Training-Free versus Training-Based Intent Classification in LLMs: Accuracy, Robustness, and Failure Modes

Nan Chen* **Zhouhao Yang*** **Soufiane Hayou**
Department of Applied Mathematics and Statistics
Johns Hopkins University

Abstract

Intent classification in Large Language Models (LLMs) involves categorizing user prompts into predefined classes. For instance, given a user prompt, the system must determine whether it primarily concerns mathematics, coding, or general text processing. Such classification enables routing prompts to specialized models optimized for specific domains, improving both accuracy and computational efficiency. In this work, we conduct a systematic study comparing training-free vs training-based approaches for intent classification. For this purpose, we consider two lightweight, training-free methods based on statistics of internal representations and compare them against MLP classifiers and linear probes. Our comprehensive empirical evaluation reveals that 1) Both training-free and training-based methods saturate easy benchmarks (mathematics vs. coding vs. natural language), 2) Training-based classifiers have an advantage on harder classification tasks (e.g. Java vs Python), and 3) Training-free methods are generally more robust to mixed-intent and adversarial prompts.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse tasks, including mathematical reasoning (Wei et al., 2022; Yao et al., 2023; Gao et al., 2023) and code generation (Li et al., 2022; Guo et al., 2024; Zhu et al., 2024). At the same time, modern LLM systems are increasingly deployed through routing architectures, where an intent classifier first identifies the user's intent and then dispatches the prompt to an appropriate downstream model or tool (OpenAI, 2025; Bocklisch et al., 2017; Bunk et al., 2020; Arora et al., 2024). Such routing can improve the efficiency and reliability of production-scale systems, but it also raises a basic question: *what kind of intent classifier should be used?*

Existing approaches to intent classification face a difficult trade-off. Direct LLM calls are flexible, but they are hard to calibrate and less convenient when reliable uncertainty estimates are needed (Bang et al., 2023; Banerjee et al., 2025). Dedicated supervised classifiers (Larson et al., 2019; Chen et al., 2019) can achieve strong task-specific performance, but they might require extensive training data and computational resources, and need retraining when the intent label space evolves. In this work, we focus on two types of lightweight intent classifiers, training-free versus training-based methods[1], and conduct a systematic empirical study to study the accuracy, robustness, and failure modes of each approach.

For training-free methods, we introduce VecStat and NormStat, which operate entirely in the prefill phase with negligible cost. The motivation is the observation that different prompt types (mathematics, coding, general text, etc.) induce distinct activation distributions. Specifically, VecStat and NormStat represent two levels of statistical compression: VecStat preserves coordinate-wise directional information, while NormStat compresses features into radial summaries and uses substantially less memory. For training-based methods, we

*Equal contribution. Corresponding authors: {nchen38,zyang145}@jh.edu

[1]We use the wording training-free vs. training-based methods to refer to whether gradient updates are needed.

consider an MLP classifier inspired by sentence-classification pipelines (Casanueva et al., 2020; Jiang et al., 2024), applied to the LLM's final projection layer. The MLP head is trained on labeled data, whereas the training-free methods require no gradient updates, only a simple statistical estimation.

We conduct extensive empirical analysis to compare the two approaches. We apply these methods to LLMs ranging from 1B to 32B parameters and evaluate intent classification at both coarse-grained and fine-grained levels across seven benchmark datasets. The empirical results reveal that *there is no one-fits-all method for intent classification*. On easy coarse-grained tasks, both training-free and training-based methods often saturate the benchmark, while on harder fine-grained tasks, training-based classifiers typically achieve higher accuracy. At the same time, this advantage does not necessarily persist under our stress tests: when prompts contain mixed intent, training-free methods provide better uncertainty estimates, and when prompts are adversarially rephrased to inject misleading content, they tend to remain more stable than the trained-based methods. Our contributions are:

- We introduce two lightweight statistical methods, NormStat and VecStat, that operate on prefill-phase, and develop theory explaining their respective strengths. We show that VecStat is advantageous when class separation is primarily directional, while NormStat is sufficient in coarser regimes where radial information is rich enough to separate data, with the additional benefit of cheaper calibration.
- We conduct a systematic empirical study comparing training-free and training-based intent classification across seven LLMs (1B–32B parameters) at both coarse and fine levels of granularity. Results show both paradigms saturate on easy tasks, while training-based methods typically outperform training-free ones on harder, fine-grained distinctions.
- We create a mixed-intent dataset and an adversarial prompt dataset to stress-test intent classifiers under mixed-intent ambiguity and misleading surface cues. We find that training-free methods are typically more robust to mixed-intent prompts and adversarial rephrasings, making them potentially attractive for practical LLM routing settings where inputs are often noisy or ambiguous.

1.1 Related Work

Task Classification Intent classification maps a user prompt to a predefined label. Classical approaches either (i) train supervised classifiers over tokenized utterances (e.g., CNNs) to produce a distribution over intents (Hashemi et al., 2016; Goo et al., 2018; He et al., 2019), or (ii) fine-tune contextual encoders, particularly BERT-based models, where hidden states feed specialized intent classification heads, often jointly trained with slot filling tasks (Chen et al., 2019; Bocklisch et al., 2017; Bunk et al., 2020). In modern LLM-based systems, intent classification serves as a critical routing mechanism that allows the selection of appropriate downstream tools and models, enforces guardrails and fallback policies, and optimizes inference cost and latency (Souha et al., 2023; Arora et al., 2024). The predominant approach involves direct LLM inference through several key techniques (Liu et al., 2023; Rodriguez et al., 2024; Wang et al., 2023b; Arora et al., 2024; Hong et al., 2024). However, the computational expense of LLM inference at scale has motivated hybrid architectures that combine fast, lightweight classifiers (including PEFT-tuned encoders) with LLMs through uncertainty-aware routing mechanisms. These systems employ confidence thresholding, entropy-based measures, or learned routing policies to reserve expensive LLM calls for ambiguous cases where simpler models exhibit high uncertainty (Arora et al., 2024).

LLMs as text encoder Recent advances in LLMs have prompted researchers to explore their use as text encoders. An interesting approach is embedding extraction where existing methods typically operate on the last layer outputs through three strategies: using the last token embedding (Ma et al., 2024; Neelakantan et al., 2022; Wang et al., 2024; Meng et al., 2024; Jiang et al., 2024), averaging across all token embeddings (Muennighoff, 2022; Muennighoff et al., 2025; BehnamGhader et al., 2024), or employing trainable modules (Lee et al., 2025; Tang & Yang, 2024). Interested readers can refer to (Tao et al., 2024; Nie et al., 2024) for a more detailed review on this topic. In contrast to these approaches, this work

addresses user-intent classification for routing where both accuracy and computational efficiency are primary considerations. Our method utilizes prefill-time outputs from general-purpose LLMs without modification or additional training. By leveraging computational intermediates already produced during LLM prefill phase, this approach avoids the storage overhead of maintaining a dedicated billion-parameter model for intent classification.

Neural Feature Analysis Our approach extracts representations Wz, where W is a pre-trained weight matrix and z is model's hidden state. This design is motivated by two lines of research. First, linear probes effectively extract semantic information from transformer representations (Alain & Bengio, 2016; Hewitt & Manning, 2019), with sparse autoencoder studies suggesting that many concepts are captured by a small number of sparse features in the activation space (Cunningham et al., 2024; Gao et al., 2024). Superposition theory provides theoretical grounding, explaining how features remain recoverable through linear projections (Elhage et al., 2022). Second, activation steering research demonstrates that intent-related behaviors can be manipulated through linear interventions in the representation space (Turner et al., 2023; Panickssery et al., 2023). Finally, activations Wz were successfully used in Hayou et al. (2025) to determine target module for LoRA finetuning, showing that activation capture data signal.

Further discussion of related works in LLM routing is deferred to Appendix F.

2 Methodology

We study training-free versus training-based intent classification in large language models. Given a prompt, the goal is to assign it to one of several intent classes, such as mathematics, code, or general text. All methods considered in this paper operate on neural features computed during the model's forward pass before autoregressive decoding (Shazeer, 2019; Ainslie et al., 2023; Chang et al., 2024; Aguirre et al., 2025; Jie et al., 2025). The difference lies in how these representations are used. Training-based methods fit a classifier head from labeled data, whereas training-free methods compare the prompt's neural feature statistics to class-specific reference statistics computed from calibration data. This distinction leads to different strengths and weaknesses. Training-based classifiers can achieve higher accuracy on harder fine-grained tasks, while training-free methods often behave more robustly when prompts are ambiguous or adversarially rephrased.

2.1 Training-free Intent Classification: A Statistical Approach

Consider an LLM with weight modules $\mathcal{M} = \{W_1, W_2, \ldots, W_p\}$, for some $p \geq 1$. We abuse the notation and use W_ℓ to refer to both the module and its weight matrix. Let $\mathbf{x} = (x_t)_{1 \leq t \leq T}$ be a prompt of T tokens. For each weight module W_k, let $(y_{\ell,t})_{1 \leq t \leq T}$ denote the output features in module W_ℓ. For instance, $(y_{\ell,t})_{1 \leq t \leq T}$ could be the output of a Query head, or the projection layer in an MLP block. Each $y_{\ell,t}$ is a d-dimensional vector given by $y_{\ell,t} = W_\ell z_{\ell,t}$, where d is the output dimension in module W_ℓ, and $z_{\ell,t}$ is the module input for token t.

Intent classification. We aim to classify the prompt $\mathbf{x}$ into one of the classes $C_1, C_2, \ldots, C_m$, where $m \geq 2$. For instance, a binary classification where C_1 is mathematics and C_2 is coding. For each module W_ℓ, we compute summary statistics from the features $\{y_{\ell,t}\}_{t=1}^{T}$ and compare them to per-class baselines: (i) for each class C_i, precompute the same statistics on *calibration data* at the same module W_ℓ; (ii) for the incoming prompt, compute the statistics at W_ℓ and measure similarity to each baseline; (iii) assign the class C_i with highest similarity.

We consider two methods: 1) *VecStat*, which relies on coordinate-wise mean and variance for classification, and 2) *NormStat*, a lighter variant that relies solely on the norm across all tokens and coordinates. Fig. 1 summarizes both methods. In the following, we present the two methods in the single-layer case. When multiple layers are used, we aggregate similarity scores across ℓ by averaging.

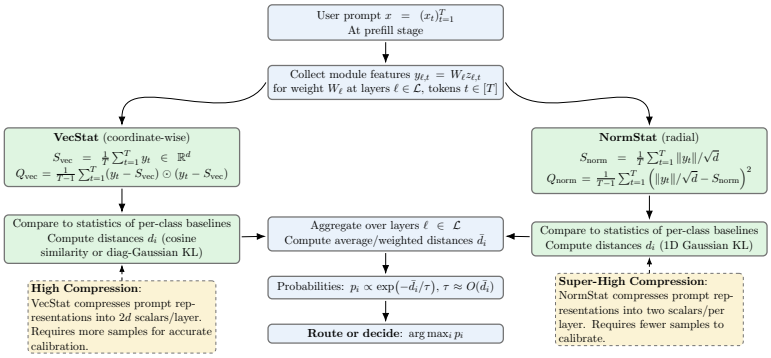


Figure 1: Flowchart of VecStat vs NormStat for prefill-time intent classification and routing.

Vector Statistic (VecStat): calculate *coordinate-wise* token means and second moments tokens

$$S_{\text{vec}} = \frac{1}{T}\sum_{t=1}^{T} y_t \in \mathbb{R}^d, \qquad Q_{\text{vec}} = \frac{1}{T-1}\sum_{t=1}^{T}(y_t - S_{\text{vec}}) \odot (y_t - S_{\text{vec}}) \in \mathbb{R}^d. \tag{1}$$

Norm Statistic (NormStat): summarize each y_t through a the norm $\|y_t\|$ and aggregate across tokens to obtain the statistics

$$S_{\text{norm}} = \frac{1}{T}\sum_{t=1}^{T} \frac{\|y_t\|}{\sqrt{d}} \in \mathbb{R}, \quad Q_{\text{norm}} = \frac{1}{T-1}\sum_{t=1}^{T}\left(\frac{\|y_t\|}{\sqrt{d}} - S_{\text{norm}}\right)^2 \in \mathbb{R}. \tag{2}$$

For both methods, we score a prompt against each class using the closed-form KL divergence between Gaussian surrogate distributions parameterized by the corresponding summary statistics (see Eqs. (5) and (6) in the Appendix). VecStat uses a diagonal Gaussian over the activation coordinates, whereas NormStat uses a univariate Gaussian over the normalized token norms $\|y_{\ell,t}\|/\sqrt{d}$. The resulting score is the exact KL divergence between the Gaussian surrogates, but it is not intended to approximate the KL divergence between the unknown underlying activation distributions. Rather, Gaussian moment matching provides a simple and computationally efficient scoring rule based on first- and second-order statistics. Figure 2 provides qualitative motivation for this choice by showing approximately bell-shaped radial statistics at representative layers, while we do not claim that LLM activations are literally Gaussian or independent.[2] We also evaluate cosine similarity for VecStat as an alternative that does not rely on Gaussian KL scoring.

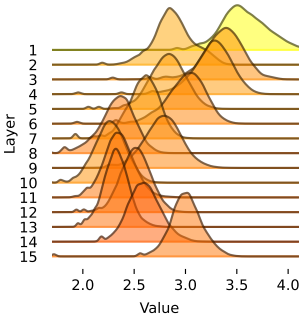


Figure 2: **Per-token query-norm distributions across layers.** Histograms of $\{\|y_{\ell,t}\|/\sqrt{d}\}_{t=1}^T$ for representative layers ℓ.

These two choices form a *statistical compression ladder*: VecStat keeps per-coordinate first and second moments, while NormStat compresses all coordinates to a single *radial* information per token and then to its mean/variance across tokens. The rest of this section develops this story rigorously: (i) we discuss when each is statistically preferable using a simplified Gaussian setting, and (ii) we connect those guarantees to *compute*, *memory*, and *calibration* costs. Proofs are in Appendix C.

Intuitive Analysis in a Gaussian Setting To compare NormStat and VecStat, we study an analytically tractable Gaussian proxy model. For a fixed class $k \in \{1, \ldots, m\}$, suppose that the token features (not necessarily representations in an LLM) satisfy

$$y_t \mid k \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu_k, \Sigma_k), \qquad t = 1, \ldots, T, \tag{3}$$

where $\Sigma_k = \text{Diag}(\sigma_{k,1}^2, \ldots, \sigma_{k,d}^2)$ is diagonal.

[2]One could estimate an empirical KL without summaries, but doing so robustly at inference time is prohibitively expensive in both compute and memory.

Expressivity. Since NormStat depends only on $\{\|y_t\|_2\}_{t=1}^T$, it is invariant under orthogonal transformations and cannot distinguish classes whose separation lies purely in direction. In contrast, VecStat is sensitive to coordinate-wise changes. Specifically, when classes share the same mean and differ only through spherical covariance scales, radial information is sufficient and NormStat is already *Bayes-optimal*. In the directional regime, where classes have equal covariance and equal mean norm but different mean directions, every radius-only rule is blind, whereas a coordinate-aware rule can achieve exponentially small error in T. See the following theorem for a formal statement. The proof is provided in Appendix C.

Theorem 1 (NormStat vs. VecStat)**.** *Consider binary classification with uniform class priors.*

1. ***Directional regime.*** *If $\Sigma_1 = \Sigma_2 = \sigma^2 I_d$, $\|\mu_1\| = \|\mu_2\|$, and $\mu_1 \neq \mu_2$, then every classifier based only on $\{\|y_t\|\}_{t=1}^T$ has error $1/2$, whereas the likelihood-ratio test satisfies*
$$\Pr(\hat{k} \neq k) \leq \exp\left(-\frac{T\|\mu_1 - \mu_2\|^2}{8\sigma^2}\right).$$

2. ***Isotropic-scale regime.*** *If $\mu_1 = \mu_2 = 0$ and $\Sigma_k = \sigma_k^2 I_d$ with $\sigma_1 \neq \sigma_2$, then the likelihood-ratio test depends only on $R_T = \sum_{t=1}^T \|y_t\|^2$. Hence radial information is sufficient, and coordinate-wise information cannot improve the Bayes risk.*

Theorem 1 describes two limiting regimes, whereas real activation distributions may contain both radial and directional differences. To illustrate an intermediate setting, write
$$y_t = s_t\mu + \varepsilon_t, \qquad \Pr(s_t = 1) = \pi, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2 I_d),$$
where $s_t \in \{\pm 1\}$, $\mu \in \mathbb{R}^d$, and $\gamma := \mathbb{E}[s_t] = 2\pi - 1$. Equivalently,
$$y_t \sim \pi\mathcal{N}(\mu, \sigma^2 I_d) + (1-\pi)\mathcal{N}(-\mu, \sigma^2 I_d).$$
The corresponding statistics satisfy
$$\mathbb{E}[S_{\text{vec}}] = \gamma\mu, \qquad \mathbb{E}[Q_{\text{vec}}] = \sigma^2 \mathbf{1}_d + (1-\gamma^2)\mu \odot \mu, \qquad \mathbb{E}[S_{\text{norm}}] = F(\|\mu\|, \sigma^2),$$
for a function F depending only on the radial parameters. Because the norm is invariant under $\mu \mapsto -\mu$, the distributions of S_{norm} and Q_{norm} do not depend on π. By contrast, VecStat retains information about π through both its mean and coordinate-wise variance. As $\pi \to \frac{1}{2}$, the first-moment signal $\mathbb{E}[S_{\text{vec}}]$ vanishes, but the second-moment term $\mu \odot \mu$ generally remains. Thus, VecStat may still retain coordinate-wise information that is invisible to NormStat, even in the symmetric-mixture case.

Calibration Cost. When using Normstat and Vecstat, an important aspect is *calibration* which we use to refer to estimating key statistics used for classification.[3] An important aspect of statistical methods is sample complexity, or more specifically, the convergence rate of key statistics in sample size. This provides an estimate of the total number of calibration samples needed to create the target statistics for classes $k \in \{1, 2, \ldots, m\}$. The next theorem show the calibration advantage of NormStat over VecStat.

Theorem 2 (Calibration cost)**.** *Fix a class k. Let $y_1, \ldots, y_N \overset{i.i.d.}{\sim} \mathcal{N}(\mu_k, \Sigma_k)$ in $\mathbb{R}^d$, where N is the number of calibration samples drawn for this class. Let $q = \mathbb{E}[\|y_1\|]$ and define $\hat{\mu}_k = N^{-1}\sum_{i=1}^N y_i$, and $\hat{q} = N^{-1}\sum_{i=1}^N d^{-1/2}\|y_i\|$. Then, for any $\delta \in (0,1)$, with probability at least $1-\delta$, we have:*

1. ***NormStat (dimension-free):*** $|\hat{q} - q| \lesssim \sqrt{\frac{\log(1/\delta)}{N}}$.

2. ***VecStat (dimension-dependent):*** $\|\hat{\mu}_k - \mu_k\|_2 \lesssim \sqrt{\frac{d + \log(1/\delta)}{N}}$.

Considering just the statistics $\hat{\mu}$ and $\hat{q}$, to obtain an estimation error of order ϵ, one needs $N = \Omega(\epsilon^{-2})$ for $\hat{q}$ and $N = \Omega(d\epsilon^{-2})$ for $\hat{\mu}$, showing the computational advantage of NormStat over VecStat. This is particularly important in data scarce regimes with few samples for each class. We discuss this in more details in the next section.

[3] Calibration for Vecstat and Normstat is the equivalent of training for MLP-based methods.

2.2 Training-based Intent Classification: MLP Classifier and Linear Probes

For a comprehensive and fair comparison with the training-free methods above, we consider a trained classifier head on top of frozen LLM features from the *prefill* phase. We use the last Transformer block and write $y_{\ell^\star,t} \in \mathbb{R}^d$ for its token features ($t = 1, \ldots, T$). Inspired by prompt/sentence classification pipelines (e.g., (Ma et al., 2024; Wang et al., 2024; Meng et al., 2024)), we build a single prompt-level vector in two ways:

$$\textbf{Avg-MLP:} \quad z_{\text{avg}} := \frac{1}{T}\sum_{t=1}^{T} y_{\ell^\star,t} = S_{\text{vec}}^{(\ell^\star)} \in \mathbb{R}^d, \qquad \textbf{Tail-MLP:} \quad z_{\text{tail}} := y_{\ell^\star,T} \in \mathbb{R}^d.$$

Given $z \in \{z_{\text{avg}}, z_{\text{tail}}\}$, we train a two-layer MLP with hidden width h with cross entropy loss. Since z_{avg} or z_{tail} is produced during prefill, the incremental latency is a single MLP forward pass. We also consider a simple linear probe variant that we call **Avg-Linear** where we use a simple projection instead of MLP. We defer discussion of training/calibration costs and practical trade-offs between training-free and training-based methods to Appendix A.

3 Experiments

In this section, we evaluate the effectiveness of NormStat, VecStat, Avg-MLP, Tail-MLP, and Avg-Linear across multiple LLMs and classification datasets. All experimental details can be found in Appendix D, and additional experimental results are presented in Appendix E. Our results provide a systematic comparison between training-free and training-based intent classification methods, showing the advantages and disadvantages of each approach. For completeness, we also compare with zero-shot direct LLM call for intent classification. The code and datasets associated with this work are publicly available at `https://github.com/Zhouhao-Yang/Training-Free-versus-Training-Based-Intent-Classification-in-LLMs`.

3.1 Experimental Setup

Classification tasks and granularities. We consider four classification tasks with different levels of granularity. Task 1 addresses coarse-grained intent classification among three broad categories: general text, mathematics, and code; Task 2 considers programming language identification; Task 3 considers natural language identification; Task 4 considers mathematical subfield classification. Task 1-3 are coarse-grained (level-1 granularity), while Task 4 is a fine-grained task (level-2 granularity).

For calibration in Task 1, we use representative datasets for each intent class: MMLU European History (Hendrycks et al., 2021a;b) for general text, GSM8K (Cobbe et al., 2021) for mathematics, and Magicoder (Wei et al., 2023) for code. We test on MMLU US History for general text, GSM8K and MATH500 (Lightman et al., 2023) for mathematics (in-distribution and out-of-distribution, respectively), and Magicoder and HumanEval (Chen et al., 2021) for code (in-distribution and out-of-distribution, respectively). For Task 2, we use Magicoder for programming language identification. For Task 3, we use the Aya dataset (Singh et al., 2024) for natural language identification, splitting each language subset into calibration and test sets. For Task 4, we use domain-specific subsets of Competition Math (Hendrycks et al., 2021c) for mathematical subfield classification. More details are provided in Appendix D.1.

Method and LLM selection We compare five classification methods: NormStat, VecStat (with two variants: cosine similarity, VecStat:Cos, and KL divergence, VecStat:KL), and training-based baselines Avg-MLP, Tail-MLP, and Avg-Linear. Training-based methods use the same calibration data for training to ensure fair comparison. We also benchmark 0-shot and 3-shot LLM calls, where model predicts intent directly from the prompt. We find that providing a high-level overview of the intent classes in the prompt is necessary for achieving a reasonable performance. All calibration prompts are truncated to 512 tokens, with training-free methods probing all linear modules. We evaluate Qwen3 and Llama models across 1B–32B scales, including base and instruction-tuned variants; see Appendix D.3 for details. Additionally, we fine-tune RoBERTa-Base end-to-end with a task-specific classification head, using the same labeled training/calibration and test splits as the other methods.

Method	FLOPs Overhead	Memory Overhead	Extendability of New Classes	Adversarial Robustness
NormStat	$O(Td)$	$O(m)$	Compute new baselines	Better
VecStat	$O(Td)$	$O(md)$	Compute new baselines	Better
MLP	$O(hd)$	$O(hd)$	Retrain a new MLP head	Worse
LLM Call	$\Omega(Td^2)$	$\sim$	Extend via prompt engineering	$\times$

Table 1: Classifier Comparison. T: prompt length; d: hidden width; m: # classes; h: MLP hidden size.

Table 2: Classification accuracy across methods on Tasks 1–3 (all level-1 granularity). Task 1 reports per-dataset accuracy for domain classification (general text, math, code). Tasks 2 and 3 report mean balanced accuracy across classes: programming language identification and natural language identification, respectively. LLM Call results omitted for Llama-3.2-1B models due to invalid outputs.

Model	Method	Task 1					Task 2	Task 3
		gsm8k	humaneval	magicoder	math500	mmlu_history	programming	natural language
Llama-3.2-1B	Avg-MLP	100.00$_{\pm 0.00}$	100.00$_{\pm 0.00}$	99.99$_{\pm 0.01}$	64.33$_{\pm 8.33}$	99.84$_{\pm 0.28}$	99.97$_{\pm 0.03}$	99.91$_{\pm 0.04}$
	Tail-MLP	99.95$_{\pm 0.04}$	100.00$_{\pm 0.00}$	99.97$_{\pm 0.03}$	98.93$_{\pm 0.42}$	99.35$_{\pm 1.13}$	99.46$_{\pm 0.19}$	99.83$_{\pm 0.07}$
	Avg-Linear	100.00$_{\pm 0.00}$	99.80$_{\pm 0.35}$	99.99$_{\pm 0.01}$	71.47$_{\pm 7.78}$	98.69$_{\pm 1.86}$	99.96$_{\pm 0.03}$	99.90$_{\pm 0.02}$
	NormStat:KL	99.49$_{\pm 0.09}$	90.85$_{\pm 0.00}$	96.39$_{\pm 0.20}$	83.40$_{\pm 0.00}$	92.48$_{\pm 0.57}$	49.21$_{\pm 1.38}$	86.53$_{\pm 1.33}$
	VecStat:KL	100.00$_{\pm 0.00}$	99.39$_{\pm 0.00}$	99.98$_{\pm 0.02}$	78.80$_{\pm 0.12}$	100.00$_{\pm 0.00}$	98.99$_{\pm 0.20}$	99.19$_{\pm 0.08}$
	VecStat:Cos	100.00$_{\pm 0.00}$	99.39$_{\pm 0.00}$	99.97$_{\pm 0.02}$	77.60$_{\pm 0.20}$	100.00$_{\pm 0.00}$	98.71$_{\pm 0.20}$	99.72$_{\pm 0.01}$
Qwen3-8B	Avg-MLP	99.42$_{\pm 0.74}$	99.59$_{\pm 0.35}$	99.99$_{\pm 0.02}$	77.27$_{\pm 11.02}$	100.00$_{\pm 0.00}$	99.97$_{\pm 0.02}$	99.92$_{\pm 0.01}$
	Tail-MLP	99.82$_{\pm 0.12}$	98.58$_{\pm 2.46}$	99.98$_{\pm 0.02}$	81.20$_{\pm 9.72}$	100.00$_{\pm 0.00}$	99.66$_{\pm 0.12}$	99.88$_{\pm 0.04}$
	Avg-Linear	99.57$_{\pm 0.24}$	100.00$_{\pm 0.00}$	100.00$_{\pm 0.00}$	83.00$_{\pm 2.03}$	100.00$_{\pm 0.00}$	99.97$_{\pm 0.02}$	99.94$_{\pm 0.01}$
	NormStat:KL	85.14$_{\pm 0.35}$	10.37$_{\pm 1.06}$	99.85$_{\pm 0.06}$	92.93$_{\pm 0.12}$	99.51$_{\pm 0.49}$	56.39$_{\pm 0.87}$	90.09$_{\pm 0.40}$
	VecStat:KL	99.95$_{\pm 0.04}$	99.59$_{\pm 0.35}$	99.99$_{\pm 0.02}$	92.20$_{\pm 0.00}$	100.00$_{\pm 0.00}$	99.23$_{\pm 0.13}$	99.20$_{\pm 0.08}$
	VecStat:Cos	100.00$_{\pm 0.00}$	95.73$_{\pm 0.61}$	99.98$_{\pm 0.03}$	94.20$_{\pm 0.00}$	100.00$_{\pm 0.00}$	99.34$_{\pm 0.14}$	99.68$_{\pm 0.03}$
	LLM Call (0-shot)	99.67$_{\pm 0.04}$	100.00$_{\pm 0.00}$	99.42$_{\pm 0.07}$	99.60$_{\pm 0.00}$	99.67$_{\pm 0.28}$	98.89$_{\pm 0.05}$	81.72$_{\pm 0.09}$
	LLM Call (3-shot)	99.14$_{\pm 0.18}$	97.15$_{\pm 0.35}$	99.43$_{\pm 0.07}$	99.87$_{\pm 0.12}$	99.51$_{\pm 0.49}$	-	-
Qwen3-32B	Avg-MLP	95.88$_{\pm 3.31}$	100.00$_{\pm 0.00}$	99.99$_{\pm 0.01}$	86.87$_{\pm 5.22}$	100.00$_{\pm 0.00}$	99.97$_{\pm 0.03}$	99.93$_{\pm 0.02}$
	Tail-MLP	99.39$_{\pm 0.00}$	100.00$_{\pm 0.00}$	99.98$_{\pm 0.00}$	96.80$_{\pm 0.40}$	99.02$_{\pm 0.49}$	98.41$_{\pm 0.39}$	99.84$_{\pm 0.01}$
	Avg-Linear	98.61$_{\pm 1.59}$	100.00$_{\pm 0.00}$	99.99$_{\pm 0.02}$	87.53$_{\pm 6.94}$	100.00$_{\pm 0.00}$	99.97$_{\pm 0.03}$	99.94$_{\pm 0.01}$
	NormStat:KL	97.93$_{\pm 0.04}$	24.59$_{\pm 0.35}$	99.78$_{\pm 0.06}$	97.93$_{\pm 0.12}$	100.00$_{\pm 0.00}$	57.02$_{\pm 1.03}$	89.62$_{\pm 0.08}$
	VecStat:KL	100.00$_{\pm 0.00}$	99.39$_{\pm 0.00}$	99.98$_{\pm 0.03}$	96.60$_{\pm 0.00}$	100.00$_{\pm 0.00}$	99.53$_{\pm 0.11}$	98.74$_{\pm 0.05}$
	VecStat:Cos	100.00$_{\pm 0.00}$	98.17$_{\pm 0.00}$	99.98$_{\pm 0.03}$	96.80$_{\pm 0.35}$	100.00$_{\pm 0.00}$	99.61$_{\pm 0.10}$	99.58$_{\pm 0.02}$
	LLM Call (0-shot)	86.91$_{\pm 0.29}$	100.00$_{\pm 0.00}$	98.69$_{\pm 0.06}$	96.27$_{\pm 0.42}$	100.00$_{\pm 0.00}$	99.82$_{\pm 0.03}$	99.16$_{\pm 0.01}$
	LLM Call (3-shot)	97.80$_{\pm 0.20}$	100.00$_{\pm 0.00}$	98.87$_{\pm 0.05}$	99.80$_{\pm 0.20}$	100.00$_{\pm 0.00}$	-	-
RoBERTa	-	100.00$_{\pm 0.00}$	100.00$_{\pm 0.00}$	99.98$_{\pm 0.02}$	42.33$_{\pm 13.05}$	100.00$_{\pm 0.00}$	99.97$_{\pm 0.05}$	99.92$_{\pm 0.16}$

Evaluation Metrics For task 1, we compute accuracy on each test dataset independently, where each dataset contains samples from a single ground-truth class. This approach ensures our evaluation is not biased by varying dataset sizes across classes. For task 2 and 3, we report mean accuracy across all classes within each task due to space constraints. This mean accuracy corresponds to the balanced accuracy metric, providing equal weight to each class regardless of test set size and effectively handling the natural class imbalance among test datasets. All experiments are run with three seeds, with mean and std reported.

3.2 Empirical Results

Table 1 summarizes the computational and practical trade-offs among intent classification methods. Table 2 reports classification results for Tasks 1–3 (all level-1 granularity) across five methods and four representative LLMs. For Task 1, we report per-dataset accuracy; for Tasks 2 and 3, we report mean balanced accuracy across classes. Table 3 provides per-subfield accuracy for Task 4 (level-2 mathematical subfield classification). Full results for all seven LLMs are reported in Appendix E.1 (level-1) and Appendix E.2 (level-2).

Computational Overhead. NormStat stores only $O(m)$ scalars and performs $O(m)$ scoring FLOPs at inference, making it the most lightweight option. VecStat stores $O(md)$ numbers and requires $O(md)$ scoring FLOPs. Meanwhile, training-based methods rely on an MLP head, requiring $O(hd)$ FLOPs and $O(hd)$ parameter storage, where h is the hidden layer dimension. Direct LLM calls incur the highest computational overhead at $\Omega(Td^2)$ FLOPs.

Level-1 classification: All methods perform well in Task 1 and 3; Task 2 exposes NormStat's limitation. For task 1 and 3, all methods achieve strong in-distribution performance. The effectiveness of NormStat, despite using only radial statistics, demonstrates itself as a computational- and memory-efficient approach when the classes are different enough. Avg-Linear performs comparably to Avg-MLP but with slightly lower accuracy, consistent with the reduced expressiveness of a linear layer versus two-layer architecture. Direct LLM inference shows improved classification accuracy with larger models, though it requires careful prompt design to achieve a reasonable performance. Adding few-shot examples can improve LLM Call performance through in-context learning, but at the cost of a longer prompt and consequently higher computational cost. Notably, Llama-3.2-1B models fail to produce valid responses, hence their results are omitted. The fine-tuned RoBERTa encoder is competitive on most level-1 tasks but falls to 42.33% on the out-of-distribution MATH500 set, where several LLM-feature methods remain above 90%. Furthermore, out-of-distribution generalization varies substantially, as evidenced by the performance difference between GSM8K and MATH500 for mathematical tasks, suggesting that different methods might capture distinct aspects of domain characteristics.

Meanwhile, task 2 reveals a clear limitation of NormStat: despite training-based methods and VecStat saturate on the classification, NormStat degrades substantially. This result is consistent with Theorem 1 that directional information in the neural feature space is critical for within-domain discrimination: since programming languages share the same broad "code" domain, their feature distributions differ primarily in direction rather than radial scale, making norm-only statistics insufficient.

Level-2 classification: Training-based methods are more accurate. Fine-grained mathematical subfield classification reveals a clear gap between training-based and training-free methods (Table 3). When averaged across subfields, both the trained LLM-feature classifiers and RoBERTa consistently outperform the training-free methods and direct LLM inference. However, no single method dominates uniformly: the best-performing method varies by subfield. These findings suggest that effective discrimination among closely related mathematical topics benefits from non-linear transformations learned through supervised training, rather than simple statistical summaries of activation distributions.

Table 3: Level-2 mathematical subfield classification results. Values represent per-subfield accuracy across seven mathematical subfields. Best accuracy in each subfield is highlighted in bold.

Model	Method	Algebra	Counting & Probability	Geometry	Intermediate Algebra	Number Theory	Prealgebra	Precalculus
Qwen3-8B	Avg-MLP	**73.94±1.63**	82.70±0.98	87.13±0.56	**81.00±3.61**	86.59±4.35	52.95±5.24	**89.36±0.42**
	Tail-MLP	68.43±1.81	80.52±2.03	89.92±1.91	73.87±6.12	82.92±5.55	**53.58±0.64**	81.10±5.72
	Avg-Linear	73.28±4.60	80.82±3.83	**90.29±2.47**	79.92±3.87	82.65±0.90	52.19±3.70	84.76±0.54
	NormStat:KL	21.01±6.35	19.03±1.50	40.19±3.56	46.38±0.91	73.45±4.22	0.73±0.59	31.98±1.31
	VecStat:KL	36.23±3.18	51.24±1.17	35.88±3.28	**81.00±1.60**	88.89±0.47	1.52±0.12	55.15±3.27
	VecStat:Cos	64.46±1.45	61.95±1.87	36.25±3.19	74.13±0.95	**89.33±0.33**	2.06±0.40	55.08±1.08
	LLM Call (0-shot)	44.49±0.22	**83.22±1.58**	78.32±2.46	67.76±0.44	52.38±2.21	18.70±1.02	64.63±1.63
Qwen3-32B	Avg-MLP	**73.25±4.75**	77.83±2.89	89.01±3.74	**84.14±3.27**	81.88±2.64	**57.92±4.22**	86.65±3.02
	Tail-MLP	66.70±8.95	78.43±2.02	88.52±4.65	79.61±3.33	74.06±11.10	50.57±10.68	80.62±3.30
	Avg-Linear	71.97±5.76	79.93±3.82	87.86±0.92	82.33±3.88	85.06±0.72	52.82±4.70	**87.53±1.89**
	NormStat:KL	31.99±2.02	21.72±0.85	40.56±3.48	44.61±1.14	75.53±2.56	0.13±0.05	31.44±1.47
	VecStat:KL	55.76±1.11	58.50±2.04	35.94±3.28	77.80±1.04	**89.87±0.34**	1.46±0.12	52.78±0.59
	VecStat:Cos	71.16±0.46	64.42±2.16	37.04±3.20	73.32±0.82	89.49±0.87	3.00±0.72	58.47±1.31
	LLM Call (0-shot)	69.28±0.36	**86.59±1.44**	**91.92±0.42**	35.81±1.07	59.61±1.31	20.04±0.51	63.28±0.12
RoBERTa	-	67.03±3.37	81.95±1.58	90.28±1.39	78.59±1.72	82.76±2.61	51.91±1.58	87.53±1.89

Distance metric comparison The cosine distance variant of VecStat consistently outperforms its KL divergence counterpart, particularly in task 4, suggesting that angular separation between prompt and baseline statistics better captures directional differences.

3.3 Uncertainty Quantification for Mixed-Intent Prompts

To evaluate whether the proposed methods can handle ambiguous prompts, we construct a mixed-intent dataset by interleaving samples from both math and code datasets at five known mix ratios, assessed across two prompt orderings (code-first and math-first). The probability outputs of each method are temperature-calibrated, and performance is mea-

sured by the RMSE between the predicted math probability and the true math fraction. Full experimental details are provided in Appendix E.3.

Figure 3 and Table 4 report results on Qwen3-1.7B; results for additional models are in Fig. 9 and Table 15. VecStat provides the most accurate uncertainty estimates across both prompt orderings, achieving the lowest calibration RMSE. Tail-MLP is the weakest method overall and the most sensitive to prompt order, consistent with its reliance on the last-token embedding. In contrast, NormStat and Avg-MLP are more stable across orderings but their prediction curves remain flatter than the ideal diagonal, suggesting they under-react to changes in mixture ratio rather than tracking them continuously.

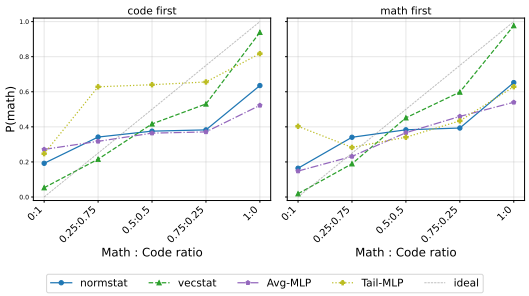


Figure 3: Predicted $P(\text{math})$ vs. mix ratio for QWEN3-1.7B. Curves are temperature-calibrated.

Table 4: Calibration RMSE for QWEN3-1.7B. $\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - p_i^*)^2}$, where $\hat{p}_i$ is the predicted math probability and p_i^* is the target math fraction, evaluated at mix-ratio points. **Bold** marks the best method per column.

Method	code-first	math-first
NormStat	0.2213	0.2098
VecStat	**0.1216**	**0.0858**
Avg-MLP	0.2451	0.1764
Tail-MLP	0.2412	0.2688

3.4 Robustness to Adversarial Attack

Adversarial Dataset. We create three adversarial variants of MATH500 at increasing levels of camouflage—Easy (lexical), Medium (structural), and Hard (genre-level) [4]. Each variant disguises math problems as code-related tasks to misguide the classifier, while remaining recognizable as math problems to human readers. For each level, we craft a dedicated prompt (see Table 16) and call GPT-4o to rewrite the problems accordingly. Furthermore, we use API-based LLM calls to GPT-5-Nano and GPT-5 to sanity-check the intended difficulty stratification.

Table 5: Accuracy on adversarial datasets.

Model	Method	Easy	Medium	Hard
GPT-5-Nano	LLM Call	93	64	34
GPT-5	LLM Call	98	98	64
Qwen3-32B	Avg-MLP	11.73±8.92	0.40±0.69	0.00
	Tail-MLP	61.60±14.67	0.93±0.81	0.20
	NormStat:KL	**92.33±0.12**	**80.67±0.12**	0.00
	VecStat:KL	70.93±0.31	32.33±0.31	0.00
	VecStat:Cos	64.87±0.58	30.33±0.64	0.00
Qwen3-8B	Avg-MLP	12.67±7.82	4.20±3.30	0.00
	Tail-MLP	36.67±31.67	1.47±2.20	0.00
	NormStat:KL	36.13±0.50	27.00±0.40	0.00
	VecStat:KL	41.00±0.69	25.00±0.35	0.00
	VecStat:Cos	**78.60±1.39**	**54.80±0.69**	0.20

Observations. Table 5 reports accuracy on three adversarial variants of MATH500 for two large-scale models (see Table 14 for full results). Performance degrades monotonically from Easy to Medium to Hard across all methods, as expected. On the Hard tier, all methods collapse to near-zero accuracy, indicating a shared failure mode. In contrast, GPT-5 achieves 64%, suggesting that identifying the underlying mathematical intent despite the bug-report framing requires substantially stronger semantic understanding. Developing lightweight classifiers with comparable robustness represents a promising avenue for future work.

Training-free methods are more robust than training-based ones. On Qwen3-8B and Qwen3-32B, the MLP-based classifiers degrade sharply, while training-free methods retain non-trivial accuracy on the Easy and Medium tiers. We attribute this to two reasons. First, for large models, training-free classifiers might require far fewer calibration samples than training-based ones. Second, training-based classifiers (MLP heads) learn a discriminative boundary directly from the token-level embedding distribution of calibration data. When

[4] The dataset is available at https://huggingface.co/datasets/nanchennn/Adv_MATH500

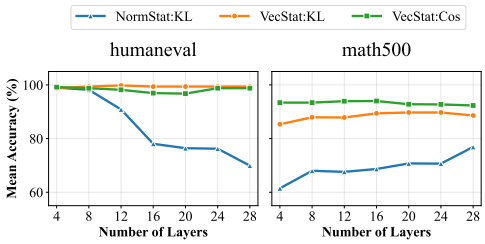


Figure 4: Effect of the number of layers on level-1 classification accuracy for Qwen3-1.7B.

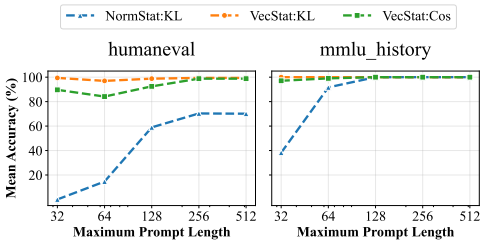


Figure 5: Effect of the maximum prompt length on level-1 classification for Qwen3-1.7B.

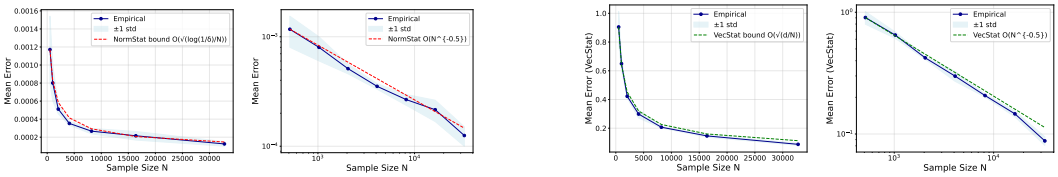
adversarial rephrasing injects coding vocabulary into a math problem, it shifts the token distribution toward the coding class, causing the learned boundary to shift.

3.5 Effect of number of probed layers and prompt length

In Fig. 4, VecStat demonstrates robust performance regardless of layer count, while NormStat exhibits dataset-dependent behavior, though both achieve competitive accuracy using only the first 12 layers out of 28. These findings imply that intent classification can be performed without completing a full forward pass, substantially saving computation costs. As shown in Fig. 5, VecStat maintains near-optimal accuracy across prompt lengths from 32 to 512 tokens, whereas NormStat is more sensitive to prompt length, plateauing at approximately 128 tokens. Additional results are presented in Appendices E.5 and E.6.

3.6 Calibration Analysis

To validate Theorem 2, we compare the empirical and theoretical convergence rates of NormStat and VecStat on the MagiCoder and present the result in Fig. 6. We vary the calibration sample size from 512 to 32768 and run with different seeds. Our results demonstrate strong match with the theoretical bounds. Both methods show $\tilde{O}(N^{-0.5})$ rate for the mean error, following the predicted theoretical curves. Notably, NormStat attains much lower absolute errors, which is consistent with its dimension-free bound, whereas VecStat sits higher due to its dimension-dependent bounds. Calibration results for other LLMs are in Appendix E.7.



(a) NormStat (b) VecStat

Figure 6: Calibration convergence analysis for Qwen3-8B on the MagiCoder dataset. Each subplot shows both linear and log-log scales comparing empirical results with theoretical bounds.

4 Conclusion

We propose two training-free methods, NormStat and VecStat and show that there is no one-size-fits-all method for intent classification. Training-based methods are stronger on fine-grained tasks, while training-free methods remain competitive on coarse-grained tasks and more robust under noisy or ambiguous prompts.

Acknowledgement

This work used GPU servers at DeltaAI at the National Center for Supercomputing Applications (NCSA) through allocation #CIS250883 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. NC was funded by the NSF–Simons Research Collaboration on the Mathematical and Scientific Foundations of Deep Learning (MoDL) (NSF DMS 2031985).

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*, 2023.

Maia Aguirre, Ariane Méndez, Arantza Del Pozo, María Inés Torres, and Manuel Torralbo. Fine-tuning medium-scale llms for joint intent classification and slot filling: A data-efficient and cost-effective solution for smes. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pp. 251–262, 2025.

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Gaurav Arora, Shreya Jain, and Srujana Merugu. Intent detection in the age of llms. *arXiv preprint arXiv:2410.01627*, 2024.

Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. Llms will always hallucinate, and we need to live with this. In *Intelligent Systems Conference*, pp. 624–648. Springer, 2025.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961*, 2024.

Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*, 2017.

Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, and Alan Nichol. Diet: Lightweight language understanding for dialogue systems. *arXiv preprint arXiv:2004.09936*, 2020.

Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders. *arXiv preprint arXiv:2003.04807*, 2020.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

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, 2021.

Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*, 2019.

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

Henry Cunningham, Ryan Huben, et al. Sparse autoencoders find highly interpretable features in language models. In *ICLR*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/1fa1ab11f4bd5f94b2ec20e794dbfa3b-Paper-Conference.pdf.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for llms. *arXiv preprint arXiv:2410.10347*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834*, 2024.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders, 2024. URL https://cdn.openai.com/papers/sparse-autoencoders.pdf.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 753–757, 2018.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Surya Narayanan Hari and Matt Thomson. Tryage: Real-time, intelligent routing of user prompts to large language models. *arXiv preprint arXiv:2308.11601*, 2023.

Homa B Hashemi, Amir Asiaee, and Reiner Kraft. Query intent detection using convolutional neural networks. In *International conference on web search and data mining, workshop on query understanding*, volume 23, 2016.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Plop: Precise lora placement for efficient finetuning of large models, 2025. URL https://arxiv.org/abs/2506.20629.

Changai He, Sibao Chen, Shilei Huang, Jian Zhang, and Xiao Song. Using convolutional neural network with bert for intent determination. In *2019 International Conference on Asian Language Processing (IALP)*, pp. 65–70. IEEE, 2019.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021c.

John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *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. 4129–4138, 2019.

Taesuk Hong, Youbin Ahn, Dongkyu Lee, Joongbo Shin, Seungpil Won, Janghoon Han, Stanley Jungkyu Choi, and Jungyun Seo. Exploring the use of natural language descriptions of intents for large language models in zero-shot intent classification. In *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 458–465, 2024.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

Ting Jiang, Shaohan Huang, Zhongzhi Luan, Deqing Wang, and Fuzhen Zhuang. Scaling sentence embeddings with large language models. In *EMNLP (Findings)*, pp. 3182–3196, 2024. URL https://aclanthology.org/2024.findings-emnlp.181.

Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. Specache: Speculative key-value caching for efficient generation of llms. *arXiv preprint arXiv:2503.16163*, 2025.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, et al. Universal model routing for efficient llm inference. *arXiv preprint arXiv:2502.08773*, 2025.

Stefan Larson, Anish Mahendran, Joseph J Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K Kummerfeld, Kevin Leach, Michael A Laurenzano, Lingjia Tang, et al. An evaluation dataset for intent classification and out-of-scope prediction. 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. 1311–1316, 2019.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. In *International Conference on Learning Representations*, volume 2025, pp. 79310–79333, 2025.

Erich Leo Lehmann and George Casella. *Theory of point estimation*. Springer, 1998.

Erich Leo Lehmann and Henry Scheffé. Completeness, similar regions, and unbiased estimation-part i. In *Selected works of EL Lehmann*, pp. 233–268. Springer, 2011.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.emnlp-demo.21.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624): 1092–1097, 2022.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35, 2023.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2421–2425, 2024.

Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. Sfrembedding-mistral: enhance text retrieval with transfer learning. *Salesforce AI Research Blog*, 3:6, 2024.

Niklas Muennighoff. Sgpt: Gpt sentence embeddings for semantic search. *arXiv preprint arXiv:2202.08904*, 2022.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *International Conference on Learning Representations*, volume 2025, pp. 45544–45613, 2025.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

Zhijie Nie, Zhangchi Feng, Mingxin Li, Cunwang Zhang, Yanzhao Zhang, Dingkun Long, and Richong Zhang. When text embedding meets large language model: a comprehensive survey. *arXiv preprint arXiv:2412.09165*, 2024.

OpenAI. Gpt-5 system card. 2025. URL https://cdn.openai.com/gpt-5-system-card.pdf.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering llama 2 via contrastive activation addition. *arXiv preprint arXiv:2312.06681*, 2023.

Juan A Rodriguez, Nicholas Botzer, David Vazquez, Christopher Pal, Marco Pedersoli, and Issam Laradji. Intentgpt: Few-shot intent discovery with large language models. *arXiv preprint arXiv:2411.10670*, 2024.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Shivalika Singh, Freddie Vargus, Daniel Dsouza, Börje F. Karlsson, Abinaya Mahendiran, Wei-Yin Ko, Herumb Shandilya, Jay Patel, Deividas Mataciunas, Laura OMahony, Mike Zhang, Ramith Hettiarachchi, Joseph Wilson, Marina Machado, Luisa Souza Moura, Dominik Krzemiński, Hakimeh Fadaei, Irem Ergün, Ifeoma Okoh, Aisha Alaagib, Oshan Mudannayake, Zaid Alyafeai, Vu Minh Chien, Sebastian Ruder, Surya Guthikonda, Emad A. Alghamdi, Sebastian Gehrmann, Niklas Muennighoff, Max Bartolo, Julia Kreutzer, Ahmet Üstün, Marzieh Fadaee, and Sara Hooker. Aya dataset: An open-access collection for multilingual instruction tuning, 2024.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. Carrot: A cost aware rate optimal router. *arXiv preprint arXiv:2502.03261*, 2025.

Adnane Souha, Charaf Ouaddi, Lamya Benaddi, and Abdeslam Jakimi. Pre-trained models for intent classification in chatbot: Comparative study and critical analysis. In *2023 6th international conference on advanced communication technologies and networking (CommNet)*, pp. 1–6. IEEE, 2023.

Dimitris Stripelis, Zijian Hu, Jipeng Zhang, Zhaozhuo Xu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Salman Avestimehr, and Chaoyang He. Tensoropera router: A multi-model router for efficient llm inference. *arXiv preprint arXiv:2408.12320*, 2024.

Yixuan Tang and Yi Yang. Pooling and attention: What are effective designs for llm-based embedding models? *arXiv preprint arXiv:2409.02727*, 2024.

Chongyang Tao, Tao Shen, Shen Gao, Junshuo Zhang, Zhen Li, Kai Hua, Wenpeng Hu, Zhengwei Tao, and Shuai Ma. Llms are also effective embedding models: An in-depth overview. *arXiv preprint arXiv:2412.12591*, 2024.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. *arXiv e-prints*, pp. arXiv–2308, 2023.

Hongyi Wang, Felipe Maia Polo, Yuekai Sun, Souvik Kundu, Eric Xing, and Mikhail Yurochkin. Fusing models with complementary expertise. *arXiv preprint arXiv:2310.01542*, 2023a.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In *ACL (1)*, pp. 11897–11916, 2024. URL https://doi.org/10.18653/v1/2024.acl-long.642.

Zhiqiang Wang, Yiran Pang, and Yanbin Lin. Large language models are zero-shot text classifiers. *arXiv preprint arXiv:2312.01044*, 2023b.

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

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. *arXiv preprint arXiv:2312.02120*, 2023.

Fangzhou Wu and Sandeep Silwal. Efficient training-free online routing for high-volume multi-llm serving. *arXiv preprint arXiv:2509.02718*, 2025.

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.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *arXiv preprint arXiv:2310.03094*, 2023.

Zesen Zhao, Shuowei Jin, and Z Morley Mao. Eagle: Efficient training-free router for multi-llm inference. *arXiv preprint arXiv:2409.15518*, 2024.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

A Additional Discussion

A.1 Training/Calibration Cost

Both training-based and training-free methods add negligible latency to intent classification and their computational overhead is minimal compared to the LLM prefill phase. The fundamental difference lies in how adaptable these methods are with evolving classification targets. When adding a new class, training-free methods maintain constant cost: *computing statistics for the new class data*. On the contrary, training-based methods face a trade-off: either (i) regenerate training embeddings for all classes (both existing and new) from scratch to enable retraining—a process whose cost grows linearly with the total number of classes, or (ii) cache all training embeddings to enable rapid retraining at the expense of storage. Fig. 7 quantifies the storage implications of option (ii) for Qwen3-32B: as the number of classes gradually increases to $1,000$, the total storage requirement reaches approximately 20GB (primarily for cached embeddings), although the MLP weights alone require only 6MB. By comparison, NormStat requires only 1.7MB for the same $1,000$ classes—a difference of over an order of magnitude. As a result, *training-free methods are particularly well-suited for dynamic intent classification* systems where new classes are frequently added or removed.

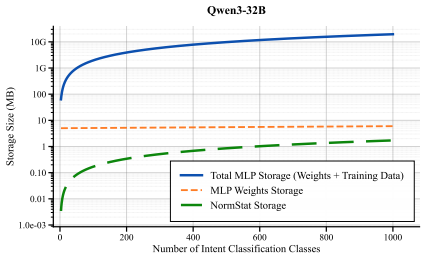


Figure 7: Storage simulation for intent classification on Qwen3-32B as the number of classes increases to 1,000.

A.2 Inference-Time Runtime Cost

Let T, d, and m denote the prompt length, LLM hidden width, and number of intent classes, respectively. Given the prefill features, the inference-time costs of the two methods are

$$NormStat = O(Td + m), \qquad VecStat = O(Td + md).$$

For both methods, the $O(Td)$ term computes the prompt statistics. NormStat then performs m scalar class comparisons, whereas VecStat compares d-dimensional coordinate-wise statistics against each class. When $m \ll T$, the shared $O(Td)$ term dominates both methods, so NormStat and VecStat have similar inference costs. In the opposite regime $m \gg T$, VecStat's $O(md)$ scoring term dominates its statistic-computation cost, and its class-scoring cost is roughly a factor of d larger than NormStat's $O(m)$ scoring cost. Nevertheless, in typical routing settings where $m \ll T(d + T)$, VecStat's scoring overhead remains small relative to the LLM prefill cost, which is $\Theta(Td^2 + T^2d)$. When m becomes very large, NormStat is preferable because its scoring cost scales only as $O(m)$.

A.3 When are training-free methods preferable?

- High-throughput, multi-tenant systems. A provider may host a large number of routers (per product, per customer, or per domain), where the intent label space evolves over time as new tools, experts, or domains are introduced. In such settings, every change in the class set would require retraining an MLP head, whereas VecStat/NormStat only require adding or removing per-class statistics—keeping the adaptation cost essentially constant as the system scales.
- Untrusted or safety-critical deployments. In systems that must handle untrusted inputs (e.g. public-facing assistants), reliable uncertainty quantification is crucial for detecting malicious or out-of-distribution prompts, and human interpretability is needed for post-hoc audits (e.g., to assess potential fairness issues). Our training-free methods directly expose calibrated per-class statistics, which can be inspected and monitored without the additional modeling and engineering complexity required to obtain well-calibrated uncertainty estimates from MLP heads.

A.4 Limitations in Experiment Design

Our evaluation studies routing an input sequence at a fixed decision point and relies primarily on public benchmark datasets, since realistic production routing logs are confidential and generally unavailable. Our mixed-intent dataset introduces controlled ambiguity by varying the ratio and ordering of math and code content, but it does not capture the full range of underspecified, conversational, domain-overlapping, or history-dependent requests encountered in deployment. Although the routed sequence could in principle be a single prompt, a concatenated conversation history, or a summarized dialogue state, our experiments instantiate it primarily with single-turn prompts and do not evaluate dynamic re-routing as a conversation evolves. Such re-routing may require a newly selected downstream model to process the full conversation history, adding prefill cost and latency. Nonetheless, evaluating these methods on realistic routing traffic and multi-turn conversations, together with the resulting trade-offs among accuracy, cost, and latency, remains important future work.

B Additional Theoretical Analysis

B.1 Two endpoints on the compression ladder

For each class $k \in [m]$, per-class baselines are computed at the same module W_ℓ on calibration data. We compare the two methods in terms of FLOPs and memory.

VecStat. **Method 1:** Compute $(S_{\text{vec}}, Q_{\text{vec}})$ as in (1). With per-class parameters $(\mu_k, \Sigma_k = \text{Diag}(\sigma_{k,1}^2, \ldots, \sigma_{k,d}^2))$. The log-likelihood ratio (LLR) between classes i and j is

$$\log \frac{p_i(Y)}{p_j(Y)} = -\frac{T}{2} \sum_{m=1}^{d} \left[\log \frac{\sigma_{i,m}^2}{\sigma_{j,m}^2} + \frac{Q_{\text{vec},m} - 2\mu_{i,m} S_{\text{vec},m} + \mu_{i,m}^2}{\sigma_{i,m}^2} - \frac{Q_{\text{vec},m} - 2\mu_{j,m} S_{\text{vec},m} + \mu_{j,m}^2}{\sigma_{j,m}^2} \right], \tag{4}$$

which is equivalently the average of coordinate-wise Gaussian KLs (since Σ_k is diagonal):

$$\sum_{m=1}^{d} \text{KL}\big(\mathcal{N}(\mu_{i,m}, \sigma_{i,m}^2) \,\|\, \mathcal{N}(\mu_{j,m}, \sigma_{j,m}^2)\big) = \frac{1}{2} \sum_{m=1}^{d} \left[\log \frac{\sigma_{j,m}^2}{\sigma_{i,m}^2} + \frac{\sigma_{i,m}^2 + (\mu_{i,m} - \mu_{j,m})^2}{\sigma_{j,m}^2} - 1 \right]. \tag{5}$$

Method 2: Using S_{vec}, classify via

$$\cos_k(S_{\text{vec}}, \mu_k) = \frac{\langle S_{\text{vec}}, \mu_k \rangle}{\|S_{\text{vec}}\| \|\mu_k\|}, \qquad \hat{b} = \arg\max_{k \in [m]} \cos_k(S_{\text{vec}}, \mu_k).$$

Costs: *Per-token compute:* $\Theta(d)$. *Prompt-state:* $O(d)$. *Baseline storage:* $O(md)$ numbers. (If only cosine scoring is used, Q_{vec} need not be stored.)

NormStat. **Method:** Compute $(S_{\text{norm}}, Q_{\text{norm}})$ as in (2), then compare $(S_{\text{norm}}, Q_{\text{norm}})$ to per-class baselines $(\mu_{x,k}, \sigma_{x,k}^2)$ via a 1D Gaussian KL:

$$\text{KL}\big(\mathcal{N}(\mu_{x,i}, \sigma_{x,i}^2) \| \mathcal{N}(\mu_{x,j}, \sigma_{x,j}^2)\big) = \frac{1}{2} \left[\log \frac{\sigma_{x,j}^2}{\sigma_{x,i}^2} + \frac{\sigma_{x,i}^2 + (\mu_{x,i} - \mu_{x,j})^2}{\sigma_{x,j}^2} - 1 \right]. \tag{6}$$

Cost. *Per-token compute:* $\Theta(d)$. *Prompt-state:* $O(1)$. *Baseline storage:* $O(m)$ scalars.

B.2 Sufficiency

We establish the minimal sufficiency of $(S_{\text{vec}}, Q_{\text{vec}})$ for the diagonal–Gaussian model, which is a classical result, see (Lehmann & Casella, 1998; Lehmann & Scheffé, 2011). Intuitively, a sufficient statistic is a lossless compression for inference about the unknown class/parameters: once $(S_{\text{vec}}, Q_{\text{vec}})$ is known, the raw sample Y contains no further information. Minimal sufficiency means no additional compression is possible without losing information—every other sufficient statistic is a measurable function of $(S_{\text{vec}}, Q_{\text{vec}})$.

Lemma 1. *Under* (3) *with diagonal* Σ_k*, the pair* $(S_{\text{vec}}, Q_{\text{vec}})$ *is a* minimal sufficient statistic *for the family* $\{\mathcal{N}(\mu_k, \Sigma_k)^{\otimes T}\}$*, and the class LLR* (4) *depends on the data only through* $(S_{\text{vec}}, Q_{\text{vec}})$*.*

The proof is provided in Appendix C.

C Proofs

C.1 Proof of Lemma 1

Proof of Lemma 1. Let $Y := (y_1, \ldots, y_T) \in \mathbb{R}^{T \times d}$. For a class k, let

$$\theta_k = (\mu_k, \sigma_{k,1}^2, \ldots, \sigma_{k,d}^2), \qquad \Sigma_k = \text{Diag}(\sigma_{k,1}^2, \ldots, \sigma_{k,d}^2).$$

Define the token-wise *sums*

$$S := \sum_{t=1}^{T} y_t \in \mathbb{R}^d, \qquad Q := \sum_{t=1}^{T} (y_t \odot y_t) \in \mathbb{R}^d,$$

and write $S_m := \sum_{t=1}^{T} y_{t,m}$, $Q_m := \sum_{t=1}^{T} y_{t,m}^2$ for coordinates $m = 1, \ldots, d$. These relate to the averaged statistics in (1) as follows:

$$S = TS_{\text{vec}}, \qquad Q = (T-1)Q_{\text{vec}} + T\big(S_{\text{vec}} \odot S_{\text{vec}}\big)$$

Since (minimal) sufficiency is invariant under invertible reparameterizations of the statistic, we may work with (S, Q) and translate back to $(S_{\text{vec}}, Q_{\text{vec}})$ via the identities above.

Sufficiency. Note that y_t are i.i.d. with density

$$p_{\theta_k}(y) = \frac{1}{(2\pi)^{d/2} \prod_{m=1}^{d} \sigma_{k,m}} \exp\left(-\frac{1}{2} \sum_{m=1}^{d} \frac{(y_m - \mu_{k,m})^2}{\sigma_{k,m}^2}\right).$$

Hence the joint density of Y under class k is

$$\begin{aligned} p_{\theta_k}(Y) &= \frac{1}{(2\pi)^{Td/2} \prod_{m=1}^{d} \sigma_{k,m}^T} \exp\left(-\frac{1}{2} \sum_{t=1}^{T} \sum_{m=1}^{d} \frac{(y_{t,m} - \mu_{k,m})^2}{\sigma_{k,m}^2}\right) \\ &= \underbrace{\frac{1}{(2\pi)^{Td/2}}}_{:=h(Y)} \cdot \underbrace{\frac{1}{\prod_{m=1}^{d} \sigma_{k,m}^T} \exp\left(-\frac{1}{2} \sum_{m=1}^{d} \frac{Q_m - 2\mu_{k,m} S_m + T\mu_{k,m}^2}{\sigma_{k,m}^2}\right)}_{:=g_{\theta_k}(S,Q)}. \end{aligned}$$

Thus $p_{\theta_k}(Y) = h(Y)\, g_{\theta_k}(S, Q)$. By the Neyman–Fisher factorization theorem, (S, Q) is sufficient for θ_k, and hence $(S_{\text{vec}}, Q_{\text{vec}})$ is sufficient by the invertible mapping above.

Minimality. Using the Lehmann–Scheffé characterization: a statistic $T(Y)$ is minimal sufficient iff for any Y, Y' the likelihood ratio $p_\theta(Y)/p_\theta(Y')$ is free of θ if and only if $T(Y) = T(Y')$. For our family,

$$\frac{p_\theta(Y)}{p_\theta(Y')} = \exp\left(-\frac{1}{2} \sum_{m=1}^{d} \frac{(Q_m - Q'_m) - 2\mu_m (S_m - S'_m)}{\sigma_m^2}\right).$$

If $(S, Q) = (S', Q')$ then this ratio equals 1, hence is parameter–free. Conversely, if for some m either $S_m \neq S'_m$ or $Q_m \neq Q'_m$, the exponent depends on μ_m (when $S_m \neq S'_m$) or on σ_m^2 (when $Q_m \neq Q'_m$); thus the ratio cannot be constant in θ.

Moreover, for classes i and j, subtracting the two log-likelihoods above yields

$$\log \frac{p_i(Y)}{p_j(Y)} = -\frac{1}{2} \sum_{m=1}^{d} \left(T \log \frac{\sigma_{i,m}^2}{\sigma_{j,m}^2} + \frac{Q_m - 2\mu_{i,m} S_m + T\mu_{i,m}^2}{\sigma_{i,m}^2} - \frac{Q_m - 2\mu_{j,m} S_m + T\mu_{j,m}^2}{\sigma_{j,m}^2}\right),$$

which is exactly (4) and depends on Y only through (S, Q), and equivalently only through $(S_{\text{vec}}, Q_{\text{vec}})$ via the identities at the start of the proof.

This establishes that $(S_{\text{vec}}, Q_{\text{vec}})$ is minimal sufficient and that the LLR depends on the sample only through this pair.

□

C.2 Proof of Theorem 1

Proof of Theorem 1. **Directional regime:** Since $\|\mu_1\| = \|\mu_2\|$, there exists an orthogonal matrix U with $U\mu_1 = \mu_2$. If $Y \sim \mathcal{N}(\mu_1, \sigma^2 I_d)$ then $UY \sim \mathcal{N}(\mu_2, \sigma^2 I_d)$ and $\|UY\| = \|Y\|$. Thus for each t, $\|y_t\| \mid k = 1$ and $\|y_t\| \mid k = 2$ have the same distribution, and by independence the vectors $(\|y_t\|)_{t=1}^T \mid k = 1$ and $(\|y_t\|)_{t=1}^T \mid k = 2$ are identically distributed. With a uniform prior, any decision rule that depends only on $\{\|y_t\|\}$ has the same acceptance probability under both classes, so its Bayes error is $1/2$.

For the log-likelihood ratio test (LRT), the log-likelihood ratio for two Gaussians with common covariance $\sigma^2 I_d$ is

$$\Lambda(y_{1:T}) = \frac{T}{\sigma^2} \langle S_{\text{vec}}, \mu_1 - \mu_2 \rangle - \frac{T}{2\sigma^2} (\|\mu_1\|^2 - \|\mu_2\|^2).$$

With equal priors the LRT accepts $k = 1$ iff $\Lambda \geq 0$. Under $\|\mu_1\| = \|\mu_2\|$, the constant term vanishes and the decision reduces to the sign of $\langle S_{\text{vec}}, \mu_1 - \mu_2 \rangle$, i.e., to $\hat{k}$ above.

Let $u := (\mu_1 - \mu_2)/\|\mu_1 - \mu_2\|$ and $Z := \langle S_{\text{vec}}, u \rangle$. Since $S_{\text{vec}} \mid k \sim \mathcal{N}(\mu_k, \frac{\sigma^2}{T} I_d)$ and $\|u\| = 1$,

$$Z \mid k \sim \mathcal{N}\left(\langle \mu_k, u \rangle, \frac{\sigma^2}{T} \right), \quad \langle \mu_1, u \rangle = \tfrac{1}{2}\|\mu_1 - \mu_2\|, \ \langle \mu_2, u \rangle = -\tfrac{1}{2}\|\mu_1 - \mu_2\|.$$

Hence, by symmetry,

$$\Pr\left(\hat{k}(y_{1:T}) \neq k\right) = \Pr_{k=1}(Z < 0) = \Phi\left(-\frac{\|\mu_1 - \mu_2\|}{2\sigma} \sqrt{T} \right) \leq \exp\left(-\frac{T}{8\sigma^2} \|\mu_1 - \mu_2\|^2 \right),$$

where Φ is the standard normal CDF and the last step uses $\Phi(-x) \leq e^{-x^2/2}$ for $x \geq 0$.

Isotropic-scale regime. Let $\phi_d(\,\cdot\,; m, \Sigma)$ denote the d-variate Gaussian density. For $k \in \{1, 2\}$, the *joint* density of $y_{1:T}$ under class k is $p_k(y_{1:T}) := \prod_{t=1}^T \phi_d\left(y_t; 0, \sigma_k^2 I_d\right)$.

With $\mu_1 = \mu_2 = 0$, $\Sigma_k = \sigma_k^2 I_d$, and $R_T = \sum_{t=1}^T \|y_t\|^2$, one can calculate the log-likelihood ratio

$$\log \frac{p_1(y_{1:T})}{p_2(y_{1:T})} = \sum_{t=1}^T \log \frac{\phi_d(y_t; 0, \sigma_1^2 I_d)}{\phi_d(y_t; 0, \sigma_2^2 I_d)} = \frac{dT}{2} \log \frac{\sigma_2^2}{\sigma_1^2} + \frac{1}{2} \left(\frac{1}{\sigma_2^2} - \frac{1}{\sigma_1^2} \right) R_T.$$

The right-hand side is an affine (hence strictly monotone when $\sigma_1 \neq \sigma_2$) function of R_T. By the Neyman–Pearson lemma, any Bayes–optimal test is a threshold on R_T, so purely radial statistics are sufficient for optimality and coordinate-wise additions cannot lower the Bayes risk. □

C.3 Proof of Theorem 2

Theorem 3. *Fix a class k. Let $y_1, \ldots, y_N \overset{i.i.d.}{\sim} \mathcal{N}(\mu_k, \Sigma_k)$ in $\mathbb{R}^d$, where N is the number of calibration samples drawn for this class. Define*

$$\hat{\mu}_k := \frac{1}{N} \sum_{i=1}^N y_i, \quad \hat{q} := \frac{1}{N} \sum_{i=1}^N \|y_i\|^2, \quad \sigma_{\max}^2 := \|\Sigma_k\|_{\text{op}}.$$

For coordinate variances, write $\sigma_{k,j}^2 := (\Sigma_k)_{jj}$ and $\widehat{\sigma}_{k,j}^2 := \frac{1}{N} \sum_{i=1}^N (y_{i,j} - \hat{\mu}_{k,j})^2$, $j = 1, \ldots, d$. Then:

1. ***NormStat (dimension-free).*** *For $q = \|y\|^2$, one has*

$$\mathbb{E}[q] = \|\mu_k\|^2 + \operatorname{Tr}(\Sigma_k), \quad \operatorname{Var}(q) = 2\operatorname{Tr}(\Sigma_k^2) + 4\mu_k^\top \Sigma_k \mu_k.$$

By Bernstein's inequality for sub-exponential variables, for all $\delta \in (0, 1)$,

$$|\hat{q} - q| \lesssim \sqrt{\operatorname{Var}(q) \log(1/\delta)/N} \quad \textit{with probability at least } 1 - \delta.$$

Normalizing by d makes the bound $O((\log(1/\delta)/N)^{1/2})$, i.e. dimension-free.

2. ***VecStat (dimension-dependent).*** *With probability at least* $1-\delta$,

$$\|\hat{\mu}_k - \mu_k\|_2 \le C_1 \sigma_{\max} \sqrt{\frac{d + \log(1/\delta)}{N}}, \quad \max_j \left|\widehat{\sigma}^2_{k,j} - \sigma^2_{k,j}\right| \le C_2 \sigma^2_{\max} \sqrt{\frac{\log(d/\delta)}{N}},$$

for absolute constants C_1, C_2. *To keep LLR plug-in error small of order* ϵ, *one needs* $N = \Omega(d/\epsilon^2_\mu)$ *for mean accuracy in* ℓ_2 *and* $N = \Omega(\log d/\epsilon^2_\sigma)$ *for variances in* ℓ_∞.

Proof of Theorem 2. **NormStat:** Denote $q_i := \|y_i\|^2$ and $Z_i := q_i - \mathbb{E}[q]$, so that $\hat{q} - \mathbb{E}[q] = \frac{1}{N}\sum_{i=1}^N Z_i$. For $y_i \sim \mathcal{N}(\mu_k, \Sigma_k)$, the centered quadratic form Z_i obeys the Hanson–Wright tail bound: there exist absolute constants $c_1, c_2 > 0$ such that for all $t > 0$,

$$\Pr\left(|Z_i| \ge t\right) \;\le\; 2\exp\left[-c_1 \min\left(\frac{t^2}{\mathrm{Var}(q)}, \frac{t}{B}\right)\right], \tag{7}$$

where $\mathrm{Var}(q) = 2\,\mathrm{Tr}(\Sigma_k^2) + 4\,\mu_k^\top \Sigma_k \mu_k$, $B = \|\Sigma_k\|_{\mathrm{op}} + \|\mu_k\|^2$. From (7), the Z_i are i.i.d. mean-zero sub-exponential. A standard Bernstein inequality for sums of independent sub-exponential variables then yields, for some absolute $c > 0$ and all $t > 0$,

$$\Pr\left(\left|\frac{1}{N}\sum_{i=1}^N Z_i\right| \ge t\right) \le 2\exp\left[-cN \min\left(\frac{t^2}{\mathrm{Var}(q)}, \frac{t}{B}\right)\right].$$

Choosing $t \lesssim \mathrm{Var}(q)/B$ and inverting the tail gives, for any $\delta \in (0,1)$,

$$|\hat{q} - \mathbb{E}[q]| \;\lesssim\; \sqrt{\frac{\mathrm{Var}(q)\log(2/\delta)}{N}} \quad \text{with probability at least } 1-\delta.$$

Since under bounded eigenvalues $\mathrm{Var}(q) = \Theta(d)$, dividing by d yields $\left|\frac{1}{d}\hat{q} - \frac{1}{d}\mathbb{E}[q]\right| \lesssim \sqrt{\frac{\log(2/\delta)}{N}}$, which is dimension-free.

VecStat: Let $z_i := \Sigma_k^{-1/2}(y_i - \mu_k) \sim \mathcal{N}(0, I_d)$. Then

$$\hat{\mu}_k - \mu_k = \Sigma_k^{1/2}\left(\frac{1}{N}\sum_{i=1}^N z_i\right) \sim \mathcal{N}\left(0,\ \frac{1}{N}\Sigma_k\right).$$

Hence, $\|\Sigma_k^{-1/2}(\hat{\mu}_k - \mu_k)\|_2^2 \sim \frac{1}{N}\chi^2_d$. Recall the standard Laurent-Massart inequalities: for any $x > 0$,

$$\Pr\left(\chi^2_d - d \ge 2\sqrt{dx} + 2x\right) \le e^{-x}, \quad \Pr\left(d - \chi^2_d \ge 2\sqrt{dx}\right) \le e^{-x}. \tag{8}$$

Applying (8) with $x = \log(1/\delta)$ and scaling by $1/N$ yields, with probability $\ge 1-\delta$,

$$\|\Sigma_k^{-1/2}(\hat{\mu}_k - \mu_k)\|_2 \le \sqrt{\frac{1}{N}\left(d + 2\sqrt{d\log(1/\delta)} + 2\log(1/\delta)\right)}.$$

Multiplying by $\|\Sigma_k^{1/2}\|_{\mathrm{op}} = \sigma_{\max}$ and using $\sqrt{a+b} \le \sqrt{a} + \sqrt{b}$ gives

$$\|\hat{\mu}_k - \mu_k\|_2 \le C_1 \sigma_{\max} \sqrt{\frac{d + \log(1/\delta)}{N}},$$

for an absolute constant $C_1 > 0$.

□

D Experiment Details

D.1 Dataset Composition and Processing Strategies

We evaluate our intent classification methods across two levels of granularity. The experimental protocol consists of two stages: calibration and classification. During calibration, we compute per-class baseline statistics (NormStat or VecStat) from calibration data passed through pretrained LLMs. During classification, we compute the same statistics for test prompts and assign labels based on the minimum KL divergence (or cosine distance) between the prompt's statistics and the calibrated per-class baselines.

Table 6: Dataset composition for Task 1: general intent classification between general text, mathematics, and code (level-1 granularity).

Category	Calibration Data	# Calibration Samples	Classification Data	# Classification Samples
General Text	MMLU (European History)	165	MMLU (US History)	204
Math	GSM8K	2,000	GSM8K MATH500	1,319 500
Code	Magicoder	2,000	Magicoder HumanEval	5,000 164

Table 7: Dataset composition for Tasks 2 and 3: programming language identification and natural language identification (both are level-1 granularity).

Task	Data Source	Classes	# Calibration Samples per Class	# Classification Samples per Class
Task 2	Magicoder	C++, C#, Java, PHP, Python, Rust, Shell, Swift, TypeScript	2,000	5,000
Task 3	Aya	Sinhala, Tamil, English, Moroccan Arabic, Japanese	512	3,000

Classification tasks and granularities. We consider four classification tasks spanning two levels of granularity.

- **Task 1** (level-1): general intent classification among three broad domains—general text, mathematics, and code. This task represents the canonical routing scenario where prompts are directed to specialized models based on broad category.
- **Task 2** (level-1): programming language identification across nine languages within code prompts.
- **Task 3** (level-1): natural language identification across five linguistically diverse languages.
- **Task 4** (level-2): fine-grained mathematical subfield classification across seven topics within mathematics.

Tasks 1–3 are coarse-grained (level-1), while Task 4 is fine-grained (level-2).

Datasets and evaluation. Tables 6 to 8 present the complete dataset composition for Tasks 1–4.

For Task 1, we calibrate using domain-representative datasets: MMLU European History for general text (165 samples), GSM8K for mathematics (2,000 samples), and Magicoder for code (2,000 samples). Classification employs both in-distribution and out-of-distribution test sets to assess generalization: MMLU US History for general text, GSM8K (in-distribution) and MATH500 (out-of-distribution) for mathematics, and Magicoder (in-distribution) and HumanEval (out-of-distribution) for code.

For Tasks 2–4, we maintain consistent calibration sizes where feasible: 2,000 samples per programming language (Task 2), 512 samples per natural language (Task 3), and 800 samples per mathematical subfield (Task 4). Classification sets contain up to 5,000 samples per class for Task 2, and up to 3,000 samples per class for Tasks 3 and 4, subject to dataset availability. Further details on each dataset are provided in Appendix D.2.

D.2 Datasets

We employ seven benchmark datasets spanning general text, mathematics, and code domains.

- **General Text.** For Task 1, we use MMLU (Hendrycks et al., 2021a;b), a comprehensive benchmark of multiple-choice questions across 57 subjects. Calibration data is drawn

Table 8: Dataset composition for Task 4: mathematical subfield classification (level-2 granularity).

Task	Data Source	Classes	# Calibration Samples per Class	# Classification Samples per Class
Task 4	Competition Math	Algebra, Counting & Probability, Geometry, Intermediate Algebra, Number Theory, Prealgebra, Precalculus	800	3,000

from the High School European History subset, and evaluation is performed on the High School US History subset, using question-and-choice pairs as input.

- **Natural Language.** For Task 3, we use the Aya dataset (Singh et al., 2024), which contains human-annotated prompts across 65 languages. We select the five linguistically diverse languages specified in Table 7, splitting each language subset into calibration and test sets, and use the input field for classification.
- **Mathematics.** We employ three benchmarks. GSM8K (Cobbe et al., 2021) provides grade-school word problems requiring multi-step reasoning; we sample 2,000 instances for calibration and use the full test set for Task 1 evaluation. MATH500 (Lightman et al., 2023) serves as an out-of-distribution test set for Task 1, containing 500 problems from the MATH benchmark. Competition Math (Hendrycks et al., 2021c) provides competition problems spanning seven mathematical subfields and is used for Task 4; we use the problem field as model input.
- **Code.** Magicoder (Wei et al., 2023) is our primary code resource, containing solutions in multiple programming languages. We use the solution field as model input for both Task 1 (general code classification) and Task 2 (programming language identification), focusing on the nine languages listed in Table 7. HumanEval (Chen et al., 2021) provides 164 function-level programming problems; we use the prompt field (containing function signatures and docstrings) as model input, serving as an out-of-distribution test set for Task 1.

All datasets are publicly available through HuggingFace Datasets (Lhoest et al., 2021). Sampling strategies and train-test splits follow the specifications in Tables 6 to 8, with test samples capped at the minimum of the specified count and available data.

D.3 Selected LLMs

We evaluate our approach on 7 pretrained large language models spanning 1B to 32B parameters, encompassing both base and instruction-tuned variants. This selection provides comprehensive coverage across model scales and training stages. We consider the following two LLM families:

- **Qwen family** (Yang et al., 2025): We evaluate four models from the Qwen3 series. The instruction-tuned variants include Qwen3-1.7B (28 layers), Qwen3-4B (36 layers), Qwen3-8B (36 layers), and Qwen3-32B (64 layers), each post-trained with supervised fine-tuning and reinforcement learning from human feedback (RLHF). Additionally, we include Qwen3-1.7B-Base to assess performance on pretrained models without alignment. For all Qwen3 evaluations, we switch on non-thinking mode to ensure consistent comparison across models.
- **Llama family** (Dubey et al., 2024): We evaluate Llama-3.2-1B (16 layers) and its instruction-tuned counterpart Llama-3.2-1B-Instruct. The instruction-tuned variant underwent supervised fine-tuning and RLHF to better align with human preferences.

This benchmark model selection enables systematic evaluation across three critical dimensions: model scale (from 1B to 32B parameters), training paradigm (pretrained-only versus post-trained), and architectural diversity (Qwen and Llama families). The substantial range in model sizes—spanning over an order of magnitude in parameters—allows us to rigorously test whether our method can effectively operate across vastly different computational

Table 9: Prompt template for direct LLM inference under level-1 intent classification setting. The template includes placeholders {`examples_block`} for optional few-shot examples and {`user_text`} for the input to be classified. The few-shot examples are taken from the calibration data.

LLM Prompt Template for Intent Classification
You are a classifier. Your task is to look at the user's input text and decide which of these three categories it belongs to: 1. General text – natural language content like sentences, questions, explanations, stories, or instructions that are not primarily mathematics or code. 2. Math – text that is primarily mathematical expressions, equations, formulas, or word problems where the main focus is on mathematics. 3. Code – text that is primarily programming code or pseudocode (any programming language, including configuration snippets or shell commands). Output rules: - If the input is general text, output: A - If the input is math, output: B - If the input is code, output: C Important Notes: - Output only a single letter: A, B, or C. - Do not output anything else (no explanation, no punctuation, no spaces). Below are some classification examples for this task: **{examples_block}** Now classify the following input accordingly and output just one letter. Input: **{user_text}** Output:

scales and model capacities. The comparison between base and aligned models reveals how post-training procedures affect our method's performance, demonstrating whether it remains equally effective for both pretrained and instruction-tuned models. The prompt used for direct LLM inference under level-1 setting is in Table 9.

D.4 More Implementation Details

To stabilize training, we normalize input features and weight matrices through the following process:

1. Input Normalization: The input tensor z is normalized to unit norm for stability:

$$z_{\text{normalized}} = \frac{z}{\|z\|_2 + \epsilon}.$$

2. Weight Normalization: The weight matrix W is normalized using its Frobenius norm:
$$W_{\text{normalized}} = \frac{W}{\sqrt{\text{mean}(W^2)} + \epsilon}.$$
3. Activation Computation: The normalized input is multiplied by the normalized weight matrix:
$$Wz = z_{\text{normalized}} \cdot W_{\text{normalized}}^T.$$

The statistics for NormStat and VecStat are computed from Wz. Note that this is likely suboptimal; in a production-scale implementation we should read Wz directly from the module's output rather than recomputing it.

D.5 Pseudo-Algorithm

Our task inference approach follows the following steps:

Algorithm 1 Intent Classification with NormStat or VecStat

Require: Input prompt x, baseline scores $\{S_1, S_2, \ldots, S_m\}$ for tasks $C_1, C_2, \ldots, C_m$, distance function "dist" (NormStat: KL; VecStat: KL or cosine similarity)
Ensure: Predicted task T_{pred} and confidence scores
1: Compute statistic scores S_x from input x
2: **for** each task C_i **do**
3: Compute distance $d_i = \text{dist}(S_x, S_i)$
4: **end for**
5: $C_{\text{pred}} = \arg\min_{C_i} d_i$
6: Compute probabilities via softmax: $p_i = \frac{\exp(-(d_i - \bar{d})/\tau)}{\sum_j \exp(-(d_j - \bar{d})/\tau)}$ (used for uncertainty quantification, $\bar{d}$ is the average, τ is the temperature)
7: **return** $C_{\text{pred}}, \{p_1, p_2, \ldots, p_m\}$

D.6 Hardware and software environment

We conducted experiments on two computational platforms based on model scales. For models up to 4B parameters, we utilized an NVIDIA L40S GPU with 48GB of memory. For larger models (Qwen3-8B and Qwen3-32B), experiments were performed on an NVIDIA Grace Hopper GH200 superchip, featuring a Grace ARM 72-core CPU with 120GB RAM and a NVIDIA H100 GPU with 96GB of memory. All experiments are implemented using Python 3.12.0 and PyTorch 2.7.0 with CUDA 12.6.

E Additional Experimental Results

E.1 Additional Results for Tasks 1–3 (Level-1)

Table 10 reports per-dataset accuracy for Task 1 (general intent classification) across all seven LLMs. Table 11 reports per-language accuracy for Task 2 (programming language identification). Table 12 reports per-language accuracy for Task 3 (natural language identification).

For Task 2, VecStat maintains near-perfect accuracy while NormStat shows substantial degradation, aligning with our theoretical prediction that directional information becomes critical for within-domain discrimination. This performance gap is consistent across all nine programming languages. In contrast, for Task 3, all methods perform strongly, with NormStat remaining competitive, reflecting stronger radial separation among natural language classes.

Table 10: Task 1 classification results for all seven LLMs (level-1 granularity). Values represent per-dataset accuracy for coarse-grained domain classification (general text, mathematics, and code).

Model	Method	gsm8k	humaneval	magicoder	math500	mmlu_history
Qwen3-1.7B	Avg-MLP	99.97±0.04	99.80±0.35	99.99±0.02	78.33±1.67	100.00±0.00
	Tail-MLP	100.00±0.00	100.00±0.00	99.97±0.02	99.00±0.00	100.00±0.00
	Avg-Linear	100.00±0.00	99.80±0.35	100.00±0.00	69.20±9.18	100.00±0.00
	NormStat:KL	97.35±0.00	70.12±0.61	99.27±0.11	76.60±0.20	100.00±0.00
	VecStat:KL	100.00±0.00	99.39±0.00	99.97±0.03	88.33±0.30	100.00±0.00
	VecStat:Cos	100.00±0.00	98.78±0.00	99.97±0.03	92.26±0.11	100.00±0.00
	LLM Call (0-shot)	51.18±0.30	99.39±0.61	97.38±0.11	80.87±0.61	82.19±1.02
Qwen3-1.7B-Base	Avg-MLP	78.09±31.54	99.80±0.35	100.00±0.00	40.60±19.91	100.00±0.00
	Tail-MLP	100.00±0.00	84.55±26.76	99.87±0.18	99.00±0.00	100.00±0.00
	Avg-Linear	79.76±0.53	99.59±0.70	100.00±0.00	30.73±4.28	100.00±0.00
	NormStat:KL	79.71±0.29	82.93±0.00	99.71±0.03	88.47±0.12	100.00±0.00
	VecStat:KL	40.46±0.10	100.00±0.00	100.00±0.0	49.06±0.95	100.00±0.00
	VecStat:Cos	99.84±0.00	99.39±0.00	99.98±0.02	92.30±0.39	100.00±0.00
	LLM Call (0-shot)	90.60±0.00	0.00±0.00	81.03±0.39	59.40±0.00	4.41±0.00
Llama-3.2-1B	Avg-MLP	100.00±0.00	100.00±0.00	99.99±0.01	64.33±8.33	99.84±0.28
	Tail-MLP	99.95±0.04	100.00±0.00	99.97±0.03	98.93±0.42	99.35±1.13
	Avg-Linear	100.00±0.00	99.80±0.35	99.99±0.01	71.47±7.78	98.69±1.86
	NormStat:KL	99.49±0.09	90.85±0.00	96.39±0.20	83.40±0.00	92.48±0.57
	VecStat:KL	100.00±0.00	99.39±0.00	99.98±0.02	78.80±0.12	100.00±0.00
	VecStat:Cos	100.00±0.00	99.39±0.00	99.97±0.02	77.60±0.20	100.00±0.00
Llama-3.2-1B-Instruct	Avg-MLP	100.00±0.00	99.59±0.35	99.99±0.02	72.40±2.91	100.00±0.00
	Tail-MLP	100.00±0.00	100.00±0.00	99.89±0.05	94.53±0.42	100.00±0.00
	Avg-Linear	100.00±0.00	99.80±0.35	100.00±0.00	66.80±7.75	100.00±0.00
	NormStat:KL	100.00±0.00	65.24±0.00	97.21±0.21	96.20±0.00	98.53±0.00
	VecStat:KL	100.00±0.00	99.39±0.00	99.96±0.01	87.87±0.11	100.00±0.00
	VecStat:Cos	100.00±0.00	99.39±0.00	99.97±0.03	85.60±0.00	100.00±0.00
Qwen3-4B	Avg-MLP	99.95±0.09	100.00±0.00	99.99±0.01	79.27±4.39	100.00±0.00
	Tail-MLP	100.00±0.00	100.00±0.00	99.96±0.02	92.27±1.17	100.00±0.00
	Avg-Linear	99.90±0.04	100.00±0.00	99.99±0.01	68.87±18.85	100.00±0.00
	NormStat:KL	96.36±0.00	35.57±0.35	99.84±0.03	89.40±0.00	99.84±0.28
	VecStat:KL	99.97±0.04	100.00±0.00	99.99±0.02	91.73±0.23	100.00±0.00
	VecStat:Cos	100.00±0.00	96.34±0.00	99.97±0.03	93.40±0.20	100.00±0.00
	LLM Call (0-shot)	91.84±0.16	100.00±0.00	99.47±0.06	98.33±0.12	95.42±0.28
Qwen3-8B	Avg-MLP	99.42±0.74	99.59±0.35	99.99±0.02	77.27±11.02	100.00±0.00
	Tail-MLP	99.82±0.12	98.58±2.46	99.98±0.02	81.20±9.72	100.00±0.00
	Avg-Linear	99.57±0.24	100.00±0.00	100.00±0.00	83.00±2.03	100.00±0.00
	NormStat:KL	85.14±0.35	10.37±1.06	99.85±0.06	92.93±0.12	99.51±0.49
	VecStat:KL	99.95±0.04	99.59±0.35	99.99±0.02	92.20±0.00	100.00±0.00
	VecStat:Cos	100.00±0.00	95.73±0.61	99.98±0.03	94.20±0.00	100.00±0.00
	LLM Call (0-shot)	99.67±0.04	100.00±0.00	99.42±0.07	99.60±0.00	99.67±0.28
Qwen3-32B	Avg-MLP	95.88±3.31	100.00±0.00	99.99±0.01	86.87±5.22	100.00±0.00
	Tail-MLP	99.39±0.00	100.00±0.00	99.98±0.00	96.80±0.40	99.02±0.49
	Avg-Linear	98.61±1.59	100.00±0.00	99.99±0.02	87.53±6.94	100.00±0.00
	NormStat:KL	97.93±0.04	24.59±0.35	99.78±0.06	97.93±0.12	100.00±0.00
	VecStat:KL	100.00±0.00	99.39±0.00	99.98±0.03	96.60±0.00	100.00±0.00
	VecStat:Cos	100.00±0.00	98.17±0.00	99.98±0.03	96.80±0.35	100.00±0.00
	LLM Call (0-shot)	86.91±0.29	100.00±0.00	98.69±0.06	96.27±0.42	100.00±0.00

E.2 Additional Results for Task 4 (Level-2)

Table 13 reports per-subfield accuracy for Task 4 (mathematical subfield classification) across all seven LLMs.

Table 11: Task 2 classification results for all seven LLMs (level-1 granularity). Values represent per-language accuracy across nine programming languages from the Magicoder dataset.

Model	Method	cpp	csharp	java	php	python	rust	shell	swift	typescript
Qwen3-1.7B	Avg-MLP	99.97±0.03	100.00±0.00	100.00±0.00	99.96±0.06	99.89±0.01	100.00±0.00	99.92±0.14	100.00±0.00	99.94±0.07
	Tail-MLP	99.54±0.09	99.57±0.06	99.41±0.26	99.89±0.11	99.35±0.27	99.32±0.17	99.92±0.14	99.81±0.07	99.52±0.03
	Avg-Linear	99.97±0.03	99.98±0.04	100.00±0.00	99.96±0.06	99.87±0.01	100.00±0.00	99.92±0.14	100.00±0.00	99.93±0.05
	NormStat:KL	53.18±0.60	53.17±1.96	40.39±0.63	57.35±1.97	60.17±0.89	58.07±0.55	88.73±1.67	59.47±0.88	49.59±0.34
	VecStat:KL	98.83±0.18	98.98±0.39	98.21±0.15	99.81±0.23	99.06±0.09	99.68±0.14	99.52±0.00	99.81±0.12	99.26±0.20
	VecStat:Cos	98.51±0.19	98.78±0.37	97.79±0.19	99.85±0.17	99.06±0.14	99.35±0.15	99.84±0.14	99.55±0.11	99.02±0.29
	LLM Call (0-shot)	94.64±0.57	99.35±0.09	88.23±0.70	99.52±0.17	99.88±0.02	98.34±0.07	99.05±0.41	99.89±0.00	89.18±0.44
Qwen3-1.7B-Base	Avg-MLP	99.97±0.03	99.96±0.04	99.97±0.03	100.00±0.00	99.89±0.05	100.00±0.00	99.92±0.14	100.00±0.00	99.96±0.04
	Tail-MLP	99.28±0.19	99.37±0.19	99.23±0.12	99.96±0.06	99.25±0.14	99.27±0.19	99.84±0.27	99.75±0.14	99.50±0.03
	Avg-Linear	99.97±0.03	99.96±0.04	99.98±0.03	100.00±0.00	99.89±0.04	100.00±0.00	99.92±0.14	100.00±0.00	99.96±0.04
	NormStat:KL	47.76±0.18	43.75±1.18	36.92±0.96	60.44±1.13	56.03±0.82	54.88±0.78	88.33±2.42	57.85±1.25	51.20±0.25
	VecStat:KL	98.29±0.13	98.37±0.53	97.00±0.21	99.66±0.11	98.63±0.09	99.45±0.12	99.68±0.14	99.64±0.12	99.17±0.19
	VecStat:Cos	98.27±0.17	98.23±0.50	96.91±0.05	99.63±0.17	98.84±0.09	99.47±0.13	99.68±0.14	99.58±0.07	98.54±0.30
Llama-3.2-1B	Avg-MLP	99.97±0.03	99.98±0.04	99.97±0.05	99.96±0.06	99.91±0.02	100.00±0.00	100.00±0.00	100.00±0.00	99.94±0.03
	Tail-MLP	99.21±0.24	99.08±0.16	99.41±0.10	99.85±0.17	99.25±0.26	99.21±0.32	99.84±0.27	99.75±0.09	99.58±0.11
	Avg-Linear	99.97±0.03	99.96±0.04	99.94±0.05	99.96±0.06	99.89±0.01	99.98±0.03	100.00±0.00	100.00±0.00	99.93±0.03
	NormStat:KL	44.23±1.37	39.54±2.26	35.09±1.07	49.25±1.60	38.65±0.71	54.91±0.78	90.95±1.56	55.63±1.24	34.67±1.81
	VecStat:KL	98.77±0.19	98.37±0.43	97.14±0.07	99.63±0.17	98.66±0.21	99.79±0.03	99.84±0.27	99.73±0.18	99.02±0.25
	VecStat:Cos	98.53±0.12	97.78±0.25	96.34±0.09	99.52±0.28	98.86±0.18	99.48±0.15	99.60±0.14	99.49±0.25	98.81±0.36
Llama-3.2-1B-Instruct	Avg-MLP	99.95±0.05	99.98±0.04	99.97±0.03	99.96±0.06	99.91±0.01	100.00±0.00	99.92±0.14	100.00±0.00	99.99±0.03
	Tail-MLP	99.07±0.39	99.23±0.15	99.12±0.40	99.55±0.22	99.27±0.11	99.00±0.49	99.84±0.27	99.70±0.16	99.64±0.04
	Avg-Linear	99.97±0.03	99.96±0.04	99.95±0.05	99.96±0.06	99.88±0.03	100.00±0.00	99.92±0.14	100.00±0.00	99.99±0.03
	NormStat:KL	41.34±1.45	45.89±0.57	33.32±1.51	45.23±2.25	46.42±1.32	55.80±0.67	89.76±2.52	57.64±0.72	34.05±2.20
	VecStat:KL	98.56±0.19	97.58±0.19	97.25±0.13	99.70±0.17	98.87±0.16	99.66±0.08	99.84±0.27	99.62±0.14	99.02±0.27
	VecStat:Cos	98.56±0.19	97.46±0.34	96.85±0.23	99.78±0.19	99.00±0.13	99.37±0.10	99.84±0.27	99.64±0.14	98.87±0.32
Qwen3-4B	Avg-MLP	99.97±0.03	99.94±0.00	100.00±0.00	99.96±0.06	99.89±0.01	100.00±0.00	100.00±0.00	100.00±0.00	99.94±0.03
	Tail-MLP	99.37±0.11	99.53±0.13	99.46±0.03	99.81±0.13	99.48±0.02	99.35±0.27	99.84±0.27	99.73±0.14	99.66±0.07
	Avg-Linear	99.97±0.03	99.96±0.07	99.98±0.03	99.96±0.06	99.89±0.02	100.00±0.00	100.00±0.00	100.00±0.00	99.93±0.05
	NormStat:KL	54.55±1.04	40.86±1.39	35.45±0.28	56.26±0.68	69.71±1.04	68.76±0.65	93.10±1.33	72.17±1.20	48.70±0.32
	VecStat:KL	99.26±0.08	98.60±0.38	97.95±0.12	99.74±0.26	99.07±0.09	99.82±0.03	99.76±0.00	99.87±0.12	99.30±0.19
	VecStat:Cos	98.92±0.09	98.86±0.44	98.18±0.05	99.89±0.19	99.13±0.08	99.53±0.10	99.84±0.14	99.79±0.13	99.26±0.23
	LLM Call (0-shot)	99.11±0.17	99.78±0.07	99.49±0.12	99.89±0.00	100.00±0.00	99.97±0.03	99.92±0.14	94.11±0.62	52.13±0.36
Qwen3-8B	Avg-MLP	99.95±0.05	100.00±0.00	100.00±0.00	99.96±0.06	99.91±0.01	100.00±0.00	100.00±0.00	100.00±0.00	99.94±0.03
	Tail-MLP	99.61±0.16	99.69±0.18	99.47±0.12	100.00±0.00	99.39±0.22	99.53±0.10	99.92±0.14	99.75±0.07	99.61±0.10
	Avg-Linear	99.97±0.03	100.00±0.00	100.00±0.00	99.96±0.06	99.91±0.03	100.00±0.00	100.00±0.00	100.00±0.00	99.93±0.03
	NormStat:KL	47.37±0.94	38.26±1.28	27.59±0.65	60.14±1.27	67.73±0.54	63.05±0.43	91.51±0.60	62.54±1.25	49.28±0.84
	VecStat:KL	99.09±0.11	98.58±0.42	97.78±0.09	99.78±0.22	99.01±0.04	99.81±0.00	99.76±0.00	99.91±0.09	99.38±0.20
	VecStat:Cos	98.90±0.24	98.96±0.38	98.49±0.03	99.85±0.13	99.13±0.12	99.55±0.07	100.00±0.00	99.81±0.12	99.35±0.19
	LM Call	99.81±0.15	100.00±0.00	99.98±0.03	100.00±0.00	100.00±0.00	100.00±0.00	99.92±0.14	100.00±0.00	90.33±0.35
Qwen3-32B	Avg-MLP	99.91±0.03	99.98±0.04	100.00±0.00	99.96±0.06	99.91±0.03	99.97±0.06	100.00±0.00	100.00±0.00	99.96±0.04
	Tail-MLP	98.01±0.41	97.27±0.95	97.99±0.53	98.77±0.22	98.26±0.21	99.21±0.06	98.97±0.60	98.71±0.48	98.50±0.07
	Avg-Linear	99.91±0.03	99.98±0.04	100.00±0.00	99.96±0.06	99.92±0.00	99.97±0.06	100.00±0.00	100.00±0.00	99.96±0.04
	NormStat:KL	53.57±0.94	36.94±1.38	25.47±1.43	59.47±0.57	61.66±1.40	67.63±0.54	89.29±0.71	69.35±1.19	49.84±1.08
	VecStat:KL	99.01±0.11	99.35±0.34	98.72±0.03	99.89±0.11	99.29±0.06	99.90±0.05	100.00±0.00	99.85±0.09	99.75±0.16
	VecStat:Cos	99.18±0.14	99.53±0.19	99.01±0.10	99.89±0.11	99.50±0.04	99.81±0.13	100.00±0.00	99.87±0.12	99.72±0.09
	LLM Call (0-shot)	99.90±0.05	100.00±0.00	99.98±0.03	99.96±0.06	99.98±0.00	100.00±0.00	99.84±0.14	100.00±0.00	98.68±0.23

E.3 Additional Results for Uncertainty Quantification for Mixed-Intent Prompts

We construct a mixed-intent dataset by interleaving samples from both math and code datasets at five known mix ratios (math:code) $r \in \{0:1,\ 0.25:0.75,\ 0.5:0.5,\ 0.75:0.25,\ 1:0\}$, assessed across two prompt orderings (code-first and math-first) to probe sensitivity to token-sequence position. For each method, a scalar temperature T is fitted via least squares to calibrate raw scores into probabilities:

$$\hat{T} = \arg\min_{T} \sum_{i} \Big(\sigma(\Delta s_i / T) - r_i \Big)^2, \quad r_i \in \{0:1,\ 0.25:0.75,\ 0.5:0.5,\ 0.75:0.25,\ 1:0\}$$

where $\Delta s_i = s_{\text{math}} - s_{\text{code}}$ is the score difference at ratio r_i ($s = -d$ for distance-based methods; $s =$ logit for MLP-based methods), and σ is the sigmoid function.

The calibration error for temperature of each method is calculated by the Root Mean Square Error (RMSE): $\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - p_i^*)^2}$, where $\hat{p}_i = P(\text{math})$ is the predicted math probability and p_i^* is the target math fraction, evaluated at mix-ratio points.

Fig. 8 presents an example of the mixed-intent prompt used in Section 3.3, constructed by concatenating code content with mathematical content. We conducted the experiments across four models, with the order *math_first* or *code_first*, with results shown in Fig. 9. Moreover, we report in Table 15 the corresponding calibration error.

Table 12: Task 3 classification results for all seven LLMs (level-1 granularity). Values represent per-language accuracy across five natural languages from the Aya dataset.

Model	Method	English	Japanese	Moroccan Arabic	Sinhala	Tamil
Qwen3-1.7B	Avg-MLP	99.74±0.12	99.99±0.02	99.97±0.03	99.99±0.02	99.94±0.04
	Tail-MLP	99.49±0.15	100.00±0.00	99.97±0.03	99.93±0.06	99.94±0.10
	Avg-Linear	99.79±0.11	100.00±0.00	99.97±0.03	99.99±0.02	99.94±0.04
	NormStat:KL	82.26±1.39	81.42±0.81	85.78±0.86	99.87±0.12	99.52±0.12
	VecStat:KL	96.27±0.34	99.83±0.03	99.97±0.00	99.99±0.02	99.93±0.06
	VecStat:Cos	98.09±0.10	99.82±0.05	99.99±0.02	99.99±0.02	99.94±0.04
	LLM Call (0-shot)	97.27±0.12	72.24±0.48	94.57±0.38	9.78±0.38	99.92±0.02
Qwen3-1.7B-Base	Avg-MLP	99.64±0.15	100.00±0.00	99.94±0.04	99.99±0.02	99.96±0.02
	Tail-MLP	99.67±0.12	100.00±0.00	99.97±0.03	99.98±0.02	99.87±0.03
	Avg-Linear	99.66±0.16	99.98±0.04	99.94±0.04	99.99±0.02	99.92±0.08
	NormStat:KL	82.58±1.88	83.08±1.35	73.59±0.57	99.83±0.09	98.43±0.17
	VecStat:KL	97.41±0.13	99.84±0.05	99.97±0.03	99.99±0.02	99.93±0.06
	VecStat:Cos	98.46±0.13	99.83±0.09	100.00±0.00	99.99±0.02	99.94±0.04
	LLM Call (0-shot)	82.37±0.30	58.67±0.67	96.14±0.41	35.30±0.88	99.26±0.07
Llama-3.2-1B	Avg-MLP	99.63±0.21	100.00±0.00	99.97±0.03	99.99±0.02	99.96±0.02
	Tail-MLP	99.60±0.18	100.00±0.00	99.97±0.03	99.87±0.00	99.71±0.28
	Avg-Linear	99.61±0.08	99.99±0.02	99.97±0.03	99.99±0.02	99.96±0.02
	NormStat:KL	77.26±6.21	57.36±3.14	98.56±0.11	99.67±0.20	99.82±0.19
	VecStat:KL	96.12±0.39	99.92±0.02	99.96±0.02	100.00±0.00	99.96±0.02
	VecStat:Cos	98.76±0.07	99.98±0.02	99.96±0.02	99.99±0.02	99.94±0.04
Llama-3.2-1B-Instruct	Avg-MLP	99.72±0.10	99.99±0.02	99.94±0.04	99.99±0.02	99.97±0.03
	Tail-MLP	99.58±0.15	100.00±0.00	99.96±0.02	99.94±0.07	99.82±0.20
	Avg-Linear	99.67±0.12	100.00±0.00	99.94±0.04	99.99±0.02	99.96±0.02
	NormStat:KL	79.47±5.07	42.79±2.16	98.54±0.18	99.86±0.13	99.86±0.07
	VecStat:KL	96.04±0.39	99.84±0.13	99.98±0.02	99.99±0.02	99.96±0.02
	VecStat:Cos	98.59±0.12	99.98±0.02	99.94±0.04	99.99±0.02	99.94±0.04
Qwen3-4B	Avg-MLP	99.71±0.10	100.00±0.00	99.94±0.04	99.99±0.02	99.96±0.02
	Tail-MLP	99.60±0.15	100.00±0.00	99.97±0.03	99.99±0.02	99.99±0.02
	Avg-Linear	99.74±0.12	100.00±0.00	99.94±0.04	99.99±0.02	99.96±0.02
	NormStat:KL	84.60±0.34	90.29±1.27	90.53±0.61	99.84±0.07	99.11±0.54
	VecStat:KL	93.99±0.82	99.77±0.09	100.00±0.00	99.99±0.02	99.93±0.06
	VecStat:Cos	97.73±0.12	99.88±0.10	100.00±0.00	99.99±0.02	99.94±0.04
	LLM Call (0-shot)	97.00±0.09	99.16±0.08	98.56±0.19	88.37±0.66	99.93±0.00
Qwen3-8B	Avg-MLP	99.70±0.06	100.00±0.00	99.94±0.04	99.99±0.02	99.97±0.00
	Tail-MLP	99.53±0.15	100.00±0.00	99.97±0.03	99.99±0.02	99.92±0.08
	Avg-Linear	99.81±0.05	100.00±0.00	99.94±0.04	99.99±0.02	99.96±0.02
	NormStat:KL	85.06±0.85	92.03±1.32	76.76±0.77	99.92±0.04	96.69±2.25
	VecStat:KL	96.27±0.38	99.82±0.05	100.00±0.00	100.00±0.00	99.93±0.06
	VecStat:Cos	98.66±0.13	99.83±0.09	100.00±0.00	99.99±0.02	99.94±0.04
	LLM Call (0-shot)	97.64±0.12	97.28±0.14	99.96±0.05	13.83±0.61	99.89±0.04
Qwen3-32B	Avg-MLP	99.76±0.17	99.99±0.02	99.94±0.04	99.99±0.02	99.96±0.04
	Tail-MLP	99.31±0.02	100.00±0.00	99.96±0.02	99.99±0.02	99.94±0.04
	Avg-Linear	99.87±0.06	100.00±0.00	99.94±0.04	99.98±0.02	99.93±0.03
	NormStat:KL	82.00±0.74	72.11±1.63	96.17±0.40	99.48±0.10	98.32±0.35
	VecStat:KL	94.08±0.27	99.68±0.18	100.00±0.00	99.99±0.02	99.94±0.04
	VecStat:Cos	98.20±0.12	99.78±0.07	100.00±0.00	99.99±0.02	99.96±0.02
	LLM Call (0-shot)	97.66±0.02	98.34±0.12	99.96±0.05	99.84±0.05	100.00±0.00

E.4 Additional Results for Robustness to Adversarial Attack

Table 14 reports full adversarial robustness results across all seven LLMs. The adversarial prompts are generated by GPT-4o using the templates described in Table 16, which specify three increasing levels of disguise: Easy adds a thin lexical code veneer, Medium introduces a function header with a prose mathematical body, and Hard embeds the problem in a

Table 13: Task 4 classification results for all seven LLMs (level-2 granularity). Values represent per-subfield accuracy across seven mathematical subfields from the Competition Math dataset.

Model	Method	Algebra	Counting & Probability	Geometry	Intermediate Algebra	Number Theory	Prealgebra	Precalculus
Qwen3-1.7B	Avg-MLP	71.78±5.57	79.48±1.91	89.98±3.46	79.33±3.96	84.02±2.56	50.00±2.29	82.79±3.37
	Tail-MLP	65.79±3.55	76.10±4.24	84.88±4.43	71.86±6.02	82.32±3.80	50.31±2.59	73.58±4.30
	Avg-Linear	64.13±3.32	75.88±0.79	89.31±1.47	81.14±3.20	79.47±4.34	52.66±2.86	79.74±1.54
	NormStat:KL	23.67±1.27	37.83±0.65	48.15±4.09	58.75±1.08	65.46±1.25	0.97±0.28	33.88±1.02
	VecStat:KL	33.29±1.39	53.71±2.00	35.88±3.28	81.88±0.58	86.26±0.62	1.46±0.05	46.07±1.82
	VecStat:Cos	57.66±0.70	61.50±2.16	36.07±3.37	73.58±0.83	86.92±0.38	2.22±0.32	51.22±1.61
	LLM Call (0-shot)	28.30±1.21	53.18±1.06	55.49±1.47	1.31±0.15	14.61±0.72	7.37±0.16	71.48±2.46
Qwen3-1.7B-Base	Avg-MLP	67.96±1.83	79.55±0.67	90.53±0.55	83.93±2.34	84.02±2.62	48.85±2.19	83.47±3.46
	Tail-MLP	70.97±4.48	80.97±2.25	90.89±1.14	74.49±2.32	81.50±6.42	51.93±7.96	78.25±2.55
	Avg-Linear	66.46±4.16	75.81±2.04	88.95±1.00	81.57±1.99	81.06±2.52	51.33±2.29	79.54±1.31
	NormStat:KL	23.35±0.70	37.23±2.12	47.36±4.20	56.65±1.30	58.84±2.38	5.67±2.17	34.82±0.82
	VecStat:KL	39.48±3.28	50.86±1.72	35.88±3.37	81.31±0.65	88.12±0.58	1.52±0.52	46.41±1.24
	VecStat:Cos	61.24±0.90	62.62±2.31	36.00±3.55	73.46±0.80	88.18±0.28	2.66±0.67	53.59±0.82
	LLM Call (0-shot)	82.37±0.30	58.67±0.67	96.14±0.41	35.30±0.88	99.26±0.07		
Llama-3.2-1B	Avg-MLP	57.99±6.53	75.13±5.42	86.22±4.37	75.94±4.00	78.43±0.09	41.67±5.38	76.02±1.42
	Tail-MLP	58.38±4.88	78.35±2.40	91.20±1.28	74.44±3.57	77.45±8.21	36.26±4.71	67.01±5.55
	Avg-Linear	53.70±6.28	70.94±5.13	81.60±9.17	74.15±2.30	80.41±7.23	44.36±5.95	76.96±7.70
	NormStat:KL	19.46±3.91	16.33±1.35	38.07±2.98	37.01±3.69	80.24±2.55	0.05±0.05	27.71±1.73
	VecStat:KL	34.57±0.92	50.34±1.92	38.19±4.30	76.13±0.97	84.89±0.59	1.38±0.12	48.17±2.66
	VecStat:Cos	44.71±0.78	59.48±2.26	44.63±5.73	72.65±0.35	83.96±1.19	2.30±0.32	52.57±0.92
Llama-3.2-1B-Instruct	Avg-MLP	64.88±4.97	78.95±1.01	86.76±4.30	76.82±5.70	85.39±0.43	45.27±2.58	84.35±2.00
	Tail-MLP	73.63±4.07	79.48±2.60	88.71±2.03	78.02±2.53	83.58±0.16	51.28±3.11	83.47±1.84
	Avg-Linear	57.23±5.70	77.83±1.72	83.36±2.16	82.14±0.69	80.57±1.53	48.75±2.43	82.05±0.96
	NormStat:KL	4.38±3.24	11.69±1.57	36.13±3.19	37.34±3.24	84.40±1.00	0.26±0.12	27.98±4.39
	VecStat:KL	40.03±2.62	45.09±1.50	35.88±3.46	74.32±1.49	88.67±0.43	1.25±0.14	46.75±2.34
	VecStat:Cos	53.20±1.76	59.85±1.80	36.79±4.02	72.72±1.04	86.59±0.19	1.78±0.09	52.17±1.54
Qwen3-4B	Avg-MLP	73.52±1.54	79.63±4.68	89.80±2.86	79.16±1.48	80.84±0.78	51.23±3.88	84.62±2.94
	Tail-MLP	68.98±9.23	82.47±1.96	82.70±2.37	67.84±7.26	81.55±3.51	46.89±4.30	79.95±1.50
	Avg-Linear	70.64±2.89	80.07±0.34	90.59±3.57	80.02±2.61	79.97±3.76	49.53±5.57	81.91±2.12
	NormStat:KL	19.57±3.97	17.15±0.91	42.26±3.41	47.00±2.72	76.79±2.63	0.60±0.25	34.28±1.70
	VecStat:KL	29.80±3.56	51.99±1.30	35.82±3.37	82.31±1.15	89.00±0.28	1.36±0.05	54.81±3.00
	VecStat:Cos	62.24±2.09	61.72±2.03	36.13±3.19	73.89±1.24	88.83±0.33	1.96±0.08	53.59±1.32
	LLM Call (0-shot)	6.05±0.17	78.88±0.81	59.14±1.17	60.21±0.32	67.16±1.78	21.39±0.77	80.69±0.20
Qwen3-8B	Avg-MLP	73.94±1.63	82.70±0.98	87.13±0.56	81.00±3.61	86.59±4.35	52.95±5.24	89.36±0.42
	Tail-MLP	68.43±1.81	80.52±2.03	89.92±1.91	73.87±6.12	82.92±5.55	53.58±0.64	81.10±5.72
	Avg-Linear	73.28±4.60	80.82±3.83	90.29±2.47	79.92±3.87	82.65±0.90	52.19±3.70	84.76±0.54
	NormStat:KL	21.01±6.35	19.03±1.50	40.19±3.56	46.38±0.91	73.45±4.22	0.73±0.59	31.98±1.31
	VecStat:KL	36.23±3.18	51.24±1.17	35.88±3.28	81.00±1.60	88.89±0.47	1.52±0.12	55.15±3.27
	VecStat:Cos	64.46±1.45	61.95±1.87	36.25±3.19	74.13±0.95	89.33±0.33	2.06±0.40	55.08±1.08
	LLM Call (0-shot)	44.49±0.22	83.22±1.58	78.32±2.46	67.76±0.44	52.38±2.21	18.70±1.02	64.63±1.63
Qwen3-32B	Avg-MLP	73.25±4.75	77.83±2.89	89.01±3.74	84.14±3.27	81.88±2.64	57.92±4.22	86.65±3.02
	Tail-MLP	66.70±8.95	78.43±2.02	88.52±4.65	79.61±3.33	74.06±11.10	50.57±10.68	80.62±3.30
	Avg-Linear	71.97±5.76	79.93±3.82	87.86±0.92	82.33±3.88	85.06±0.72	52.82±4.70	87.53±1.89
	NormStat:KL	31.99±2.02	21.72±0.85	40.56±3.48	44.61±1.14	75.53±2.56	0.13±0.05	31.44±1.47
	VecStat:KL	55.76±1.11	58.50±2.04	35.94±3.28	77.80±1.04	89.87±0.34	1.46±0.12	52.78±0.59
	VecStat:Cos	71.16±0.46	64.42±2.16	37.04±3.20	73.32±0.82	89.49±0.87	3.00±0.72	58.47±1.31
	LLM Call (0-shot)	69.28±0.36	86.59±1.44	91.92±0.42	35.81±1.07	59.61±1.31	20.04±0.51	63.28±0.12

bug-report framing. Across all levels, the underlying mathematical task is held fixed while misleading code-like cues are progressively injected, allowing us to test whether classifiers follow semantic intent or surface form. The resulting dataset is therefore intended as a controlled stress test rather than simulating typical human-authored adversarial prompts or real production failures. GPT5-Nano and GPT5 is used only to sanity-check the intended difficulty ordering, not to validate the realism of the generated prompts; accordingly, the results characterize robustness to this specific family of synthetic rephrasings.

The main observations from Section 3.4 hold consistently: performance degrades monotonically with difficulty, and training-free methods are substantially more robust than training-based ones for larger Qwen models. RoBERTa similarly drops to 10.60% on Easy and 5.60% on Medium, showing that the robustness gap is not specific to MLP heads on LLM features. We highlight several additional patterns.

Effect of post-training. Comparing Qwen3-1.7B against Qwen3-1.7B-Base reveals a striking degradation in the base model: almost all methods drop to near-zero even on the Easy tier, indicating that post-training alignment substantially improves adversarial robustness across both paradigms.

Consistency across Qwen scales. The robustness advantage of training-free methods over Avg-MLP holds across all Qwen model sizes. VecStat:Cos is the strongest training-free

Table 14: Robustness to adversarial attack: accuracy on adversarial MATH500 across all seven LLMs and three difficulty tiers (Easy, Medium, Hard).

Model	Method	Easy	Medium	Hard
Qwen3-1.7B	Avg-MLP	21.07±1.55	12.07±0.61	0.00±0.00
	Tail-MLP	89.13±13.28	54.87±33.39	0.00±0.00
	NormStat:KL	62.67±0.61	47.73±0.50	0.00±0.00
	VecStat:KL	64.93±0.46	34.87±0.76	0.00±0.00
	VecStat:Cos	81.27±0.50	63.60±0.20	0.20±0.00
Qwen3-1.7B-Base	Avg-MLP	2.60±2.60	0.87±1.17	0.00±0.00
	Tail-MLP	69.73±19.06	40.73±23.60	0.00±0.00
	NormStat:KL	10.33±0.23	1.20±0.00	0.00±0.00
	VecStat:KL	0.53±0.12	0.00±0.00	0.00±0.00
	VecStat:Cos	15.53±0.46	8.13±0.12	0.00±0.00
Qwen3-32B	Avg-MLP	11.73±8.92	0.40±0.69	0.00±0.00
	Tail-MLP	61.60±14.67	0.93±0.81	0.20±0.00
	NormStat:KL	92.33±0.12	80.67±0.12	0.00±0.00
	VecStat:KL	70.93±0.31	32.33±0.31	0.00±0.00
	VecStat:Cos	64.87±0.58	30.33±0.64	0.00±0.00
Qwen3-4B	Avg-MLP	18.47±3.19	10.27±2.34	0.00±0.00
	Tail-MLP	60.73±16.21	8.33±3.83	0.00±0.00
	NormStat:KL	31.87±0.31	28.33±0.42	0.00±0.00
	VecStat:KL	36.33±0.12	23.33±0.23	0.00±0.00
	VecStat:Cos	79.47±0.31	55.80±0.72	0.20±0.00
Qwen3-8B	Avg-MLP	12.67±7.82	4.20±3.30	0.00±0.00
	Tail-MLP	36.67±31.67	1.47±2.20	0.00±0.00
	NormStat:KL	36.13±0.50	27.00±0.40	0.00±0.00
	VecStat:KL	41.00±0.69	25.00±0.35	0.00±0.00
	VecStat:Cos	78.60±1.39	54.80±0.69	0.20±0.00
Llama-3.2-1B	Avg-MLP	27.07±8.03	14.80±4.16	0.47±0.42
	Tail-MLP	95.73±2.20	86.80±8.01	6.87±11.89
	NormStat:KL	87.60±0.00	59.20±0.35	5.33±0.64
	VecStat:KL	67.67±0.12	29.67±0.64	1.80±0.00
	VecStat:Cos	56.73±0.58	32.40±0.40	2.33±0.12
Llama-3.2-1B-Instruct	Avg-MLP	24.60±9.04	19.13±6.13	0.87±0.99
	Tail-MLP	72.20±8.72	31.27±8.88	0.20±0.20
	NormStat:KL	97.60±0.00	85.13±0.23	11.47±0.81
	VecStat:KL	85.73±0.12	60.00±0.53	3.40±0.40
	VecStat:Cos	77.33±0.12	56.40±0.35	4.13±0.12
RoBERTa	-	10.60±4.72	5.60±7.66	0.00

method for smaller and mid-size Qwen models, while NormStat:KL dominates for Qwen3-32B. This model-dependence of the best training-free variant motivates treating method selection as a deployment decision informed by the target model family and scale.

E.5 The effect of the number of layers considered

See Fig. 10.

Code+Math

```static func calcWidth(text: String) -> CGFloat {
    let constants = StoreSortBaseCell.Constants()
    let textWidth = HelperManager.textWidth(text, font: constants.titleLabelFont)
    return constants.horizontalSpace
        + textWidth
        + constants.spaceBetweenStarAndTitleLabel
        + constants.starImageViewSize
        + constants.spaceBetweenSeparatorAndImageView
        + constants.verticalSeparatorWidth
        + constants.spaceBetweenSeparatorAndImageView
        + constants.dropDownButtonWidth
        + constants.horizontalSpace
} ```

In the solution, the `calcWidth` function takes the input text and calculates the width based on the provided constants and helper function. It first initializes the constants and then uses the `textWidth` helper function to calculate the width of the input text. Finally, it computes the total width based on the constants and returns the result as a floating-point number.

-Jame will turn 27 in 5 years. In 8 years his cousin will be 5 years younger than twice his age. How many years separate the age of the two now?

Figure 8: Mixed-intent prompt example combining Swift code (width calculation function) and a mathematical problem used for uncertainty quantification analysis.

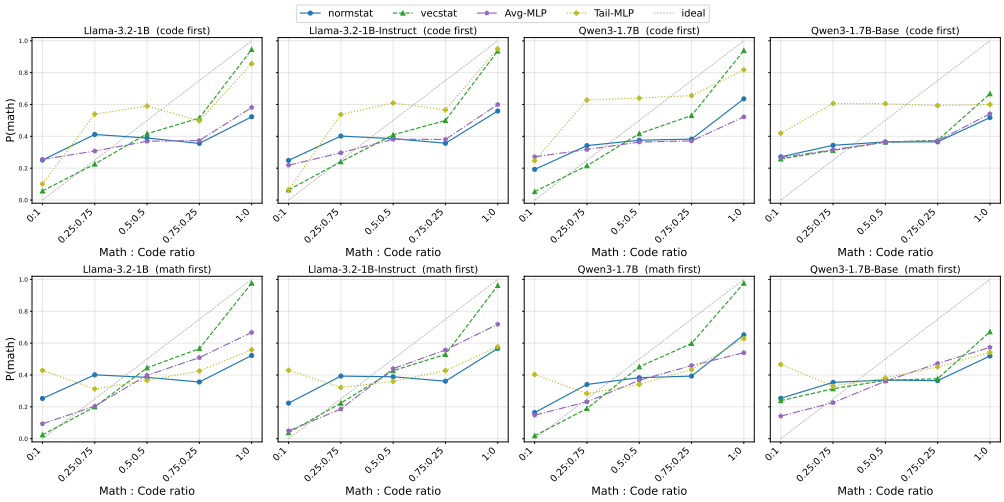



Figure 9: Predicted $P(\text{math})$ vs. mix ratio for all models and prompt orderings. Each curve is temperature-calibrated; the dashed diagonal is the ideal response. Upper and lower subplots show *code-first* and *math-first* prompt orderings, respectively.

### E.6 The effect of the maximum prompt length

See Fig. 11.

### E.7 Calibration Convergence Analysis

See Fig. 12.
```

Table 15: Calibration error of each method per model and prompt ordering. **Bold** marks the best method per row.

Model	Order	normstat	vecstat	Avg-MLP	Tail-MLP
Llama-3.2-1B	code-first	0.2535	**0.1286**	0.2384	0.2038
	math-first	0.2525	**0.1002**	0.1408	0.2787
Llama-3.2-1B-Instruct	code-first	0.2514	**0.1371**	0.2239	0.1819
	math-first	0.2419	0.1183	**0.1089**	0.2799
Qwen3-1.7B	code-first	0.2213	**0.1216**	0.2451	0.2412
	math-first	0.2098	**0.0858**	0.1764	0.2688
Qwen3-1.7B-Base	code-first	0.2493	**0.2396**	0.2431	0.2910
	math-first	0.2455	0.2335	**0.1710**	0.2861

Table 16: Adversarial prompt templates at three difficulty levels, each used with GPT-4o. A shared system prompt instructs the model to preserve all mathematical facts, numbers, equations, and constraints from the original, ensuring identical answers and identical reasoning; only framing, vocabulary, formatting, and surrounding scenario may change. Original mathematics problems are inserted at {`problem`}.

Easy	**Medium**	**Hard**
Increasing depth of disguise: lexical → structural → narrative		
Goal: Keep the original math problem mostly intact and add only a thin lexical code veneer.	*Goal:* Present the problem with a short untyped Python function header followed by a purely prose mathematical description.	*Goal:* Embed the problem in a realistic internal bug report for a misbehaving helper function.
• Preserve original sentence order and phrasing. • Name exactly one programming language. • Add a minimal wrapper (`solve()` or `compute()`). • Rephrase the final ask as "what value should the function return?" • Add only 2–5 code-signaling terms: *function*, *return*, *parameter*, *variable*, *input*. • No code block; no extra variables. • Length cap: $\leq 1.5\times$ original.	• Begin with one untyped `def` header; use plain parameter names matching the problem's given quantities; no type annotations; no return-type arrow. • Prose body after the header describes the problem in ordinary sentences, preserving every mathematical fact. • End by asking what value or expression the function should return. • No type annotations, backticks, code blocks, docstrings, function body, loops, conditionals, or data structures. • Length cap: $\leq 2\times$ original.	• Issue title referencing a function name. • 2–4 structured sections (e.g., CONTEXT, REPRO, OBSERVED, EXPECTED); CONTEXT must embed all mathematical facts sufficiently to solve the problem. • Example call with original problem values; output marked `?` (e.g., `f(a=3, b=4)` `?`). • End with a 1-sentence reviewer-style comment. • No function body, stack traces, logs, diff hunks, or file paths. • Length cap: $\leq 4\times$ original.

Shared system constraints (all levels): preserve every mathematical fact, symbol, equation, and relationship from the original; the rewritten problem must have the same answer and require the same mathematical reasoning; do not add programming-knowledge requirements, hints, edge cases, type checks, implementation details, or extra assumptions; output only the rewritten problem text.

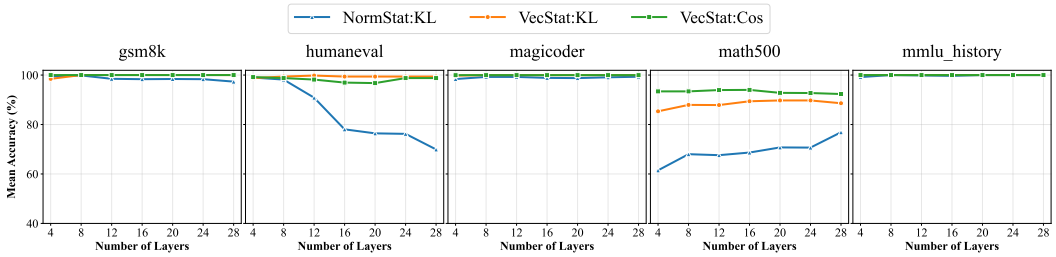


Figure 10: Effect of the number of layers on level-1 classification accuracy for Qwen3-1.7B.

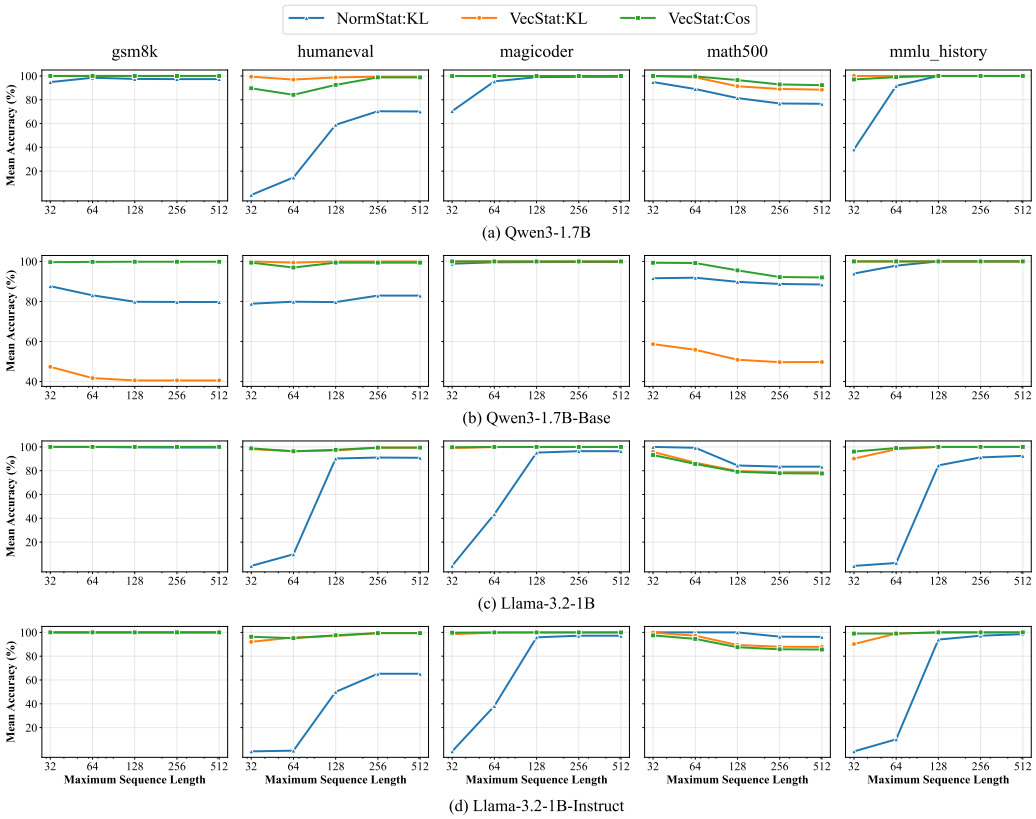
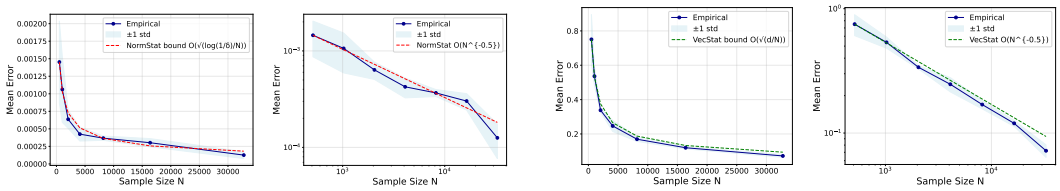


Figure 11: Effect of the maximum prompt length on level-1 classification accuracy for Qwen3-1.7B (a), Qwen3-1.7B-Base (b), Llama-3.2-1B (c), and Llama-3.2-1B-Instruct (d)..

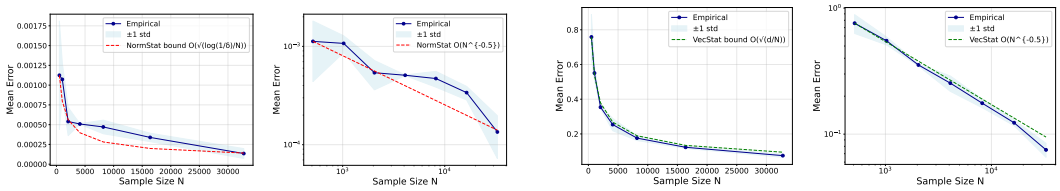
F More Related Works

LLM Routing Early work on LLM routing either ensembles outputs from multiple models (Jiang et al., 2023; Wang et al., 2023a) or uses cascades that query models sequentially by capability (Aggarwal et al., 2023; Chen et al., 2023; Yue et al., 2023), but both incur high latency and cost due to multiple calls per query. Subsequent approaches train learned routers–model-based predictors–that estimate per-query quality or cost and select the target LLM (Hari & Thomson, 2023; Stripelis et al., 2024; Feng et al., 2024; Dekoninck et al., 2024; Somerstep et al., 2025; Jitkrittum et al., 2025); these reduce unnecessary calls but introduce nontrivial training and maintenance overhead. Meanwhile, there are also training-free routers which choose among LLMs using lightweight ranking or budget-aware criteria (Zhao et al., 2024; Wu & Silwal, 2025). In contrast, our training-free statistical method is cheaper still because it operates entirely within a single LLM's prefill: we compute simple



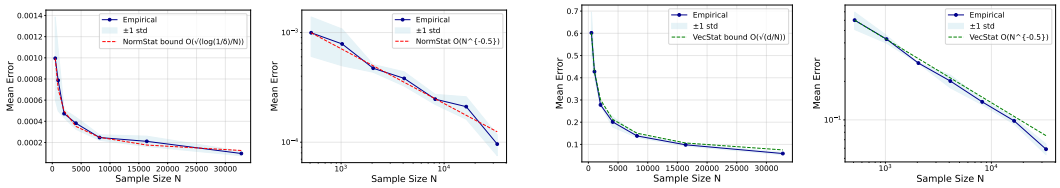
(a) Llama-3.2-1B-Instruct (NormStat)

(b) Llama-3.2-1B-Instruct (VecStat)



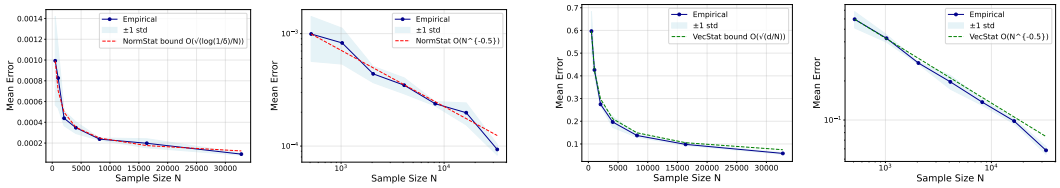
(c) Llama-3.2-1B (NormStat)

(d) Llama-3.2-1B (VecStat)



(e) Qwen3-1.7B-Base (NormStat)

(f) Qwen3-1.7B-Base (VecStat)



(g) Qwen3-1.7B (NormStat)

(h) Qwen3-1.7B (VecStat)

Figure 12: Calibration convergence analysis for different models and methods on the MagiCoder dataset. Each subplot shows both linear and log-log scales comparing empirical results with theoretical bounds. NormStat (norm method) uses dimension-free bounds while VecStat (projection method) uses dimension-dependent bounds.

statistics of internal activations to obtain fast, calibrated intent probabilities that serve as an efficient router without extra forward passes or router training.