EVALUATING THE DIVERSITY AND QUALITY OF LLM GENERATED CONTENT

Alexander Shypula[1]*, **Shuo Li**[1], **Botong Zhang**[1],
Vishakh Padmakumar[2], **Kayo Yin**[3], **Osbert Bastani**[1]

[1]University of Pennsylvania, [2]New York University, [3]UC Berkeley

ABSTRACT

Recent work suggests that preference-tuning techniques—including Reinforcement Learning from Human Preferences (RLHF) methods like PPO and GRPO, as well as alternatives like DPO—reduce diversity, creating a dilemma given that such models are widely deployed in applications requiring diverse outputs. To address this, we introduce a framework for measuring *effective semantic diversity*—diversity among outputs that meet quality thresholds—which better reflects the practical utility of large language models (LLMs). Using open-ended tasks that require no human intervention, we find counterintuitive results: although preference-tuned models—especially those trained via RL—exhibit reduced lexical and syntactic diversity, they produce greater effective semantic diversity than SFT or base models, not from increasing diversity among high-quality outputs, but from generating more high-quality outputs overall. We discover that preference tuning reduces syntactic diversity while preserving semantic diversity—revealing a distinction between diversity in form and diversity in content that traditional metrics often overlook. Our analysis further shows that smaller models are consistently more parameter-efficient at generating unique content within a fixed sampling budget, offering insights into the relationship between model scaling and diversity. These findings have important implications for applications that require diverse yet high-quality outputs, from creative assistance to synthetic data generation.

1 INTRODUCTION

As large language models (LLMs) increasingly serve as tools for ideation, synthetic data generation, and creative assistance, their ability to produce *diverse yet high-quality* outputs has become critically important. While recent advances have dramatically improved the performance of these models, relatively little attention has been paid to systematically measuring and optimizing for this dual objective of quality and diversity.

Consider a user asking an LLM to generate story premises or a researcher using an LLM to create synthetic training data. In these scenarios, the utility of the model depends not only on generating a coherent set of responses but on producing a set of outputs that cover a meaningful range of ideas or training examples. This stands in contrast to traditional LLM evaluation paradigms that optimize for a single correct answer.

The challenge, however, lies in defining what constitutes meaningful diversity. Diversity without quality can be trivial to achieve: random tokens are maximally diverse but utterly useless. What we seek instead is diversity among outputs that meet a threshold of quality or acceptability. We term this *effective semantic diversity* and argue that it represents a more accurate measure of an LLM's practical utility in open-ended generation tasks.

This distinction is particularly relevant in the current landscape of post-training LLMs, where preference-tuned models, through methods like Direct Preference Optimization (DPO), Proximal Policy Optimization (PPO), and Group Relative Policy Optimization (GRPO), are becoming standard and often yield high quality LLMs. While these methods have been successful in aligning models

*Corresponding author: shypula@seas.upenn.edu

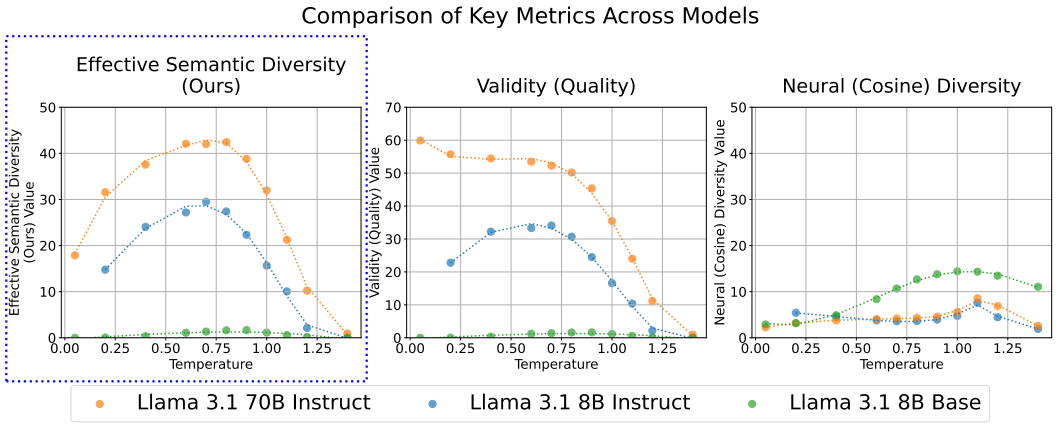

Figure 1: Diversity and Quality metrics when modulating the Temperature parameter across different models. We use the CODEBERTSCORE (Zhou et al., 2023) model for neural cosine diversity.

with human preferences, their impact on output diversity has been disputed. The dominant research narrative remains that preference tuning harms diversity. Moreover, a great deal of other open questions remain, such as the impact of model size on diversity and the ability to generate unique content.

In this paper, we introduce a principled framework for measuring this notion of high-quality diversity that:

- Requires no human evaluation at inference time
- Accounts for the fundamental quality-diversity interplay
- Allows for meaningful comparison across model families and training techniques

In Figure 1 we show our implementation of effective semantic diversity for three LLAMA-3.1 family models when modulating the temperature parameter. We find that on its own, a diversity metric like neural cosine diversity is highest for the base LLM despite an extremely low proportion of its generations being valid (quality). Our implementation of effective semantic diversity penalizes excessively-low and high temperature values as well as models that greatly struggle to generate coherent content.

Using our framework, our experiments and analysis reveal several counterintuitive findings. We find that preference-tuning reduces semantic diversity relative to supervised fine tuning (SFT): but only if we restrict ourselves to the subset of high-quality generations, which generally gets larger with preference-tuning. As a result, preference-tuned models produce greater effective diversity than their SFT and base counterparts, where the increase in the proportion of high-quality outputs more than compensates for the reduction of diversity within high-quality generations. Nevertheless, we also observe a pattern where preference tuning, and especially reinforcement learning (RL) algorithms, reduce syntactic and lexical structures while preserving semantic diversity, suggesting an important distinction between diversity of form and diversity of content that linguistics has long recognized. Furthermore, we also find a pattern where RL post-training is associated with significantly less lexical and syntactic diversity relative to DPO post-training. Interestingly, this pattern differs from the effect of increasing model size, where we find that increases in effective diversity are not associated with a reduction in lexical or syntactic diversity. Additionally, in an attempt to understand which models are most efficient per parameter in generating unique programs, for example in generating unique synthetic-data examples, we generally find that going smaller, to as small as 500 million parameters, is preferable.

These insights come at a critical moment as the world increasingly looks to LLMs to assist in open-ended tasks that span ideation, synthetic data creation, and creative assistance. Our insights deepen the understanding of how post-training strategies meaningfully affect diversity, and our framework can be used to evaluate how post-training strategies of the future meaningfully affect diversity.

2 Background and Related Work

LLM Alignment with Preference Tuning. As LMs (Bengio et al., 2000; Radford et al., 2019) have become more powerful, work has been done to improve their instruction following ability and to mitigate the likelihood of generating undersirable content. RLHF with PPO (Schulman et al., 2017; Ziegler et al., 2019; Ouyang et al., 2022) has emerged as a highly effective technique in aligning LLMs with human preferences. Subsequently, numerous alternative methods to PPO have been proposed, such as DPO (Rafailov et al., 2024), Rejection Sampling (Touvron et al., 2023), and GRPO (Shao et al., 2024). In numerous works, it has been reported that over-optimizing for the reward model eventually leads to incoherent or undesirable outputs (Ziegler et al., 2019; Stiennon et al., 2020; Korbak et al., 2022). It is suggested that an optimization strategy which places all probability on the highest reward outcome is optimal for this loss function, and will inevitably lead to distribution collapse (Lanchantin et al., 2025). In preference-tuning LLMs, the KL-divergence regularization has been instrumental to mitigating this effect as it allows the preference-tuned model to keep some of the distributional properties of the base LLM (Korbak et al., 2022). Therefore in-theory, KL-regularized RL should preserve some attributes of diversity present in base models.

Approaches to Measuring Diversity. Due to the cost of human evaluation, popular methods for automatically measuring diversity fall into lexical or neural approaches. In earlier days when LMs often could struggle to generate rich content, lexical metrics were often used to measure diversity. These generally involve calculating summary statistics using n-grams, such as **Distinct-N** (Li et al., 2016; Du & Black, 2019) and **Self-BLEU** (a modified BLEU metric) (Zhu et al., 2018) and are still popular today (Guo et al., 2024; Shaib et al., 2024). As early foundation models for language representation, such as BERT (Devlin et al., 2019) became popular and used to model sentence similarity (Zhang et al., 2019; Reimers & Gurevych, 2019) and code similarity (Feng et al., 2020; Zhou et al., 2023), they were eventually proposed to measure diversity in natural language (Lai et al., 2020; Tevet & Berant, 2021; Stasaski & Hearst, 2022), but have not been applied for programming language diversity.

While lexical and neural diversity metrics have been used for evaluating if one LLM is more diverse than another (Kirk et al., 2023; Guo et al., 2024), to our knowledge, no work has actually evaluated if neural diversity metrics reflect diversity of effective semantic content in LLM-generated outputs. When LMs may run the gamut from tiny models which may often produce incoherent content, to large and powerful models with highly sophisticated abilities, different safety attributes, and even variations in "style," it is unclear if existing diversity metrics are capable of robustly reflecting meaningful semantic content across such a wide range of distributions. Models used for evaluating diversity like Sentence-BERT (Reimers & Gurevych, 2019) are generally trained and evaluated on human text; however, for diversity they're expected to generalize to the most diverse sets of LLM-generated text which may be off-distribution. The closest attempt to evaluate lexical and neural diversity metrics is found in Tevet & Berant (2021) which evaluates if neural diversity metrics can reflect the temperature parameter used for sampling as well as diversity in human written content: for the latter it was found that neural models underperform human judgment.

Diversity of LLM Content. Due to the high cost of human evaluation, most insights into the nature of novelty and diversity of LLMs are built upon linguistic and neural measures of diversity. Zhang et al. (2020) introduce the notion of a quality/diversity tradeoff in natural language generation in evaluating sampling algorithms like nucleus sampling (Holtzman et al., 2019) and temperature sampling. McCoy et al. (2023) investigate whether smaller LMs exhibit linguistic novelty by evaluating combinations of n-grams not present in the training corpus. As expectations rose with the advent of more chat-based LLMs like ChatGPT (Ouyang et al., 2022; OpenAI et al., 2023), the ability of these models to serve as creative assistants and propose diverse responses has come into question. It has been noted that the first release of ChatGPT was incapable generating diverse jokes (Jentzsch & Kersting, 2023). In a more recent study, Kirk et al. (2023) fine-tuned 7B AlpacaFarm (Dubois et al., 2024) models with SFT and PPO for neural text summarization, and found PPO reduces the lexical and neural diversity of summarized content. Padmakumar & He (2023) show that human-written essays assisted by an RLHF-tuned model are less diverse than those assisted by a base model when assessed using neural diversity measures. Guo et al. (2024) benchmark the lexical, syntactic, and neural diversity of 7B parameter language models, but do not probe whether or not preference-tuned models are more / less diverse diverse than SFT or base model alternatives. A separate line of work has probed LLMs from the perspective of social cognition to understand if LLMs reflect the same opinions and

conceptual associations as macro-level populations of humans (Santurkar et al., 2023; Murthy et al., 2024). While these works have found that LLMs may not reflect the diversity of *entire populations* of people, it is highly unclear how this would impact the nature of effectively generating diverse content especially when placed under competing constraints of quality and helpfulness.

3 Measuring Effective Semantic Diversity

3.1 Problem Formulation

Let $\mathcal{D} = \{x_i\}_{i=1}^{N}$ denote a dataset of prompts, where each prompt x_i is designed to elicit a range of possible outputs from a language model. For each prompt x_i, we generate K outputs $\mathcal{P}_i = \{g_i^1, g_i^2, \ldots, g_i^K\} \sim f(\cdot \mid x_i)$, where f denotes the generative distribution of the language model under evaluation.

To evaluate the overall diversity of a model on the dataset $\mathcal{D}$, we compute the average diversity score:

$$\text{AvgDiv}_m = \frac{1}{N} \sum_{i=1}^{N} \text{Div}_m(\mathcal{P}_i), \tag{1}$$

where Div_m is a diversity metric and m denotes the specific type of metric being used. In our formulation below, Div_m refers to our proposed effective semantic diversity metric. However, Div_m could represent any diversity metric, including lexical metrics like Distinct n-grams and average cosine distance.

Validity function. To determine diversity at the individual prompt level, we define a binary validity function $V : \mathcal{G} \to \{0, 1\}$, where $\mathcal{G}$ is the space of generations:

$$V(g) = \begin{cases} 1 & \text{if } g \text{ is valid (i.e., meets quality threshold),} \\ 0 & \text{otherwise.} \end{cases}$$

Semantic funtion. Crucially, we determine the semantic uniqueness of generations based on their meaning rather than superficial textual differences. Two outputs may differ significantly in their lexical form yet convey identical semantic content—a distinction that purely lexical or syntactic metrics fail to capture. Our goal is to define a diversity metric over a set of generations that captures both this semantic distinctiveness and the validity of outputs.

We define a semantic function $S : \mathcal{G} \to \mathcal{S}$, where $\mathcal{S}$ is the semantic space. Two generations $g, g' \in \mathcal{G}$ are said to be **semantically equivalent** if and only if $S(g) = S(g')$. The detailed definition of S on each domain is described in Section 3.2.

We then formalize effective semantic diversity in two complementary ways:

Semantic diversity. We define the **effective semantic diversity** of a generation set $\mathcal{P}_i$ as the number of semantically unique valid generations, normalized by the total number of generations in the set:

$$\text{Div}_{\text{fixed}}(\mathcal{P}_i) = \frac{|\text{Set}(\{S(g_i^k) \mid g_i^k \in \mathcal{P}_i,\ V(g_i^k) = 1\})|}{K}, \tag{2}$$

where $\text{Set}(\cdot)$ denotes the set of unique elements and $|\cdot|$ denotes cardinality. This measures the proportion of generations that are both valid/high-quality and semantically unique within the group of generations.

However, we found that Equation (2) can be confounded by the number of samples under consideration if analyzing a variable-sized subset of valid-only programs; see Appendix A.2 for more detail. To address potential sample size effects, we adopt the pairwise diversity metric suggested in Tevet & Berant (2021):

$$\text{Div}_{\text{pair}}(\mathcal{P}_i) = \frac{1}{\binom{K}{2}} \sum_{\substack{g_i^j, g_i^k \in \mathcal{P}_i, \\ 1 \le j < k \le K}} d_{\text{sem}}(g_i^j, g_i^k), \tag{3}$$

Input Description:

- Multiple datasets.
- Each dataset consists of four real numbers: a, b, c, d.
- There are no more than 30 datasets.

Example Input:

```
35.68 139.77 51.15 359.82
01.37 103.92 41.78 272.25
51.15 359.82 -34.58 301.52
```

Function Signature:

Write a function `f(inputs)` that processes the list of tuples where each tuple contains four real numbers.

```
from typing import List, Tuple
def f(inputs: List[Tuple[float, float, float, float]]):
    '''
    inputs: a list of tuples, where each tuple contains four real
    ↪  numbers
    '''
```

Figure 2: An example of an open-ended problem description from our dataset.

where the semantic distance function $d_{\text{sem}} : \mathcal{G} \times \mathcal{G} \to \{0, 1\}$ is symmetric, i.e., $d_{\text{sem}}(g_i^j, g_i^k) = d_{\text{sem}}(g_i^k, g_i^j)$, and defined as:

$$d_{\text{sem}}(g_i^j, g_i^k) = \begin{cases} 0 & \text{if } V(g_i^j) = V(g_i^k) = 0, \\ 0 & \text{if } V(g_i^j) = V(g_i^k) = 1 \text{ and } S(g_i^j) = S(g_i^k), \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

This pairwise approach normalizes by the total number of possible pairs, making it robust to variations in the number of valid generations across different prompts.

3.2 Dataset, Validity, and Semantic Equivalence Checking

We chose to instantiate methods of validity and semantic equivalence checking using programs. Where possible, we found it preferable to avoid using LLMs to evaluate LLMs for diversity, especially if we seek to evaluate increasingly powerful LLMs with weaker LLMs. And furthermore, we do this because there has been no empirical work studying the robustness of this general approach for comparing LLM content. Research in programming language semantics has a rich history in the formalization of program correctness, termination, and semantic equivalence (Hoare, 1969; Plotkin, 1981; Floyd, 1993; Pierce, 2002). Using program execution on test cases, we can confirm two programs are semantically distinct if they compute different values over the same suite of input test cases[1]. Additionally, we can impose constraints of validity or quality that are easy to check: such as the program always returning a value without syntax or other runtime errors.

The key question, however, is to define open-ended programming tasks. Generating programs as an open-ended task is not necessarily radical: in chat applications, users may desire to brainstorm programs or code projects. Moreover, generating synthetic programming tasks is already an important part of improving foundation LLMs for code (Dubey et al., 2024; Shypula et al., 2024).

We constructed our dataset by adapting competitive programming-style problems into open-ended abstracted programming tasks. In Figure 2, we show an example problem description from our dataset. For each problem description in our dataset, we provide an "Input Description" in natural language specifying the input format, an "Example Input" demonstrating potential inputs the function would handle, and a "Function Signature" providing a concrete specification of the function name and typing hints for the inputs. We manually reviewed, and if necessary, edited all final descriptions,

[1] *N.B.* Test cases may fail to test edge cases where two programs may differ. However, instead of over-reporting correctness, this phenomenon would only *penalize* diversity when two programs are equivalent on most test cases and hypothetically differ on some edge-case, which we find tolerable.

removing any potential references to the original programming task, standardizing the function name to $f(...)$, and using highly generic argument names. We used competitive programming problems from CODENET (Puri et al., 2021) and accompanying test cases from ALPHACODE (Li et al., 2022) as a starting point for our dataset. In total we have 108 unique programming tasks / prompts that were adapted from CODENET into this open-ended format. Using each of these, we can sample an arbitrary number of generations to calculate $\text{Div}_m(\mathcal{P}_i)$. Further details about the test set, its creation, and our execution environment are provided in Appendix A.3.

4 EXPERIMENTAL SETUP

Diversity Metrics. We now formalize the diversity metric $\text{Div}_m(\mathcal{P}_i)$ with particular focus on our proposed notion of **effective semantic diversity.** As introduced in Section 3.1, this metric relies on the choice of valid function V and semantic function S.

Let $\mathcal{T} = \{t_1, \ldots, t_L\}$ be a fixed set of test inputs. For any generated program g, let $g(t_l)$ denotes its execution on test case $t_l \in \mathcal{T}$, and o_l denotes the corresponding output. We define the **validity function** $V : \mathcal{G} \to \{0, 1\}$ as follows:

$$V(g) = \begin{cases} 1 & \text{if } g(t_l) \text{ executes without error and produces non-null output } \forall l \in \{1, \ldots, L\}, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, a program g is considered valid if it runs without raising any errors (e.g., `SyntaxError`, `ValueError`) and produces non-null outputs for all test cases in $\mathcal{T}$.

Next, we define the **semantic function** $S : \mathcal{G} \to \mathcal{O}^L$, where $\mathcal{O}$ denotes the output space. The function maps a program g to its output trace on the test set:

$$S(g) = (g(t_1), g(t_2), \ldots, g(t_L)) .$$

Two programs g and g' are considered **semantically equivalent** if and only if their outputs are identical across all test cases:

$$S(g) = S(g') \iff g(t_l) = g'(t_l) \quad \forall l \in \{1, \ldots, L\}.$$

For the remaining diversity metrics, we adopt Equation (3) and specify the semantic distance function $d_{\text{sem}}(g, g')$ used. For **Lexical diversity,** we use Expectation-Adjusted Distinct n-grams (**EAD**) with n-grams of length n=4. For **syntactic diversity,** we adapt the Distinct-N metric to the Abstract Syntax Tree (**AST**) of each generated program: to isolate the syntactic structure of a program (e.g., for-loop instead of recursion) from superficial choices (e.g., variable names), we canonicalize all program identifiers. For two programs, we calculate the ratio of the number of unique subtrees of height H across both programs to the total number of subtrees of height H in both programs, where H=4. In order to measure **neural diversity,** we adapt existing methods of neural diversity metrics (Tevet & Berant, 2021) to our domain using CODEBERTSCORE (Zhou et al., 2023). We use CODEBERTSCORE because it closely resembles the models used in the NLP literature to evaluate diversity (Tevet & Berant, 2021) and has been evaluated as effective in the evaluation of neural codes. We also attempted to use CODELLAMA-7B-INSTRUCT embeddings and ICESCORE (Zhuo, 2024) to evaluate diversity and found CODELLAMA similar to CODEBERTSCORE and ICESCORE to correlate strongly with temperature, such as the lexical diversity metric. See the the additional results in Appendix A.1.

Models and Sampling. The main empirical questions we seek to answer in our work surround the effects of different post-training algorithms and model size on diversity (SFT, DPO, PPO, GRPO, etc). In order to isolate the effects of different post-training strategies, we utilize the TULU-2 (Ivison et al., 2023) and LLAMA-2 (Touvron et al., 2023) as well as the TULU-3 (Lambert et al., 2024) and LLAMA3.1 (Dubey et al., 2024) families of models. TULU-2 consists of post-trained LLAMA-2 models, and TULU-3 consists of post-trained LLAMA3.1 models. In Table 1, we provide a categorization of the post-training methods. Using these we can compare a SFT-tuned model to a DPO-tuned model from the same family. For each problem description x_i in our dataset, we generate $K = 32$ programs, yielding 3,456 total programs sampled for each experiment. In cases where models are post-trained with numerous algorithms, we often categorize the model with the most aggressive algorithm used, e.g. we would categorize a model adapted with DPO and PPO as PPO.

Model Family	Base	SFT	DPO	RL
LLaMA 2 7B	Llama-2-7b-hf	tulu-2-7b	tulu-2-dpo-7b	Llama-2-7b-chat-hf (PPO)
LLaMA 2 70B	Llama-2-70b-hf	tulu-2-70b	tulu-2-dpo-70b	Llama-2-70b-chat-hf (PPO)
LLaMA 3.1 8B	Llama-3.1-8B	Tulu-3-8B-SFT	Tulu-3-8B-DPO Llama-3.1-8B-Instruct	Tulu-3-8B (PPO) Tulu-3.1-8B (GRPO)
LLaMA 3.1 70B	Llama-3.1-70B	Tulu-3-70B-SFT	Tulu-3-70B-DPO Llama-3.1-70B-Instruct	Tulu-3-70B (PPO)

Table 1: **Model Post-Training Categorization.** We organize all models under consideration by their base model and post-training method. All models to the right of "Base" are post-trained with the specified method from the base model.

In addition to our questions regarding the effects of post-training and model size, we also investigate which sizes and classes of models are most *efficient* on a *per-parameter basis* in generating unique examples. This is an attempt to understand which types of models could be optimal for large-scale synthetic data generation. We relax the strict necessity to pair models by the same base model, and use models from QWEN2.5-CODER (Hui et al., 2024), QWEN2.5 (Yang et al., 2024), and as well as DEEPSEEK-R1-DISTILL (Guo et al., 2025) family. Our goal was to test a broader set of sizes as well as models tuned specifically for reasoning tasks (i.e. DEEPSEEK-R1).

Prompt selection. The choice of prompt can affect the nature of generations. Because of this, we created three separate prompt templates: a zero-shot prompt, a two-shot prompt, and a two-shot prompt with chain-of-thought reasoning. This design allows us to probe generation behavior across a variety of settings. The few-shot examples included in the prompts were simple, manually written examples shared across all problems in the dataset. We provide the examples in Appendix A.5.

Statistical tests. We aim to determine if a factor such as post-training algorithm or model size increases or decreases diversity compared to another. To rigorously summarize these results, we report the two-tailed Wilcoxon Signed-Rank Test (to measure significance) and Cohen's D (to measure effect size) for AvgDiv_m over the entire dataset for each statistic. A benefit of the non-parametric Wilcoxon statistical test is that we can make rigorous conclusions even if only limited samples are available. We always pair models from the same family and vary whether the model is larger

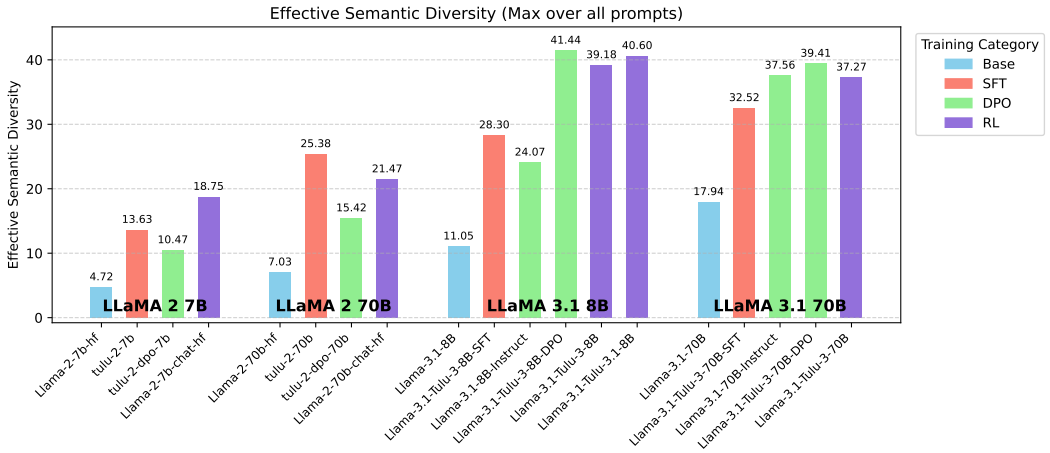

Figure 3: **Effective semantic diveresity scores for all 19 models evaluated in our experiments, grouped by model family.** Each bar is color-coded according to the post-training method, as categorized in Table 1

	Validity		Semantic		Syntactic		Lexical		Neural	
Comparison	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**
BASE VS. INST (ALL)	**<0.001**	**1.33**	**<0.001**	**1.34**	**<0.001**	**-1.10**	**0.028**	-0.47	**<0.001**	**-1.53**
BASE VS. INST-SFT	**<0.001**	**1.37**	**<0.001**	**1.36**	0.266	-0.34	0.339	0.21	0.339	-0.42
BASE VS. INST-DPO	**0.006**	**1.11**	**0.005**	**1.10**	**0.014**	-0.84	**<0.001**	**-1.68**	**<0.001**	**-2.37**
BASE VS. INST-RL	**<0.001**	**1.67**	**<0.001**	**1.54**	**0.012**	**-1.20**	**0.035**	-0.66	**<0.001**	**-2.12**
BASE VS. INST-PREF	**<0.001**	**1.34**	**<0.001**	**1.29**	**<0.001**	**-1.45**	**0.001**	-0.77	**<0.001**	**-2.29**
SFT VS. DPO	0.151	0.51	0.266	0.39	**<0.001**	**-0.90**	**0.003**	**-0.63**	**<0.001**	**-1.52**
SFT VS. RL	**0.004**	**3.19**	**0.004**	**2.49**	**0.004**	**-2.43**	**0.004**	**-1.20**	**0.004**	**-2.29**
SFT VS. PREF	**<0.001**	**0.97**	**0.002**	0.77	**<0.001**	**-1.30**	**<0.001**	**-0.83**	**<0.001**	**-1.86**
DPO VS. RL	1.0	0.10	0.301	-0.14	**0.039**	-0.60	**0.004**	**-0.78**	1.0	-0.01
SM. VS. LG.	**0.030**	0.20	**0.017**	0.26	0.170	0.15	0.485	0.09	0.269	0.12
SM. VS. LG. BASE	0.063	0.57	0.063	0.56	0.063	0.74	0.063	0.69	**0.031**	**0.80**
SM. VS. LG. INST	0.111	0.20	0.070	0.27	0.683	0.06	0.539	-0.11	0.759	-0.05
SM. VS. LG. PREF	0.330	0.13	0.277	0.20	0.454	0.10	0.330	-0.19	0.389	0.14

Table 2: **Model Comparison Results.** Results from Wilcoxon's Signed-Rank Test p-values: **W (p)**, and Effect Size measured by Cohen's D: **ES (d)**. Bold p-values are below 0.05, and bold d-values have an absolute value greater than 0.8 (large effect size). For effect sizes, positive values indicate the second model type in the comparison has higher values. "Inst" = Instruction-tuned (Post-Trained), "Pref" = Preference-tuned and combines RL and DPO models. The columns correspond to metrics in Section 4 (e.g. "Semantic" corresponds to effective semantic diversity).

	Semantic		Syntactic		Lexical		Neural	
Comparison	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**	**W (p)**	**ES (d)**
SFT VS. DPO	0.052	-0.71	0.791	-0.35	1.000	-0.12	**0.016**	-0.89
SFT VS. RL	**0.004**	**-3.12**	0.426	-0.28	0.820	0.08	**0.004**	**-2.25**
SFT VS. PREF	**<0.001**	**-1.31**	0.473	-0.32	0.946	-0.05	**<0.001**	**-1.39**
DPO VS. RL	0.301	-0.43	**0.012**	**-1.16**	**0.008**	**-1.24**	0.570	-0.05

Table 3: **Validity-Controlled Model Comparison Results.** Bold p-values are below 0.05, and bold d-values have an absolute value greater than 0.8 (large effect size)

fine-tuned with a different strategy *while fixing all other factors* unless otherwise noted. For example, when isolating model size, we would compare LLAMA3.1-8B-INSTRUCT with Zero-Shot prompting to LLAMA3.1-70B-INSTRUCT with Zero-Shot prompting, and so on. Because of the differences in post-training pipelines and to avoid confounding factors, we avoid pairing TULU post-trained models with LLAMA preference-tuned models.

5 EXPERIMENTAL RESULTS

5.1 EFFECT OF POST-TRAINING AND PREFERENCE-TUNING ON DIVERSITY

Preference tuning: more semantic diversity at the cost of less diversity in form. In Table 2, we summarize our results across all diversity metrics when comparing base models and their instruction-tuned counterparts, as well as comparisons between post-training techniques. Each row is a comparison between two different fine-tuning approaches; for each metric, the columns show the statistical significance **W (p)** and effect size **ES (d)**. A statistically significant, positive effect means that the second post-training strategy is greater than the first in terms of that metric.

We find a very clear trend: all post-training techniques increase both effective semantic diversity and validity over base models. We also find that RL in particular improves dramatically on SFT when it comes to effective semantic diversity. Additionally, we find an interesting pattern where preference-tuning broadly induces a dramatic decrease in the syntactic and lexical diversity of generations.

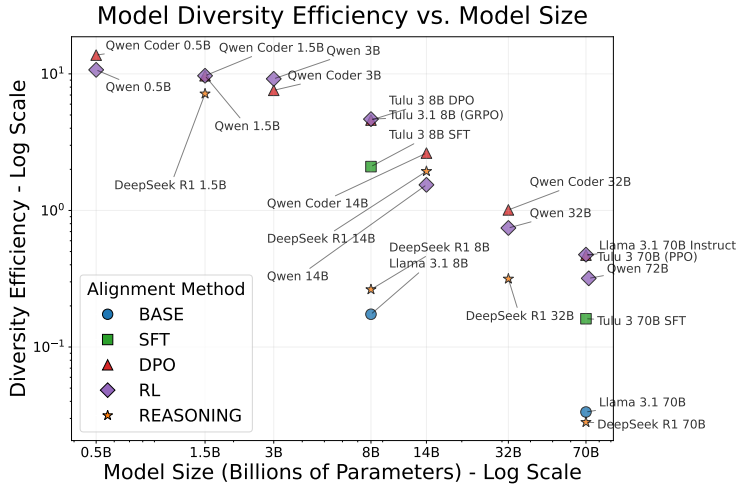

Figure 4: **Model efficiency**: We plot the parameter efficiency of a model in generating unique examples vs. model size (log-scale).

Semantic diversity driven by higher quality. In Table 3, we perform similar statistical tests as in Table 2, only with the subset of generations that were valid. We find that within this subset preference-tuned models generally have more semantic duplicates than their SFT-tuned counterparts. This suggests that while preference-tuning, especially RL, is liable to increase semantic duplicates in the subset of high quality generations, the volume of high quality generations more than compensates for this effect. Lastly, while we do not find a statistically-significant pattern between DPO and RL for effective semantic diversity, we find strong evidence and large effect sizes that RL-tuned models have less syntactic and lexical diversity than associated DPO-tuned models.

5.2 Effects of Model Size

Larger models increase semantic diversity without reducing the diversity of form. In Table 2, we show the results when comparing small and large models of the same family. We generally see higher semantic diversity in larger models without sacrificing lexical/syntactic diversity. We believe there is an intuitive justification: larger models have a higher capacity to model a greater amount of distributions in its corpus and their relative strengths in helpfulness and quality is well-documented.

Smaller models may be more compute-efficient to generate a unique example. While we find that larger models may be more diverse per-generation, we explore the question of which models are most efficient with regard to a computation budget. Consider a researcher trying to generate a large number of unique, synthetic programs: is it more advantageous to take fewer samples from a very-large LLM, many samples from a smaller LM, or is there a “sweet spot" in-between? In Figure 4 we explore this question by plotting the parameter-efficiency for diversity vs. the model’s size, where this efficiency is the effective semantic diversity using Equation (2) of a model normalized by the number of parameters of a model. For our fixed sampling budget of 32 generations, smaller models are consistently more efficient *on a parameter basis*. We believe that further work should be done to investigate if this pattern continues to hold as the number of samples increases to ensure semantically unique programs do not saturate faster for certain model classes than others. This is one of many potential directions for future work that could further leverage our framework.

6 Discussion and Conclusion

We have proposed a novel strategy for studying effective semantic diversity where we leverage code execution to measure and balance quality and diversity. Using this methodology, we perform extensive empirical analysis of LLM diversity. We find counterintuitive insights into the nature of how factors like post-training and model size meaningfully affect diversity. We encourage further work broadening empirical questions asked and exploring domains like natural language in the future.

REFERENCES

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 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. 4171–4186, 2019.

Wenchao Du and Alan W Black. Boosting dialog response generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

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 preprint arXiv:2407.21783*, 2024.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Robert W Floyd. Assigning meanings to programs. In *Program Verification: Fundamental Issues in Computer Science*, pp. 65–81. Springer, 1993.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yanzhu Guo, Guokan Shang, and Chloé Clavel. Benchmarking linguistic diversity of large language models. *arXiv preprint arXiv:2412.10271*, 2024.

Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A Smith, Iz Beltagy, et al. Camels in a changing climate: Enhancing lm adaptation with tulu 2. *arXiv preprint arXiv:2311.10702*, 2023.

Sophie Jentzsch and Kristian Kersting. Chatgpt is fun, but it is not funny! humor is still challenging large language models. *arXiv preprint arXiv:2306.04563*, 2023.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity. *arXiv preprint arXiv:2310.06452*, 2023.

Tomasz Korbak, Ethan Perez, and Christopher L Buckley. Rl with kl penalties is better viewed as bayesian inference. *arXiv preprint arXiv:2205.11275*, 2022.

Yi-An Lai, Xuan Zhu, Yi Zhang, and Mona Diab. Diversity, density, and homogeneity: Quantitative characteristic metrics for text collections. *arXiv preprint arXiv:2003.08529*, 2020.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Jack Lanchantin, Angelica Chen, Shehzaad Dhuliawala, Ping Yu, Jason Weston, Sainbayar Sukhbaatar, and Ilia Kulikov. Diverse preference optimization. *arXiv preprint arXiv:2501.18101*, 2025.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In Kevin Knight, Ani Nenkova, and Owen Rambow (eds.), *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 110–119, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1014. URL https://aclanthology.org/N16-1014.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

R Thomas McCoy, Paul Smolensky, Tal Linzen, Jianfeng Gao, and Asli Celikyilmaz. How much do language models copy from their training data? evaluating linguistic novelty in text generation using raven. *Transactions of the Association for Computational Linguistics*, 11:652–670, 2023.

Sonia K Murthy, Tomer Ullman, and Jennifer Hu. One fish, two fish, but not the whole sea: Alignment reduces language models' conceptual diversity. *arXiv preprint arXiv:2411.04427*, 2024.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Vishakh Padmakumar and He He. Does writing with language models reduce content diversity? *arXiv preprint arXiv:2309.05196*, 2023.

Benjamin C Pierce. *Types and programming languages*. MIT press, 2002.

Gordon D Plotkin. A structural approach to operational semantics. 1981.

Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Shibani Santurkar, Esin Durmus, Faisal Ladhak, Cinoo Lee, Percy Liang, and Tatsunori Hashimoto. Whose opinions do language models reflect? In *International Conference on Machine Learning*, pp. 29971–30004. PMLR, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Chantal Shaib, Joe Barrow, Jiuding Sun, Alexa F Siu, Byron C Wallace, and Ani Nenkova. Standardizing the measurement of text diversity: A tool and a comparative analysis of scores. *arXiv preprint arXiv:2403.00553*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Alexander G Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob R. Gardner, Yiming Yang, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ix7rLVHXyY.

Katherine Stasaski and Marti A Hearst. Semantic diversity in dialogue with natural language inference. *arXiv preprint arXiv:2205.01497*, 2022.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

Guy Tevet and Jonathan Berant. Evaluating the evaluation of diversity in natural language generation. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 326–346, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.25. URL https://aclanthology.org/2021.eacl-main.25.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan. Trading off diversity and quality in natural language generation. *arXiv preprint arXiv:2004.10450*, 2020.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. Codebertscore: Evaluating code generation with pretrained models of code. *arXiv preprint arXiv:2302.05527*, 2023.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pp. 1097–1100, 2018.

Terry Yue Zhuo. Ice-score: Instructing large language models to evaluate code. In *Findings of the Association for Computational Linguistics: EACL 2024*, pp. 2232–2242, 2024.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A APPENDIX

A.1 ADDITIONAL NEURAL DIVERSITY METRICS

In addition to using CODEBERTSCORE, we also demonstrate neural diversity with respect to temperature sweeps similar to those in Figure 1 using CODELLAMA-7B-INSTRUCT (Roziere et al., 2023) embeddings. We also use an LLM-as-a-judge to calculate pairwise similarity using ICESCORE (Zhuo, 2024) using `gpt-4o-mini` as the LLM-judge. We find that ICESCORE correlates heavily with temperature and does not seem to reflect quality. Similarly with CODELLAMA embeddings, the LLAMA8B-BASE model's generations are consistently considered more diverse than the INSTRUCT variants despite the base model generally producing far more low-quality and invalid outputs.

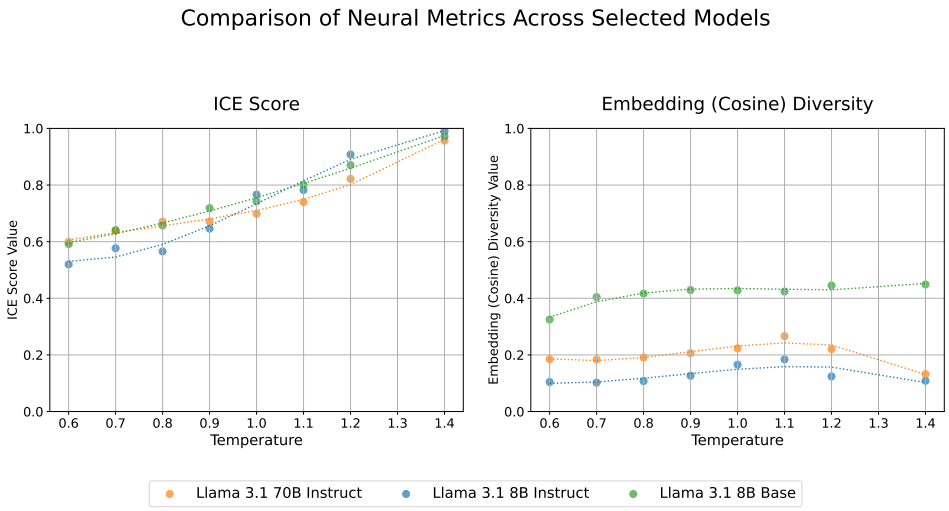

Figure 5: Neural diversity metrics when modulating the Temperature parameter across different models. We report neural diversity metrics using the ICESCORE (Zhuo, 2024) and cosine diversity of CODELLAMA-7B-INSTRUCT embeddings.

A.2 SAMPLE SIZE CONFOUNDING DIVERSITY AND ANALYSIS OF PAIRWISE DIVERSITY METRIC

In practice, we found that varying the size of the subset under consideration could dramatically affect the calculation of Equation (2): for the same model, as samples tended to get larger, this metric would decrease. When we use this metric for most of our experiments, it is sufficient, because we keep the number of samples constant. However, when we analyze the subset of valid-only programs, the size of this population may vary from model to model depending on the average quality of generations. Thus it is necessary to use Equation (3).

Analysis of Pairwise Diversity Metric. We provide an analysis of Equation (3). For a given large language model f, we assume that only a finite number K of distinct semantic meanings can be generated by f. We first establish that the original semantic diversity metric converges to zero as the number of sampled responses tends to infinity. Specifically, the original semantic diversity metric is defined as

$$\frac{N}{n},$$

where N is the number of distinct semantic clusters, and n is the number of sampled responses. Since only a finite number of semantic meanings can be generated by f, the number of semantic clusters N is bounded above, implying the existence of a constant $C_1 > 0$ such that $N \geq C_1$. Therefore, we have

$$\frac{N}{n} \geq \frac{C_1}{n} \quad \text{for all sufficiently large } n.$$

Now, observe that

$$\lim_{n\to\infty} \frac{C_1}{n} = 0,$$

so applying the squeeze theorem, we conclude that

$$\lim_{n\to\infty} \frac{N}{n} = 0.$$

Next, we show that the new metric defined in Equation (3), converges to a constant as $n \to \infty$. As before, we assume that there are K distinct semantic meanings in total, and let π_k denote the proportion of responses corresponding to the k-th semantic meaning. This implies that the number of times each semantic meaning is sampled is $\pi_k n$, where $\sum_{k=1}^{K} \pi_k = 1$. Thus, we have

$$\sum_{p_i,p_j\in P, i>j} m_{\text{dist}}(p_i, p_j) = \sum_{k\neq h} (\pi_k n) \cdot (\pi_h n) = \sum_{k\neq h} (\pi_k \pi_h) n^2,$$

where the summation is taken over distinct semantic meanings k and h, and $m_{\text{dist}}(p_i, p_j)$ measures the semantic distance between generations. Moreover, the number of possible response pairs is

$$\binom{|P|}{2} = \frac{n(n-1)}{2}.$$

Thus, the new metric becomes

$$\frac{1}{\binom{|P|}{2}} \sum_{p_i,p_j\in P, i>j} m_{\text{dist}}(p_i, p_j) = \frac{2\sum_{k\neq h}(\pi_k \pi_h) n^2}{n(n-1)} = 2\sum_{k\neq h}(\pi_k \pi_h) + \frac{2\sum_{k\neq h}(\pi_k \pi_h)}{n-1}.$$

Finally, we have

$$\lim_{n\to\infty} \left\{ 2\sum_{k\neq h}(\pi_k \pi_h) + \frac{2\sum_{k\neq h}(\pi_k \pi_h)}{n-1} \right\} = 2\sum_{k\neq h} \pi_k \pi_h.$$

That is, as $n \to \infty$, the value of the new metric converges to the constant value:

$$2\sum_{k\neq h} \pi_k \pi_h.$$

A.3 Dataset Creation and Additional Details

We created our dataset through a multi-step process starting from the CodeNet dataset and testcases from AlphaCode. The process involved problem standardization, language model assistance for scaling, and manual validation.

Initial Processing. We began by randomly selecting a single problem description from IBM CodeNet. We used this to as a seed to construct examples for a 1-shot prompt for a Language Model to assist us. Specifically we manually wrote one example of the following outputs that should be generated for each individual problem description from CodeNet

1. A canonicalized problem description
2. A wrapper function that would take any generation for that function, parse inputs for the function, and instrument the generation with the entire suite of test cases
3. A property-based testing function for generating additional test cases when needed

Dataset Expansion. We then wrote prompt templates for both `gpt-3.5-turbo-0125` and `gpt-4o-2024-11-20` (depending on the iteration). These templates were designed to take our 1-shot prompt, and then prompt the LLM to repeat the same for the new example we select from CodeNet. We randomly sampled over 300 programs from CodeNet and attempted to generate the three components for each original problem description from CodeNet. We then saved these into individual files, and also wrote them into an HTML document for manual review.

Manual Review and Selection. For the over 300 problem ids that were processed, we then manually inspected the output components checking for the following criteria:

1. The language model correctly formatted inputs into our desired format

2. The problem appeared novel relative to the existing problems we had already added to our dataset.

We tracked the original CodeNet problem IDs and validation results in a spreadsheet. These were the 108 examples then used for our dataset.

Manual Editing of Problem Descriptions. After selecting our problems, we then manually went through each of the three components, and manually edited them to be correct: a large amount required revisions, as the LLM assistant made mistakes. We saved our manually edited components to be further processed.

Test Case Integration. For each of the individual problem descriptions, we then merged test cases from CODENET and ALPHACODE. We required at least 10 test cases per problem. For the three problems that lacked sufficient test cases, we used our property-based testing scripts to generate 100 additional cases, such that we would have sufficient coverage. We then saved the canonicalized problem descriptions, the function to extract and parse input test cases, and the input test cases into a dataset.

Final Checks. During experimental validation, we found and fixed multiple faulty problem description argument parsing functions. We then saved these corrected version as our final dataset.

Additional Dataset Details. In our experiments, we instruct LLMs to return their outputs. We maintain a wrapper script that executes the function $f(...)$ for each input, obtains the returned result, and then serializes the object as well as its type (recursively) into a string. We then utilize these strings as the basis for our semantic comparisons. For each generation, we execute all test cases and capture the resulting outputs.

Because LLMs often generate natural language to accompany generated programs, extracting programs from the generations is a non-trivial task, especially for pre-trained models. We developed an extraction utility that extracts not only the target function $f(...)$, but also any helper functions and imports that may be relevant. To safely execute programs at scale, we perform all execution inside an isolated DOCKER container to prevent adverse consequences of blindly executing LLM outputs.

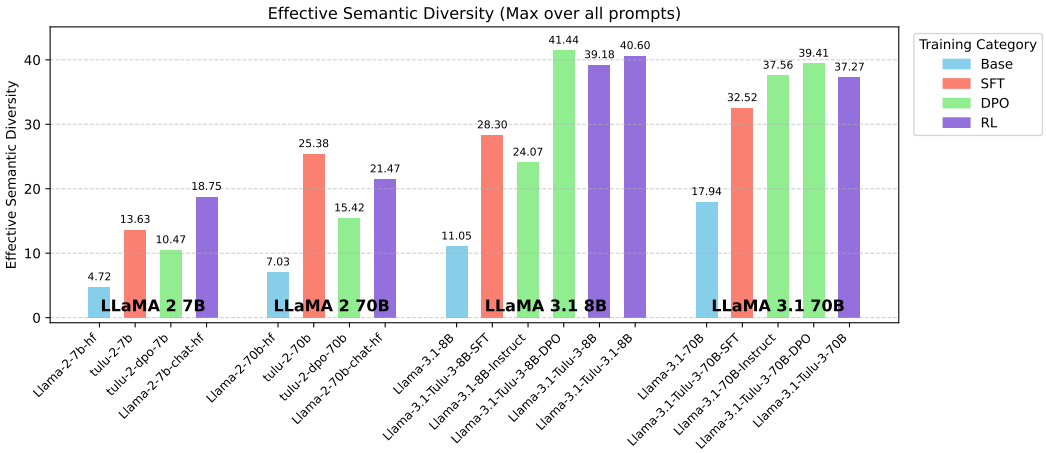

Figure 6: **Effective semantic diveresity scores for all 19 models evaluated in our experiments, grouped by model family.** Each bar is color-coded according to the post-training method, as categorized in Table 1

A.4 Additional Information the Syntactic Diversity Metric

We extract and process all Abstract Syntax Trees (ASTs) using Python's AST library with Python version 3.12.0, and report metrics for subtrees of height 4. Because syntactically incorrect programs do not parse, we can only calculate this metric over the subset of syntactically correct generations.

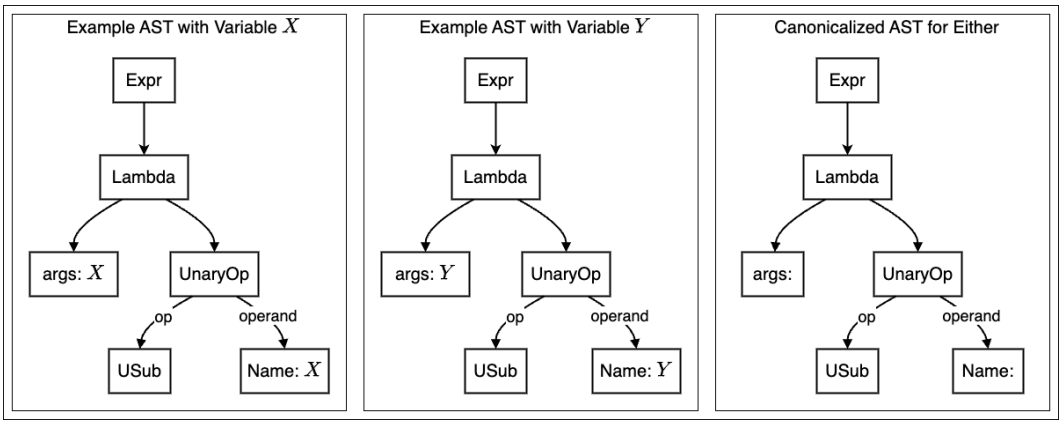

Figure 7: An Example of Canonicalizing an Abstract Syntax subtree used in the Distinct-CAST metric. The expression under consideration is for a simple lambda expression that negates a given variable. The first two ASTs are not equal because of the usage of the variables X and Y, respectively, even though they are alpha-equivalent expressions. The AST on the far right canonicalizes identifier names such as arguments and variables so that both expressions would be equivalent.

A.5 Prompts Used in Experiments

In Figure 8, Figure 9, and Figure 10, we show the prompting templates we use across all experiments.

```
{problem_description}

Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds.
```

Figure 8: **Zero-Shot Prompt**: Our template for our zero-shot prompt, where the problem description would be input inside the curly braces.

````
### Input Description:
1. An integer \( N \) (1
    ≤ \(N\) ≤ 10000), representing some quantity or size.
### Example Input:
```
1000
```
### Function Signature:
Write a function `f(N)` that takes in the input.
```python
def f(N: int):
    '''
    N: an integer
    '''
Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds.
def f(N: int):
    print(n**2)
### Input Description:
1. A floating point number \( N \) (1
    ≤ \(N\) ≤ 10000), representing some quantity or size.
### Example Input:
```
143.23
```
### Function Signature:
Write a function `f(N)` that takes in the input.
```python
def f(N: float):
    '''
    N: a float
    '''
Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds.
def f(N: float):
    i = 0
    while N > 1:
        N = N / 2
        i += 1
    print(i)
{problem_description}
Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds.
````

Figure 9: **Two-Shot Prompt**: Our template for our two-shot prompt, where the problem description would be input near the end inside the curly braces.

````
### Input Description:
1. An integer \( N \) (1 ≤ \(N\) ≤ 10000), representing some quantity or size.
### Example Input:
```
1000
```
### Function Signature:
Write a function `f(N)` that takes in the input.
```python
def f(N: int):
    '''
    N: an integer
    '''
Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds. First describe the
↪  function you would write, then implement it.
The following function will print out the square of the input
↪  number. We will take the square using the ** operator in
↪  Python within the print statement.
def f(N: int):
    print(n**2)
### Input Description:
1. A floating point number \( N \) (1 ≤ \(N\) ≤ 10000), representing some quantity or size.
### Example Input:
```
143.23
```
### Function Signature:
Write a function `f(N)` that takes in the input.
```python
def f(N: float):
    '''
    N: a float
    '''
Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds. First describe the
↪  function you would write, then implement it.
The following function will calculate the number of times the
↪  input number can be divided by 2 before it becomes less than
↪  1. We will increment a counter variable i each time we
↪  divide the number by 2 inside a while loop.
def f(N: float):
    i = 0
    while N > 1:
        N = N / 2
        i += 1
    print(i)
{problem_description}
Now please implement the function f; do not return anything, the
↪  function f should print the result of the operation.
It should terminate within 30 seconds. First describe the
↪  function you would write, then implement it.
````

Figure 10: **Two-Shot Chain-of-Thought Prompt**: Our template for our two-shot Chain-of-Thought prompt, where the problem description would be input near the end inside the curly braces.