicts:

$$\hat{y}_{\text{RAIL}}(x) = \arg\min_{v \in \mathbb{R}} \mathbb{E}_{y \sim p(\cdot|x)} \left[\ell \left(\text{float}(y), v \right) \right].$$

This is equivalent to the Bayes-optimal decision rule for the target loss ℓ (Kumar & Byrne, 2004). For mean squared error, the optimal prediction is the mean of the posterior. In practice, this posterior is approximated either by sampling multiple completions or, when feasible (e.g., ordinal regression), by enumerating a grid of candidate values.

As summarized in Table 2, RAIL thus trades higher inference cost to better align textual predictions with numerical proximity. Lukasik et al. (2024) show that RAIL improves over greedy and self-consistency decoding (Wang et al., 2023) on semantic text similarity and sentiment analysis tasks. Subsequent work further strengthens RAIL through regression-aware fine-tuning (Lukasik et al., 2025), which we discuss in Section 2.3.

Predictive heads. Predictive heads (Fernandes et al., 2023; Zhuang et al., 2023; Nguyen et al., 2024; Tang et al., 2025) avoid text generation by predicting continuous scalars directly from internal LLM representations. Given an input x, let $H(x) \in \mathbb{R}^{S \times d}$ denote the token-level hidden states of the LLM, where S is the sequence length and d is the LLM's hidden dimension. Predictive head methods first summarize token representations into a fixed-size vector, and then apply a downstream regressor:

$$\hat{y}_{\text{HEAD}}(x) = f_{\text{REG}}(\phi(H(x))).$$

Here, ϕ typically represents a static compression heuristic, such as mean-pooling across token sequences (Li et al., 2020) or indexing the representation of a specific token, such as

[CLS] (Reimers & Gurevych, 2019). The regressor f_{REG} varies by implementation. *Vector-based* heads employ an auxiliary module, such as a dense linear layer (Zhuang et al., 2023) or a 2-layer MLP (Tang et al., 2025). *Logit-based* heads repurpose the LLM's language modeling head by extracting the logit of a designated vocabulary token (Fernandes et al., 2023; Zhuang et al., 2023) to emit as predictions.

2.3 Training Regimes

Complementary to the inference mechanism, LLM regression methods also differ in *which parameters are updated* and *which loss objective is used*. As summarized in Table 1, these choices are largely separable from the inference family, though some objectives are more naturally paired with certain prediction rules.

No training. The simplest regime uses the LLM as-is, so all task adaptation occurs through the prompt rather than gradient updates. This regime is most natural for autoregressive decoding, but can also be combined with regression-aware inference by applying a regression-aware decision rule to the output distribution of a frozen model (e.g., RAIL).

Head-only training. Predictive heads typically freeze the LLM backbone and only train auxiliary head parameters. Vector-based heads update the regressor f_{REG}, while logit-based heads update the linear matrix $W \in \mathbb{R}^{d \times |\mathcal{Y}|}$ that projects model hidden states to vocabulary tokens. Since predictive heads emit continuous scalars, they can directly optimize numerical error using classic regression loss functions, such as mean squared error ($\mathcal{L}_{\text{MSE}}$).

LLM backbone adaptation: PEFT and SFT. The LLM backbone can be adapted via parameter-efficient fine-tuning (e.g., LoRA (Hu et al., 2022)) or full supervised fine-tuning. For autoregressive decoding, the model is typically optimized using token-level cross-entropy to reproduce the textualized target string exactly:

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{|str(y^*)|} \log p(y_t \mid x, y_{<t}).$$

While $\mathcal{L}_{\text{CE}}$ does not directly minimize numerical error, large-scale training can lift generation to match or exceed scalar alternatives (Song et al., 2024; Song & Bahri, 2025; Akhauri et al., 2025; 2026). However, these methods rely on massive proprietary datasets (e.g., telemetry logs) to learn numerical continuity and are impractical for most applications.

Regression-aware fine-tuning. For regression-aware inference, one may still fine-tune the LLM with $\mathcal{L}_{\text{CE}}$, but a more natural objective is to directly optimize the same posterior statistic used at runtime. RAFT (Lukasik et al., 2025) operationalizes this by minimizing:

$$\mathcal{L}_{\text{RAFT}} = \Big(y^* - \mathbb{E}_{y \sim p}[\text{float}(y)]\Big)^2,$$

thereby aligning training with the Bayes-optimal estimator used by RAIL. This is computationally more expensive, since the posterior must be approximated during training as well, and inference still requires multiple samples or candidate enumeration (see Table 2).

2.4 How is RELISH different?

Similar to existing predictive heads, RELISH also compresses token representations into a fixed-size vector for regression, preserving high training and inference efficiency. However, prior predictive heads are limited by an *information extraction bottleneck*. Specifically, vector-based methods collapse token representations using task-agnostic heuristics such as mean pooling, which can wash out localized or distributed signals before the downstream regressor sees them. Logit-based heads avoid explicit pooling, but bind the prediction to a designated vocabulary logit at a specific output position. This architecture forces the model to encode all regression-relevant signals into that output position's representation.

Consequently, any localized or distributed cues omitted from this representation cannot be recovered at prediction time.

In contrast, RELISH mitigates this information extraction bottleneck with an iterative refinement mechanism, in which the predictive head repeatedly updates a learned summary of token representations for regression. By repeatedly attending over all token representations, this learned summary can retain localized cues while accumulating distributed signals spread across multiple tokens. Concretely, RELISH maintains a learned latent state $r^{(0)}$ and repeatedly refines r through cross-attention over the token representations $H(x)$:

$$r^{(i)} = \text{Refine}\left(r^{(i-1)}, H(x)\right), \qquad i = 1, \ldots, L,$$

The final prediction is then emitted by a linear regressor:

$$\hat{y}_{\text{RELISH}}(x) = w^\top r^{(L)} + b.$$

By leveraging trainable latent queries rather than static pooling or indexing, RELISH is related in spirit to DeepMind's Perceiver IO (Jaegle et al., 2022) and BLIP-2 architectures (Li et al., 2023). Unlike prior latent-query architectures designed for general encoding or multimodal bridging, RELISH is specifically engineered for regression by iteratively distilling token-level hidden states into a single latent state optimized for scalar prediction. As summarized in Table 2, our design preserves the training and inference efficiency of predictive heads while providing a more robust summarization mechanism.

Beyond efficiency, RELISH offers conceptual advantages over autoregressive and regression-aware methods by directly predicting scalars from LLM representations. Autoregressive methods serialize continuous targets into discrete tokens, but token likelihoods do not naturally encode numerical proximity. Regression-aware methods attempt to mitigate this limitation by estimating Bayes-optimal statistics from the LLM's induced output distribution. Collectively, these methods essentially attempt to calibrate probability mass over serialized numeric candidates. Consequently, predictions may be sensitive to the *verbalization gap* (Yuchi et al., 2026), tokenization artifacts, candidate-set design, and learned semantic associations from pre-training. § B provides additional details on how RELISH differentiates from existing LLM-based regression methods.

3 Method

3.1 Setup

Given an input sequence x of length S, we extract token-level representations from the penultimate layer (see § C.1 for justification) of a frozen decoder-only LLM:

$$H(x) = \text{LLM}_{\text{frozen}}(x) \in \mathbb{R}^{S \times d}.$$

Following Nguyen et al. (2024), Song et al. (2024), and Tang et al. (2025), we apply y-normalization to standardize ground-truth labels (y^*) using training-split statistics (μ_y, σ_y). We train the model to predict an unbounded standardized target:

$$z = \frac{y^* - \mu_y}{\sigma_y + \epsilon}, \qquad \hat{y} = \hat{z}(\sigma_y + \epsilon) + \mu_y,$$

where $\epsilon > 0$ is a small constant for numerical stability.

3.2 Architecture

RELISH mitigates the information extraction bottleneck of existing predictive heads with an iterative refinement mechanism over token-level LLM representations. Rather than collapsing the token sequence into a fixed vector in a single step, RELISH maintains a learned latent state that is repeatedly refined via cross-attention over the input tokens before being projected to the final scalar prediction.

We first project token-level hidden states to a head dimension d_h:

$$X(x) = H(x)W, \qquad W \in \mathbb{R}^{d \times d_h},$$

so that $X(x) \in \mathbb{R}^{S \times d_h}$ serves as the token-level memory. This projection is optional if $d_h = d$. See § C.1 on why we recommend including this step.

Let $r^{(0)} \in \mathbb{R}^{d_h}$ denote a learned latent state. For $i = 1, \ldots, L$, RELISH updates this latent state with Transformer-style residual refinement blocks:

$$\tilde{r}^{(i)} = LN\big(r^{(i-1)} + \mathrm{MHA}\big(r^{(i-1)}, X(x), X(x)\big)\big), \qquad r^{(i)} = LN\big(\tilde{r}^{(i)} + \mathrm{FFN}\big(\tilde{r}^{(i)}\big)\big).$$

Here, LN is a post-layer normalization (Vaswani et al., 2017), FFN is a feed-forward MLP, and $\mathrm{MHA}(\cdot)$ is a multi-head cross-attention with the latent state as query and token-level representations as keys and values. For each attention head m out of M total heads,

$$Q_m = r^{(i-1)} W_m^Q, \qquad K_m = X(x) W_m^K, \qquad V_m = X(x) W_m^V,$$

$$\mathrm{Attn}_m(Q_m, K_m, V_m) = \mathrm{softmax}\left(\frac{Q_m K_m^\top}{\sqrt{d_h / M}}\right) V_m.$$

After L refinement steps, the final state is mapped to an unbounded standardized prediction:

$$\hat{z} = w^\top r^{(L)} + b,$$

where $w \in \mathbb{R}^{d_h}$ and $b \in \mathbb{R}$ are trainable parameters.

Although RELISH adopts components from transformers, RELISH is not simply a stack of L additional transformer layers trained on top of the LLM. A standard transformer layer is a *sequence-processing* module. Queries, keys, and values are all derived from the same token sequence $H(x)$, self-attention updates every token representation, and layer propagation aims to output new token sequences $H'(x)$. In contrast, RELISH is a *summarization* module. A RELISH layer uses the learned latent state $r^{(i-1)}$ as the query, which cross-attends to frozen LLM token representations as the keys and values. Each subsequent refinement layer updates only the latent state $r^{(i)}$ and leaves the token representations unchanged.

3.3 Training Objective

We train all RELISH parameters end-to-end while keeping the LLM backbone frozen, i.e., the projection W, latent state $r^{(0)}$, refinement blocks, and linear weights (w, b). We optimize the Huber loss (Huber, 1992) on standardized targets:

$$\mathcal{L}_{\mathrm{Huber}}(\hat{z}, z) = \begin{cases} \frac{1}{2}(\hat{z} - z)^2, & |\hat{z} - z| \le \delta, \\ \delta\big(|\hat{z} - z| - \frac{1}{2}\delta\big), & \text{otherwise}, \end{cases}$$

where $\delta > 0$ is the Huber threshold.

4 Evaluation

4.1 Experimental Setup

LLM Backbones We employ four LLM backbones: **Llama 3.1 8B Instruct** (Grattafiori et al., 2024), **Qwen 3 8B and 32B** (Yang et al., 2025), and **Gemma 3 27B Instruct** (Team et al., 2025).

Datasets We evaluate on three text regression tasks: semantic textual similarity (STS) (Enevoldsen et al., 2025), machine translation quality estimation (WMT) (Specia et al., 2020), and code quality estimation (Song et al., 2026). For STS, models predict a scalar score representing the degree of semantic equivalence between two sentences. The STS-Benchmark (STS-B) (Enevoldsen et al., 2025) contains English sentence pairs from captions and forums

	Unsupervised			Supervised							Ours
Metric	**Zero**	**Many**	**RAIL**	**Lin.**	**MLP**	**L+Zero**	**L+Lin.**	**L+MLP**	**L+RAIL**	**RAFT**	**RELISH**
$r \uparrow$	$55.0_{\pm 0.3}$	$52.3_{\pm 0.8}$	$56.0_{\pm 0.1}$	$44.4_{\pm 0.0}$	$56.3_{\pm 0.4}$	$57.2_{\pm 0.4}$	$44.9_{\pm 0.0}$	$55.9_{\pm 0.5}$	$61.8_{\pm 0.5}$	$66.1_{\pm 0.3}$	$\mathbf{72.9}_{\pm \mathbf{0.1}}$
$\rho \uparrow$	$55.5_{\pm 0.2}$	$52.6_{\pm 1.0}$	$58.8_{\pm 0.1}$	$42.5_{\pm 0.0}$	$53.7_{\pm 0.4}$	$56.1_{\pm 0.5}$	$43.0_{\pm 0.0}$	$53.4_{\pm 0.3}$	$62.9_{\pm 0.6}$	$65.3_{\pm 0.5}$	$\mathbf{71.4}_{\pm \mathbf{0.3}}$
NRMSE $\downarrow$	$25.1_{\pm 0.0}$	$22.2_{\pm 0.5}$	$25.5_{\pm 0.0}$	$23.1_{\pm 0.0}$	$17.8_{\pm 0.3}$	$18.9_{\pm 0.1}$	$23.5_{\pm 0.2}$	$17.7_{\pm 0.1}$	$17.9_{\pm 0.1}$	$15.7_{\pm 0.2}$	$\mathbf{12.6}_{\pm \mathbf{0.2}}$

Table 3: Predictive performance macro-averaged across six datasets, four LLMs, and three runs for each baseline. We report Pearson (r) and Spearman (ρ) correlations ($\uparrow$ higher is better) and range-Normalized Root Mean Squared Error (NRMSE) ($\downarrow$ lower is better), all scaled to %. L+ denotes LoRA-based fine-tuning. We also report standard deviation ($\pm$).

on a 0–5 scale. The semantic relatedness subset of Sentences Involving Compositional Knowledge (SICK-R) (Marelli et al., 2014) is on a 1–5 scale. Table 5 reports dataset split sizes.

For WMT, models rate how accurately a machine translation preserves the source language's meaning without human references. The 2020 Multilingual Quality Estimation Task 1 (Specia et al., 2020) provides scores on a 0–100 scale for English–Chinese, Russian–English, and Sinhala–English following Wang et al. (2022). For code, models assess the correctness and readability of statistical code. The StatLLM dataset (Song et al., 2026) comprises 207 scientific workflow tasks, for which GPT-3.5 (OpenAI, 2022), GPT-4 (OpenAI, 2023), and Llama 3.1 70B (Grattafiori et al., 2024) generated SAS code with expert ratings on a 0–50 scale.

Metrics We report Pearson (r), Spearman (ρ), and range-normalized root mean squared error (NRMSE) in Table 3. Because our datasets have different gold label ranges, raw RMSEs are not directly comparable. We therefore report NRMSE $= \text{RMSE}/(y_{\max} - y_{\min})$, which normalizes prediction error on the same scale across datasets. Tables 17, 18, 19, 20, 21, and 22 report raw RMSE scores for each individual dataset.

Baselines We compare RELISH against baselines from three LLM-based regression families (Table 2). For autoregressive decoding, we evaluate zero-shot and many-shot (128 demonstrations) prompting. For regression-aware inference, we adopt the best-performing variants from the original studies: sample-based RAIL (Lukasik et al., 2024) and grid-based RAFT (Lukasik et al., 2025). For predictive heads, we compare to linear regression (Lin.) and a two-layer MLP, with the MLP hidden size chosen to match RELISH's trainable parameter count for each LLM backbone. We adopt the Huber loss and y-normalization for predictive head baselines and RELISH only. All baselines are evaluated on both frozen and LoRA fine-tuned LLM backbones, except for LoRA with many-shot prompting, since the benefits of in-context learning often diminish after fine-tuning (Yin et al., 2024; Duan et al., 2024).

We use $M = 8$, $d_h = 256$, and $L = 3$ for RELISH. See § C for additional details on RELISH (§ C.1), LLM backbones (§ C.2), datasets (§ C.3), metrics (§ C.4), and baselines (§ C.5).

4.2 Results: Predictive Performance

Table 3 shows that RELISH consistently outperforms both unsupervised and supervised baselines across four LLM backbones and six datasets. **Figure 1** shows detailed results for Pearson correlation relative to the zero-shot performance of the underlying LLM backbones. Across all evaluated configurations, RELISH consistently achieves the largest positive gains. In contrast, the other baselines are unreliable: they may yield comparable or modest improvements in some cases, but can degrade below zero-shot performance (negative bars).

Full predictive performance tables are reported in Tables 17, 18, 19, 20, 21, and 22 (Appendix). There, we observe that RELISH achieves the highest predictive performance in almost all cases, except for a minor case in STS-B where Spearman is comparable to LoRA + RAIL (Lukasik et al., 2024) for Gemma 3 27B Instruct. See § D for extended discussions.

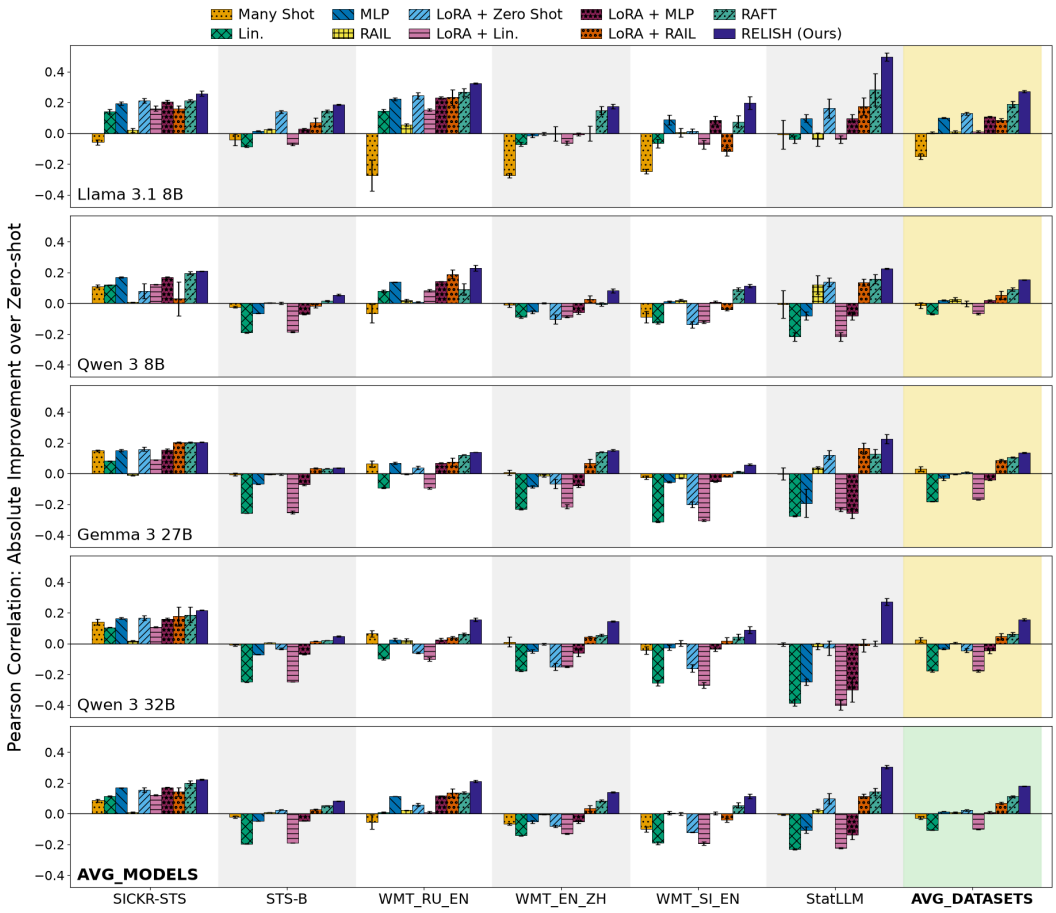

Figure 1: **Detailed results across datasets (columns) and LLMs (rows).** The rightmost column averages over all datasets, and the bottom row averages over LLMs. Bars show absolute improvement in Pearson (r) correlation over the zero-shot baseline (the "ground zero" line). Error bars denote variability across 3 independently-seeded runs. All metrics are computed using gold test labels. RELISH consistently performs best across all settings.

We conduct ablations in § F to validate RELISH's design choices. § F.1 compares RELISH trained with Huber and MSE, showing that gains are primarily architectural rather than loss-dependent and that RELISH still outperforms RAFT under squared-loss settings. § F.2 ablates refinement depth L, suggesting that attention pooling alone ($L = 1$) is strong but iterative updates ($L > 1$) can improve predictive performance. § F.3 and § F.4 investigate the data efficiency of RELISH compared to other baselines. Results show that while RAFT performs best under scarce data regimes (< 256 examples), RELISH scales better with moderate-to-large supervision (> 512 examples) while being substantially faster to train than RAFT. § F.5 indicates that RELISH adds negligible inference overhead relative to simple predictive heads ($\sim$1% slower), while being much faster than autoregressive and regression-aware baselines. Finally, § F.6 shows that input-dependent warm starts for $r^{(0)}$, such as last-token or mean-pooling, do not consistently lead to better predictive performance or faster convergence over the default random initialization. We discuss limitations in § G.

4.3 Results: Training Footprint

RELISH trains a small set of modules atop a frozen LLM backbone, yielding a small trainable footprint. **Table 4** shows that **RELISH consistently requires only $\sim$3.4–3.7M trainable parameters across all LLM backbones (0.01–0.04%)**, whereas LoRA's trainable budget (and

Method	Llama 3.1 8B	Qwen 3 8B	Gemma 3 27B	Qwen 3 32B
LoRA / RAFT	21.0M (0.26%)	21.8M (0.27%)	116.5M (0.42%)	134.2M (0.41%)
RELISH	**3.4M (0.04%)**	**3.4M (0.04%)**	**3.7M (0.01%)**	**3.7M (0.01%)**

Table 4: **How trainable parameter sizes vary for different LLM backbones.** We report the total trainable parameters and their % relative to the underlying LLM. RAFT applies a regression-aware loss to LoRA adapters, sharing the same trainable footprint as LoRA with cross-entropy loss. While LoRA's budgets scale with model size, RELISH remains essentially invariant to the backbone, scaling instead with the hidden dimension (Table 7).

by extension RAFT's, which utilizes the same adapter architecture) grows with model size (0.26–0.42%). Moreover, **this gap widens as LLM backbones scale.** Moving from 8B-class models to 27B–32B models increases LoRA parameters substantially, but RELISH remains nearly unchanged. Holding other parameters constant, **RELISH's trainable components scale primarily with the backbone hidden dimension** (e.g., 4096 vs. 5120 units), rather than the total number of model parameters. See Table 7 and § E for details.

5 Discussion

5.1 Broader Applications

RELISH may also benefit a broader set of use cases, such as reward modeling (RM) (Ouyang et al., 2022; Wang et al., 2024a; Xu et al., 2024; Liu et al., 2024), LLM-as-a-judge (LLMaaJ) (Zheng et al., 2023; Li et al., 2025a; Chiang et al., 2025), and confidence estimation (Kadavath et al., 2022; Tian et al., 2023; Xiong et al., 2024; Steyvers et al., 2025). For example, TRACT (Chiang et al., 2025) fine-tunes LLMaaJ to output rubric-based scalar scores alongside chain-of-thought rationales under a regression-aware objective. See § G for discussion of how RELISH could be extended with chain-of-thought (CoT) rationales.

However, the supervised training underlying RELISH may represent a practical barrier for adoption in some settings. Many *reference-free* LLMaaJ systems leverage black-box models that rely solely on prompt steering rather than supervised training (Li et al., 2024; 2025a). Likewise, many confidence estimation methods obtain scores directly from black-box models through verbalization without supervision (Kadavath et al., 2022; Tian et al., 2023; Xiong et al., 2024; Steyvers et al., 2025). RM differs more fundamentally: its scalar rewards are typically learned from pairwise preference data and are only meaningful for comparing responses under the same prompt (Ouyang et al., 2022; Wang et al., 2024a). We thus view these applications as potential extensions to this work.

5.2 Point Prediction vs. Distribution Elicitation

This work focuses on natural language regression, where the goal is to make a point estimate given textual inputs. A complementary line of work treats LLMs as probabilistic predictors and studies how to best elicit *distributions* for uncertainty quantification (Requeima et al., 2024; Vedula et al., 2025; Wang et al., 2025; Hsu et al., 2026; Kambhatla et al., 2026; Piskorz et al., 2026). While conceptually related, how best to elicit distributions from LLMs is beyond the scope of this work. Nonetheless, RELISH could yield distributions by replacing the linear regressor with quantile heads trained with pinball loss.

6 Conclusion

We proposed RELISH, a novel, lightweight architecture for LLM-based natural language regression that iteratively refines a learned latent state via cross-attention. Across six datasets and four LLM backbones, RELISH consistently outperforms prior state-of-the-art regression baselines. In addition, RELISH requires only ∼3.4–3.7M trainable parameters (0.01–0.04% additional overhead) and avoids LoRA scaling bottlenecks. Promising applications include uncertainty quantification, reward modeling, and LLM-as-a-judge tasks.

Acknowledgments

We appreciate the valuable feedback we received on this manuscript from our anonymous reviewers, Junyi Jessy Li, Leqi Liu, and members of the Artificial Intelligence and Human-Centered Computing (AI&HCC) lab at UT Austin. This work was supported by the NSF under Cooperative Agreement 2421782 and the Simons Foundation grant MPS-AI-00010515, awarded to the NSF-Simons AI Institute for Cosmic Origins (CosmicAI)[1]. This research was also supported by computational resources provided by the Texas Advanced Computing Center (TACC) at The University of Texas at Austin. The statements made herein are solely the opinions of the authors and do not reflect the views of the sponsoring agencies.

References

Yash Akhauri, Bryan Lewandowski, Cheng-Hsi Lin, Adrian N Reyes, Grant C Forbes, Arissa Wongpanich, Bangding Yang, Mohamed S Abdelfattah, Sagi Perel, and Xingyou Song. Performance prediction for large systems via text-to-text regression. *arXiv [cs.LG]*, June 2025.

Yash Akhauri, Xingyou Song, Arissa Wongpanich, Bryan Lewandowski, and Mohamed S. Abdelfattah. Regression language models for code. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=utTapVWtc7.

AI Anthropic. System card: Claude opus 4 & claude sonnet 4. *Claude-4 Model Card*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and Others. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.*, 33:1877–1901, 2020.

Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649(8099):1139–1146, January 2026.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In Steven Bethard, Marine Carpuat, Marianna Apidianaki, Saif M. Mohammad, Daniel Cer, and David Jurgens (eds.), *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001/.

David Chen and William Dolan. Collecting highly parallel data for paraphrase evaluation. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea (eds.), *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 190–200, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL https://aclanthology.org/P11-1020/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv [cs.LG]*, July 2021.

[1] https://www.cosmicai.org/

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 96934–96990, 2025.

Cheng-Han Chiang, Hung-Yi Lee, and Michal Lukasik. TRACT: Regression-aware fine-tuning meets chain-of-thought reasoning for LLM-as-a-judge. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2934–2952, Stroudsburg, PA, USA, 2025. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv [cs.AI]*, March 2018.

Róbert Csordás, Christopher D Manning, and Chris Potts. Do language models use their depth efficiently? In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 160313–160362. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/eae3af0f5868f0a2eceb74208966d55b-Paper-Conference.pdf.

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

Hanyu Duan, Yixuan Tang, Yi Yang, Ahmed Abbasi, and Kar Yan Tam. Exploring the relationship between in-context learning and instruction tuning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 3197–3210, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.182. URL https://aclanthology.org/2024.findings-emnlp.182/.

Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Diganta Misra, Shreeya Dhakal, Jonathan Rystrøm, Roman Solomatin, Ömer Veysel Çağatan, Akash Kundu, Martin Bernstorff, Shitao Xiao, Akshita Sukhlecha, Bhavish Pahwa, Rafał Poświata, Kranthi Kiran GV, Shawon Ashraf, Daniel Auras, Björn Plüster, Jan Philipp Harries, Loïc Magne, Isabelle Mohr, Dawei Zhu, Hippolyte Gisserot-Boukhlef, Tom Aarsen, Jan Kostkan, Konrad Wojtasik, Taemin Lee, Marek Suppa, Crystina Zhang, Roberta Rocca, Mohammed Hamdy, Andrianos Michail, John Yang, Manuel Faysse, Aleksei Vatolin, Nandan Thakur, Manan Dey, Dipam Vasani, Pranjal A Chitale, Simone Tedeschi, Nguyen Tai, Artem Snegirev, Mariya Hendriksen, Michael Günther, Mengzhou Xia, Weijia Shi, Xing Han Lù, Jordan Clive, Gayatri K, Maksimova Anna, Silvan Wehrli, Maria Tikhonova, Henil Shalin Panchal, Aleksandr Abramov, Malte Ostendorff, Zheng Liu, Simon Clematide, Lester James Validad Miranda, Alena Fenogenova, Guangyu Song, Ruqiya Bin Safi, Wen-Ding Li, Alessia Borghini, Federico Cassano, Lasse Hansen, Sara Hooker, Chenghao Xiao, Vaibhav Adlakha, Orion Weller, Siva Reddy, and Niklas Muennighoff. MMTEB: Massive multilingual text embedding benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zl3pfz4VCV.

Patrick Fernandes, Daniel Deutsch, Mara Finkelstein, Parker Riley, André Martins, Graham Neubig, Ankush Garg, Jonathan Clark, Markus Freitag, and Orhan Firat. The devil is in the errors: Leveraging large language models for fine-grained machine translation evaluation. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz (eds.),

Proceedings of the Eighth Conference on Machine Translation, pp. 1066–1083, Stroudsburg, PA, USA, 2023. Association for Computational Linguistics.

Sanjana Gautam, Houjiang Liu, Yujin Choi, and Matthew Lease. How researchers navigate accountability, transparency, and trust when using ai tools in early-stage research: A think-aloud study. In *Proceedings of the 2026 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '26, pp. 1494–1512, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725968. doi: 10.1145/3805689.3812262.

Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 946–958, Online, 2020. Association for Computational Linguistics.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

Google DeepMind. Gemini 3.1 pro model card. Technical report, Google DeepMind, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/. Accessed: 2026-03-10.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Francisco Guzmán, Peng-Jen Chen, Myle Ott, Juan Pino, Guillaume Lample, Philipp Koehn, Vishrav Chaudhary, and Marc'Aurelio Ranzato. The flores evaluation datasets for low-resource machine translation: Nepali–english and sinhala–english. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 6098–6111, 2019.

Lifeng Han, Alan Smeaton, and Gareth Jones. Translation quality assessment: A brief survey on manual and automatic methods. In *Proceedings for the first workshop on modelling translation: Translatology in the digital age*, pp. 15–33, 2021.

Sai Hao, Hao Zeng, Hongxin Wei, and Bingyi Jing. RACER: Risk-aware calibrated efficient routing for large language models. In *Forty-third International Conference on Machine Learning*, 2026.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, October 2020.

Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

Chi-Yang Hsu, Alexander Braylan, Yiheng Su, Matthew Lease, and Omar Alonso. PIE: Performance interval estimation for free-form generation tasks. *arXiv [cs.CL]*, January 2026.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1068, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.67. URL https://aclanthology.org/2023.emnlp-main.67/.

Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pp. 492–518. Springer, 1992.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J Henaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=fILj7WpI-g.

Sebastian Joseph, Syed M. Husain, Stella Offner, Stéphanie Juneau, Paul Torrey, Adam Bolton, Juan Farias, Niall Gaffney, Greg Durrett, and Junyi Jessy Li. Astrovisbench: A code benchmark for scientific computing and visualization in astronomy. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *arXiv [cs.CL]*, July 2022.

Gauri Kambhatla, Sanjana Gautam, Angela Zhang, Alexander Liu, Ravi Srinivasan, Junyi Jessy Li, and Matthew Lease. Improving the distributional alignment of LLMs using supervision. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34277–34305, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1583. URL https://aclanthology.org/2026.acl-long.1583/.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V Le, and Orhan Firat. BIG-bench extra hard. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26473–26501, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1285. URL https://aclanthology.org/2025.acl-long.1285/.

Shankar Kumar and William Byrne. Minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pp. 169–176, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics. URL https://aclanthology.org/N04-1022/.

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 9119–9130, 2020.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From generation to judgment: Opportunities and challenges of LLM-as-a-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2757–2791, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.138. URL https://aclanthology.org/2025.emnlp-main.138/.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19730–19742. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/li23q.html.

Long-Fei Li et al. Provably efficient online RLHF with one-pass reward modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 166987–167016, 2026.

Xiangyang Li, Xiaopeng Li, Kuicai Dong, Zhangquanhu, Rongju Ruan, Xinyi Dai, Yasheng Wang, and Ruiming Tang. Humanity's last code exam: Can advanced LLMs conquer human's hardest code competition? In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 21122–21137, Stroudsburg, PA, USA, 2025b. Association for Computational Linguistics.

Houjiang Liu, Yujin Choi, Sanjana Gautam, Gabriel Jaffe, Soo Young Rieh, and Matthew Lease. Who owns creativity and who does the work? trade-offs in llm-supported research ideation. *arXiv preprint arXiv:2601.12152*, 2026.

Tianqi Liu, Wei Xiong, Jie Ren, Lichang Chen, Junru Wu, Rishabh Joshi, Yang Gao, Jiaming Shen, Zhen Qin, Tianhe Yu, Daniel Sohn, Anastasia Makarova, Jeremiah Zhe Liu, Yuan Liu, Bilal Piot, Abe Ittycheriah, Aviral Kumar, and Mohammad Saleh. RRM: Robust reward model training mitigates reward hacking. In *The Thirteenth International Conference on Learning Representations*, October 2024.

Michal Lukasik, Harikrishna Narasimhan, Aditya Krishna Menon, Felix Yu, and Sanjiv Kumar. Regression aware inference with LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13667–13678, Stroudsburg, PA, USA, November 2024. Association for Computational Linguistics.

Michal Lukasik, Zhao Meng, Harikrishna Narasimhan, Yin-Wen Chang, Aditya Krishna Menon, Felix Yu, and Sanjiv Kumar. Better autoregressive regression with LLMs via regression-aware fine-tuning. *International Conference on Learning Representations*, 2025: 69871–69901, May 2025.

Fengbo Ma, Zixin Rao, Xiaoting Li, Zhetao Chen, Hongyue Sun, Yiping Zhao, Xianyan Chen, and Zhen Xiang. IntrAgent: An LLM agent for content-grounded information retrieval through literature review. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 674–715, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.29. URL https://aclanthology.org/2026.acl-long.29/.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pp. 216–223, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL https://aclanthology.org/L14-1314/.

Yuchun Miao et al. The energy loss phenomenon in RLHF: A new perspective on mitigating reward hacking. In *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, 2025.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In Andreas Vlachos and Isabelle Augenstein (eds.), *Proceedings of*

the 17th Conference of the European Chapter of the Association for Computational Linguistics, pp. 2014–2037, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.148. URL https://aclanthology.org/2023.eacl-main.148/.

Tung Nguyen, Qiuyi Zhang, Bangding Yang, Chansoo Lee, Jorg Bornschein, Sagi Perel, Yutian Chen, and Xingyou Song. Predicting from strings: Language model embeddings for bayesian optimization, 2024. URL https://openreview.net/forum?id=L5nW2DxI5h.

OpenAI. Introducing ChatGPT. https://openai.com/blog/chatgpt, 2022. Accessed: 2026-07-03.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. URL https://arxiv.org/abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Julianna Piskorz, Kasia Kobalczyk, and Mihaela van der Schaar. Eliciting numerical predictive distributions of LLMs without auto-regression. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=SsuBd46twl.

Jingyuan Qi, Zian Jia, Minqian Liu, Wangzhi Zhan, Junkai Zhang, Xiaofei Wen, Jingru Gan, Jianpeng Chen, Qin Liu, Mingyu Derek Ma, Bangzheng Li, Haohui Wang, Adithya Kulkarni, Muhao Chen, Dawei Zhou, Ling Li, Wei Wang, and Lifu Huang. MetaScientist: A human-AI synergistic framework for automated mechanical metamaterial design. In Nouha Dziri, Sean (Xiang) Ren, and Shizhe Diao (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations)*, pp. 404–436, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-191-9. doi: 10.18653/v1/2025.naacl-demo.34. URL https://aclanthology.org/2025.naacl-demo.34/.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

James Requeima, John F Bronskill, Dami Choi, Richard E Turner, and David Duvenaud. LLM processes: Numerical predictive distributions conditioned on natural language. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, November 2024.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, 2016. Association for Computational Linguistics.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=WGXb7UdvTX.

Xingyou Song and Dara Bahri. Decoding-based regression. *Transact. Mach. Learn. Res.*, May 2025.

Xingyou Song, Oscar Li, Chansoo Lee, Bangding Yang, Daiyi Peng, Sagi Perel, and Yutian Chen. OmniPred: Language models as universal regressors. *Transact. Mach. Learn. Res.*, December 2024.

Xinyi Song, Lina Lee, Kexin Xie, Xueying Liu, Xinwei Deng, and Yili Hong. StatLLM: A dataset for evaluating the performance of large language models in statistical analysis. *Scientific Data*, 13(1):369, feb 2026. ISSN 2052-4463. doi: 10.1038/s41597-026-06731-4. URL https://doi.org/10.1038/s41597-026-06731-4.

Lucia Specia, Frédéric Blain, Marina Fomicheva, Erick Fonseca, Vishrav Chaudhary, Francisco Guzmán, and André F. T. Martins. Findings of the WMT 2020 shared task on quality estimation. In Loïc Barrault, Ondřej Bojar, Fethi Bougares, Rajen Chatterjee, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Alexander Fraser, Yvette Graham, Paco Guzman, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, André Martins, Makoto Morishita, Christof Monz, Masaaki Nagata, Toshiaki Nakazawa, and Matteo Negri (eds.), *Proceedings of the Fifth Conference on Machine Translation*, pp. 743–764, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.wmt-1.79. URL https://aclanthology.org/2020.wmt-1.79/.

Aarohi Srivastava et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj. Featured Certification.

M Steyvers, Heliodoro Tejeda, Aakriti Kumar, Catarina Belém, Sheer Karny, Xinyue Hu, Lukas Mayer, and P Smyth. What large language models know and what people think they know. *Nature Machine Intelligence*, pp. 1–11, January 2025.

Eric Tang, Bangding Yang, and Xingyou Song. Understanding LLM embeddings for regression. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=Wt6Iz5XNIO.

Gemma Team et al. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher Manning. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5433–5442, Stroudsburg, PA, USA, 2023. Association for Computational Linguistics.

Robert Vacareanu, Vlad Andrei Negru, Vasile Suciu, and Mihai Surdeanu. From words to numbers: Your large language model is secretly a capable regressor when given in-context examples. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=LzpaUxcNFK.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Nikhita Vedula, Dushyanta Dhyani, Laleh Jalali, Boris N Oreshkin, Mohsen Bayati, and Shervin Malmasi. Quantile regression with large language models for price prediction. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 12396–12415, Stroudsburg, PA, USA, 2025. Association for Computational Linguistics.

Mauricio Villavicencio, Sitong Pan, and Qianwen Wang. Not all uncertainty is equal: How uncertainty granularity shapes human verification in llm-assisted decision making. In *Proceedings of the 2026 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '26, pp. 3804–3857, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725968. doi: 10.1145/3805689.3812329.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5307–5315, Hong Kong, China, 2019. Association for Computational Linguistics.

Binghai Wang, Rui Zheng, Lu Chen, Yan Liu, Shihan Dou, Caishuang Huang, Wei Shen, Senjie Jin, Enyu Zhou, Chenyu Shi, et al. Secrets of rlhf in large language models part ii: Reward modeling. *arXiv preprint arXiv:2401.06080*, 2024a.

Victor Wang, Michael J Q Zhang, and Eunsol Choi. Improving LLM-as-a-judge inference with the judgment distribution. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23173–23199, Stroudsburg, PA, USA, November 2025. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024b. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

Yuxia Wang, Daniel Beck, Timothy Baldwin, and Karin Verspoor. Uncertainty estimation and reduction of pre-trained models for text regression. *Trans. Assoc. Comput. Linguist.*, 10: 680–696, June 2022.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. Berxit: Early exiting for bert with better fine-tuning and extension to regression. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, pp. 91–104, 2021.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gjeQKFxFpZ.

Dehong Xu, Liang Qiu, Minseok Kim, Faisal Ladhak, and Jaeyoung Do. Aligning large language models via fine-grained supervision. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 673–680, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-short.62. URL https://aclanthology.org/2024.acl-short.62/.

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*, 2025.

Qingyu Yin, Xuzheng He, Chak Tou Leong, Fan Wang, Yanzhao Yan, Xiaoyu Shen, and Qiang Zhang. Deeper insights without updates: The power of in-context learning over fine-tuning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4138–4151, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.239. URL https://aclanthology.org/2024.findings-emnlp.239/.

Fengting Yuchi, Li Du, and Jason Eisner. LLMs know more about numbers than they can say. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics: Human Language Technologies (EACL)*, Rabat, Morocco, March 2026. Oral presentation.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. In *The First Conference on Language Modeling*, 2024.

Bowen Zhang and Chunping Li. Advancing semantic textual similarity modeling: A regression framework with translated ReLU and smooth K2 loss. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11882–11893, Stroudsburg, PA, USA, November 2024. Association for Computational Linguistics.

Yang Zhang, Yanfei Dong, and Kenji Kawaguchi. Investigating layer importance in large language models. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 469–479, Stroudsburg, PA, USA, November 2024. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Tianshi Zheng, Zheye Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17733–17750, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.895. URL https://aclanthology.org/2025.emnlp-main.895/.

Banghua Zhu, Michael Jordan, and Jiantao Jiao. Principled reinforcement learning with human feedback from pairwise or k-wise comparisons. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 2023.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. RankT5: Fine-tuning T5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2308–2313, New York, NY, USA, July 2023. ACM.

A Appendix Organization

We organize the Appendix as follows:

- **Extended Background (§ B)** expands related works (§ 2) on clarifying the operational costs of sample- and grid-based RAIL/RAFT (§ B.1), conceptual limitations of RAFT (§ B.2), and why RELISH is preferable to [CLS] pooling (§ B.3).
- **Experimental Setup (§ C)** complements § 4.1 with granular details on RELISH (§ C.1), LLM backbones (§ C.2), datasets (§ C.3), metrics (§ C.4), and baselines (§ C.5).
- **Predictive Performance (§ D)** expands on the main results (§ 4.2) by detailing the metrics aggregation pipeline (§ D.1), providing additional tables and figures (§ D.2), and analyzing both high-level (§ D.3) and dataset-specific (§ D.4) patterns across STS, WMT, and code quality estimation.
- **Training Footprint (§ E)** expands on the parameter efficiency and training footprint discussion from § 4.3.
- **Ablation Studies (§ F)** complements the main predictive performance results (§ 4.2) by comparing RELISH trained with Huber versus MSE loss (§ F.1), examining the model's sensitivity to the refinement depth L (§ F.2), and evaluating supervised-data scaling (§ F.3), training runtime (§ F.4), inference wall-clock throughput (§ F.5), and warm-start initialization (§ F.6).
- **Limitations (§ G)** concludes with a discussion of the method's limitations and avenues for future work, including CoT (§ G) and RAFT-style hybridization (§ G) as future extensions.

B Expanded Related Work and Conceptual Comparisons

In this section, we provide additional details on how RELISH differentiates from prior LLM-based regression approaches. We first clarify the operational costs of sample- and grid-based RAIL/RAFT (§ B.1), then discuss conceptual limitations of RAFT (§ B.2) and why RELISH is preferable to [CLS] pooling (§ B.3).

B.1 Operational Costs of Sample- and Grid-based RAIL/RAFT

To compute Bayes-optimal statistics, regression-aware approaches such as RAIL (Lukasik et al., 2024) and RAFT (Lukasik et al., 2025) must estimate the posterior of the LLM's induced output distribution through *multiple samples* or *enumerating a grid of candidates* (e.g., '1', '2', '3', '4', '5'). It follows that the operational costs of sample-based variants scale linearly with the number of samples, and are generally more expensive compared to one pass autoregressive methods and predictive heads.

For grid-based RAIL and RAFT, Lukasik et al. (2025) demonstrate (see their Table 2) that when grid candidates are single tokens (e.g., '0' or '1'), the computation simplifies to extracting the next-token probability, implying that inference overhead may be minimal for ordinal regression contexts. However, our setting uses continuous targets serialized to three decimal places (e.g., '1.250'), which are typically multi-token strings. Here, the probability of generating a multi-token candidate must be computed autoregressively as the product of conditional token probabilities:

$$p(\mathrm{str}(y) \mid x) = \prod_{j=1}^{|\mathrm{str}(y)|} p(t_j \mid x, t_{<j}).$$

It follows that the operational cost of grid-based variants scales both with the grid size and tokenized length of the numeric string targets. Since RAFT also fine-tunes the LLM backbone, these per-candidate probabilities must also remain in the computational graph for backpropagation, thereby imposing additional memory usage. In contrast, RELISH directly predicts a scalar from a single forward pass using a lightweight predictive head.

B.2 Why is RELISH Conceptually Preferable to RAFT?

Beyond efficiency, RELISH offers conceptual advantages over RAFT (Lukasik et al., 2025) by directly mapping text to scalars, whereas RAFT indirectly derives a prediction from probability distributions over textualized scalars. Specifically, RAFT computes a weighted sum over a grid of numeric string candidates. Thus, RAFT is structurally constrained by the modality mismatch between tokens and scalars. This is problematic for several reasons.

First, RAFT is prone to the **verbalization gap**, where LLMs internally encode numerical magnitudes with high precision but fail to express this knowledge through generated tokens (Yuchi et al., 2026). Abstractly, RAFT can be viewed as calibrating the LLM's ability to assign precise relative probability mass (logits) to textualized numeric targets in order to "triangulate" an accurate weighted sum. Thus, if the model's latent numerical reasoning is not perfectly translated to verbalized logits, then the resulting expectation becomes inherently noisy.

As an example of the verbalization gap, RAFT is prone to **probability leakage**. As noted in Equation 10 of Lukasik et al. (2025), RAFT derives weights from the probability distribution over the *entire* vocabulary, not just the grid candidates. Thus, if the LLM is uncertain, with probability mass distributed across both candidate and non-candidate tokens, the total weights for the grid targets may not sum to 1. This means that the weighted sum can artificially deflate toward zero, potentially biasing lower-magnitude predictions. Consequently, RAFT may conflate numerical magnitude with model uncertainty, since it is hard to discern whether a low-value output is genuine or merely a sign of probability leakage in verbalization.

Second, this possibility for probability leakage can lead to *asymmetrical bounding and inference instability*. While Lemma 3 in Lukasik et al. (2025) proves that the model can represent any target in the grid interval $[a, b]$ via a convex combination, this is not guaranteed under leakage, when the grid probabilities sum to less than 1. Effectively, RAFT may emit a prediction below the lower endpoint through verbalization failure rather than genuine extrapolation. Furthermore, RAFT is mathematically constrained to predict values no greater than the upper endpoint of the grid. Thus, RAFT cannot generalize to out-of-distribution magnitudes or unbounded continuous targets.

Trying to mitigate these generalization limitations is non-trivial, since simply increasing the grid range can introduce instability. Per Lemma 2 in Lukasik et al. (2025), a small ϵ-shift in the probability distribution can result in a lower-bound prediction error of $\geq \frac{b}{4}^2\epsilon$. Consequently, for large grid candidates, minor miscalibrations at inference time can drastically skew the final prediction.

Third, the verbalization gap is exacerbated by *tokenization fragility*. As shown by Wallace et al. (2019) and Geva et al. (2020), transformers often employ subword tokenizers such as BPE variants (Sennrich et al., 2016) that fracture numbers into token fragments, disrupting numerical continuity. Since RAFT requires computing the conditional probability of generating textualized numeric targets, the method may be sensitive to superficial formatting. For instance, the probability of generating "1" versus "1.000" might differ substantially, since the LLM must aggregate probability mass across multiple fragments to generate the latter. Thus, calibrating logits for multi-token grid candidates can be unstable.

Finally, RAFT may be brittle due to *semantic entanglement* from pre-training, which fine-tuning may not fully overcome. For instance, how an LLM encodes "5" is highly contextual (e.g., "5 o'clock" versus "5%"). As a result, token representations may be pulled in multiple semantic directions during fine-tuning. This is supported by the performance degradation observed in Lukasik et al. (2025) when numeric grid candidates are replaced with their date equivalents (e.g., using "January" for "1"). Forcing the LLM to represent grid candidates purely as continuous ordinal values can therefore make the resulting probability distribution highly brittle.

RELISH bypasses the above limitations by avoiding token probabilities entirely. Instead, RELISH directly maps LLM hidden states to a continuous scalar, thereby avoiding the verbalization gap and probability leakage. Likewise, RELISH avoids tokenization fragility

and semantic entanglement by avoiding textualized numeric targets altogether. While RAFT is mathematically bounded by its grid candidates and potentially fragile for wide intervals, RELISH predicts an unbounded standardized scalar through a continuous predictive head.

B.3 Why Prefer RELISH over [CLS] Pooling?

RELISH is preferable over [CLS] pooling for two reasons. First, many modern LLMs are decoder-only and thus lack a native [CLS] token. In encoder-only models like BERT, [CLS] is explicitly learned during pre-training, since the model uses the [CLS] token representation to predict whether two input segments are consecutive sentences (Devlin et al., 2019). In contrast, decoder-only LLMs are trained to generate tokens autoregressively. Therefore, they do not need to learn an analogous [CLS]-like token, since there is no sequence-level pre-training objective that uses it.

Second, RELISH avoids any structural overhead introduced by [CLS] pooling. Specifically, a [CLS]-like solution for decoder-only LLMs would require either adding a new vocabulary token or selecting a special unused token, if any, for further training, so that this token learns how to attend to other tokens. If the token is new, the tokenizer and embedding layers would also need to be updated.

In contrast, RELISH offers a self-contained module that separates representation learning from predictive forecasting. RELISH does not modify the LLM backbone in any way. Instead, RELISH learns a standalone predictive head that operates on frozen LLM token representations to make accurate predictions. This modularity confines task-specific fine-tuning to the predictive head and allows the same architectural design to generalize to different model backbones.

For decoder-only LLMs, a cheap [CLS]-like solution might be to use the representation of the last token, such as [EOS]. We do not benchmark against this approach because Tang et al. (2025) show that predictive heads using last-token representations perform worse than those using mean pooling. Lukasik et al. (2025) similarly show that RAFT outperforms [CLS] pooling predictive heads. Since RELISH outperforms RAFT in our experiments, we focus our comparisons on stronger baselines rather than ablating pooling variants.

C Expanded Experimental Setup

In this section, we provide further details of the experimental setup discussed in § 4.1 on RELISH (§ C.1), LLM backbones (§ C.2), datasets (§ C.3), metrics (§ C.4) and baselines (§ C.5).

To ensure statistical robustness and reproducibility of our empirical results, we conduct three independent runs of experiments using three different random seeds (42, 1234, and 2026) to initialize pseudo-random number generation. We then report aggregated metrics across trials to account for this natural variance. Table 6 summarizes all hyperparameters.

C.1 RELISH

For RELISH, we keep the LLM backbone frozen and train only the projection matrix $W \in \mathbb{R}^{d \times d_h}$, the latent state $r^{(0)}$, the attention module, and the linear output linear. We use a learning rate of $1e-4$, eight attention heads, a hidden size of $d_h = 256$, and a Huber $\delta = 1.0$ for all datasets and LLM backbones. Our FFN residual MLP consists of two linear layers with a hidden size of $4 * d_h$ interpolated by a GELU (Hendrycks & Gimpel, 2016). See Table 6 for all hyperparameters.

What if the initial projection is omitted? In § 3, we noted that the initial projection from the frozen LLM backbone states $H(x) \in \mathbb{R}^{S \times d}$ into $X(x) \in \mathbb{R}^{S \times d_h}$ using $W \in \mathbb{R}^{d \times d_h}$ is mathematically optional whenever the chosen head dimension matches the backbone's hidden dimension ($d_h = d$). If this optional step is not taken, the token-level memory directly utilizes the raw LLM hidden states (i.e., $X(x) = H(x)$).

Omitting this initial projection is highly parameter-inefficient. By directly utilizing the raw hidden states ($X(x) = H(x)$), the model bypasses the projection matrix W, reducing the total parameter count by d^2 initially. However, the L downstream refinement layers now operate at the full LLM backbone's dimension d instead of d_h. Within each refinement layer, each multi-head cross-attention block introduces at least $\approx 3d^2$ additional parameters, since each projection matrix for the query, value, and key (W_m^Q, W_m^K, W_m^V) now operate in d. Similarly, each feed-forward network introduces $\approx 8d^2$ additional parameters, assuming our 2-layer setup where each layer is now of shape $d \times 4d$. Thus, omitting the initial projection at least contributes $\mathcal{O}(L \cdot 11d^2)$ additional parameters. Consequently, the d^2 parameters saved by dropping W are vastly overshadowed by the quadratic parameter explosion in the refinement steps.

Empirically, this quadratic scaling no longer renders RELISH as lightweight. Under our configuration with $L = 3$, setting $d_h = d$ scales the trainable footprint to approximately $32d^2$. For our LLM backbones with large hidden state dimensions (e.g., $d \geq 4096$ versus $d_h = 256$, see Table 7), this effectively scales overall parameters by a factor of ~ 512, requiring over hundreds of millions of parameters. As a result, this full-dimension configuration introduces *substantially more trainable parameters* than even LoRA. We suspect that this quadratic scaling will yield diminishing returns in predictive performance that will not justify this explosive additional overhead.

Moreover, beyond the severe parameter inefficiency, omitting this projection step forces the iterative refinement mechanism to operate directly on the raw pre-trained representation space, prohibiting the model's ability to transform those frozen representations into a specialized, task-specific manifold before the cross-attention interactions begin. Therefore, we do not experiment with setting $d_h = d$ in our experiments.

Why the Penultimate Layer? Our design to extract representations from the penultimate layer (directly before the language modeling head) follows standard practice in LLM-based regression. Tang et al. (2025) fit an MLP head on mean-pooled token representations after a full forward pass of a transformer. Likewise, Lukasik et al. (2025) benchmark RAFT against predictive head baselines that mean-pool output embeddings from the penultimate layer, following RankT5 (Zhuang et al., 2023). Traditionally, canonical embedding models such as Sentence BERT (Reimers & Gurevych, 2019) emit a vector representation by pooling hidden-layer representations. In reward modeling, perhaps the most prominent application of LLM-based regression, Zhu et al. (2023) note that features are usually derived from LLMs by removing the language-modeling head and attaching a linear scalar head directly to the final-layer hidden states. Later reward modeling systems adopt a similar formulation (Miao et al., 2025; Li et al., 2026).

Nevertheless, the optimal layer for regression may not always be the penultimate layer. Different LLMs naturally differ in architecture, including depth, backbone structure, and attention mechanism, so intermediate layers may serve different functions across models (Zhang et al., 2024; Skean et al., 2025; Csordás et al., 2025). Consequently, the best layer or layer combination is likely model-specific. In contrast, the penultimate layer is a robust, model-agnostic choice because it is the definitive stop before the language-modeling head across architectures. Therefore, we do not investigate other layers or weighted layer combinations in this study.

C.2 LLM Backbones

We source our LLM backbones directly from public Huggingface checkpoints: **Llama-3.1-8B-Instruct**,[2] **Qwen 3-8B**,[3] **Qwen 3-32B**,[4] and **Gemma-3-27B-Instruct**.[5].

[2] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
[3] https://huggingface.co/Qwen/Qwen3-8B
[4] https://huggingface.co/Qwen/Qwen3-32B
[5] https://huggingface.co/google/gemma-3-27b-it

Dataset Name	Train	Validation	Test	Score Range
STS-B	5,749	1,500	1,380	[0, 5]
SICKR-STS	6,948	1,985	994	[1, 5]
WMT_EN_ZH	7,000	1,000	1,000	[0, 100]
WMT_RU_EN	7,000	1,000	1,000	[0, 100]
WMT_SI_EN	7,000	1,000	1,000	[0, 100]
StatLLM	321	100	200	[0, 50]

Table 5: Dataset split statistics.

C.3 Datasets

We further provide details on the origin, composition, annotation, and post-processing of the six natural language regression datasets used in our experiments. Table 5 summarizes the splits we adopt for each dataset.

STS-Benchmark (STS-B) STS-B[6] is a compilation of the English datasets used in the SemEval STS shared tasks from 2012 to 2017. Cer et al. (2017) curated STS-B to provide a standard benchmark for the STS community. Sentence pairs are sampled from three distinct domains: image captions, video captions, and news headlines. Annotations were collected via crowdsourcing (Amazon Mechanical Turk), where workers assigned scores from 0 to 5 based on the degree of semantic equivalence. We followed the split created by Enevoldsen et al. (2025).

SICKR-STS Derived from the Sentences Involving Compositional Knowledge (SICK) dataset (Marelli et al., 2014), SICKR-STS[7] focuses specifically on semantic relatedness. The sentences originated from the 8K ImageFlickr (Hodosh et al., 2013) and MSR-Video Description (Chen & Dolan, 2011) datasets and were subsequently processed to include specific linguistic variations, such as negation and passive-to-active voice transformations. Like STS-B, it was annotated by crowdsourced workers on a 5-point Likert scale. We followed the split created by Enevoldsen et al. (2025).

WMT2020 Multilingual Quality Estimation (MLQE) Datasets The MLQE[8] "Task 1: Predicting sentence-level DA" involves predicting the quality of neural machine translation (NMT) outputs, without access to human references. We select three subsets to test model performance across varying data availability: English-to-Chinese (En–Zh; high-resource), Russian-to-English (Ru–En; medium-resource), and Sinhala-to-English (Si–En; low-resource). The source sentences for En–Zh and Si–En are primarily from Wikipedia, while Ru–En includes a mix of Wikipedia and Reddit content. Each machine translation was directly assessed (DA) by professional translators who scored the output on a fine-grained, ordinal 0-100 scale based on its accuracy relative to the source sentence. We adopt the splits created by Specia et al. (2020) directly.

StatLLM StatLLM (Song et al., 2026) evaluates the quality of LLM-generated SAS code for scientific tasks. While STS and WMT are canonical tasks in text regression, StatLLM comprises 207 scientific workflow tasks, each paired with code generated by GPT-3.5 (OpenAI, 2022), GPT-4 (OpenAI, 2023), and Llama 3.1 70B (Grattafiori et al., 2024), yielding a total of $207 * 3 = 621$ examples. This is notably smaller than the STS and WMT datasets, but code generation is also a more complex task, and examples are generally longer than those in STS and WMT. Also, whereas STS and WMT inputs are scored against references, each input in StatLLM is a single generated SAS program plus task instructions. StatLLM thus provides a different form of regression task for evaluating the generalization of methods.

[6] https://huggingface.co/datasets/mteb/stsbenchmark-sts
[7] https://huggingface.co/datasets/mteb/sickr-sts
[8] https://huggingface.co/datasets/wmt/wmt20_mlqe_task1

Our interest in StatLLM is further motivated by emerging needs of AI-for-science (AI4Science) (Zheng et al., 2025) where LLM use is becoming increasingly prevalent across a variety of use cases: to generate research ideas (Liu et al., 2026; Gautam et al., 2026), synthesize literature (Ma et al., 2026), and run computational experiments (Qi et al., 2025). While existing AI4Science benchmarks primarily evaluate whether LLM-generated artifacts produce correct outputs (Chen et al., 2025; Joseph et al., 2025), estimating the quality of analytical code at runtime remains largely underexplored. Such inference-time quality estimates can assist scientists or autonomous agents in scientific workflows and to triage generated code before execution, flagging problematic snippets for human review (Villavicencio et al., 2026) or routing outputs back to an LLM for further revision (Hao et al., 2026).

Each example in StatLLM is evaluated by human experts along ten rubric criteria grouped under three dimensions: readability, executability, and correctness. The underlying rubric criteria use a five-point Likert scale, ranging from 1 (strongly disagree) to 5 (strongly agree). The dataset[9] does not provide per-criterion ratings but only aggregated scores. Readability aggregates five criteria (5–25), executability aggregates two criteria (0–10), and correctness aggregates three criteria (0–15). The overall quality score is the sum of these three aggregate scores, yielding a final prediction target in the range 0–50. In our experiments, we directly predict the overall quality score without querying for intermediate dimensions.

C.4 Metrics

Range-Normalized Root Mean Square Error Because our datasets use heterogeneous label scales (e.g., STS-B: 0–5, SICK-R: 1–5, WMT: 0–100, StatLLM: 0–50), raw RMSE is not directly comparable across datasets. In addition, direct macro averaging over raw RMSE can disproportionately upweight the translation datasets, since the prediction errors are generally larger for the WMT task. In Table 3, we therefore report range-normalized RMSE (NRMSE), defined as $\text{NRMSE} = \text{RMSE}/(y_{\max} - y_{\min})$, or dividing the raw RMSE by each dataset's gold-score range. We then macro-average NRMSE across datasets, models, and runs. This yields an error metric on a common unitless scale, where lower values indicate smaller prediction error relative to the particular dataset's full scoring range.

C.5 Baselines

This section provides further implementation details for the baseline methods (Table 2) benchmarked in our work, including specific hyperparameter configurations for both the frozen and LoRA-fine-tuned regimes.

C.5.1 Autoregressive Decoding

Zero-shot Prompting We evaluate the base capabilities of the LLM backbones using the system prompts and formatting templates detailed in Table 11, 12, 13, 14, 15, and 16. We limit model outputs to a maximum of 10 new tokens. We set temperature = 0.7 across three independent seed runs to control for output variance. The numeric score is directly extracted from the generated text using Python's FLOAT function. Note that this means the LLM could generate potentially multiple equivalent string representations of the same numeric candidate, such as '1.0' versus '1.00'. Since we employed only instruction-following models, all LLM backbones produced valid numeric strings.

Many-shot Prompting As shown in Vacareanu et al. (2024), LLMs can be capable regressors when given in-context demonstrations. We therefore also experiment with many-shot learning in addition to zero-shot prompting. We provide $k = 128$ in-context demonstrations for all datasets, which already incurs a $\approx 10\times$ slowdown in inference across all LLM backbones, so we did not experiment with larger numbers of demonstrations. Examples are randomly sampled from the training split for each independent run, but are consistent for trials using the same random seed. We use the same formatting instructions as the zero-shot case, prefixing the user message with formatted examples before appending the test query.

[9] https://github.com/yili-hong/StatLLM

As noted in the main body, we do not evaluate many-shot in the LoRA fine-tuning regime because the parameter updates during fine-tuning are intended to internalize the task logic, rendering in-context demonstrations (which were already seen during training) redundant.

C.5.2 Regression-Aware Baselines

RAIL Regression-aware inference for LLMs (RAIL) (Lukasik et al., 2024) avoids text decoding by treating the numeric prediction as a decision-theoretic problem. We implement the RAIL posterior mean approach, where the model's output distribution over numeric tokens is used to calculate an expected value. We limit the vocabulary to digits 0–9 and the decimal point, normalizing the probabilities over these tokens at each step of a fixed-length numeric sequence generation. Following Lukasik et al. (2024), we sampled 16 outputs from the LLM backbone to approximate the posterior with an effective temperature of 0.25.

RAFT Regression-aware fine-tuning (RAFT) (Lukasik et al., 2025) extends RAIL by fine-tuning the model to minimize a special regression-aware loss (see Table 1) rather than the standard token-level cross-entropy. In our experiments, RAFT uses the same hyperparameter setup as the standard LoRA decoding baselines, but optimizes the numeric expected value computed via the RAIL procedure using mean squared error. Following Lukasik et al. (2025), we do not sample like RAIL but instead pick $\mathcal{Y}_{\text{grid}}$ values that correspond to the gold-score range for each dataset. For STS-B, we use $\mathcal{Y}_{\text{grid}} = \{0, 1, 2, 3, 4, 5\}$. For SICKR-STS, we use $\mathcal{Y}_{\text{grid}} = \{1, 2, 3, 4, 5\}$. For WMT datasets, we use $\mathcal{Y}_{\text{grid}} = \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. For StatLLM, we use $\mathcal{Y}_{\text{grid}} = \{0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. Following Lukasik et al. (2025), we do not normalize grid candidates for RAFT. We observe that all LLM backbones produced continuous scalars and achieved good predictive performance even with only a few grid candidates.

C.5.3 Predictive Head Baselines

Linear Regression (Lin.) We first extract token hidden states from the penultimate layer of the transformer, $H(x) \in \mathbb{R}^{S \times d}$, and an attention mask $m \in \{0, 1\}^S$. We then compute the masked mean-pooled representation:

$$\bar{h}(x) = \frac{\sum_{t=1}^{S} m_t \, H_t(x)}{\sum_{t=1}^{S} m_t}.$$

Finally, we apply a single linear layer to predict the (standardized) scalar target:

$$\hat{z} = \mathbf{w}^\top \bar{h}(x) + b.$$

We omit the y-normalization and de-normalization specified in Section 3 for brevity.

Multi-Layer Perceptron (MLP) We similarly first compute $\bar{h}(x)$ for the MLP predictive head. Following Tang et al. (2025), we use a 2-layer MLP regressor

$$u_1 = \text{Drop}\big(\text{RELU}(W_1 \bar{h} + b_1)\big), \quad u_2 = \text{Drop}(\text{RELU}(W_2 u_1 + b_2)), \quad \hat{z} = W_3 u_2 + b_3,$$
$$W_1 \in \mathbb{R}^{d \times h}, \quad W_2 \in \mathbb{R}^{h \times h}, \quad W_3 \in \mathbb{R}^{h \times 1},$$

with RELU activations (Glorot et al., 2011) and dropout rate 0.1 after each hidden layer. To approximately match the training capacity of RELISH, we select hidden size h by parameter matching. This yields $h = 704$ for Llama 3.1 8B Instruct and Qwen 3 8B, and $h = 640$ for Gemma 3 27B Instruct and Qwen 3 32B. We fine-tune both the linear regressor head and the 2-layer MLP using Huber loss with $\delta = 1.0$ and a learning rate of $1e-4$. See Table 6 for all hyperparameters.

C.5.4 LLM Training Configurations

All LLM backbones were trained using the AdamW optimizer with a linear learning rate scheduler and a warmup phase of 10% of the total training steps. LLM training was conducted for 5 epochs with an effective batch size of 32 and a learning rate of $5e-5$. We stop the training early after 2 epochs if the validation RMSE does not improve.

Hyperparameter	Value
Shared	
Seed Values	42, 1234, 2026
Effective Batch Size	32
Early Stopping Patience	2
Warmup Ratio	0.1
Weight Decay	0.01
Dropout	0.1
Precision	bfloat16
LoRA / RAFT	
Learning Rate	5×10^{-5}
Max Epochs	5
Rank (r)	8
Alpha (α)	16
Target Modules	All Linear Layers (`q, k, v, o, gate, up, down`)
Predictive Heads / RELISH	
Learning Rate	1×10^{-4}
Max Epochs	10
RELISH d_h	256
Predictive Head d_h	704 (Llama & Qwen-8B) or 640 (Gemma & Qwen-32B); see § C.5.3
L	3
FFN Hidden Size	1024
Huber δ	1.0

Table 6: Hyperparameter configurations.

Frozen Regime In the frozen regime, the LLM backbone parameters θ are fixed. For RELISH, Linear, and MLP baselines, only the weights of the predictive heads are updated. We used a learning rate of $1e-4$, the Huber loss, and a maximum of 10 epochs, with a patience of 2 epochs to improve the validation loss.

LoRA Regime For fine-tuning, we apply Low-Rank Adaptation (Hu et al., 2022) to all linear projections within the transformer blocks: the attention modules (`q_proj, k_proj, v_proj, o_proj`) and the feed-forward network modules (`gate_proj, up_proj, down_proj`). We use a learning rate of 5×10^{-5}. Detailed hyperparameters are provided in Table 6. For all LoRA variants besides RAFT, we adopt the standard cross entropy loss on textualized targets, so we do not apply y-normalization here.

D Full Predictive Performance Results

In this section, we expand on the predictive performance results discussed in § 4.2. First, we discuss how we aggregated results across datasets, LLMs, and seeds in Table 3. We then introduce additional figures and tables (§ D.2). Next, we discuss the high-level trends and provide some intuition on why RELISH works so well (§ D.3). Finally, we dive into each dataset and analyze trends locally (§ D.4).

D.1 Aggregation Pipeline for Table 3

Table 3 reports the macro-averaged results for Pearson correlation (r), Spearman correlation (ρ), and range-normalized Root Mean Squared Error (NRMSE). In this section, we specify the aggregation pipeline. Namely, we report macro-averaged means across metrics, combined with seed-to-seed standard deviations in $\pm$. We adopt this aggregation strategy to measure both the generalizability of regression-baselines across diverse tasks and LLMs and their stability across training runs.

Granular Metric Computation For each metric $\ell \in \{r, \rho, \text{NRMSE}\}$, we first compute the score $x_{d,m,s}^{(k,\ell)}$ for a given baseline k, evaluated on dataset d, using LLM backbone m,

under some random seed s. Note that NRMSE normalizes RMSE by the label range of the respective dataset d to ensure comparability across tasks with vastly different target scales:

$$\text{NRMSE}_{d,m,s}^{(k)} = \frac{\text{RMSE}_{d,m,s}^{(k)}}{y_d^{\max} - y_d^{\min}}$$

All reported metrics are scaled by 100 to be reported as percentages.

Macro-averaging Across Datasets, LLM backbones, and Seeds For a given method k and metric ℓ, we compute a macro-mean $a_s^{(k,\ell)}$ for each seed s by averaging the scores across all available dataset-model pairs $\mathcal{P}$:

$$a_s^{(k,\ell)} = \frac{1}{|\mathcal{P}|} \sum_{(d,m)\in\mathcal{P}} x_{d,m,s}^{(k,\ell)}$$

Table 3 then reports the mean (μ) and standard deviation (σ) of $a_s^{(k,\ell)}$:

$$\mu^{(k,\ell)} = \frac{1}{|\mathcal{S}|} \sum_{s\in\mathcal{S}} a_s^{(k,\ell)}$$

$$\sigma^{(k,\ell)} = \sqrt{\frac{1}{|\mathcal{S}|-1} \sum_{s\in\mathcal{S}} \left(a_s^{(k,\ell)} - \mu^{(k,\ell)}\right)^2}$$

Why not Micro-Average? We chose to report macro-averages rather than micro-averages, which would have first pooled individual predictions across all datasets before computing the respective metric. While this can be more intuitive, micro-averaging is highly susceptible to datasets with larger samples or higher variance, which can disproportionately skew the overall result. By computing the macro-mean $a_s^{(k,\ell)}$, we treat every dataset-model pair as an equal vote. Consequently, the superior predictive performance of RELISH in Table 3 reflects robust and consistent gains across diverse datasets and LLM backbones, rather than potentially lifted by a few datasets or LLM backbones with large improvements.

In addition, we highlight that our reported σ (seed-to-seed variance) is *not* task heterogeneity, or the variance of metrics across different datasets and models. Task heterogeneity would be much larger, since datasets inherently vary in difficulty. Instead, our reported $\pm$ isolates stability across different seeds (training variance). The relatively tight $\pm$ bounds shown in Table 3 confirm that the performance gains achieved by RELISH are statistically stable and not the artifact of a "lucky" random initialization.

D.2 Additional Figures and Tables

Figures 2 and 3 report detailed results for Spearman correlation and the range-normalized root mean squared error (NRMSE) relative to the zero-shot performance of the underlying LLM backbones, respectively. For Figures 1 and 2, the absolute improvement is calculated as $r_{RELISH} - r_{zero_shot}$ and $\rho_{RELISH} - \rho_{zero_shot}$, respectively. For Figure 3, we flip the order and compute the absolute improvement as $NRMSE_{zero_shot} - NRMSE_{RELISH}$, since lower is better.

Full predictive performance tables reporting raw RMSE, Pearson, and Spearman correlation across four LLM backbones and six datasets are reported in Tables 17, 18, 19, 20, 21, and 22. We do not report range-normalized RMSE there since each table compares baseline-LLM pairs within each dataset. Nevertheless, one may simply convert raw RMSE to NRMSE by dividing each dataset by its appropriate gold-score range (e.g., STS-B: 0–5, SICK-R: 1–5, WMT: 0–100, StatLLM: 0–50).

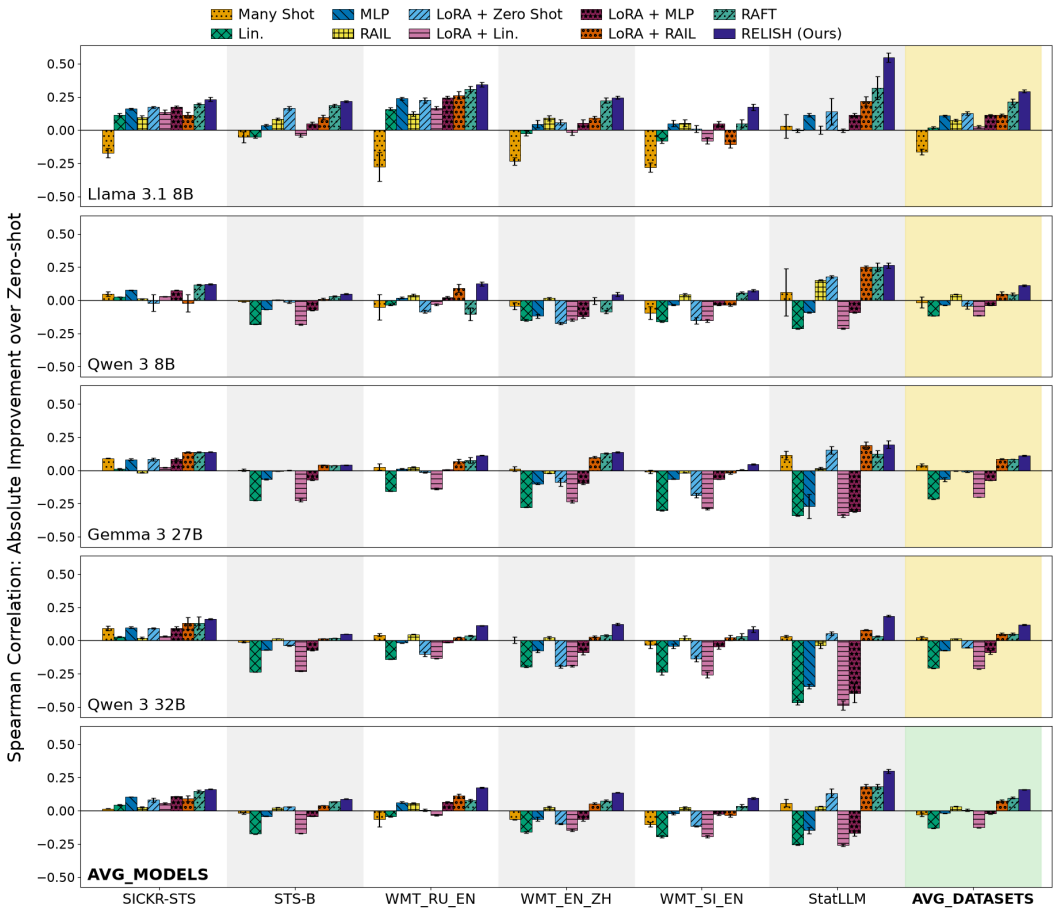

Figure 2: **Spearman (ρ) correlation across datasets (columns) and LLMs (rows).** The rightmost column averages over all datasets, and the bottom row averages over LLMs. Whereas earlier Figure 1 showed Pearson correlation, this figure shows absolute improvement in Spearman correlation over the zero-shot baseline (the "ground zero" line). Error bars denote variability across 3 independently-seeded runs. All metrics are computed using gold test labels. RELISH consistently performs best across all settings.

D.3 High Level Patterns

D.3.1 The Verbalization Gap: Why Prompting is Suboptimal

Consistent with Vacareanu et al. (2024), our results confirm that LLMs are capable regressors with learned numerical reasoning capabilities. Instead, the main challenge is how to *best extract numerical predictions from LLMs*. Specifically, while pre-trained LLMs encode regression-relevant signals in their hidden states, existing inference methods are suboptimal at exploiting what the LLM already knows.

Autoregressive (prompt-based) methods are the simplest, directly modeling numeric targets as text. Zero-shot prompting establishes a reasonable baseline (see Table 3), indicating that although LLMs are not explicitly trained as regressors, the internalized scale and relational priors (e.g., "low" versus "high") learned during pre-training are sufficient to verbalize approximate numeric values at inference time (Yuchi et al., 2026).

However, autoregressive approaches are fundamentally limited by the **verbalization gap** (Yuchi et al., 2026): while LLM internal representations may accurately encode a scalar, the stochastic process of next-token generation often introduces noise, preventing the model from accurately translating its internal estimates into the correct string of digits. Our many-

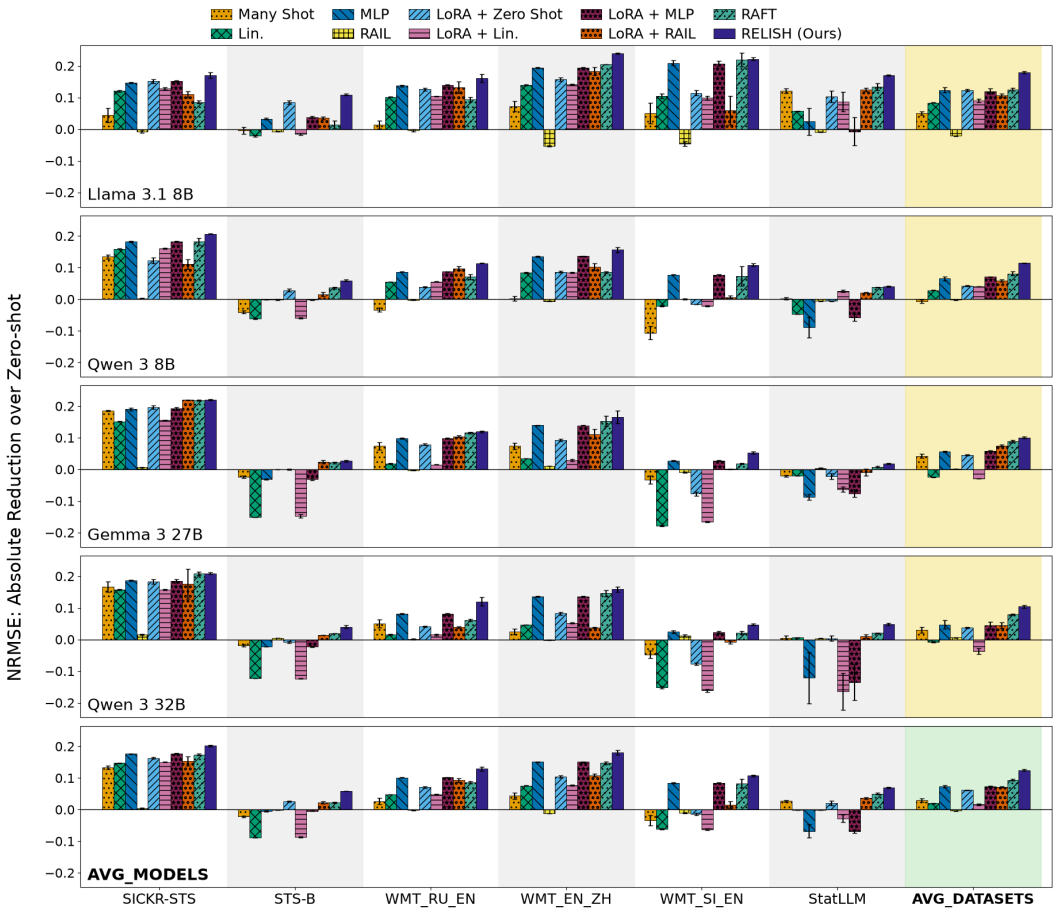

Figure 3: **NRMSE results across datasets (columns) and LLMs (rows).** The rightmost column averages over all datasets, and the bottom row averages over LLMs. Whereas earlier Figure 1 showed Pearson correlation, this figure shows absolute improvement in range-normalized root mean squared error (NRMSE) over the zero-shot baseline (the "ground zero" line). Error bars denote variability across 3 independently-seeded runs. All metrics are computed using gold test labels. RELISH consistently performs best across all settings.

shot results (see Table 3) exemplify this gap. Contrary to expectations, adding in-context demonstration may actually degrade correlation metrics (e.g., $r = 57.8$ compared to $r = 61.0$ for zero-shot). We attribute this degradation to *contextual distraction*: rather than calibrating the model's internal scale, the presence of fixed exemplars introduces formatting biases and a documented tendency to regress toward the mean of the provided demonstrations (Vacareanu et al., 2024). Consequently, simply providing more context fails to bridge the verbalization gap, as the model remains limited by the autoregressive generation process, which is poorly aligned with the numerical nature of regression targets.

D.3.2 The Summarization Bottleneck: Why Static Pooling for Predictive Heads is Suboptimal

Predictive heads bypass the verbalization gap by treating the LLM as a feature extractor rather than a text generator. Specifically, predictive heads attach an auxiliary regressor to the model's internal representations, aiming to directly "read out" numeric values that the autoregressive generator often fails to articulate. However, our results indicate that standard predictive heads are constrained by a **summarization bottleneck**: mean-pooled representations are often too coarse to capture the fine-grained signal necessary for regression.

For instance, our results show that a linear regressor over mean-pooled hidden states is the weakest baseline, significantly underperforming even zero-shot prompting in terms of Pearson and Spearman correlation (e.g., $r = 52.9$ versus $r = 61.0$, and $\rho = 50.3$ versus $\rho = 60.9$, respectively). We attribute this to *information collapse*, where static averaging washes out the localized token-level features essential for precise numeric estimation (e.g., the log-magnitude signals identified by Yuchi et al. (2026)). While a non-linear MLP head ($r = 64.7, \rho = 71.7$) can improve predictive performance, this approach remains bottlenecked by a fixed-vector summary and thus is suboptimal compared to RELISH. In summary, while existing predictive heads avoid the noise of token generation, they introduce their own limitations by compressing high-dimensional hidden states into static vector representations that are too restrictive for regression.

D.3.3 Regression-aware Inference Helps, but Remains Suboptimal.

Regression-aware baselines such as RAIL (Lukasik et al., 2024) and RAFT (Lukasik et al., 2025) attempt to bridge the verbalization gap by aligning the decoding process with continuous scalar predictions. These methods thus implicitly assume that LLMs already possess the internal reasoning capabilities for regression, but better verbalization techniques are needed to obtain accurate numerical estimates. However, regression-aware inference remains an indirect and computationally expensive solution to the extraction problem.

Empirically, Table 3 shows that RAIL can outperform zero-shot prompting on correlation metrics, but not on RMSE. A plausible explanation is that RAIL improves *ranking* without improving *scale calibration*. In particular, the constrained decoding procedure may preserve relative ordering across examples while shrinking predictions toward high-probability regions, clipping extreme values, or discretizing outputs too coarsely. Such effects can increase bias in the mean prediction even as they reduce variance, leading to better correlation but worse RMSE. More broadly, because RAIL still inherits the backbone LLM's imperfect numeric calibration, averaging samples from its induced predictive distribution can stabilize outputs without correcting systematic errors in magnitude. We therefore view RAIL as a partial mitigation of the generation-regression mismatch, rather than a guaranteed improvement over zero-shot next-token prediction.

RAFT outperforms all autoregressive methods. However, its empirical gains are not consistent across all settings and can underperform baselines such as the MLP predictive head (e.g., see Figure 1 for Qwen 3 8B on WMT_RU_EN). Computationally, RAFT requires an expensive pipeline: LoRA fine-tuning, regression-aware training, and regression-aware inference.

In contrast, RELISH addresses the verbalization gap and summarization bottleneck more directly. Rather than forcing the generation process to behave like a regressor, RELISH learns a task-specific summarization over token-level representations to predict scores in a single pass. This approach is not only more computationally efficient but also more architecturally natural: if the log-magnitude signals are already present in the hidden states (Yuchi et al., 2026), directly learning to extract them is more intuitive than attempting to elicit them through sampled or constrained decoding.

D.3.4 RELISH Provides Architectural Gains Beyond Head Capacity Alone.

The gains from RELISH cannot be explained by simply adding more trainable parameters. Our MLP baseline is parameter-matched to RELISH for each LLM backbone, yet RELISH still consistently outperforms it across six datasets. This indicates that the improvements are architectural rather than capacity alone, suggesting that iterative token-level summarization is more effective for scalar prediction than static mean pooling. This interpretation is further supported by the comparison to the LoRA + zero-shot baseline. Although LoRA updates substantially more parameters, it remains less performant than RELISH. This suggests that, for regression tasks such as semantic sentence similarity and machine translation quality estimation, the primary bottleneck is not necessarily acquiring new task-relevant knowledge through backbone adaptation, but extracting an accurate scalar prediction from representations that already encode substantial numerical signal. From this perspective,

LoRA may help with task adaptation by calibrating to expected output ranges, but it is less effective than RELISH's iterative summarization mechanism.

D.3.5 RELISH Provides Structural Gains Beyond Calibration Alone.

The gains from RELISH are not solely due to calibration. Across Figures 1, 2, and 3, RELISH consistently achieves the largest gains across Pearson, Spearman, and RMSE. This is significant because a calibration method that merely rescales predictions might lower RMSE or increase Pearson (e.g., LoRA fine-tuning), but such a method would not necessarily change the relative ranking of test queries. The consistent gains in Spearman correlation across LLM backbones and datasets demonstrate that RELISH actually improves the LLM's ability to rank instances by similarity or quality. Thus, RELISH is not just calibrating model predictions, but also learning how to make more accurate numerical predictions using a higher-resolution regression signal that is otherwise lost to the noise of verbalization or the coarseness of static pooling.

D.3.6 RELISH Helps Most When Regression Evidence is Localized or Distributed.

Examining results at the task level helps clarify when RELISH's architectural design provides the greatest benefit. The gains are most pronounced when regression-relevant evidence is localized or distributed across the input, rather than captured by a single global similarity signal. STS generally requires a global interpretation of semantic relatedness, which prompting and mean-pooled heads can often capture reasonably well. In contrast, WMT quality estimation depends on localized translation errors, omissions, and subtle meaning shifts that may be spread across source–translation pairs. StatLLM similarly requires identifying correctness, executability, and readability evidence across longer code inputs. Such fine-grained signals may be washed out by static pooling or obscured by autoregressive verbalization, whereas RELISH can repeatedly query token-level representations through cross-attention to preserve both localized and distributed cues.

D.4 Dataset Specific Patterns

STS-B (Table 17): For STS-B, improvements over baselines are generally modest. Because STS relies on global semantic similarity, larger LLM backbones already perform quite well with standard autoregressive or regression-aware decoding methods. Consequently, as the backbone scales, the headroom for improvement shrinks. This is evident on Gemma 3 27B Instruct, where RELISH's Spearman correlation ($\rho = 0.923$) effectively ties with the computationally heavier LoRA+RAIL baseline. Nevertheless, RELISH maintains a definite lead in both Pearson and RMSE. Thus, even where baseline methods are already strong, RELISH remains the most robust across all LLM backbones and metrics.

SICKR-STS (Table 18): Similarly, for SICKR-STS, improvements over baselines are generally modest. However, in this dataset, predictive heads are surprisingly strong baselines, suggesting that the mean-pooled representations retained much of the information embedded in full token-level hidden states. Yet, even in this highly competitive setting, RELISH consistently squeezes additional performance across all four backbones. For instance, on Gemma 3 27B Instruct, RELISH narrowly but definitively outperforms the strongest alternatives, including RAFT and LoRA+RAIL. This demonstrates that even when existing baselines perform reasonably well, iterative latent refinement still captures valuable, task-relevant signals that are missed by verbalization and static aggregation.

WMT Ru–En (Table 19): RELISH's architectural advantages are especially evident on WMT_RU_EN. This dataset exposes the weaknesses of generation-based methods: many-shot prompting yields inconsistent gains, and RAIL frequently trails behind learned predictive heads. RELISH capitalizes on this, achieving substantial margins over RAFT and LoRA baselines across all backbones. This trend reinforces our prior hypothesis that machine translation quality estimation requires selectively analyzing localized cues spread across source-translation pairs, a task RELISH excels at with its token-level latent querying.

WMT En–Zh (Table 20): This translation dataset poses the most difficult task in our suite, yielding significantly lower correlation metrics across all methods and LLM backbones. Under these challenging conditions, standard methods may not perform well. Notably, even LoRA-adapted methods (particularly LoRA+Zero Shot and LoRA+Linear) degrade sharply, struggling to maintain stability. RAFT, typically the second-best baseline, may actually underperform simpler baselines like LoRA+RAIL on Qwen 3 8B. In contrast, RELISH remains highly robust, consistently achieving the highest predictive performance.

WMT Si-En (Table 21): WMT_SI_EN is meant to test the capabilities of LLMs on a low-resource language setting, such as translating from Sinhala to English. Interestingly, the correlation scores here are higher than those in WMT_EN_ZH. For instance, RELISH reaches the highest Pearson correlation of $r = 0.669$ on Gemma 3 27B Instruct here, versus a best of $r = 0.567$ on Qwen 3 32B for WMT_EN_ZH. We suspect that since the translation is in English, the LLM backbones can rely on strong English priors to avoid major structural errors that often appear in low-resource translation (Guzmán et al., 2019; Han et al., 2021). Nevertheless, RELISH performs best across all four backbones, suggesting that its token-level summarization remains effective even under low-resource settings.

StatLLM (Table 22): As noted in § C.3, StatLLM provides a qualitatively different test case because code generation is a more complex task, and because each example consists of a generated SAS program plus task context, rather than a sentence pair or source–translation pair. That said, we have also noted the smaller scale of this dataset compared to others, and future work on larger scale code generation datasets may reveal additional patterns.

RELISH is seen to achieve the best results across all four LLM backbones. This suggests that RELISH's gains are not solely due to cross-attention aligning two input segments. Instead, the results support our broader interpretation that RELISH improves regression by learning which parts of the input are most informative, including localized correctness issues and distributed readability signals in longer single-input programs.

E Full Training Footprint Results

Table 7 compares the exact trainable component count across all evaluated LLM backbones for RELISH, LoRA (Hu et al., 2022), and RAFT (Lukasik et al., 2025). Notably, while Table 4 and Table 7 show that LoRA and RAFT have the same trainable parameter count, in practice, RAFT incurs a higher cost because the posterior for each training example must still be approximated via sampling or enumerating grid candidates.

Despite being the most performant method, RELISH also fine-tunes significantly fewer parameters than LoRA and RAFT, which use a regression-aware loss for LoRA fine-tuning. For instance, on the 8B-class models (Llama 3.1 and Qwen 3), RELISH introduces merely 3.4 million trainable parameters, or roughly 0.04% of the LLM backbone. In contrast, standard LoRA and RAFT adaptation requires over 20 million parameters, or roughly 0.26% of the LLM backbone, representing a roughly six-fold increase in parameter overhead. This gap is even more significant for larger models. For Qwen 3 32B, LoRA and RAFT require fine-tuning roughly 37 times as many parameters as RELISH.

In addition, unlike LoRA and RAFT, RELISH's training footprint does **not** scale with the LLM backbone. Rather, the dominant components in RELISH are the projection matrix $W \in \mathbb{R}^{d \times d_h}$ and the iterative refinement layers consisting of the multi-head attention (MHA), the FFN, and the LayerNormalization (LN). Thus, the main hyperparameters that affect the trainable capacity of RELISH are the head dimension d_h, the number of refinement layers L, and parameter sizes within each refinement layer, such as the hidden size of the FFN ($4 * d_h$). Conversely, this means that RELISH's training footprint **does not scale with the size of the LLM backbone**. In summary, RELISH is not performant merely because it fine-tunes more parameters, but by providing a more architecturally efficient summarization mechanism.

Method	Trainable (n)	% of Model
Llama 3.1 8B Instruct	($d = 4096, N = 8,030,261,248$)	
LoRA/RAFT	20,971,520	0.2612%
RELISH	**3,418,625**	**0.0426%**
Qwen 3 8B	($d = 4096, N = 8,190,735,360$)	
LoRA/RAFT	21,823,488	0.2664%
RELISH	**3,418,625**	**0.0417%**
Gemma 3 27B Instruct	($d = 5376, N = 27,432,406,640$)	
LoRA/RAFT	116,502,528	0.4247%
RELISH	**3,746,305**	**0.0137%**
Qwen 3 32B	($d = 5120, N = 32,762,123,264$)	
LoRA/RAFT	134,217,728	0.4097%
RELISH	**3,680,769**	**0.0112%**

Table 7: **Detailed parameter footprints across architectures.** N represents the total number of backbone parameters and d is the model hidden dimension. **Trainable (n)** is the number of parameters updated during training, and **% of Model** denotes n/N. Note that while baseline footprints scale with the total parameter count N, RELISH's footprint remains largely invariant to backbone size, scaling primarily as a function of d.

F Ablation Results

F.1 How sensitive is RELISH to the choice of loss function?

	Fine-tuned Baselines		RELISH Variants	
Metric	**LoRA+MLP**	**RAFT**	**RELISH (MSE)**	**RELISH (Huber)**
$r \uparrow$	$64.8_{\pm 0.1}$	$71.5_{\pm 0.4}$	$74.2_{\pm 0.4}$	$\mathbf{76.3}_{\pm 0.2}$
$\rho \uparrow$	$61.7_{\pm 0.1}$	$69.0_{\pm 0.2}$	$71.3_{\pm 0.4}$	$\mathbf{74.0}_{\pm 0.1}$
NRMSE $\downarrow$	$16.7_{\pm 0.1}$	$16.6_{\pm 0.3}$	$13.5_{\pm 0.1}$	$\mathbf{13.3}_{\pm 0.3}$

Table 8: Ablating RELISH with Huber versus MSE loss. We report macro-averaged Pearson r and Spearman ρ correlation (higher is better), as well as range-normalized RMSE (NRMSE, lower is better) in %. Metrics are averaged across the five STS/WMT datasets, excluding StatLLM, four LLM backbones, and three independently seeded runs. We also report standard deviation ($\pm$).

Table 3 shows that RELISH consistently improves over prior LLM-regression baselines. But where do the gains come from? In this section, we investigate **whether the improvements are due to better architectural design or simply an artifact of the Huber loss** (Huber, 1992). We conduct ablations on the five STS/WMT datasets, excluding StatLLM. Another motivation is to enable fairer comparisons with RAFT (Lukasik et al., 2025). While RAFT currently employs a squared loss (Table 1), RAFT could potentially use a Huber formulation by penalizing large prediction errors with a L1 loss.

Thus, we ablate RELISH trained with the standard mean-squared error (MSE) loss. We keep y-normalization, as we observe that training collapses without it. Specifically, without label normalization, all predictive heads will default to predicting the dataset label mean as the simplest solution to minimize numerical error.

Theoretically, the Huber loss helps stabilize training by behaving like MSE near the numeric target while down-weighting occasional large residuals, which is attractive in our setting because our evaluation spans heterogeneous tasks with different score ranges and annotation processes. At the same time, since label normalization already caps large residuals, we

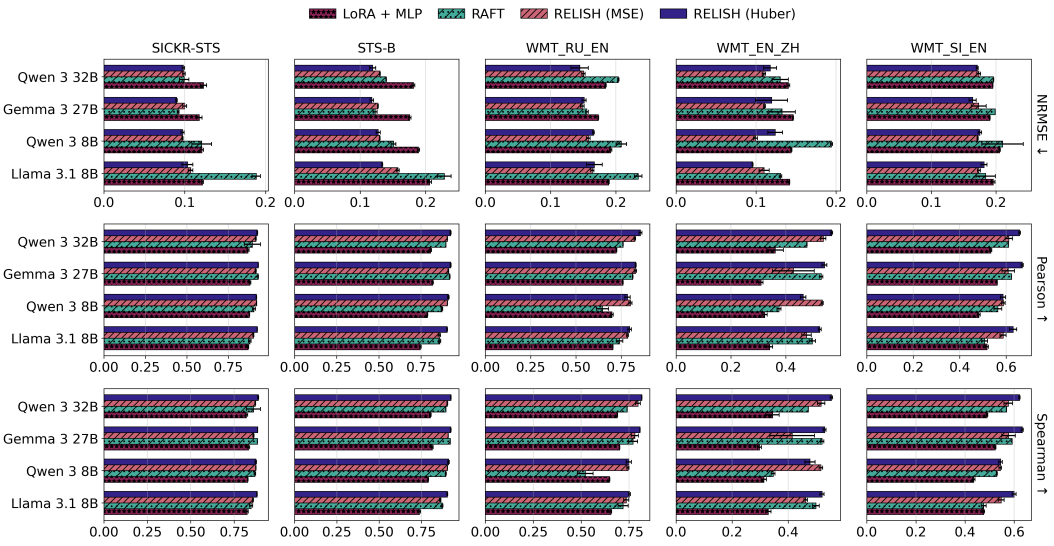

Figure 4: **RELISH loss ablations**. We compare LoRA + MLP, RAFT, and two RELISH variants trained with MSE or Huber across the five STS/WMT datasets, excluding StatLLM (columns) and four LLM backbones (rows). Metrics are NRMSE (lower is better), Pearson correlation (higher is better), and Spearman correlation (higher is better). Error bars show variability across 3 seeds. Overall, MSE and Huber perform similarly, but Huber is modestly better in most settings.

hypothesize that the performance difference between RELISH trained with Huber and with MSE will be modest.

Following the aggregation protocol detailed in § D.1, we report macro-averaged metrics and standard deviations in Table 8. For reference, we duplicate the two most performant baselines from Table 3, namely LoRA + MLP and RAFT (Lukasik et al., 2025). We then provide more fine-grained result breakdowns per dataset and LLM backbone in Figure 4.

Based on Table 8, on average, we observe that **the improvements from RELISH are likely due to improved architectural design, and not an artifact of the Huber loss**. Compared to the other baselines, both variants of RELISH yield higher overall predictive performance. Notably, on NRMSE, RELISH trained with Huber or MSE is very similar (13.5 vs. 13.3), but the Huber loss yields greater improvements on the correlation metrics (76.3 vs. 74.2, 74.0 vs. 71.3, respectively). Nevertheless, the main gains are likely from the iterative-refinement design of RELISH, and using Huber over MSE yields modest improvements in addition to the gains from the architecture.

Figure 4 confirms the overall pattern observed in Table 8. Across the five STS/WMT datasets, four LLM backbones, and three metrics, the RELISH (MSE) and RELISH (Huber) bars are usually close, often nearly overlapping, indicating that **RELISH is generally robust to the loss function**. This affirms our prior intuition that after label normalization, extreme residual magnitudes become less dominant, so Huber no longer has a structural advantage over MSE. Nevertheless, for the majority of panels, RELISH (Huber) outperforms RELISH (MSE), slightly scoring lower for NRMSE and higher for Pearson and Spearman. This suggests that although Huber does not help with output calibration, since labels are already normalized, it can help RELISH learn cleaner monotonic relationships when ranking test queries.

Looking more closely at Figure 4, there are also settings where RELISH with MSE outperforms the original Huber variant. Most notably, for Llama 3.1 8B and Qwen 3 8B, RELISH (MSE) ties with or exceeds RELISH (Huber) across several translation datasets. Similar reversals occasionally appear for larger LLM backbones, but less frequently. We rarely observe any reversals for the semantic similarity datasets.

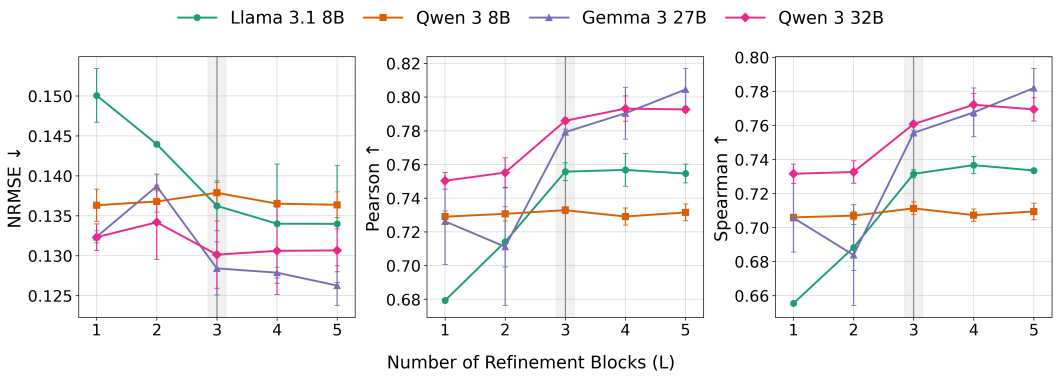

Figure 5: **RELISH refinement depth ablations.** We systematically vary the refinement depth of RELISH from 1 to 5. We report range-normalized root mean square error (NRMSE; lower is better), Pearson correlation, and Spearman correlation (higher is better) macro-averaged across four LLM backbones and the five STS/WMT datasets, excluding StatLLM. Error bars denote the standard deviation across three random seeds. The $L = 1$ configuration isolates the performance of a single-step, attention-based pooling baseline. For most LLM backbones, multiple refinement iterations ($L \geq 2$) substantially enhance predictive performance, with diminishing returns past $L = 3$ (our default setting, shaded in grey). Notably, the optimal depth varies by LLM architecture: Qwen 3 8B achieves peak performance at $L = 1$, while the larger Gemma 3 27B continues to benefit from refinement even at higher depths.

One possible explanation is that smaller LLM backbones are more capacity-limited, so their main bottleneck is representational expressivity rather than robustness to occasional large residuals. Hence, the fully quadratic signal from MSE may sometimes be more preferable, especially on noisier translation tasks where fitting fine-grained score variation is already difficult. In contrast, larger LLM backbones may be better able to exploit Huber's robustness. Once the representation is sufficiently strong, down-weighting harder or noisier examples can more reliably improve ranking consistency. Of course, the main empirical trend remains unchanged: RELISH performs strongly under both losses, and Huber is better overall and in most settings, but only modestly so.

Finally, we note that **RELISH's gains are not due to using a more robust objective than other regression baselines**. On the contrary, the baselines we compare to in Table 3 typically employ loss functions that are at least as robust as Huber. For the predictive head methods (linear and MLP), we use the same Huber objective. For RAFT, a special regression-aware loss is employed. The only exception might be the LoRA fine-tuned autoregressive methods, but their capacity is larger (more trainable parameters), and yet RELISH consistently outperforms them all. Hence, the gains from RELISH cannot be explained solely by the Huber loss. The iterative architecture offers a clear advantage over existing LLM-regression methods.

F.2 How sensitive is RELISH to the number of iterative refinement blocks (L)?

Previously, § F.1 demonstrated that the improvements from RELISH are primarily driven by architectural design rather than the Huber loss, which only modestly improves predictive performance. Furthermore, § 2.4 highlighted two core architectural differentiations of RELISH from prior predictive head methods: learning a latent attention-based query for robust summarization, and iteratively refining this latent state. In this section, we aim to **disentangle the observed gains of RELISH between the attention-pooling mechanism and the iterative nature of the refinement process**.

Specifically, we systematically vary the number of refinement blocks, L, from 1 to 5. Notably, the $L = 1$ configuration serves as a strict learned, attention-based query-pooling baseline, as the output prediction is computed in a single pass without subsequent iterations. By comparing $L = 1$ against variants where $L \geq 2$, we can explicitly isolate the performance gains attributable to repeated latent state updates versus attention-pooling alone.

Figure 5 reports LLM-specific metrics macro-averaged across the five STS/WMT datasets, excluding StatLLM, with $\pm$ denoting standard deviation across seeds. The primary takeaway is that **although attention-pooling alone serves as a strong baseline, iterative refinement blocks substantially enhance predictive performance**. While a single pass of token-level cross-attention ($L = 1$) outperforms naive pooling, it remains suboptimal compared to the full RELISH architecture. Across the evaluated LLM backbones and datasets, the $L = 1$ configuration generally yields the highest range-normalized root mean square error (NRMSE; lower is better) and the lowest Pearson and Spearman correlations (higher is better). This confirms that merely attending to token representations is insufficient to fully capture the regression target from the frozen latent space. As L increases, we observe sharp reductions in NRMSE alongside notable improvements in both correlation metrics, demonstrating that the latent state requires at least one subsequent pass to re-attend to the token sequence and properly contextualize its initial estimate.

Following the initial performance surge at $L = 2$, we observe a clear pattern of convergence and diminishing returns. The NRMSE curves rapidly flatten, and all predictive metrics become highly stable across $L = 3$ through $L = 5$, exhibiting only marginal fluctuations. Collectively, these patterns justify our default choice of $L = 3$. Importantly, RELISH does not exhibit optimization instabilities or degradation at these deeper refinement levels.

However, we note distinct exceptions to these trends across model architectures. For Qwen 3 8B, iterations yield minimal benefits; the model achieves its lowest NRMSE at $L = 1$, with performance slightly degrading as refinement blocks are added. This suggests that Qwen 3 8B's frozen representations may already encode regression features in a highly localized, accessible manner, allowing efficient extraction in a single pass, whereas subsequent iterations merely inject noise or induce overfitting. We do not observe this behavior in the larger Qwen 3 32B model. We hypothesize this discrepancy stems from the 32B model's larger hidden dimension (5120 vs. 4096), which may dilute regression-relevant information and necessitate iterative contextualization.

Conversely, while Gemma 3 27B Instruct clearly benefits from added iterations, its predictive performance does not plateau at higher depths. Instead, all three metrics continue to improve through $L = 5$, suggesting further gains are possible. We suspect this is similarly tied to the model's hidden size; Gemma 3 27B features the largest hidden dimension in our suite (5376; see Table 7), necessitating deeper iterative refinement to fully isolate task-relevant signals. Therefore, while $L = 3$ serves as a robust empirical default, the optimal refinement depth is ultimately intrinsically tied to the latent representation geometry of the specific LLM backbone.

F.3 How much supervised data does RELISH need?

Since RELISH trains parameters from scratch, a natural consideration is **how much supervised data RELISH requires**. Towards this end, we now investigate how regression performance scales with supervised data availability on the STS-B and WMT_EN_ZH datasets. We subsample the training set at 64, 128, 256, 512, 1,024, and 2,048 training instances, while keeping the validation and test sets fixed. We also replicate the full-dataset performance reported in our paper for reference (5749 for STS-B and 7000 for WMT_EN_ZH). We conduct experiments across the same four LLMs and three random seeds as detailed in § C.

We compare RELISH against MLP and RAFT (Lukasik et al., 2025). MLP is the strongest predictive head baseline, while RAFT is the strongest supervised regression-aware baseline. For a method-level comparison, we report the RMSE, Pearson, and Spearman correlation averaged across four LLM backbones and three random seeds. To ensure fairness, sampled subsets are nested so that trials with more training instances always include those with fewer (e.g., the 128-example set must contain the 64-example set).

Figure 6 shows that predictive performance for each method varies depending on the metric and the amount of labeled data. On correlation metrics, MLP consistently performs the worst in all cases, indicating that mean-pooled predictive heads struggle to rank examples accurately even with more supervision. RAFT performs best in scarce data regimes: on both STS-B and WMT_EN_ZH, RAFT achieves the best Pearson and Spearman correlations with

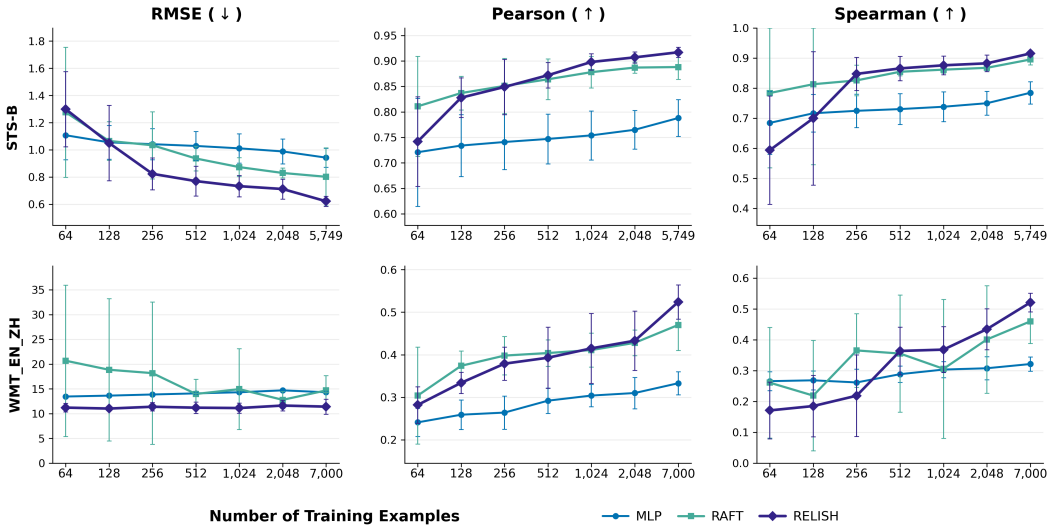

Figure 6: **How predictive performance scales with data availability.** We compare RELISH against MLP and RAFT on STS-B and WMT_EN_ZH as the number of training examples increases from 64 to the full training set (5,749 for STS-B and 7,000 for WMT_EN_ZH). We report RMSE (lower is better), Pearson correlation (higher is better), and Spearman correlation (higher is better), averaged across four LLM backbones and three random seeds. Error bars denote standard error across the corresponding 12 model-seed runs. RAFT performs best under extremely low data regimes ($<$ 256 examples), but RELISH benefits more from additional supervision ($>$ 256–512 examples).

64–256 examples. However, RELISH improves more steadily with additional supervision. On STS-B, RELISH surpasses RAFT by 512 examples and continues to widen the gap as the number of training instances increases. On WMT_EN_ZH, RELISH remains competitive up to 512 examples, but outperforms RAFT with 1,024 or more examples.

The RMSE trends are slightly different. RELISH achieves the lowest RMSE in most cases, except for scarce data settings on STS-B (64 and 128 examples). Overall, RAFT is more performant when labels are extremely scarce ($<$ 256 examples), whereas RELISH benefits more from moderate-to-large ($>$ 256–512 examples) supervised datasets. Thus, a promising future direction might be to explore whether further improvements are possible by combining the best aspects of RELISH and RAFT (see § G).

F.4 How does training overhead scale with supervised data?

We also measure the training overhead of RELISH versus RAFT across different supervision regimes on STS-B and WMT_EN_ZH, following the setup specified in § F.3. Specifically, we benchmark the total training wall-clock on a compute node with an Intel Xeon Platinum 8468 ("Sapphire Rapids") CPU and 4x NVIDIA H100. Figure 7 shows that RAFT is consistently slower to train than RELISH, and the gap widens as the number of training instances grows. Even with only 64 examples, RAFT is already 1.6$\times$ slower on STS-B and 5.6$\times$ slower on WMT_EN_ZH. Using the full training dataset split, the gap grows to 17.7$\times$ on STS-B and 48.9$\times$ on WMT_EN_ZH. Together with Figure 6, these results suggest that while RAFT performs best when labels are extremely scarce, RELISH might still be preferred for its lightweight architecture. Once moderate supervision is available, RELISH is both more accurate and more efficient to train.

F.5 How efficient is RELISH at inference time?

Related to training efficiency, we also investigate **how efficient RELISH is at inference time**. RELISH makes one forward pass through the LLM backbone and emits a prediction

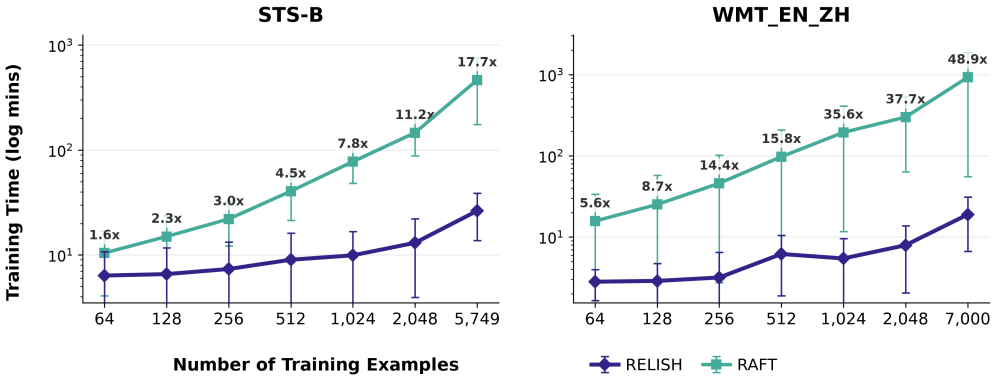

Figure 7: **Training overhead across different data scales.** We compare RELISH and RAFT on STS-B and WMT_EN_ZH as the number of training examples increases from 64 to the full training set. Training time is reported in minutes on a log scale, averaged across four LLM backbones and three random seeds. Error bars denote standard error across 12 model-seed runs, and text labels report the RAFT/RELISH runtime ratio at each training size. RELISH is consistently faster to train than RAFT.

Dataset	Method	Throughput (ex/s) ↑	Latency/RELISH ↓
STS-B	MLP	$15.58_{\pm 8.63}$	0.99×
	Linear	$15.57_{\pm 8.61}$	0.99×
	RELISH	$15.35_{\pm 8.36}$	1.00×
	Zero-shot	$3.17_{\pm 2.53}$	4.85×
	RAFT	$2.92_{\pm 1.70}$	5.26×
	RAIL	$1.24_{\pm 0.78}$	12.43×
	Many-shot	$1.03_{\pm 0.71}$	14.95×
WMT_EN_ZH	MLP	$15.57_{\pm 8.53}$	0.99×
	Linear	$15.57_{\pm 8.57}$	0.99×
	RELISH	$15.34_{\pm 8.29}$	1.00×
	Zero-shot	$2.38_{\pm 1.97}$	6.45×
	RAFT	$1.17_{\pm 0.98}$	13.12×
	RAIL	$1.08_{\pm 0.69}$	14.21×
	Many-shot	$0.62_{\pm 0.44}$	24.86×

Table 9: **Inference efficiency of RELISH.** We report throughput as the number of examples processed per second, averaged across four LLM backbones and three random seeds. Latency/RELISH compares each baseline's wall-clock latency against RELISH, where larger values indicate slower inference. RELISH adds negligible inference overhead compared to simple predictive heads and is faster than autoregressive and regression-aware methods.

using a lightweight predictive head, but the additional refinement layers may impose additional overhead compared to a simple MLP head. Toward this end, we measure inference wall-clocks for all benchmarked LLM-based regression baselines on the test sets of STS-B and WMT_EN_ZH. All experiments are conducted on a compute node with an Intel Xeon Platinum 8468 ("Sapphire Rapids") CPU and 4x NVIDIA H100.

Table 9 reports 'Throughput (ex/s)' by dividing the number of corresponding test set examples by the total wall-clock time. Final metrics are then averaged across four LLMs and three seeds within each dataset to yield a method-level overview. For ease of comparison, we also show a 'Latency/RELISH' column that compares each baseline's relative throughput against RELISH.

Dataset	RELISH Variant	RMSE ↓	Pearson ↑	Spearman ↑	Training Steps ↓
STS-B	Last Token	$0.677_{\pm 0.059}$	$0.898_{\pm 0.019}$	$0.897_{\pm 0.017}$	$1035_{\pm 267}$
	Mean Pool	$0.688_{\pm 0.051}$	$0.894_{\pm 0.017}$	$0.893_{\pm 0.016}$	$\mathbf{960}_{\pm \mathbf{310}}$
	Random (Ours)	$\mathbf{0.623}_{\pm \mathbf{0.037}}$	$\mathbf{0.917}_{\pm \mathbf{0.010}}$	$\mathbf{0.916}_{\pm \mathbf{0.009}}$	$1080_{\pm 188}$
WMT_EN_ZH	Last Token	$\mathbf{11.088}_{\pm \mathbf{1.092}}$	$0.501_{\pm 0.029}$	$0.493_{\pm 0.025}$	$493_{\pm 297}$
	Mean Pool	$11.365_{\pm 1.206}$	$0.498_{\pm 0.027}$	$0.489_{\pm 0.024}$	$493_{\pm 249}$
	Random (Ours)	$11.410_{\pm 1.521}$	$\mathbf{0.524}_{\pm \mathbf{0.040}}$	$\mathbf{0.521}_{\pm \mathbf{0.030}}$	$\mathbf{420}_{\pm \mathbf{174}}$

Table 10: **Warm start ablations for $r^{(0)}$.** We compare the default random initialization of $r^{(0)}$ against input-dependent last-token and mean-pooling warm start configurations. Metrics are averaged across four LLM backbones and three random seeds. Bold indicates the best variant within each dataset and metric. Overall, random initialization appears to be the most robust choice, matching or exceeding the predictive performance of other warm-start variants while maintaining training efficiency.

We observe that RELISH adds negligible inference overhead compared to simple predictive heads (only $\sim 1\%$ slower). In contrast, autoregressive and regression-aware methods incur more overhead. On STS-B, these methods increase latency by factors ranging from 4.85x to 14.95x relative to RELISH. This gap widens on the WMT_EN_ZH dataset, where these baselines incur latency penalties ranging from 6.45x to 24.86x. Ultimately, these results suggest that RELISH is both efficient during training (fewer trainable parameters; see § 4.3) and inference (high throughput).

F.6 Does warm-starting $r^{(0)}$ improve RELISH?

For results reported in § 4.2, we randomly initialized the learned latent state $r^{(0)}$ for all experiments. Another variant is to initialize $r^{(0)}$ from an input-dependent representation, which might give the refinement blocks a warm start and reduce the number of optimization steps needed.

Toward this end, we compare RELISH with different warm-start configurations. Namely, we compare random (reported in § 4.2) to last-token and mean-pooled initializations for $r^{(0)}$ on the STS-B and WMT_EN_ZH datasets across the same four LLMs and three seeds. We follow the same experimental setup details in § C and only change the initialization scheme. To measure whether warm-start can lead to faster convergence, we also report 'Training Steps' as the number of 'optim.step()' counts. Since we do not use gradient accumulation, one step roughly corresponds to one backpropagation step for each training batch. We use a batch size of 32 for all variants, so training steps are comparable across models within a dataset, but not directly across datasets since WMT_EN_ZH has more training data (7k vs. 5.7k). For readability, we aggregate results over all LLMs and seeds for each configuration.

Table 10 shows that input-dependent warm starts do not yield consistent benefits. Mean-pool initialization slightly reduces training steps on STS-B only, and last-token initialization slightly lowers RMSE on WMT_EN_ZH only. On the other hand, the default random initialization achieves the best Pearson and Spearman correlations on both datasets, the best RMSE on STS-B, and the fewest training steps on WMT_EN_ZH. We therefore retain the simpler random initialization as the most robust default.

G Limitations and Future Work

Reliance on LLM backbones. Because RELISH operates over a frozen LLM backbone, its predictive capability is inherently bounded by the quality of the LLM's pre-trained representations. That is, RELISH is designed to robustly extract and refine regression-relevant signals that are *already embedded* in model hidden states. Thus, for tasks that require

novel or out-of-distribution domain knowledge, we would expect some fine-tuning of the LLM backbone to be necessary.

Computational Scaling with Sequence Length. RELISH adds minimal trainable parameters (∼3.4–3.7M) and avoids the heavy inference costs of autoregressive generation or posterior approximation of regression-aware inference methods. However, its attention-based pooling mechanism scales linearly with the input sequence length. For exceptionally long inputs (e.g., estimating the quality of entire research papers), cross-attention operations over token-level representations could become a computational bottleneck.

Scope of Evaluated Tasks. Our evaluation covers semantic similarity, reference-free translation quality estimation, and code quality estimation. Future work may test how well RELISH generalizes to further regression tasks, such as continuous sentiment ratings, product prices, or average product ratings. Recent work by Vedula et al. (2025) suggests product-related prediction as a promising direction, and its authors helpfully suggested two public resources—Amazon Reviews 2018[10] and 2023[11]—that could support such extensions.

Combining RELISH with RAFT. Since RAFT (Lukasik et al., 2025) performs better under sparse data settings, combining RELISH with RAFT is an interesting direction for future work. A simple extension may replace RELISH's linear regressor with a RAFT-style decision-theoretic inference rule. Rather than linearly projecting the final latent state $r^{(L)}$ to a single scalar, the predictive head could project $r^{(L)}$ into logits over a grid of candidates, convert those logits into weights, and compute the final scalar as an expectation over grid values. This hybrid would preserve RELISH's learned token-summary mechanism while using RAFT's grid-based predictor, but it would also reintroduce training and inference overhead that scales with the number of grid candidates.

Leveraging Chain of Thought Rationales. While predictive head methods do not leverage CoT in the same way that autoregressive methods do, RELISH could still leverage CoT. One way to do this would be to incorporate CoT rationales into task inputs. Since RELISH operates on token representations, any rationales included in the prompt can be attended to by the learned latent query. Alternatively, instead of taking CoT rationales as inputs, one could generate them as outputs. For example, future work could explore a multi-task approach in which the LLM backbone is fine-tuned to output CoT rationales, while RELISH makes predictions conditioned on both the original input and generated traces.

Beyond the work required above to leverage CoT, evaluation would also need to be significantly expanded. A fair study would evaluate all regression baselines, with and without CoT rationales, across various datasets and LLMs. Beyond making relative comparisons between methods, one would need to carefully analyze how changes in performance stem from the CoT rationales rather than from the regression methods themselves. One would also need to decide which types of CoT rationales to evaluate: human annotation, self-generated by prompting the LLM backbone, or generated by another teacher LLM.

Finally, leveraging CoT could actually hurt performance. Prior work suggests that incorporating rationales requires careful design (Huang et al., 2023). Chiang et al. (2025) show that fine-tuning on self-generated lower-quality rationales reduces performance. Similarly, systems like Quiet-STaR rely on filtering or rewarding useful rationales rather than indiscriminately training against all generated rationales (Zelikman et al., 2024). Our present study fairly compares all methods without CoT and, for the reasons above, leaves a robust investigation of CoT in regression to future work.

[10] https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/
[11] https://amazon-reviews-2023.github.io/

Dataset: STS-B	

System Prompt	`You are an expert human annotator for semantic textual similarity (STS). You will see English sentence pairs and judge how similar in MEANING they are.`
	
	`Scoring instructions:`
	`- Output a real-valued similarity score from 0.000 to 5.000, inclusive.`
	`- 5.000 = same meaning / paraphrases.`
	`- 2.500 = partially similar with important differences.`
	`- 0.000 = completely unrelated meaning.`
	
	`CRITICAL FORMAT RULE:`
	`Return ONLY the numeric score, with EXACTLY three digits after the decimal point (e.g., 3.000). Do NOT include any words, punctuation, units, labels, or explanation.`

User Message	`Sentence A: {sentence_a}`
	`Sentence B: {sentence_b}`
	
	`Score (0.000 to 5.000 only):`

Table 11: Full prompt template for the STS-Benchmark dataset.

Dataset: SICKR-STS	

System Prompt	`You are an expert human annotator for semantic textual similarity (STS) for Sentences Involving Compositional Knowledge. You will see sentence pairs and judge how similar in MEANING they are.`
	
	`Scoring instructions:`
	`- Output a real-valued similarity score from 1.000 to 5.000, inclusive.`
	`- 5.000 = same meaning / paraphrases; 1.000 = completely unrelated.`
	
	`CRITICAL FORMAT RULE:`
	`Return ONLY the numeric score, with EXACTLY three digits after the decimal point (e.g., 3.000). Do NOT include any words, punctuation, units, labels, or explanation.`

User Message	`Sentence A: {sentence_a}`
	`Sentence B: {sentence_b}`
	
	`Score (1.000 to 5.000 only):`

Table 12: Full prompt template for the SICKR-STS dataset.

Dataset: WMT_EN_ZH	
System Prompt	You are an expert human annotator for translation quality / semantic textual similarity (STS). You will see sentence pairs where Sentence A is in English (source) and Sentence B is in Chinese (translation). You must judge how well the meaning is preserved from English to Chinese. Scoring instructions: - Output a real-valued translation quality / similarity score from 0.000 to 100.000. - 100.000 = perfect meaning preservation (excellent translation / paraphrases). - 50.000 = partially similar meaning but with important differences or missing info. - 0.000 = completely unrelated or very bad translation. CRITICAL FORMAT RULE: Return ONLY the numeric score with EXACTLY three digits after the decimal point. Do NOT include any words, punctuation, units, labels, or explanation.
User Message	Sentence A (English source): {sentence_a} Sentence B (Chinese translation): {sentence_b} Score (0.000 to 100.000 only):

Table 13: Full prompt template for the WMT English-Chinese quality estimation task.

Dataset: WMT_RU_EN	
System Prompt	You are an expert human annotator for translation quality / semantic textual similarity (STS). You will see sentence pairs where Sentence A is in Russian (source) and Sentence B is in English (translation). You must judge how well the meaning is preserved from Russian to English. Scoring instructions: - Output a real-valued translation quality / similarity score from 0.000 to 100.000. - 100.000 = perfect meaning preservation (excellent translation / paraphrases). - 50.000 = partially similar meaning but with important differences or missing info. - 0.000 = completely unrelated or very bad translation. CRITICAL FORMAT RULE: Return ONLY the numeric score with EXACTLY three digits after the decimal point. Do NOT include any words, punctuation, units, labels, or explanation.
User Message	Sentence A (Russian source): {sentence_a} Sentence B (English translation): {sentence_b} Score (0.000 to 100.000 only):

Table 14: Full prompt template for the WMT Russian-English quality estimation task.

Dataset: WMT_SI_EN	
System Prompt	You are an expert human annotator for translation quality / semantic textual similarity (STS). You will see sentence pairs where Sentence A is in Sinhala (source) and Sentence B is in English (translation). You must judge how well the meaning is preserved from Sinhala to English. Scoring instructions: - Output a real-valued translation quality / similarity score from 0.000 to 100.000. - 100.000 = perfect meaning preservation (excellent translation / paraphrases). - 50.000 = partially similar meaning but with important differences or missing info. - 0.000 = completely unrelated or very bad translation. CRITICAL FORMAT RULE: Return ONLY the numeric score with EXACTLY three digits after the decimal point. Do NOT include any words, punctuation, units, labels, or explanation.
User Message	Sentence A (Sinhala source): {sentence_a} Sentence B (English translation): {sentence_b} Score (0.000 to 100.000 only):

Table 15: Full prompt template for the WMT Sinhala-English quality estimation task.

Dataset: StatLLM	
System Prompt	You are a human expert in statistical programming using the Statistical Analysis System (SAS) language. You will see a statistical analysis task, dataset description, and SAS code. You must judge the quality of the provided SAS code within its intended context. Scoring instructions: - Output a real-valued total score from 0.000 to 50.000, inclusive. - 50.000 = excellent code: correct, executable, readable, concise, and produces the necessary clear output. - 25.000 = partially correct code: addresses some of the task but has important mistakes, omissions, or unclear output. - 0.000 = unusable code: does not address the task, is likely non-executable, or would produce wrong/missing output. CRITICAL FORMAT RULE: Return ONLY the numeric score with EXACTLY three digits after the decimal point. Do NOT include any words, punctuation, units, labels, or explanation.

User Message	Statistical analysis task:
	{problem_description}
	
	Dataset name: {dataset_name}
	
	Dataset description:
	{data_description}
	
	Generated SAS code:
	``sas
	{generated_sas_code}
	``
	
	Total score (0.000 to 50.000 only):

Table 16: Full prompt template for the StatLLM SAS code quality estimation task.

Table 17: Comparison on **STS-B** (Semantic Textual Similarity Benchmark). Metrics are mean ± std over 3 seeds and rounded to 3 decimals. Lower is better for RMSE (↓); higher is better for correlations (↑).

Dataset: STS-Benchmark (Semantic Textual Similarity Benchmark)				
LLM encoder	**Inference**	**RMSE ↓**	**Pearson ↑**	**Spearman ↑**
LLAMA 3.1 8B Instruct	Zero Shot	$1.215_{\pm 0.010}$	$0.717_{\pm 0.006}$	$0.688_{\pm 0.010}$
	Many Shot	$1.235_{\pm 0.045}$	$0.675_{\pm 0.031}$	$0.635_{\pm 0.030}$
	Linear	$1.323_{\pm 0.000}$	$0.632_{\pm 0.000}$	$0.637_{\pm 0.000}$
	MLP	$1.051_{\pm 0.015}$	$0.732_{\pm 0.009}$	$0.725_{\pm 0.010}$
	RAIL	$1.252_{\pm 0.007}$	$0.743_{\pm 0.002}$	$0.771_{\pm 0.003}$
	LoRA + Zero Shot	$0.788_{\pm 0.032}$	$0.858_{\pm 0.014}$	$0.853_{\pm 0.017}$
	LoRA + Linear	$1.294_{\pm 0.004}$	$0.647_{\pm 0.002}$	$0.651_{\pm 0.002}$
	LoRA + MLP	$1.030_{\pm 0.015}$	$0.745_{\pm 0.006}$	$0.738_{\pm 0.007}$
	LoRA + RAIL	$1.041_{\pm 0.029}$	$0.788_{\pm 0.026}$	$0.785_{\pm 0.021}$
	RAFT	$1.142_{\pm 0.052}$	$0.861_{\pm 0.006}$	$0.873_{\pm 0.004}$
	RELISH (Ours)	$0.669_{\pm 0.004}$	$0.905_{\pm 0.002}$	$0.904_{\pm 0.002}$
QWEN3 8B	Zero Shot	$0.935_{\pm 0.008}$	$0.857_{\pm 0.002}$	$0.864_{\pm 0.002}$
	Many Shot	$1.140_{\pm 0.029}$	$0.833_{\pm 0.008}$	$0.854_{\pm 0.003}$
	Linear	$1.241_{\pm 0.000}$	$0.667_{\pm 0.000}$	$0.683_{\pm 0.000}$
	MLP	$0.938_{\pm 0.008}$	$0.792_{\pm 0.002}$	$0.794_{\pm 0.001}$
	RAIL	$0.941_{\pm 0.000}$	$0.861_{\pm 0.000}$	$0.868_{\pm 0.000}$
	LoRA + Zero Shot	$0.793_{\pm 0.014}$	$0.857_{\pm 0.006}$	$0.853_{\pm 0.010}$
	LoRA + Linear	$1.235_{\pm 0.004}$	$0.670_{\pm 0.001}$	$0.682_{\pm 0.001}$
	LoRA + MLP	$0.945_{\pm 0.008}$	$0.788_{\pm 0.002}$	$0.790_{\pm 0.003}$
	LoRA + RAIL	$0.857_{\pm 0.032}$	$0.841_{\pm 0.011}$	$0.872_{\pm 0.010}$
	RAFT	$0.754_{\pm 0.018}$	$0.874_{\pm 0.005}$	$0.894_{\pm 0.003}$
	RELISH (Ours)	$0.638_{\pm 0.015}$	$0.912_{\pm 0.005}$	$0.911_{\pm 0.005}$
GEMMA 3 27b Instruct	Zero Shot	$0.721_{\pm 0.002}$	$0.890_{\pm 0.001}$	$0.883_{\pm 0.000}$
	Many Shot	$0.841_{\pm 0.022}$	$0.884_{\pm 0.009}$	$0.885_{\pm 0.009}$
	Linear	$1.477_{\pm 0.000}$	$0.632_{\pm 0.000}$	$0.655_{\pm 0.000}$
	MLP	$0.875_{\pm 0.004}$	$0.822_{\pm 0.002}$	$0.817_{\pm 0.003}$
	RAIL	$0.719_{\pm 0.000}$	$0.884_{\pm 0.000}$	$0.878_{\pm 0.000}$
	LoRA + Zero Shot	$0.720_{\pm 0.009}$	$0.884_{\pm 0.006}$	$0.884_{\pm 0.002}$
	LoRA + Linear	$1.460_{\pm 0.028}$	$0.637_{\pm 0.010}$	$0.657_{\pm 0.011}$
	LoRA + MLP	$0.878_{\pm 0.010}$	$0.818_{\pm 0.005}$	$0.814_{\pm 0.005}$
	LoRA + RAIL	$0.603_{\pm 0.025}$	$0.924_{\pm 0.003}$	$0.923_{\pm 0.003}$
	RAFT	$0.615_{\pm 0.015}$	$0.920_{\pm 0.002}$	$0.920_{\pm 0.000}$
	RELISH (Ours)	$0.589_{\pm 0.013}$	$0.926_{\pm 0.003}$	$0.923_{\pm 0.002}$
QWEN3 32B	Zero Shot	$0.796_{\pm 0.001}$	$0.876_{\pm 0.001}$	$0.874_{\pm 0.001}$
	Many Shot	$0.892_{\pm 0.016}$	$0.866_{\pm 0.004}$	$0.865_{\pm 0.007}$
	Linear	$1.408_{\pm 0.000}$	$0.627_{\pm 0.000}$	$0.639_{\pm 0.000}$
	MLP	$0.909_{\pm 0.003}$	$0.805_{\pm 0.002}$	$0.803_{\pm 0.002}$
	RAIL	$0.774_{\pm 0.001}$	$0.882_{\pm 0.000}$	$0.888_{\pm 0.000}$
	LoRA + Zero Shot	$0.838_{\pm 0.012}$	$0.842_{\pm 0.004}$	$0.837_{\pm 0.005}$
	LoRA + Linear	$1.414_{\pm 0.002}$	$0.628_{\pm 0.001}$	$0.643_{\pm 0.002}$
	LoRA + MLP	$0.907_{\pm 0.011}$	$0.807_{\pm 0.005}$	$0.803_{\pm 0.005}$
	LoRA + RAIL	$0.730_{\pm 0.001}$	$0.891_{\pm 0.000}$	$0.890_{\pm 0.001}$
	RAFT	$0.699_{\pm 0.003}$	$0.897_{\pm 0.000}$	$0.895_{\pm 0.000}$
	RELISH (Ours)	$0.595_{\pm 0.024}$	$0.925_{\pm 0.003}$	$0.924_{\pm 0.002}$

Table 18: Comparison on **SICKR_STS** (Sentences Involving Compositional Knowledge, Revised). Metrics are mean ± std over 3 seeds and rounded to 3 decimals. Lower is better for RMSE (↓); higher is better for correlations (↑).

Dataset: SICKR_STS (Sentences Involving Compositional Knowledge, Revised)				
LLM encoder	**Inference**	**RMSE ↓**	**Pearson ↑**	**Spearman ↑**
LLAMA 3.1 8B Instruct	Zero Shot	$1.095_{\pm 0.012}$	$0.666_{\pm 0.015}$	$0.650_{\pm 0.014}$
	Many Shot	$0.918_{\pm 0.090}$	$0.608_{\pm 0.019}$	$0.477_{\pm 0.036}$
	Linear	$0.612_{\pm 0.000}$	$0.807_{\pm 0.000}$	$0.764_{\pm 0.000}$
	MLP	$0.509_{\pm 0.007}$	$0.858_{\pm 0.004}$	$0.809_{\pm 0.007}$
	RAIL	$1.126_{\pm 0.003}$	$0.686_{\pm 0.002}$	$0.746_{\pm 0.002}$
	LoRA + Zero Shot	$0.489_{\pm 0.017}$	$0.878_{\pm 0.008}$	$0.822_{\pm 0.007}$
	LoRA + Linear	$0.581_{\pm 0.004}$	$0.826_{\pm 0.001}$	$0.787_{\pm 0.002}$
	LoRA + MLP	$0.488_{\pm 0.005}$	$0.869_{\pm 0.004}$	$0.824_{\pm 0.007}$
	LoRA + RAIL	$0.656_{\pm 0.045}$	$0.825_{\pm 0.011}$	$0.762_{\pm 0.015}$
	RAFT	$0.754_{\pm 0.021}$	$0.879_{\pm 0.008}$	$0.847_{\pm 0.009}$
	RELISH (Ours)	$0.413_{\pm 0.027}$	$0.924_{\pm 0.001}$	$0.881_{\pm 0.002}$
QWEN3 8B	Zero Shot	$1.216_{\pm 0.005}$	$0.708_{\pm 0.002}$	$0.752_{\pm 0.001}$
	Many Shot	$0.679_{\pm 0.027}$	$0.820_{\pm 0.011}$	$0.801_{\pm 0.019}$
	Linear	$0.579_{\pm 0.000}$	$0.829_{\pm 0.000}$	$0.778_{\pm 0.000}$
	MLP	$0.483_{\pm 0.007}$	$0.877_{\pm 0.002}$	$0.829_{\pm 0.002}$
	RAIL	$1.204_{\pm 0.001}$	$0.717_{\pm 0.000}$	$0.763_{\pm 0.001}$
	LoRA + Zero Shot	$0.727_{\pm 0.034}$	$0.789_{\pm 0.050}$	$0.732_{\pm 0.064}$
	LoRA + Linear	$0.575_{\pm 0.002}$	$0.831_{\pm 0.001}$	$0.782_{\pm 0.002}$
	LoRA + MLP	$0.484_{\pm 0.008}$	$0.876_{\pm 0.002}$	$0.828_{\pm 0.002}$
	LoRA + RAIL	$0.774_{\pm 0.065}$	$0.738_{\pm 0.110}$	$0.732_{\pm 0.065}$
	RAFT	$0.484_{\pm 0.049}$	$0.904_{\pm 0.010}$	$0.869_{\pm 0.005}$
	RELISH (Ours)	$0.389_{\pm 0.008}$	$0.918_{\pm 0.002}$	$0.874_{\pm 0.003}$
GEMMA 3 27b Instruct	Zero Shot	$1.241_{\pm 0.005}$	$0.727_{\pm 0.002}$	$0.746_{\pm 0.002}$
	Many Shot	$0.501_{\pm 0.011}$	$0.875_{\pm 0.007}$	$0.838_{\pm 0.005}$
	Linear	$0.633_{\pm 0.000}$	$0.808_{\pm 0.000}$	$0.758_{\pm 0.000}$
	MLP	$0.479_{\pm 0.013}$	$0.876_{\pm 0.006}$	$0.829_{\pm 0.007}$
	RAIL	$1.214_{\pm 0.000}$	$0.715_{\pm 0.000}$	$0.730_{\pm 0.000}$
	LoRA + Zero Shot	$0.458_{\pm 0.017}$	$0.886_{\pm 0.011}$	$0.830_{\pm 0.008}$
	LoRA + Linear	$0.618_{\pm 0.002}$	$0.815_{\pm 0.001}$	$0.770_{\pm 0.002}$
	LoRA + MLP	$0.472_{\pm 0.013}$	$0.880_{\pm 0.006}$	$0.832_{\pm 0.007}$
	LoRA + RAIL	$0.362_{\pm 0.002}$	$0.929_{\pm 0.001}$	$0.883_{\pm 0.001}$
	RAFT	$0.368_{\pm 0.004}$	$0.929_{\pm 0.002}$	$0.883_{\pm 0.001}$
	RELISH (Ours)	$0.359_{\pm 0.004}$	$0.930_{\pm 0.001}$	$0.885_{\pm 0.001}$
QWEN3 32B	Zero Shot	$1.231_{\pm 0.008}$	$0.708_{\pm 0.003}$	$0.727_{\pm 0.005}$
	Many Shot	$0.564_{\pm 0.064}$	$0.849_{\pm 0.016}$	$0.820_{\pm 0.012}$
	Linear	$0.602_{\pm 0.000}$	$0.813_{\pm 0.000}$	$0.755_{\pm 0.000}$
	MLP	$0.484_{\pm 0.005}$	$0.873_{\pm 0.003}$	$0.826_{\pm 0.008}$
	RAIL	$1.170_{\pm 0.000}$	$0.726_{\pm 0.001}$	$0.747_{\pm 0.001}$
	LoRA + Zero Shot	$0.497_{\pm 0.024}$	$0.874_{\pm 0.014}$	$0.820_{\pm 0.007}$
	LoRA + Linear	$0.602_{\pm 0.002}$	$0.816_{\pm 0.002}$	$0.758_{\pm 0.005}$
	LoRA + MLP	$0.493_{\pm 0.016}$	$0.868_{\pm 0.006}$	$0.820_{\pm 0.008}$
	LoRA + RAIL	$0.528_{\pm 0.180}$	$0.885_{\pm 0.060}$	$0.859_{\pm 0.041}$
	RAFT	$0.398_{\pm 0.023}$	$0.894_{\pm 0.049}$	$0.860_{\pm 0.040}$
	RELISH (Ours)	$0.393_{\pm 0.006}$	$0.924_{\pm 0.002}$	$0.888_{\pm 0.003}$

Table 19: Comparison on **WMT_RU_EN** (Russian → English). Metrics are mean ± std over 3 seeds and rounded to 3 decimals. Lower is better for RMSE (↓); higher is better for correlations (↑).

Dataset: WMT_RU_EN (Russian → English)				
LLM encoder	**Inference**	**RMSE ↓**	**Pearson ↑**	**Spearman ↑**
LLAMA 3.1 8B Instruct	Zero Shot	$32.853_{\pm 0.160}$	$0.473_{\pm 0.008}$	$0.409_{\pm 0.012}$
	Many Shot	$31.481_{\pm 1.342}$	$0.198_{\pm 0.106}$	$0.133_{\pm 0.098}$
	Linear	$22.637_{\pm 0.000}$	$0.619_{\pm 0.000}$	$0.567_{\pm 0.000}$
	MLP	$19.161_{\pm 0.045}$	$0.695_{\pm 0.001}$	$0.646_{\pm 0.004}$
	RAIL	$33.237_{\pm 0.175}$	$0.528_{\pm 0.005}$	$0.530_{\pm 0.007}$
	LoRA + Zero Shot	$20.264_{\pm 0.426}$	$0.718_{\pm 0.018}$	$0.633_{\pm 0.018}$
	LoRA + Linear	$22.385_{\pm 0.101}$	$0.627_{\pm 0.001}$	$0.575_{\pm 0.001}$
	LoRA + MLP	$18.910_{\pm 0.076}$	$0.704_{\pm 0.001}$	$0.655_{\pm 0.005}$
	LoRA + RAIL	$19.661_{\pm 1.973}$	$0.706_{\pm 0.048}$	$0.672_{\pm 0.026}$
	RAFT	$23.410_{\pm 0.572}$	$0.740_{\pm 0.018}$	$0.719_{\pm 0.027}$
	RELISH (Ours)	$16.711_{\pm 1.219}$	$0.796_{\pm 0.010}$	$0.752_{\pm 0.004}$
QWEN3 8B	Zero Shot	$27.977_{\pm 0.052}$	$0.556_{\pm 0.008}$	$0.625_{\pm 0.007}$
	Many Shot	$31.292_{\pm 0.658}$	$0.490_{\pm 0.054}$	$0.574_{\pm 0.090}$
	Linear	$22.492_{\pm 0.000}$	$0.636_{\pm 0.000}$	$0.592_{\pm 0.000}$
	MLP	$19.336_{\pm 0.142}$	$0.695_{\pm 0.007}$	$0.644_{\pm 0.002}$
	RAIL	$28.246_{\pm 0.012}$	$0.574_{\pm 0.001}$	$0.663_{\pm 0.003}$
	LoRA + Zero Shot	$24.112_{\pm 0.256}$	$0.565_{\pm 0.012}$	$0.541_{\pm 0.007}$
	LoRA + Linear	$22.376_{\pm 0.042}$	$0.640_{\pm 0.001}$	$0.595_{\pm 0.000}$
	LoRA + MLP	$19.210_{\pm 0.158}$	$0.699_{\pm 0.007}$	$0.648_{\pm 0.002}$
	LoRA + RAIL	$18.203_{\pm 0.640}$	$0.745_{\pm 0.023}$	$0.717_{\pm 0.027}$
	RAFT	$20.834_{\pm 0.809}$	$0.647_{\pm 0.033}$	$0.522_{\pm 0.041}$
	RELISH (Ours)	$16.560_{\pm 0.144}$	$0.785_{\pm 0.015}$	$0.749_{\pm 0.013}$
GEMMA 3 27b Instruct	Zero Shot	$27.124_{\pm 0.113}$	$0.692_{\pm 0.005}$	$0.694_{\pm 0.002}$
	Many Shot	$19.757_{\pm 1.330}$	$0.754_{\pm 0.021}$	$0.721_{\pm 0.027}$
	Linear	$25.315_{\pm 0.000}$	$0.599_{\pm 0.000}$	$0.539_{\pm 0.000}$
	MLP	$17.273_{\pm 0.043}$	$0.759_{\pm 0.003}$	$0.706_{\pm 0.005}$
	RAIL	$27.445_{\pm 0.028}$	$0.687_{\pm 0.001}$	$0.720_{\pm 0.001}$
	LoRA + Zero Shot	$19.241_{\pm 0.185}$	$0.729_{\pm 0.007}$	$0.680_{\pm 0.003}$
	LoRA + Linear	$25.625_{\pm 0.043}$	$0.597_{\pm 0.003}$	$0.555_{\pm 0.008}$
	LoRA + MLP	$17.300_{\pm 0.059}$	$0.759_{\pm 0.001}$	$0.700_{\pm 0.002}$
	LoRA + RAIL	$16.709_{\pm 0.336}$	$0.766_{\pm 0.030}$	$0.763_{\pm 0.016}$
	RAFT	$15.545_{\pm 0.223}$	$0.813_{\pm 0.002}$	$0.771_{\pm 0.023}$
	RELISH (Ours)	$15.126_{\pm 0.320}$	$0.830_{\pm 0.003}$	$0.807_{\pm 0.001}$
QWEN3 32B	Zero Shot	$26.487_{\pm 0.220}$	$0.699_{\pm 0.009}$	$0.703_{\pm 0.003}$
	Many Shot	$21.403_{\pm 1.086}$	$0.763_{\pm 0.014}$	$0.746_{\pm 0.012}$
	Linear	$24.908_{\pm 0.000}$	$0.602_{\pm 0.000}$	$0.564_{\pm 0.000}$
	MLP	$18.346_{\pm 0.036}$	$0.725_{\pm 0.002}$	$0.687_{\pm 0.003}$
	RAIL	$26.402_{\pm 0.024}$	$0.721_{\pm 0.001}$	$0.749_{\pm 0.001}$
	LoRA + Zero Shot	$22.409_{\pm 0.353}$	$0.639_{\pm 0.014}$	$0.601_{\pm 0.019}$
	LoRA + Linear	$24.970_{\pm 0.114}$	$0.597_{\pm 0.003}$	$0.569_{\pm 0.002}$
	LoRA + MLP	$18.419_{\pm 0.103}$	$0.723_{\pm 0.003}$	$0.689_{\pm 0.002}$
	LoRA + RAIL	$22.618_{\pm 0.014}$	$0.740_{\pm 0.002}$	$0.728_{\pm 0.000}$
	RAFT	$20.373_{\pm 0.101}$	$0.759_{\pm 0.001}$	$0.740_{\pm 0.001}$
	RELISH (Ours)	$14.464_{\pm 1.328}$	$0.856_{\pm 0.008}$	$0.816_{\pm 0.001}$

Table 20: Comparison on **WMT_EN_ZH** (English → Chinese). Metrics are mean ± std over 3 seeds and rounded to 3 decimals. Lower is better for RMSE (↓); higher is better for correlations (↑).

Dataset: **WMT_EN_ZH** (English → Chinese)				
LLM encoder	**Inference**	**RMSE ↓**	**Pearson ↑**	**Spearman ↑**
LLAMA 3.1 8B Instruct	Zero Shot	$33.543_{\pm 0.217}$	$0.348_{\pm 0.013}$	$0.276_{\pm 0.018}$
	Many Shot	$26.312_{\pm 1.674}$	$0.075_{\pm 0.010}$	$0.042_{\pm 0.013}$
	Linear	$19.648_{\pm 0.000}$	$0.277_{\pm 0.000}$	$0.253_{\pm 0.000}$
	MLP	$14.198_{\pm 0.057}$	$0.331_{\pm 0.012}$	$0.321_{\pm 0.011}$
	RAIL	$38.840_{\pm 0.096}$	$0.345_{\pm 0.004}$	$0.367_{\pm 0.002}$
	LoRA + Zero Shot	$17.827_{\pm 0.338}$	$0.345_{\pm 0.033}$	$0.335_{\pm 0.003}$
	LoRA + Linear	$19.434_{\pm 0.034}$	$0.285_{\pm 0.001}$	$0.259_{\pm 0.001}$
	LoRA + MLP	$14.183_{\pm 0.040}$	$0.341_{\pm 0.010}$	$0.329_{\pm 0.010}$
	LoRA + RAIL	$15.238_{\pm 1.016}$	$0.347_{\pm 0.037}$	$0.368_{\pm 0.008}$
	RAFT	$13.060_{\pm 0.169}$	$0.497_{\pm 0.012}$	$0.499_{\pm 0.012}$
	RELISH (Ours)	$9.555_{\pm 0.062}$	$0.524_{\pm 0.006}$	$0.521_{\pm 0.007}$
QWEN3 8B	Zero Shot	$28.012_{\pm 0.156}$	$0.381_{\pm 0.004}$	$0.433_{\pm 0.007}$
	Many Shot	$27.780_{\pm 0.743}$	$0.370_{\pm 0.012}$	$0.389_{\pm 0.030}$
	Linear	$19.608_{\pm 0.000}$	$0.291_{\pm 0.000}$	$0.283_{\pm 0.000}$
	MLP	$14.459_{\pm 0.126}$	$0.325_{\pm 0.009}$	$0.315_{\pm 0.008}$
	RAIL	$28.603_{\pm 0.025}$	$0.384_{\pm 0.001}$	$0.450_{\pm 0.003}$
	LoRA + Zero Shot	$19.322_{\pm 0.203}$	$0.279_{\pm 0.026}$	$0.261_{\pm 0.004}$
	LoRA + Linear	$19.539_{\pm 0.043}$	$0.295_{\pm 0.002}$	$0.285_{\pm 0.001}$
	LoRA + MLP	$14.395_{\pm 0.088}$	$0.323_{\pm 0.010}$	$0.313_{\pm 0.009}$
	LoRA + RAIL	$17.809_{\pm 1.278}$	$0.409_{\pm 0.025}$	$0.430_{\pm 0.033}$
	RAFT	$19.431_{\pm 0.139}$	$0.376_{\pm 0.007}$	$0.346_{\pm 0.007}$
	RELISH (Ours)	$12.386_{\pm 0.921}$	$0.464_{\pm 0.010}$	$0.478_{\pm 0.018}$
GEMMA 3 27b Instruct	Zero Shot	$28.496_{\pm 0.006}$	$0.389_{\pm 0.006}$	$0.394_{\pm 0.002}$
	Many Shot	$21.105_{\pm 0.975}$	$0.396_{\pm 0.019}$	$0.406_{\pm 0.016}$
	Linear	$25.082_{\pm 0.000}$	$0.160_{\pm 0.000}$	$0.116_{\pm 0.000}$
	MLP	$14.567_{\pm 0.076}$	$0.305_{\pm 0.006}$	$0.295_{\pm 0.005}$
	RAIL	$27.340_{\pm 0.009}$	$0.373_{\pm 0.002}$	$0.372_{\pm 0.001}$
	LoRA + Zero Shot	$19.245_{\pm 0.327}$	$0.322_{\pm 0.028}$	$0.306_{\pm 0.029}$
	LoRA + Linear	$25.617_{\pm 0.360}$	$0.174_{\pm 0.009}$	$0.159_{\pm 0.008}$
	LoRA + MLP	$14.626_{\pm 0.054}$	$0.310_{\pm 0.008}$	$0.298_{\pm 0.007}$
	LoRA + RAIL	$17.318_{\pm 1.510}$	$0.457_{\pm 0.029}$	$0.493_{\pm 0.008}$
	RAFT	$13.242_{\pm 1.656}$	$0.529_{\pm 0.006}$	$0.521_{\pm 0.006}$
	RELISH (Ours)	$11.941_{\pm 2.003}$	$0.541_{\pm 0.009}$	$0.530_{\pm 0.005}$
QWEN3 32B	Zero Shot	$27.628_{\pm 0.056}$	$0.421_{\pm 0.006}$	$0.432_{\pm 0.007}$
	Many Shot	$25.149_{\pm 0.998}$	$0.433_{\pm 0.025}$	$0.434_{\pm 0.018}$
	Linear	$22.955_{\pm 0.000}$	$0.246_{\pm 0.000}$	$0.236_{\pm 0.000}$
	MLP	$14.048_{\pm 0.134}$	$0.372_{\pm 0.015}$	$0.354_{\pm 0.011}$
	RAIL	$27.796_{\pm 0.023}$	$0.419_{\pm 0.001}$	$0.456_{\pm 0.002}$
	LoRA + Zero Shot	$19.358_{\pm 0.410}$	$0.272_{\pm 0.022}$	$0.237_{\pm 0.019}$
	LoRA + Linear	$22.394_{\pm 0.086}$	$0.270_{\pm 0.002}$	$0.241_{\pm 0.001}$
	LoRA + MLP	$14.037_{\pm 0.179}$	$0.361_{\pm 0.030}$	$0.344_{\pm 0.024}$
	LoRA + RAIL	$23.888_{\pm 0.141}$	$0.465_{\pm 0.001}$	$0.462_{\pm 0.001}$
	RAFT	$13.012_{\pm 1.022}$	$0.477_{\pm 0.001}$	$0.472_{\pm 0.000}$
	RELISH (Ours)	$11.759_{\pm 0.812}$	$0.567_{\pm 0.004}$	$0.554_{\pm 0.004}$

Table 21: Comparison on **WMT_SI_EN** (Sinhala → English). Metrics are mean ± std over 3 seeds and rounded to 3 decimals. Lower is better for RMSE (↓); higher is better for correlations (↑).

Dataset: WMT_SI_EN (Sinhala → English)				
LLM encoder	**Inference**	**RMSE ↓**	**Pearson ↑**	**Spearman ↑**
LLAMA 3.1 8B Instruct	Zero Shot	$40.343_{\pm 0.720}$	$0.433_{\pm 0.028}$	$0.427_{\pm 0.020}$
	Many Shot	$35.311_{\pm 3.971}$	$0.186_{\pm 0.020}$	$0.149_{\pm 0.054}$
	Linear	$29.768_{\pm 0.000}$	$0.368_{\pm 0.000}$	$0.348_{\pm 0.000}$
	MLP	$19.460_{\pm 0.198}$	$0.521_{\pm 0.004}$	$0.477_{\pm 0.004}$
	RAIL	$44.966_{\pm 0.055}$	$0.438_{\pm 0.001}$	$0.481_{\pm 0.010}$
	LoRA + Zero Shot	$28.894_{\pm 1.483}$	$0.448_{\pm 0.041}$	$0.438_{\pm 0.046}$
	LoRA + Linear	$30.483_{\pm 0.089}$	$0.361_{\pm 0.001}$	$0.348_{\pm 0.002}$
	LoRA + MLP	$19.570_{\pm 0.275}$	$0.518_{\pm 0.006}$	$0.475_{\pm 0.004}$
	LoRA + RAIL	$34.396_{\pm 4.982}$	$0.315_{\pm 0.018}$	$0.319_{\pm 0.021}$
	RAFT	$18.411_{\pm 1.496}$	$0.507_{\pm 0.012}$	$0.476_{\pm 0.010}$
	RELISH (Ours)	$18.152_{\pm 0.474}$	$0.630_{\pm 0.016}$	$0.600_{\pm 0.008}$
QWEN3 8B	Zero Shot	$28.314_{\pm 0.161}$	$0.472_{\pm 0.007}$	$0.471_{\pm 0.008}$
	Many Shot	$38.964_{\pm 1.962}$	$0.383_{\pm 0.035}$	$0.376_{\pm 0.046}$
	Linear	$30.422_{\pm 0.000}$	$0.345_{\pm 0.000}$	$0.313_{\pm 0.000}$
	MLP	$20.558_{\pm 0.108}$	$0.483_{\pm 0.005}$	$0.436_{\pm 0.006}$
	RAIL	$28.305_{\pm 0.034}$	$0.493_{\pm 0.000}$	$0.516_{\pm 0.002}$
	LoRA + Zero Shot	$29.880_{\pm 0.190}$	$0.334_{\pm 0.021}$	$0.319_{\pm 0.023}$
	LoRA + Linear	$30.448_{\pm 0.121}$	$0.349_{\pm 0.004}$	$0.318_{\pm 0.004}$
	LoRA + MLP	$20.553_{\pm 0.120}$	$0.482_{\pm 0.006}$	$0.435_{\pm 0.005}$
	LoRA + RAIL	$27.689_{\pm 0.447}$	$0.432_{\pm 0.015}$	$0.435_{\pm 0.014}$
	RAFT	$21.034_{\pm 3.215}$	$0.564_{\pm 0.017}$	$0.529_{\pm 0.002}$
	RELISH (Ours)	$17.506_{\pm 0.286}$	$0.586_{\pm 0.011}$	$0.545_{\pm 0.007}$
GEMMA 3 27b Instruct	Zero Shot	$21.685_{\pm 0.137}$	$0.612_{\pm 0.003}$	$0.587_{\pm 0.004}$
	Many Shot	$25.006_{\pm 1.386}$	$0.587_{\pm 0.012}$	$0.576_{\pm 0.011}$
	Linear	$39.492_{\pm 0.000}$	$0.298_{\pm 0.000}$	$0.286_{\pm 0.000}$
	MLP	$19.024_{\pm 0.067}$	$0.558_{\pm 0.003}$	$0.523_{\pm 0.003}$
	RAIL	$22.617_{\pm 0.009}$	$0.583_{\pm 0.002}$	$0.570_{\pm 0.002}$
	LoRA + Zero Shot	$29.368_{\pm 0.522}$	$0.410_{\pm 0.017}$	$0.401_{\pm 0.014}$
	LoRA + Linear	$38.217_{\pm 0.058}$	$0.307_{\pm 0.004}$	$0.298_{\pm 0.002}$
	LoRA + MLP	$19.048_{\pm 0.051}$	$0.561_{\pm 0.001}$	$0.522_{\pm 0.003}$
	LoRA + RAIL	$21.554_{\pm 0.063}$	$0.591_{\pm 0.002}$	$0.566_{\pm 0.003}$
	RAFT	$19.892_{\pm 0.004}$	$0.623_{\pm 0.000}$	$0.592_{\pm 0.000}$
	RELISH (Ours)	$16.384_{\pm 0.555}$	$0.669_{\pm 0.005}$	$0.633_{\pm 0.005}$
QWEN3 32B	Zero Shot	$21.809_{\pm 0.411}$	$0.568_{\pm 0.019}$	$0.536_{\pm 0.020}$
	Many Shot	$26.538_{\pm 0.839}$	$0.524_{\pm 0.008}$	$0.505_{\pm 0.009}$
	Linear	$36.996_{\pm 0.000}$	$0.314_{\pm 0.000}$	$0.298_{\pm 0.000}$
	MLP	$19.405_{\pm 0.066}$	$0.541_{\pm 0.004}$	$0.496_{\pm 0.003}$
	RAIL	$20.587_{\pm 0.031}$	$0.570_{\pm 0.001}$	$0.556_{\pm 0.004}$
	LoRA + Zero Shot	$29.508_{\pm 0.120}$	$0.407_{\pm 0.009}$	$0.399_{\pm 0.008}$
	LoRA + Linear	$37.975_{\pm 0.099}$	$0.298_{\pm 0.001}$	$0.277_{\pm 0.001}$
	LoRA + MLP	$19.504_{\pm 0.025}$	$0.533_{\pm 0.005}$	$0.489_{\pm 0.003}$
	LoRA + RAIL	$22.659_{\pm 0.117}$	$0.587_{\pm 0.002}$	$0.557_{\pm 0.001}$
	RAFT	$19.616_{\pm 0.045}$	$0.610_{\pm 0.000}$	$0.569_{\pm 0.000}$
	RELISH (Ours)	$17.110_{\pm 0.148}$	$0.658_{\pm 0.004}$	$0.621_{\pm 0.003}$

Table 22: Comparison on **StatLLM** (code quality estimation). Metrics are mean ± std over 3 seeds and rounded to 3 decimals. Lower is better for RMSE (↓); higher is better for correlations (↑).

Dataset: StatLLM				
LLM encoder	**Inference**	**RMSE ↓**	**Pearson ↑**	**Spearman ↑**
LLAMA 3.1 8B Instruct	Zero Shot	$26.170_{\pm 0.121}$	$0.060_{\pm 0.025}$	$0.038_{\pm 0.013}$
	Many Shot	$13.988_{\pm 0.777}$	$0.052_{\pm 0.072}$	$0.067_{\pm 0.078}$
	Linear	$20.478_{\pm 0.000}$	$0.022_{\pm 0.000}$	$0.032_{\pm 0.000}$
	MLP	$23.720_{\pm 4.167}$	$0.158_{\pm 0.000}$	$0.153_{\pm 0.000}$
	RAIL	$27.078_{\pm 0.112}$	$0.020_{\pm 0.021}$	$0.039_{\pm 0.019}$
	LoRA + Zero Shot	$15.838_{\pm 1.675}$	$0.222_{\pm 0.063}$	$0.176_{\pm 0.088}$
	LoRA + Linear	$17.452_{\pm 2.971}$	$0.022_{\pm 0.000}$	$0.032_{\pm 0.000}$
	LoRA + MLP	$26.890_{\pm 4.574}$	$0.158_{\pm 0.000}$	$0.153_{\pm 0.000}$
	LoRA + RAIL	$13.657_{\pm 0.395}$	$0.234_{\pm 0.061}$	$0.256_{\pm 0.048}$
	RAFT	$12.746_{\pm 1.029}$	$0.344_{\pm 0.099}$	$0.357_{\pm 0.078}$
	RELISH (Ours)	$9.177_{\pm 0.096}$	$0.557_{\pm 0.016}$	$0.584_{\pm 0.023}$
QWEN3 8B	Zero Shot	$13.996_{\pm 0.049}$	$0.239_{\pm 0.026}$	$0.245_{\pm 0.005}$
	Many Shot	$13.779_{\pm 0.453}$	$0.232_{\pm 0.087}$	$0.305_{\pm 0.173}$
	Linear	$18.659_{\pm 0.000}$	$0.022_{\pm 0.000}$	$0.032_{\pm 0.000}$
	MLP	$22.834_{\pm 3.309}$	$0.158_{\pm 0.000}$	$0.153_{\pm 0.000}$
	RAIL	$14.569_{\pm 0.081}$	$0.358_{\pm 0.035}$	$0.395_{\pm 0.004}$
	LoRA + Zero Shot	$14.532_{\pm 0.104}$	$0.377_{\pm 0.003}$	$0.423_{\pm 0.009}$
	LoRA + Linear	$11.324_{\pm 0.398}$	$0.022_{\pm 0.000}$	$0.032_{\pm 0.000}$
	LoRA + MLP	$19.687_{\pm 1.129}$	$0.158_{\pm 0.000}$	$0.153_{\pm 0.000}$
	LoRA + RAIL	$11.912_{\pm 0.087}$	$0.375_{\pm 0.008}$	$0.497_{\pm 0.004}$
	RAFT	$10.204_{\pm 0.131}$	$0.396_{\pm 0.005}$	$0.494_{\pm 0.030}$
	RELISH (Ours)	$9.875_{\pm 0.192}$	$0.465_{\pm 0.030}$	$0.507_{\pm 0.022}$
GEMMA 3 27b Instruct	Zero Shot	$11.386_{\pm 0.025}$	$0.298_{\pm 0.003}$	$0.373_{\pm 0.005}$
	Many Shot	$13.465_{\pm 0.343}$	$0.296_{\pm 0.038}$	$0.488_{\pm 0.035}$
	Linear	$13.346_{\pm 0.000}$	$0.022_{\pm 0.000}$	$0.032_{\pm 0.000}$
	MLP	$20.110_{\pm 0.930}$	$0.105_{\pm 0.091}$	$0.102_{\pm 0.088}$
	RAIL	$10.966_{\pm 0.019}$	$0.337_{\pm 0.003}$	$0.390_{\pm 0.005}$
	LoRA + Zero Shot	$13.543_{\pm 0.876}$	$0.419_{\pm 0.027}$	$0.526_{\pm 0.031}$
	LoRA + Linear	$17.684_{\pm 0.829}$	$0.064_{\pm 0.013}$	$0.033_{\pm 0.011}$
	LoRA + MLP	$18.990_{\pm 1.185}$	$0.040_{\pm 0.035}$	$0.066_{\pm 0.008}$
	LoRA + RAIL	$12.362_{\pm 1.111}$	$0.461_{\pm 0.037}$	$0.560_{\pm 0.025}$
	RAFT	$10.581_{\pm 0.258}$	$0.427_{\pm 0.030}$	$0.499_{\pm 0.020}$
	RELISH (Ours)	$9.584_{\pm 0.242}$	$0.522_{\pm 0.033}$	$0.569_{\pm 0.030}$
QWEN3 32B	Zero Shot	$13.064_{\pm 0.040}$	$0.406_{\pm 0.021}$	$0.498_{\pm 0.016}$
	Many Shot	$12.515_{\pm 0.686}$	$0.401_{\pm 0.014}$	$0.529_{\pm 0.012}$
	Linear	$12.462_{\pm 0.000}$	$0.022_{\pm 0.000}$	$0.032_{\pm 0.000}$
	MLP	$25.223_{\pm 8.128}$	$0.158_{\pm 0.000}$	$0.153_{\pm 0.000}$
	RAIL	$12.700_{\pm 0.052}$	$0.387_{\pm 0.003}$	$0.458_{\pm 0.004}$
	LoRA + Zero Shot	$12.695_{\pm 0.790}$	$0.378_{\pm 0.054}$	$0.551_{\pm 0.028}$
	LoRA + Linear	$29.472_{\pm 5.804}$	$0.007_{\pm 0.012}$	$0.011_{\pm 0.019}$
	LoRA + MLP	$26.600_{\pm 5.662}$	$0.105_{\pm 0.091}$	$0.102_{\pm 0.088}$
	LoRA + RAIL	$12.072_{\pm 0.593}$	$0.394_{\pm 0.028}$	$0.576_{\pm 0.012}$
	RAFT	$11.003_{\pm 0.083}$	$0.407_{\pm 0.006}$	$0.529_{\pm 0.012}$
	RELISH (Ours)	$8.246_{\pm 0.298}$	$0.678_{\pm 0.009}$	$0.684_{\pm 0.019}$