Uncertainty Awareness of Large Language Models Under Code Distribution Shifts: A Benchmark Study

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have been widely applied in programming language analysis to enhance human productivity. Yet, their reliability can be compromised by various code distribution shifts, leading to inconsistent outputs. While probabilistic methods are known to mitigate such impact through uncertainty calibration and estimation, their efficacy in the language domain remains underexplored compared to their application in image-based tasks. In this work, we first introduce a large-scale benchmark dataset, incorporating three realistic patterns of code distribution shifts at varying intensities. Then we thoroughly investigate state-of-the-art probabilistic methods applied to LLMs using these shifted code snippets. We observe that these methods generally improve the uncertainty awareness of LLMs, with increased calibration quality and higher uncertainty estimation (UE) precision. However, our study also reveals varied performance dynamics across different criteria (e.g., calibration error vs misclassification detection) and the trade-off between efficacy and efficiency, highlighting necessary methodological selection tailored to specific contexts.

1 Introduction

Large language models (LLMs) have achieved impressive performance in code generation and analysis (Rozière et al., 2023). On the other side, current LLMs that are fine-tuned for specific tasks typically assume the test dataset is independently and identically distributed (i.i.d. or *in-distribution*) with the training dataset (Snoek et al., 2019). This makes the reliability of such models for generalization a challenge, as real-world scenarios often come with *distribution shifts* (also referred to as data drifts), such as codebase updates stemming from changes in library versions, or new developers' contribution, leading to discrepancies in test dataset distribution and consequently, a degradation in the quality of generated code by the models.

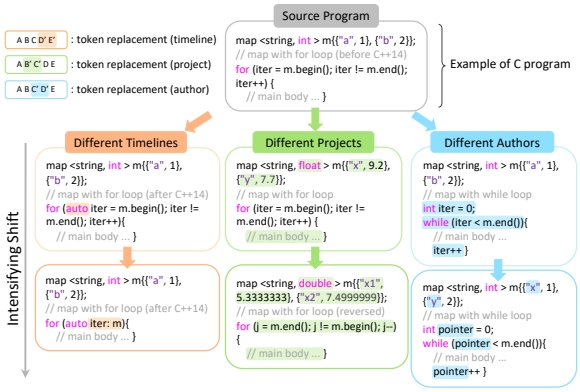


Figure 1: Three real-world code distribution shifts: TIMELINE SHIFT, PROJECT SHIFT, and AUTHOR SHIFT, on C++ snippets of *for* loops.

Addressing this challenge requires an understanding of the *uncertainty awareness* of LLMs. On the one hand, previous works show that deep models tend to be overconfident in their predictions despite low quality (Zablotskaia et al., 2023; Xu et al., 2022), particularly with shifted inputs. This issue, known as miscalibration, highlights the disparity between a model's predictive confidence and its actual accuracy. Uncertainty calibration, therefore, emerges as a critical solution to improve prediction quality under distribution shifts. Despite its significance, such a notion has not received much attention in the code generation literature. Uncertainty estimation (UE), however, is also vital even post-calibration, as mistakes are inevitable and it aids in assessing the reliability of model predictions, guiding decision-making on whether to accept or *abstain* from specific predictions (Vazhentsev et al., 2022). UE methodologies include detecting error-prone instances, i.e., misclassification detection, and identifying out-of-distribution (OOD) instances (van Amersfoort et al., 2020), etc.

To investigate the uncertainty awareness of LLMs in the context of code distribution shifts, we define three prevalent shift patterns that cover

most real-world evolution scenarios (Dilhara et al., 2023): code changes due to library or API updates across TIMELINE SHIFT, code changes resulting from PROJECT SHIFT that fulfill similar functions, code changes derived from AUTHOR SHIFT, as shown in Figure 1. Unlike previous works in the language domain that only explore shift patterns (Nie et al., 2022; Hu et al., 2023), we further introduce a series of fine-grained shift *intensities*. This allows for a more dynamic investigation of the performance of different methods under varying degrees of shifts, as underscored by Snoek et al. (2019). We create a benchmark dataset by extracting and synthesizing Java code snippets from open-source projects, aligning them with each identified shift pattern and respective intensities.

Leveraging this benchmark, we present the first comprehensive study[1] of relative effectiveness of cutting-edge probabilistic methods in improving LLM's prediction quality. Specifically, we examine both classic approaches such as Monte Carlo dropout (Gal and Ghahramani, 2016), deep ensemble (Lakshminarayanan et al., 2017), and more recent techniques such as adversarial mutation (Wang et al., 2019), dissector (Wang et al., 2020). Our findings reveal interesting dynamic patterns of efficacy. For instance, adversarial method, though excels in identifying out-of-distribution (OOD) examples, falls short in precisely predicting misclassifications; ensemble method is more robust against severe shifts compared to the post-hoc calibration (Guo et al., 2017). Additionally, we uncover a trade-off between calibration efficacy and efficiency for these methods, e.g., DE incurs a 50-fold increase in latency compared to the deterministic baseline, underscoring the importance of selecting appropriate methods under specific task requirements.

Our contributions are:

- We develop a large-scale dataset incorporating three realistic code distribution shifts with varying intensities and an OOD pattern.
- We adapt various probabilistic methods to the LLM setup and conduct an extensive benchmark study of their resultant uncertainty awareness in the distribution shift context.
- We demonstrate that while probabilistic methods generally mitigate the adverse effect of distributional shifts on LLM performance, their efficacy varies from specific task, model, and evaluation context, notably entailing a compromise in computation efficiency.

[1] Our code and data will be released after the review period.

2 Related Work

Code Distribution Shifts Recent studies have explored distribution shifts and evaluation methods to assess the reliability of deep models. For instance, semantic-preserving code transformations (Rabin et al., 2021) present various shifts, ranging from common refactoring like variable renaming to more intrusive operations such as loop exchange. Nie et al. (2022) evaluate the impacts of cross-project and time-segmented shifts on code summarization performance. Hu et al. (2023) assess the robustness of code analysis models under five shifts, including task, programmer, time-stamp, token, and concrete syntax tree. These evaluation strategies resemble our defined shifts, e.g., time-segmented and time-stamp function similarly to TIMELINE SHIFT. However, some of these shifts, such as concrete syntax tree, are synthetic and unrealistic compared to our shift patterns that accommodate real-world evolution scenarios. Also, our fine-grained shift intensities allow a more thorough understanding of the impacts of varying shift scales.

Uncertainty Calibration Uncertainty is a natural aspect of predictive models, and modeling it properly can be crucial for reliable decision-making. In recent years, research efforts have been directed towards quantifying uncertainty in deep learning (DL) models (Tran et al., 2022; Vazhentsev et al., 2022), which can be broadly classified into two categories: aleatoric and epistemic. Psaros et al. (2023) introduce several aleatoric uncertainty methods, such as Vanilla (Hendrycks and Gimpel, 2017) and Temperature Scaling (Guo et al., 2017). It also thoroughly explores epistemic uncertainty techniques, including both Bayesian Neural Networks (BNNs) like Laplace approximation (MacKay, 1992), variational inference (Blundell et al., 2015), dropout-based methods (Gal and Ghahramani, 2016; Kingma et al., 2015), and non-Bayesian methods such as ensembles (Lakshminarayanan et al., 2017), SWAG (Maddox et al., 2019), and SNGP (Liu et al., 2020). This research lays the theoretical ground for our choice of uncertainty methods. Hu et al. (2023) alternatively applies uncertainty techniques, such as ODIN (Liang et al., 2018) and Mahalanobis (Lee et al., 2018), to detect OOD examples. While it contributes significantly to the field, our study goes beyond identifying OOD datasets and offers a practical method for mitigating shift impacts via abstaining from low-quality predictions.

		Train / Dev	Shift1	Shift2	Shift3
TIMELINE	Snippet size	12.66 / 12.64	12.58	12.62	12.71
	Snippets	388,577 / 98,830	466,759	535,314	548,453
	Vocab	17,858	17,874	17,849	17,862
	KL↑	*0.0*	0.15	0.26	0.32
	Cosine↓	*0.0*	0.96	0.94	0.91
PROJECT	Snippet size	11.83 / 11.88	15.56	14.79	13.56
	Snippets	172,591 / 41,216	106,607	100,279	116,714
	Vocab	16,977	15,833	14,797	15,212
	KL↑	*0.0*	1.54	1.85	1.99
	Cosine↓	*0.0*	0.87	0.79	0.74
AUTHOR	Snippet size	16.08 / 15.28	15.85	14.61	15.32
	Snippets	197,096 / 42,569	118,987	86,013	84,250
	Vocab	16,118	16,180	16,834	16,566
	KL↑	*0.0*	0.12	0.35	0.66
	Cosine↓	*0.0*	0.91	0.87	0.81

Table 1: Statistics of the three datasets with intensifying shifts (shift1 → shift3): TIMELINE SHIFT, PROJECT SHIFT, and AUTHOR SHIFT. Snippet size is the average number of lines in each snippet.

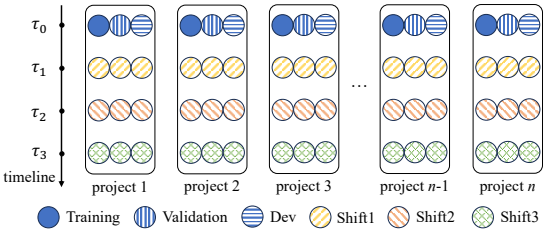


Figure 2: TIMELINE SHIFT where all the n projects are evaluated chronologically.

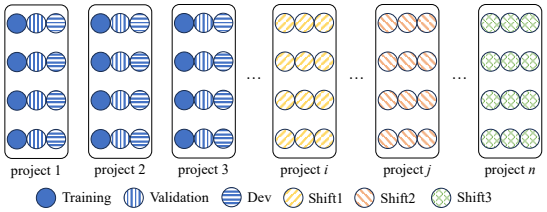


Figure 3: PROJECT SHIFT with *cross-project* splits.

3 Dataset Configuration

3.1 Motivation of Our Study

Previous works (Kojima et al., 2022) have demonstrated the impressive zero-shot capability of LLMs on reasoning tasks, yet we observe these models still fall short in complex tasks like code analysis (see results in Table 12), necessitating fine-tuning or further refinements for improved reliability. For example, consider a scenario where an LLM is fine-tuned on source-code of a project P. Over time, P undergoes various changes such as file modifications and version updates, transforming to a new version P'. Evaluating the model performance on P' is crucial to assess its reliability under distribution shifts across timelines (TIMELINE SHIFT), potentially saving training resources. Additionally, it's also crucial to determine if the model can be directly used for a different project with similar functionalities (PROJECT SHIFT). Furthermore, code contributions from new developers, who possess distinct coding styles, introduce variability in coding patterns (AUTHOR SHIFT). These three real-world, commonly occurring code shifts define the focus of this project.

3.2 Benchmark Code Datasets

To represent the three code shift patterns, we investigate seven open-source Java projects collected from the *Java-small* benchmark[2]: *elasticsearch*, *gradle*, *presto*, *wildfly*, *hadoop*, *hibernate-orm*, and *spring-framework*. They are language software for distributed computing and are studied by existing code analysis literature (Alon et al., 2019a,b). We extract Java files from each project as raw code snippets[3] and use Abstract Syntax Trees (ASTs) for tokenization. To enable a more fine-grained study, we create three levels of shift intensities (intensified from shift1 to shift3) for each shift pattern. The intensities are measured in two criteria: 1) Kullback–Leibler (KL) divergence between the token histogram distribution of each shifted set and dev set, and 2) cosine similarity between the code embeddings of each shifted set and dev set. The dataset statistics are shown in Table 1.

- **Timeline Shift** As shown in Figure 2, we consider four different release timelines, i.e., τ_0, τ_1, τ_2, τ_3, and collect Java files written by the same set of authors across times. Specifically, for each timeline, we check the commit histories of all projects and collect data from their corresponding release versions.
- **Project Shift** As shown in Figure 3, we calculate the KL divergence between each pair of the seven projects, and select the top-4 projects, namely *hibernate-orm*, *presto*, *spring-framework*, *wildfly*, whose token distributions have least average divergence with others, as the training (and dev) set. The remaining three sets with increasing KL scores, namely *hadoop*, *gradle*, and *elasticsearch* are treated as shift1, shift2, and shift3 datasets.
- **Author Shift** As shown in Figure 4, we split Java files into four groups. Although multiple pro-

[2] https://s3.amazonaws.com/code2seq/datasets/java-small.tar.gz

[3] We define a *code snippet* as a single, complete function extracted from Java source code. Each function, including its signature and body, is treated as an independent unit of code.

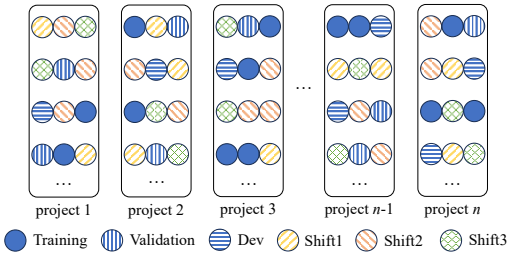


Figure 4: AUTHOR SHIFT with *cross-author* splits. Each color represents a unique author.

grammers may have contributed to these files, we organize them based on the primary contributor, creating a semblance of *author uniqueness* within each group. In this way, the AUTHOR SHIFT implicitly exists in our setup, as each group represents the work of a distinct set of programmers. Given the considerable size of the project and the presence of hundreds of contributors, we select four authors who have made the most significant number of commits throughout the project histories. The Java files committed by these authors are selected to form the datasets.

4 Experiments

4.1 Experiment Setup

Probabilistic methods have been applied to improve the reliability of LLMs. In this study, we focus on the state-of-the-art CodeLlama (Rozière et al., 2023), and defer results on other models, such as Code2Vec (Alon et al., 2019b), CodeBERT (Feng et al., 2020), CodeGPT (Lu et al., 2021), to Appendix C. Their implementation details are described in Appendix A.2. We evaluate the effect of following methods on mitigating distribution shifts, and detail their computation and theoretical analyses in Appendix B:

- **Vanilla Baseline** Our deterministic base model is CodeLlama-7B. The UE is measured as the maximum softmax probability (also referred to as winning score (WS) (Hendrycks and Gimpel, 2017)) across the softmax space.
- **Temperature Scaling (TS)** Guo et al. (2017) propose a post-hoc calibration that learns a scalar parameter $T_{\text{ts}} > 0$ based on the validation set $\mathbb{D}_{\text{val}}$ and align models' softmax probability more closely with the actual accuracy. TS "softens" the vanilla logit l^c with T_{ts} to obtain a new predictive distribution $p(y = c|x) = \frac{e^{l^c/T_{\text{ts}}}}{\sum_k e^{l^k/T_{\text{ts}}}}$. Since TS does not change the maximum of the softmax function, the class prediction remains *unchanged*. The UE is measured as the Shannon entropy of the new predictive distribution.
- **Monte-Carlo Dropout (MCD)** Gal and Ghahramani (2016) introduce a Bayesian method that estimates *inner-model* epistemic uncertainty using the Monte-Carlo average of T_{mcd} dropout samples, which are generated using the same architecture but with different random seeds at dropout layers. We define calibrated outputs as the average of the T_{mcd} dropout samples. Following Vazhentsev et al. (2022), we consider three UE techniques: sampled winning score (SWS), probability variance (PV) (Gal et al., 2017; Smith and Gal, 2018), and Bayesian active learning by disagreement (BALD) (Houlsby et al., 2011).
- **Deep Ensemble (DE)** Lakshminarayanan et al. (2017) measure the *cross-model* epistemic uncertainty by training T_{de} independent vanilla models and averages all. We use the same model architecture but different initial seeds. We quantify uncertainty via the same UE methods as MCD.
- **Mutation Testing (MT)** Wang et al. (2019) propose an adversarial method that measures the sensitivity of input to model mutation operations, effectively quantifying uncertainty regarding how close it is to the decision boundary. For a given model, MT first obtains a set of mutated models through mutation operators, such as Gaussian Fuzzing (GF), and Weight Shuffling (WS), then calibrates predictive distribution as the average softmax over all mutated models. The UE is defined as the label change rate (LCR).
- **Dissector (DS)** Wang et al. (2020) train a classification layer (snapshot) S_l after each intermediate layer l using linear regression and estimates uncertainty by assuming a correctly classified input should induce increasing confidence across the hidden layers. We compute the average softmax over the resultant snapshots as calibrated outputs. The UE measures how uniquely the final prediction is supported by each snapshot, i.e., snapshot-profile-validity (SPV), whose weight parameters are formulated in three growth patterns: linear, logarithmic (log), and exponential (exp).

To evaluate the performance of these methods, we consider two most-studied code analysis tasks (their preprocessing and evaluation details are described in Appendix A.1):

- **Code Completion (CC)** We consider token-level CC (Kim et al., 2021), which is analogous to language modeling of code generation. Specifically,

Pattern	Method	Code Completion						Code Summarization					
		Dev	Shift1	Shift2	Shift3	Avg↑	Avg Rank↓	Dev	Shift1	Shift2	Shift3	Avg↑	Avg Rank↓
TIMELINE	Base / TS	71.97	71.84	69.25	68.46	70.38	4.75	48.04	47.82	46.98	46.89	47.43	5.00
	MCD	72.94	72.68	71.38	70.40	**71.85**	**1.25**	48.83	48.57	47.87	47.38	48.16	2.50
	DE	72.60	71.74	71.30	70.64	71.57	2.25	49.58	49.54	48.66	48.63	**49.10**	**1.00**
	MT	72.02	71.84	69.28	68.47	70.40	3.5	48.20	47.95	47.14	47.24	47.63	4.00
	DS	72.92	71.31	69.50	67.51	70.31	3.25	48.30	48.42	48.27	47.51	48.13	2.50
PROJECT	Base / TS	68.05	65.76	65.46	64.76	66.01	4.75	53.64	52.86	48.63	46.84	50.49	4.50
	MCD	69.79	65.81	65.60	64.82	66.51	2.50	54.53	54.13	49.12	47.24	51.26	1.75
	DE	70.18	67.79	66.71	65.70	**67.60**	**1.00**	54.46	53.44	49.91	48.80	**51.65**	**1.50**
	MT	68.14	65.77	65.54	64.96	66.10	3.00	54.12	53.02	48.65	47.99	50.95	3.00
	DS	68.98	65.79	65.36	64.70	66.21	3.75	53.46	52.79	48.89	47.02	50.54	4.25
AUTHOR	Base / TS	73.69	72.97	72.77	71.43	72.72	4.75	50.94	49.32	48.13	44.50	48.22	4.00
	MCD	74.27	73.95	72.84	71.52	73.15	2.00	51.87	50.29	48.30	46.82	**49.32**	**2.25**
	DE	75.07	74.71	73.23	72.55	**73.89**	**1.00**	51.17	49.99	48.74	46.23	49.03	**2.25**
	MT	73.75	73.11	73.08	71.49	72.86	3.25	50.95	49.37	48.14	45.54	48.50	3.50
	DS	73.63	72.99	72.64	71.55	72.70	4.00	51.02	49.43	48.24	46.07	48.69	3.00

Table 2: F-1 scores and ranking of different methods for CodeLlama across in-distribution and shifted datasets. Results for other models are shown in Table 13. All methods consistently produce lower F-1 under intensifying shifts. Probabilistic methods (except TS) outperform the vanilla baseline.

we randomly mask a token in each code snippet as the missing part and feed the preceding context to LLMs. We evaluate the top-k sub-token F-1 of the predicted masked tokens.

- **Code Summarization (CS)** We focus on method name prediction (Alon et al., 2019a; Jain et al., 2021) which aims to describe names for code bodies (functions, classes, etc.). It is a challenging yet vital part of readable and maintainable code (Nie et al., 2022). We evaluate the top-k sub-token F-1 of the predicted names.

4.2 Uncertainty Calibration Quality

Prediction Performance We first compare the post-calibration prediction accuracy of various methods, by extending standard training, validation, and testing (dev) protocols to evaluations on shifted datasets. As shown in Table 2, we observe a correlation between performance degradation and shift intensity, e.g., TIMELINE SHIFT incurs a minor reduction in F-1 compared to the other shift patterns. Despite the decline of calibration quality due to intensifying shifts, probabilistic methods consistently mitigate this effect, as evidenced by improved F-1 relative to the baseline (and TS) across shifted datasets. Notably, DE and MCD achieve superior F-1 scores. Other probabilistic methods also show promising results: MT, whose combined predictions can be viewed as another ensemble technique, shows a lesser F-1 drop across shifted datasets. Interestingly, AUTHOR SHIFT has less impact on CC than CS. This suggests different programmers' design and implementation logic plays little role when decision-making largely relies on *code syntax*, such as predicting the next token based on common programming language grammars and rules. Additionally, we observe a more apparent calibration effect on smaller models (see Table 13), e.g., DE achieves an over 10% F-1 gain over the baseline for Code2Vec in CS, highlighting a challenge in calibrating LLMs.

Expected Calibration Error (ECE)↓ We measure the difference in expectation between confidence and accuracy using the ECE metric (Naeini et al., 2015) to evaluate how well the estimated model probabilities have been calibrated:

$$\text{ECE} = \sum_{k=1}^{K} \frac{|B_k|}{n} |\text{conf}(B_k) - \text{acc}(B_k)| \quad (1)$$

Here we group n sample predictions into K interval bins and define B_k as the set of indices of examples whose prediction confidence lies in the k^{th} bin $B_k = (\frac{k-1}{K}, \frac{k}{K}]$, where its accuracy and confidence are defined as $\text{acc}(B_k) = \frac{1}{|B_k|} \sum_{i \in B_k} \mathbf{1}(\hat{y}_i = y_i)$ and $\text{conf}(B_k) = \frac{1}{|B_k|} \sum_{i \in B_k} \hat{p}_i$. In code analysis, $\hat{y}_i$ is the predicted token, and $\hat{p}_i$ is the corresponding probability. As shown in Figure 5, probabilistic methods consistently outperform the deterministic baseline. Among them, TS, MCD, and MT better diminish the ECE, suggesting they facilitate maintaining competitive model prediction quality under distribution shifts. DE exhibits slightly improved ECE quality over the baseline, while DS falls short in reducing the calibration errors.

Rank Correlation with Quality Score↑ We evaluate how Spearman's rank correlation between calibrated probabilities and accuracy changes with

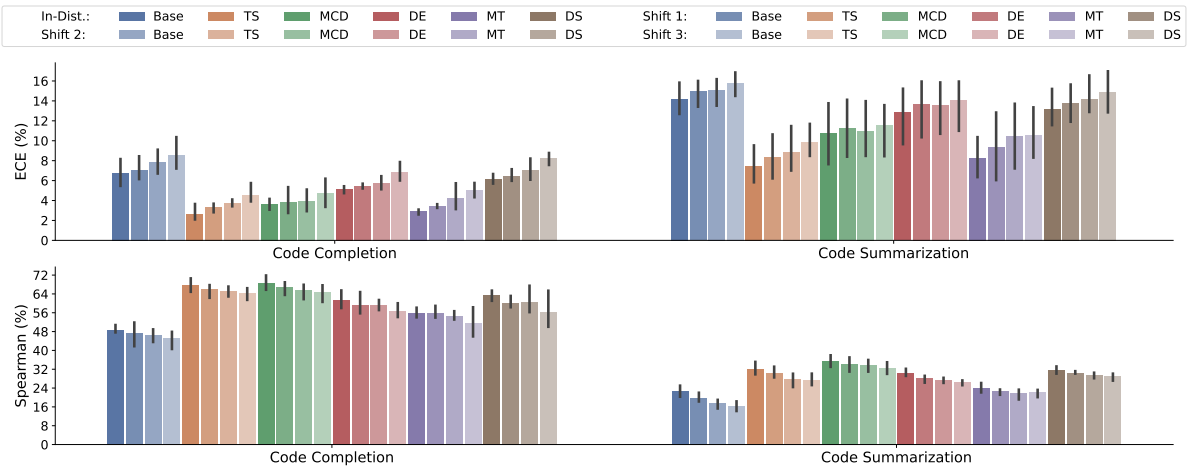


Figure 5: ECE↓ and Spearman's Rank Correlation↑ of different methods for CodeLlama. Results for other models are shown in Figure 13. Error bars indicate variations across the three shift patterns.

intensifying shifts. As shown in Figure 5, we observe a general trend demonstrating the calibration of probabilistic methods increases the correlation under distribution shifts. MCD and TS exhibit superior calibration capacity with higher correlation scores, while MT demonstrates limitations in improving correlation over the baseline.

4.3 Uncertainty Estimation (UE) Precision

To comprehensively assess the quality of UE methods, We consider three widely-applied metrics:

- **Area Under the ROC curve (AUC)**↑ Hendrycks and Gimpel (2017) suggest evaluating the quality of UE using AUC. It is interpreted as the probability that a misclassified example has a greater uncertainty score than a correctly classified one.
- **Area Under the Precision-Recall curve (AUPR)**↑ To better handle the situation when the positive class and negative class have greatly differing base rates, AUPR is also suggested to evaluate the quality of UE. The PR curve plots the relationship between precision and recall.
- **Brier**↓ Brier (1950) measure the mean squared error of the uncertainty scores assigned to each sample and the actual outcome.

Misclassification Detection In this experiment, we evaluate how the uncertainty scores $u \in [0, 1]$ of different UE techniques correlate with the mistakes $\bar{e} = \mathbf{1}(y_i \neq \hat{y}_i)$ of LLMs under distribution shifts. An effective UE method should produce higher scores for mistakes. Figure 6 presents the misclassification detection performance of different UE methods across all three shift patterns. We observe a general trend of decreasing UE quality as shift intensity grows, as evidenced by lower AUC and Brier scores for all methods in the shift3 set. Probabilistic methods generally outperform the baseline, cautioning the significance of calibration. Among them, DS achieves the highest AUC and rather low Brier scores under distribution shifts, suggesting that cross-layer consistency largely determines the reliability of model predictions. Ensemble method (DE) not only shows promising results but also shows relative robustness against intensifying shifts by reducing the model variance. Whereas adversarial method (MT) produces even lower quality than the baseline in misclassification detection. This indicates that the adversarial assumption may not necessarily apply to model mistake detection and cause false alarms. Furthermore, MT is more sensitive to different shift patterns, as evidenced by larger error bars in all three metrics.

Selective Prediction Selective prediction refers to selectively predicting high-quality outputs while rejecting the low-quality outputs (Ren et al., 2023). It measures how effective and efficient a UE method can improve the performance by allowing a model to abstain from predicting highly uncertain instances, assuming an effective method indicates high uncertainty for low-quality outputs. As shown in Figure 7, at a given abstention rate τ, we filter out the lowest τ-fraction uncertain examples based on the UE scores and compute the average F-1 score of the remaining examples. We observe that all probabilistic methods improve the quality of CodeLlama to a large extent, especially in the shift3 set. Similar to misclassification detection, DE and DS are more efficient: they achieve higher F-1 by rejecting the same proportion of code examples. MT slightly improves over the baseline, while

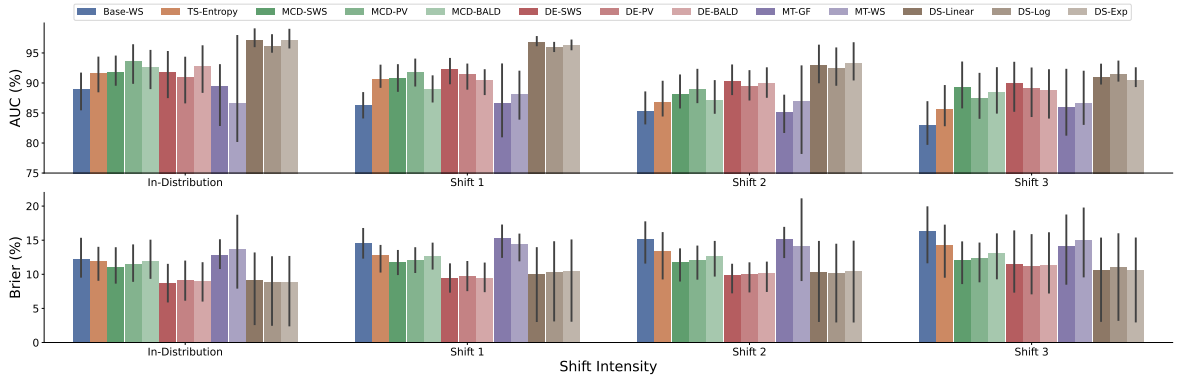


Figure 6: AUC↑ and Brier↓ results for detecting CodeLlama mistakes using different methods with corresponding UE techniques in CC under intensifying shifts. Error bars indicate variations across the three shift patterns. Results in CS and AUPR↑ results in both tasks are demonstrated in Figure 8 and Figure 9.

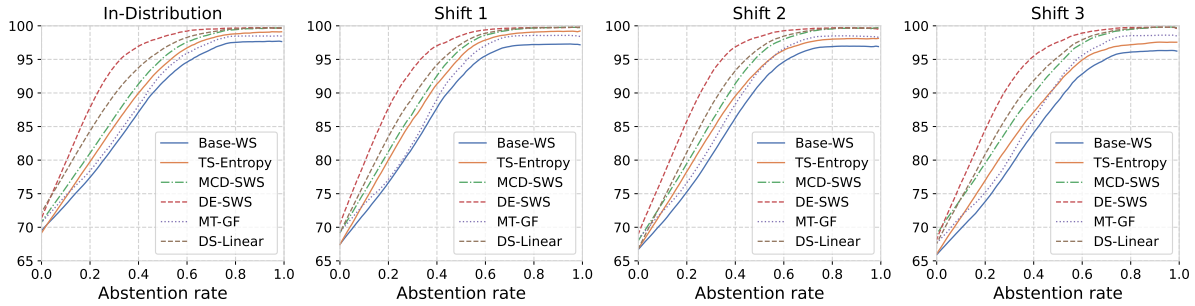


Figure 7: F-1 vs. Abstention curve for different uncertainty methods in CC under distribution shifts. For each method, we report one corresponding UE result. Each line is the average F-1 score over the three shift patterns. Results in CS and remaining UE results are shown in Figure 10 and Figure 11.

TS is prone to degradation under larger shifts, as evidenced by the smaller highest F-1 scores.

OOD Detection Besides the three shift patterns, we further evaluate the UE quality under a more severe shift: projects that are designed for *different programming paradigms* (PARADIGM SHIFT). We consider two real-world projects, BigCloneEval[4] (in-distribution) and Ruoyiplus[5] (OOD). The former is a Java language tool for analyzing code clone detection, while the latter is a J2EE development system for backend management. We extract Java files from both projects as raw snippets. Statistics of the processed dataset are detailed in Appendix A.3. Table 3 summarizes the OOD detection precision for CodeLlama of various UE methods. Interestingly, MT consistently outperforms others in detecting OOD code examples, with higher AUC, AUPR, and lower Brier scores. This suggests its adversarial assumption effectively applies to severely shifted inputs when considering their distance to the model's decision boundary. DS also achieves top-rank results, showing the sensitivity of snapshot validity to distribution shifts.

Method	UE	Code Completion				Code Summarization			
		AUC↑	AUPR↑	Brier↓	Rank↓	AUC↑	AUPR↑	Brier↓	Rank↓
Base	WS	69.99	78.18	23.78	8.67	59.63	69.47	22.67	10.67
TS	Entropy	70.68	78.56	25.31	9.00	60.18	70.30	25.84	11.67
MCD	SWS	69.25	81.71	20.34	4.67	64.84	72.80	24.27	9.00
MCD	PV	73.12	77.08	31.96	10.67	65.06	74.25	22.13	6.67
MCD	BALD	72.62	77.34	30.76	10.33	67.79	71.81	24.03	7.67
DE	SWS	66.53	79.94	24.34	8.33	63.97	78.94	25.78	9.00
DE	PV	63.78	79.41	25.83	10.67	62.45	80.81	28.01	9.00
DE	BALD	64.29	79.55	25.36	9.67	61.92	79.30	26.33	9.67
MT	GF	77.24	80.69	19.28	2.67	75.38	83.42	17.81	**1.33**
MT	WS	78.25	81.33	17.50	**1.33**	73.80	85.65	19.49	1.67
DS	Linear	73.63	79.41	20.44	6.00	68.62	79.27	21.35	4.33
DS	Log	73.79	80.83	23.96	4.67	66.73	80.39	22.73	6.00
DS	Exp	74.89	80.26	21.37	4.33	67.46	81.08	22.16	4.33

Table 3: AUC↑, AUPR↑, Brier↓ and average rankings of different UE methods for OOD detection.

5 Further Analysis

5.1 Calibration Quality in OOD Samples

We further evaluate the calibration quality of probabilistic methods in handling OOD examples, as shown in Table 4. We observe a clear drop in quality, e.g., lower F-1 scores, of all methods, compared to the results from shifted datasets in Table 2. This demonstrates the challenge of severe shifts for uncertainty calibration. Interestingly, TS, despite

[4] https://github.com/jeffsvajlenko/BigCloneEval
[5] https://github.com/kongshanxuelin/ruoyiplus

Method	Code Completion				Code Summarization			
	F-1↑	ECE↓	Corr↑	Rank↓	F-1↑	ECE↓	Corr↑	Rank↓
Base	55.48	17.95	55.68	4.83	29.99	34.11	14.54	5.83
TS	55.48	18.73	58.33	4.33	29.99	29.67	21.88	5.16
MCD	57.18	17.95	58.96	2.16	33.32	24.83	30.72	2.00
DE	58.79	16.32	58.33	**1.83**	35.15	12.69	33.38	**1.00**
MT	56.86	14.29	55.68	3.50	30.12	25.48	22.66	3.67
DS	57.04	23.11	56.04	4.33	30.19	28.47	27.40	3.33

Table 4: Calibration performance of different methods for CodeLlama in handling OOD code examples.

showing a top calibration performance in milder shifts, does not maintain such an edge in this severe case. This suggests that post-hoc calibration using the validation set can hardly be effective on severely shifted instances. Ensemble and Bayesian methods, however, emerge as the most reliable techniques in highly uncertain environments.

5.2 Empirical Findings

Method	Calibration			UE			Score↑
	F-1	ECE	Corr	MD	SP	OOD	
Base	✗	✗	✗	✗	✗	✗	*0.0*
TS	✗	✓✓	✓✓	✓	✓	✗	6
MCD	✓✓	✓✓	✓✓	✓	✓	✗	**8**
DE	✓✓	✓	✓	✓✓	✓✓	✗	**8**
MT	✓	✓✓	✗	✗	✓	✓✓	6
DS	✓	✗	✓	✓✓	✓✓	✓	7

Table 5: Overall comparison of studied methods. MD, SP are abbreviations of misclassification detection and selective prediction. ✓✓ denotes top-ranked, ✓ presents above-baseline, and ✗ means equal/below-baseline.

Table 5 summarizes the overall performance of studied methods concerning their calibration ability and UE quality. 1) For calibration ability, we observe that MCD achieves the best performance with top-ranked F-1 and calibration quality. Other methods also show promising results: DE is better at predicting F-1, while MT and TS exhibit better calibration quality. DS, in contrast, falls short in calibrating LLMs under distribution shifts. 2) For UE quality, however, we see DS produces the overall best results, with top-ranked low-quality output detection precision (i.e., detecting mistakes and abstaining) and distribution sensitivity (i.e., detecting OOD samples). Other probabilistic methods also improve UE quality compared to the baseline: DE is superior in misclassification detection and selective prediction, while MT is more sensitive to distribution shifts in OOD detection. This performance gap between different criteria suggests the hypotheses of certain methods fit in only specific scenarios with loss of generalizability.

Method	Calibration	UE	Total↓ (Relative)	Snippet↓
Base	*0.0*	65	65 (×1)	**0.27**
TS	69	75	144 (×2)	0.61
MCD*	5973	157.83	6130.83 (×94)	25.85
DE*	3162	87.39	3249.39 (×50)	13.50
MT*	7917	132.26	8049.94 (×124)	33.94
DS*	135	71.39	206.39 (×3)	0.87

Table 6: Overhead (s) for different methods. Per snippet ($\times 10^{-3}$s) is the instance-level overhead. For method* containing multiple UE results, we report the average number as they cost similar overhead.

5.3 Efficiency Analysis

Table 6 compares the overhead of different methods, where the calibration overhead of DE is the averaged per epoch overhead for fine-tuning T_{de} independent CodeLlama models. The hardware setting is detailed in Appendix A.4. We observe a general trade-off between efficacy and efficiency: methods requiring sophisticated calibration such as MCD, DE, generally produce promising results but cost larger overhead. Their practical overhead may even inflate: the complexity of DS is proportional to the number of hidden layers used for training snapshots. In time-sensitive tasks such as online code generation, TS or vanilla baseline may be preferred for selective prediction, especially when deploying computation-intensive models.

6 Conclusion

LLMs are prone to performance degradation under various code distribution shifts. This study identifies three real-world code distribution shift patterns that adversely affect LLMs' prediction quality. We investigate five cutting-edge probabilistic methods, focusing on both their calibration capabilities and uncertainty estimation (UE) efficacy in these shifting contexts. Our findings reveal that these methods generally alleviate the adverse effect of distribution variations, leading to improved accuracy and low-quality output detection precision. Our analyses also uncover performance variation across different evaluation criteria and various shift intensities, due to limited application scenarios of method hypotheses. For instance, adversarial assumption (MT) aligns well with severe OOD instances but little with misclassified ones; ensemble method is more robust against severe shifts compared to post-hoc calibration. We further highlight a general trade-off between efficacy and efficiency, cautioning the importance of choosing appropriate methods under specific circumstances.

Limitations

This study investigates the uncertainty awareness of LLMs under various code distribution shifts, using advanced probabilistic methods. While our benchmark datasets mainly focus on the Java programming language, the scope of real-world code distribution shifts covers a variety of other languages, such as Python and C++. Our methodologies for dataset creation and experimental designs hold the potential for broad applicability across these diverse programming contexts. Furthermore, our current research regards code analysis tasks as token-level classification (although token-level CC is analogous to language modeling). In our future work, we aim to extend our evaluation to generative tasks, such as comment generation and code search, to lead to more comprehensive understanding of LLMs' capabilities. However, these generative tasks typically demand human annotations when creating shifted datasets (e.g., comments that include time-sensitive information may need human calibrations). This requirement underscores the need for more automated approaches to address distribution shifts in the future.

References

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019a. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019b. code2vec: learning distributed representations of code. *Proc. ACM Program. Lang.*, 3(POPL):40:1–40:29.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1613–1622. JMLR.org.

Glenn W Brier. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3.

Malinda Dilhara, Danny Dig, and Ameya Ketkar. 2023. PYEVOLVE: automating frequent code changes in python ML systems. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 995–1007. IEEE.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1050–1059. JMLR.org.

Yarin Gal, Riashat Islam, and Zoubin Ghahramani. 2017. Deep Bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1183–1192. PMLR.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Neil Houlsby, Ferenc Huszar, Zoubin Ghahramani, and Máté Lengyel. 2011. Bayesian active learning for classification and preference learning. *CoRR*, abs/1112.5745.

Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Mike Papadakis, Lei Ma, and Yves Le Traon. 2023. Codes: Towards code model generalization under distribution shift. In *45th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results, NIER@ICSE, Melbourne, Australia, May 14-20, 2023*, pages 1–6. IEEE.

Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph Gonzalez, and Ion Stoica. 2021. Contrastive code representation learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5954–5971, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. 2021. Code prediction by feeding trees to transformers. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 150–162. IEEE.

Diederik P. Kingma, Tim Salimans, and Max Welling. 2015. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2575–2583.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *NeurIPS*.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6402–6413.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7167–7177.

Yufei Li, Xiao Yu, Yanchi Liu, Haifeng Chen, and Cong Liu. 2023. Uncertainty-aware bootstrap learning for joint extraction on distantly-supervised data. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1349–1358, Toronto, Canada. Association for Computational Linguistics.

Shiyu Liang, Yixuan Li, and R. Srikant. 2018. Enhancing the reliability of out-of-distribution image detection in neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Jeremiah Z. Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax-Weiss, and Balaji Lakshminarayanan. 2020. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

David JC MacKay. 1992. Bayesian interpolation. *Neural computation*, 4(3):415–447.

Wesley J. Maddox, Pavel Izmailov, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2019. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13132–13143.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. 2015. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2901–2907. AAAI Press.

Pengyu Nie, Jiyang Zhang, Junyi Jessy Li, Ray Mooney, and Milos Gligoric. 2022. Impact of evaluation methodologies on code summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4936–4960, Dublin, Ireland. Association for Computational Linguistics.

Apostolos F. Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. 2023. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *J. Comput. Phys.*, 477:111902.

Md Rafiqul Islam Rabin, Nghi DQ Bui, Ke Wang, Yijun Yu, Lingxiao Jiang, and Mohammad Amin Alipour. 2021. On the generalizability of neural program models with respect to semantic-preserving program transformations. *Information and Software Technology*, 135:106552.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J. Liu. 2023. Out-of-distribution detection and selective generation for conditional language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Lewis Smith and Yarin Gal. 2018. Understanding measures of uncertainty for adversarial example detection. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 560–569. AUAI Press.

Jasper Snoek, Yaniv Ovadia, Emily Fertig, Balaji Lakshminarayanan, Sebastian Nowozin, D. Sculley, Joshua V. Dillon, Jie Ren, and Zachary Nado. 2019. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13969–13980.

Dustin Tran, Jeremiah Z. Liu, Michael W. Dusenberry, Du Phan, Mark Collier, Jie Ren, Kehang Han, Zi Wang, Zelda Mariet, Huiyi Hu, Neil Band, Tim G. J. Rudner, Karan Singhal, Zachary Nado, Joost van Amersfoort, Andreas Kirsch, Rodolphe Jenatton, Nithum Thain, Honglin Yuan, Kelly Buchanan, Kevin Murphy, D. Sculley, Yarin Gal, Zoubin Ghahramani, Jasper Snoek, and Balaji Lakshminarayanan. 2022. Plex: Towards reliability using pretrained large model extensions. *CoRR*, abs/2207.07411.

Joost van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. 2020. Uncertainty estimation using a single deep deterministic neural network. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9690–9700. PMLR.

Artem Vazhentsev, Gleb Kuzmin, Artem Shelmanov, Akim Tsvigun, Evgenii Tsymbalov, Kirill Fedyanin, Maxim Panov, Alexander Panchenko, Gleb Gusev, Mikhail Burtsev, Manvel Avetisian, and Leonid Zhukov. 2022. Uncertainty estimation of transformer predictions for misclassification detection. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8237–8252, Dublin, Ireland. Association for Computational Linguistics.

Hongjun Wang and Yisen Wang. 2022. Self-ensemble adversarial training for improved robustness. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2020. Dissector: input validation for deep learning applications by crossing-layer dissection. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 727–738. ACM.

Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 1245–1256. IEEE / ACM.

Shusheng Xu, Xingxing Zhang, Yi Wu, and Furu Wei. 2022. Sequence level contrastive learning for text summarization. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 11556–11565. AAAI Press.

Polina Zablotskaia, Du Phan, Joshua Maynez, Shashi Narayan, Jie Ren, and Jeremiah Liu. 2023. On uncertainty calibration and selective generation in probabilistic neural summarization: A benchmark study. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2980–2992, Singapore. Association for Computational Linguistics.

A Settings

A.1 Preprocessing and Evaluation

Preprocessing Each Java file is converted into a string for tokenization and preprocessing. We normalize uncommon literals for better user experience. As suggested by (Lu et al., 2021), we don't require models to identify literals such as names, IP address, phone numbers or numeric literals, and thus normalize them by pre-defined special tokens. Considering that frequently-used literals may contain useful information, e.g., "__main__" or "utf-8", we preserve the 200 most frequent string and 30 most frequent numeric literals. These literals will be normalized by tokens in "<STR_LIT:utf-8>" format, while uncommon literals are replaced by <STR_LIT> or <NUM_LIT>. We add <s> and </s> to indicate the start and the end of one snippet of code, and <MASK> one token for prediction in each snippet line.

Code Completion CC aims to predict next code token given context of previous tokens, is a one of the most widely used features in software development through IDEs. An effective code completion tool could improve software developers' productivity. We evaluate each model in terms of sequence classification by top-k sub-token F-1 score following (Lu et al., 2021), which is analogous to language modeling. For instance, consider the function "countllines" with $k = 2$. We tokenize it into sub-tokens: "count" and "lines", using a Llama pretrained tokenizer. Then we query WordNet (Miller, 1995) to obtain the top-2 synonyms for each sub-token: "tally", "total" (for "count"), and "rows",

"stripes" (for "lines"). Combinations of synonyms like "lines|tally" and "count|rows" are treated as exact matches. Partial matches like "count" implies full precision but low recall, while extra matches like "count|blank|lines" suggests full recall but low precision. Out-of-vocabulary (OOV) sub-tokens are treated as false negatives, which reduces the recall performance.

Code Summarization Existing studies regarding CS consists of two main tasks: comment generation and method name prediction. In this work, we focus on MNP, which aims to predict method names based on the context provided by code snippets. Similar to CC, function names are also composed of sub-tokens which implies non-unique ground truths, and we evaluate the top-2 sub-token F-1 score of the predicted method names.

A.2 Models Setup

We use the Hugging Face pre-trained model *codellama/CodeLlama-7b-hf*. Besides CodeLlama, we also investigate three relatively smaller models that are popular in solving code analysis tasks: Code2Vec (Alon et al., 2019b), CodeBERT (Feng et al., 2020), and CodeGPT (Lu et al., 2021). Their sizes and configuration are shown in Table 7. For CodeBERT and CodeGPT, we use their Hugging Face pre-trained models: *microsoft/codebert-base*, *microsoft/CodeGPT-small-java-adaptedGPT2*. The training hyperparameters of each model are tuned using grid search. We fine-tune each model for 100 epochs, with a learning rate of 2e-5 and weight decay of 1e-4, using the AdamW optimizer (Loshchilov and Hutter, 2019). The training batch size is 8. For Code2Vec, we select the hidden dimension based on the validation accuracy. For each experiment, we run the model 5 times and report the average numbers as the evaluation results. Below are pre-defined list from which we select hyperparameter values:

Learning rate: [5e-6, 7e-9e-6, 1e-5, 2e-5, 5e-5, 7e-5, 1e-4, 2e-4, 5e-4];
Number of epochs: $\{n \in \mathbb{N} | 15 \leq n \leq 150\}$;
Batch size: [2, 4, 6, 8, 10, 12, 14, 16];
Weight decay: [0, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1];
Hidden dim.: [32, 64, 128, 256, 512, 1024].

A.3 Datasets

Timeline Shift The four timelines we chose for training/dev, shift1, shift2, and shift3 are June 2020, December 2020, July 2021, and January 2022.

OOD Both BigCloneEval and Ruoyiplus are relatively large datasets, and we extracted 15,000 Java files in both projects. We randomly select 2,000 / 500 Java files from BigCloneEval as the training/dev datasets, and 500 Java files from Ruoyiplus as the OOD test set. Table 8 shows their statistics after preprocessing, where we see a much more severe shift compared to the previous three according to the higher KL and lower cosine scores.

A.4 Hardware

All of our experiments, including the fine-tuning of LLMs, uncertainty calibration, and evaluation, are conducted on an NVIDIA A6000 Ada Server, as shown in Table 9.

B Probabilistic Methods

In this section, we illustrate the additional details for the probabilistic methods, providing a comprehensive understanding of their characteristics and how their properties may facilitate LLM calibration. For Monte-Carlo Dropout, Deep Ensemble, Mutation Testing, and Dissector, the essential idea is to get an approximation of the predictive posterior of a model by averaging the probabilities from several independent models (also referred to the *committee size*).

Vanilla Baseline Vanilla logits (after softmax) are directly used as predictive probabilities without calibration. The UE method is winning score:

$$u_{\text{ws}}(x) = 1 - \max_{c \in C} p(y = c|x) = 1 - \max_{c \in C} p_t^c \quad (2)$$

The time complexity is $O(B)$, where B is the complexity of the base model, e.g., CodeLlama.

Temperature Scaling TS is a post-hoc calibration method based on a hold-out validation set. It works effectively on in-distribution sets but can be prone to degradation under distribution shifts, as shown by our study. We tune the parameter T_{ts} using a validation set, by minimizing the negative log-likelihood (NLL) loss between the scaled logits and labels with the LBFGS optimizer for 50 training epochs. The time complexity is $O(1) + O(B) = O(B)$. The UE method is Shannon entropy over the new predictive distribution:

$$u_{\text{entropy}} = -\sum_{c=1}^{C} \frac{e^{l^c/T_{\text{ts}}}}{\sum_k e^{l^k/T_{\text{ts}}}} \log \left(\frac{e^{l^c/T_{\text{ts}}}}{\sum_k e^{l^k/T_{\text{ts}}}} \right) \quad (3)$$

Model	Hidden Dim.	Attention Heads	Layers	Parameters
Code2Vec (Alon et al., 2019b)	256	*0*	6	14,519,554
CodeBERT (Feng et al., 2020)	768	12	12	135,917,634
CodeGPT (Lu et al., 2021)	768	12	12	135,699,456
CodeLlama (Rozière et al., 2023)	4096	32	32	6,667,448,320

Table 7: Model size and configuration in our experiments.

	BigCloneEval	Ruoyiplus
Paradigm	Language tool	J2EE platform
Snippet size	10.74/10.73	9.01
# Snippets	15,652/3,896	3,768
KL↑	*0.0*	3.44
Cosine↓	*0.0*	0.68
Vocab	13,828	11,704

Table 8: Statistics of the BigCloneEval (in-distribution) and Ruoyiplus (OOD) datasets.

Monte-Carlo Dropout MCD is a Bayesian technique leveraging dropout regularization during training inference. By sampling multiple predictions, MCD measures how *inner-model* parameters align with the data distribution and thus the confidence in its predictions. We set the number of dropout samples T_{mcd} as 10 and the dropout rate as 0.1. The time complexity is $O(B \times T_{\text{mcd}})$. The three UE methods are formulated as:

$$u_{\text{sws}} = 1 - \frac{1}{T_{\text{mcd}}} \sum_{t=1}^{T_{\text{mcd}}} \max_{c \in C} p_t^c \tag{4}$$

$$u_{\text{pv}} = \frac{1}{C} \sum_{c=1}^{C} \left(\frac{1}{T_{\text{mcd}}} \sum_{t=1}^{T_{\text{mcd}}} (p_t^c - \overline{p_t^c})^2 \right) \tag{5}$$

$$u_{\text{bald}} = \frac{1}{T_{\text{mcd}}} \sum_{t,c} p_t^c \log p_t^c - \frac{1}{C} \sum_{c=1}^{C} \overline{p_t^c} \log \overline{p_t^c} \tag{6}$$

Deep Ensemble DE is a non-Bayesian method that trains several independent deterministic models, with each capturing different features of the data and output more precise and robust predictions when combined. Its uncertainty captures the *cross-model* parametric uncertainty. Our study also proves its robustness in both top-ranked calibration ability and uncertainty estimation precision. We train $T_{\text{de}} = 5$ deterministic CodeLlama models and averages all. In the original work, each model is trained with different architectures or subsets or training data. We focus on CodeLlama and apply self-ensemble (Wang and Wang, 2022) in our experiments, which has been shown helpful for boosting LLM performance (Li et al., 2023). The time complexity is $O(B \times T_{\text{de}})$. The three UE methods are the same as MCD but across ensemble models.

Mutation Testing MT assumes adversarial inputs are closer to the model decision boundary and thus more sensitive to the model mutation operations. (Wang et al., 2019) propose four DNN model mutation operators, GF, WS, NS, and NAI, as shown in Table 10. For each operator, we mutate 100 times to obtain a series of mutated models $\mathcal{M} = \{f_i' | 1 \leq i \leq 100\}$ where $|\mathcal{M}| = 100$. In other words, the calibrated predictive distribution and UE results are obtained from the 100 mutated models. To save space, we only report the UE results of MT-GF and MT-WS in the paper, as the other two techniques produce similar patterns. The time complexity is $O(B + |\mathcal{M}| \times F)$, where F is the complexity of each mutation operation. As $|\mathcal{M}|$ is typically large and F is related to the size of deep models, the calibration overhead of MT can be extremely large in practice. The UE method is the label change rate (LCR) between the original output and mutated model outputs:

$$u_{\text{lcr}}(x) = \frac{|\{f_i'(x) \in \mathcal{M} | f_i'(x) \neq f(x)\}|}{|\mathcal{M}|} \tag{7}$$

Dissector DS assumes a correctly classified inputs should have an increasing intermediate consistency across hidden layers, and is thus effective in misclassification detection. We select the 1st, 3rd, 5th, 7th, and 9th transformer layers of LLMs to train the corresponding snapshots. Each snapshot is a linear layer trained using stochastic gradient descent (SGD) with cross-entropy loss for 10 epochs. Suppose the original model f predicts x as label $\hat{y}$, the snapshot validity (SV) score first measures how $\hat{y}$ is uniquely supported by the probability vector in this snapshot. If $\hat{y}$ is aligned with the highest probability, Dissector measures uniqueness as how much $\hat{y}$'s associated probability $p_{S_l}^{\hat{y}}$ exceeds the

CPU	Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz
CPU Cores	32
GPU	NVIDIA RTX 6000 Ada Generation
GPU Memory	50 GB

Table 9: Hardware configuration in the experiments.

Mutation Operator	**Level**	**Description**
Gaussian Fuzzing (GF)	Weight	Fuzz weights of LLMs by Gaussian Distribution
Weight Shuffling (WS)	Neuron	Shuffle selected weights with ratio 25%
Neuron Switch (NS)	Neuron	Switch two neurons within each layer of LLMs
Neuron Activation Inverse (NAI)	Neuron	Deactivate the status of a neuron with ratio 25%

Table 10: Model mutation operators in Mutation Testing (MT).

Grow Type	**Formula**	**Reduced Formula**	**Number of Parameters**
Linear	$\alpha = ax + l$	$\begin{cases} \alpha = x \\ \alpha = wx + 1 \end{cases}$	$2 \rightarrow 1$
Logarithmic	$\alpha = a \log_b(lx + k_1) + k_2$	$\begin{cases} \alpha = \ln x \\ \alpha = w \ln x + 1 \\ \alpha = w \ln(\beta x + 1) + 1 \end{cases}$	$5 \rightarrow 2$
Exponential	$\alpha = ae^{lx+b_1} + b_2$	$\begin{cases} \alpha = e^{\beta x} \\ \alpha = we^{\beta x} + 1 \end{cases}$	$4 \rightarrow 2$

Table 11: Parameter reduction for modeling weights in Dissector.

Pattern	**Code Completion**				**Code Summarization**			
	Dev	**Shift1**	**Shift2**	**Shift3**	**Dev**	**Shift1**	**Shift2**	**Shift3**
TIMELINE	0.44	0.55	0.46	0.43	0.03	0.06	0.06	0.09
PROJECT	0.44	0.21	0.69	0.35	0.08	0.08	0.02	0.08
AUTHOR	0.57	0.94	0.82	1.15	0.21	0.19	0.16	0.15
Pattern	**Dev**		**OOD**		**Dev**		**OOD**	
PARADIGM	0.41		*0.0*		0.08		0.12	

Table 12: Zero-shot F-1 scores ($\times 10^{-2}$) of CodeLlama across all the shifted and OOD datasets.

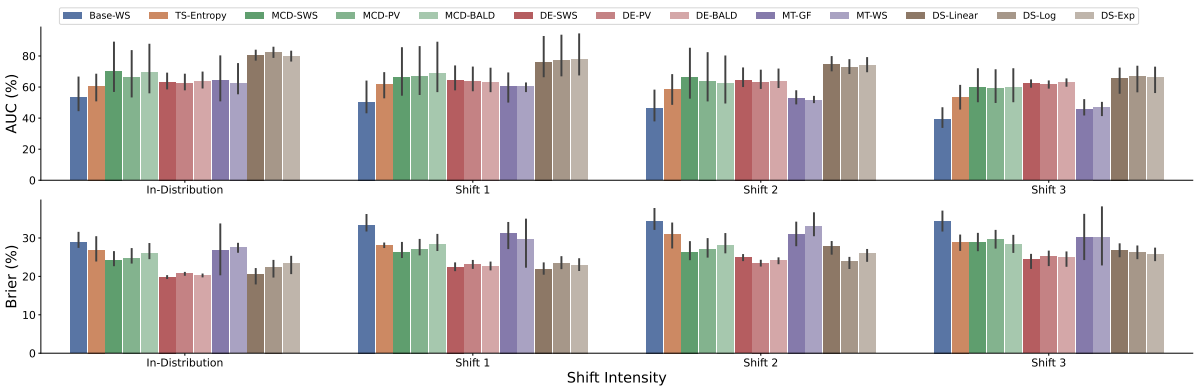


Figure 8: AUC↑ and Brier↓ results for detecting CodeLlama mistakes using different methods in CS under intensifying shifts. Error bars indicate variations across the three shift patterns.

second highest probability $p_{S_l}^{c_{(2)}}$ (with label $c_{(2)}$). Otherwise, Dissector measures how much the actual highest probability $p_{S_l}^{c_{(1)}}$ (with label $c_{(1)}$) exceeds that of $\hat{y}$:

$$\mathrm{SV}(\hat{y}, S_l) = \begin{cases} \frac{p_{S_l}^{\hat{y}}}{p_{S_l}^{\hat{y}} + p_{S_l}^{c_{(2)}}}, & \hat{y} = \underset{c}{\operatorname{argmax}}\, p_{S_l}^{c} \\ 1 - \frac{p_{S_l}^{c_{(1)}}}{p_{S_l}^{\hat{y}} + p_{S_l}^{c_{(1)}}}, & \text{otherwise} \end{cases} \tag{8}$$

The SV score is in range [0, 1], with a larger value indicating more uniquely the current snapshot supports the final prediction result $\hat{y}$. The UE method is profile validity (PV), i.e., the normalized sum of SV scores across layers:

$$u_{\text{spv}} = \frac{\sum_{l \in L} \alpha_l \cdot \mathrm{SV}_l(\hat{y}, S_l)}{\sum_{l \in L} \alpha_l} \tag{9}$$

Here L is the set of hidden layers for training snapshots and α_l is the associated weight for the l^{th} layer. Table 11 lists general formulas for computing weight α values regarding the three growth types, based on the DS assumption that a within-input should have increasing confidence across the intermediate layers of a model. To reduce parameters, (Wang et al., 2020) also reduce these formulas with one (w) or two (w and β) parameters only. The time complexity is $O(B + |L| \times W)$, where L is the set of hidden layers used for training snapshots, and W is the complexity of each snapshot submodel.

C Additional Results

Zero-Shot Performance Table 12 presents the zero-shot performance of CodeLlama in two popular code analysis tasks. We observe a noticeable challenge: the model produces nearly zero F-1 in all datasets. This suggests current capabilities of CodeLlama in zero-shot settings, especially when deployed for solving complex tasks, are still inadequate, underscoring further fine-tuning in real-world code analysis applications.

Misclassification Detection Figure 8 shows the AUC and Brier results of misclassification detection for different methods in CS. We observe a similar pattern as in CC: DS is superior in validating low-quality predictions or mistakes. Figure 9 presents the AUPR results of misclassification detection under the three shifted datasets. Consistent with the AUC and Brier performance, we also observe a degradation of quality of UE methods under distribution shifts, as evidenced by lower AUPR on the shift3 set. Probabilistic methods generally improve the awareness of the mistakes over the vanilla baseline. DE, MCD, and DS achieve promising AUPR results. Interestingly, post-hoc calibration (TS) is prone to degradation under intensifying shifts.

Selective Prediction Figure 10 demonstrates the selective prediction results in CS. We observe probabilistic methods generally lead to higher F-1 performance with the same rejection rate compared to the vanilla baseline. Among them, DE and DS still achieve higher efficiency. However, compared to the results in CC, the highest F-1 is much smaller and cannot reach nearly 100%. This is because the prediction quality of the vanilla model in CS is much lower than in CC, causing the UE a bit more challenge. Figure 11 illustrates the selective prediction results of the remaining UE methods. We observe different UE techniques that correspond to the same probabilistic method produce similar results. DE and DS are efficient in achieving the same prediction quality with a lower abstention rate. Figure 12 demonstrates the abstention prediction performance for both in-distribution and OOD examples. We observe a more severe degradation of efficiency. Among them, DE and DS still achieve higher F-1 scores in the OOD dataset, while the MT is relatively less effective compared to other probabilistic methods. This again suggests its UE assumption is sensitive to severe shifts but less relevant to the reliability of model predictions.

Calibration Quality for Different Models In addition to CodeLlama, we further evaluate the calibration performance for Code2Vec, CodeBERT, and CodeGPT. Table 13 summarizes their F-1 scores across the three shifted datasets. Similar to results from CodeLlama, probabilistic methods consistently outperform the vanilla baseline, among which DE exhibits particularly notable improvements. We observe a general decline in F-1 relative to CodeLlama, with the smaller Code2Vec model experiencing a significant reduction (approximately 20% in CC and 15% in CS). This bias highlights the varied capabilities of these models in processing code instances. Interestingly, we observe that the calibration effect of probabilistic methods is more apparent for the small Code2Vec model, e.g., DE increases the F-1 score of Code2Vec from 29.33% to 40.29% in CS for the shift3 dataset. This suggests that powerful models may challenge the calibration effect. Figure 13 displays the ECE and

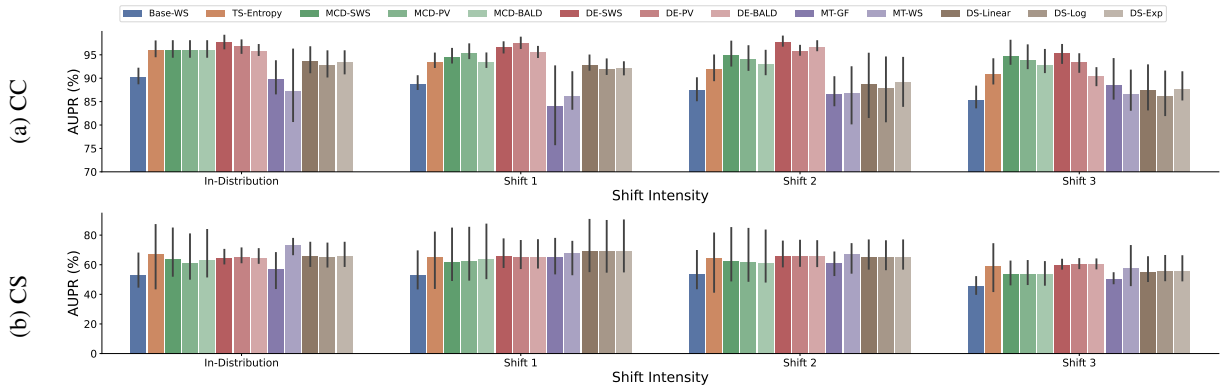


Figure 9: AUPR↑ results for detecting CodeLlama mistakes using different methods in (a) CC and (b) CS, with intensity shifts. Error bars indicate AUPR variations across the three shift patterns.

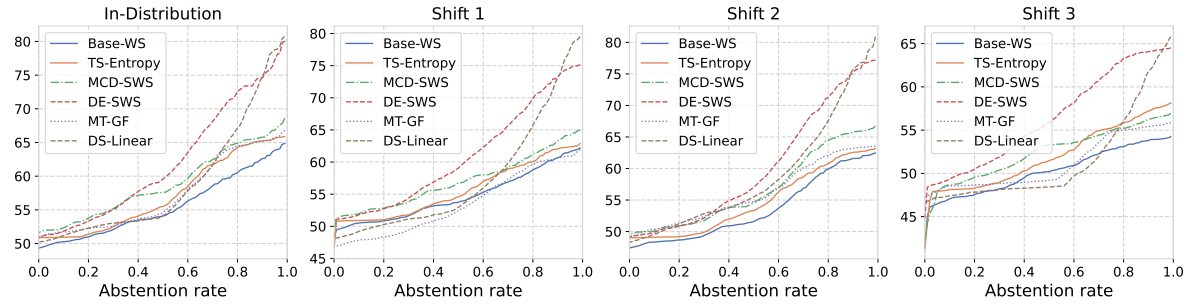


Figure 10: F-1 vs. Abstention curve for CodeLlama using different methods in CS, with intensifying shifts. Each line is the average F-1 score over the three shift patterns.

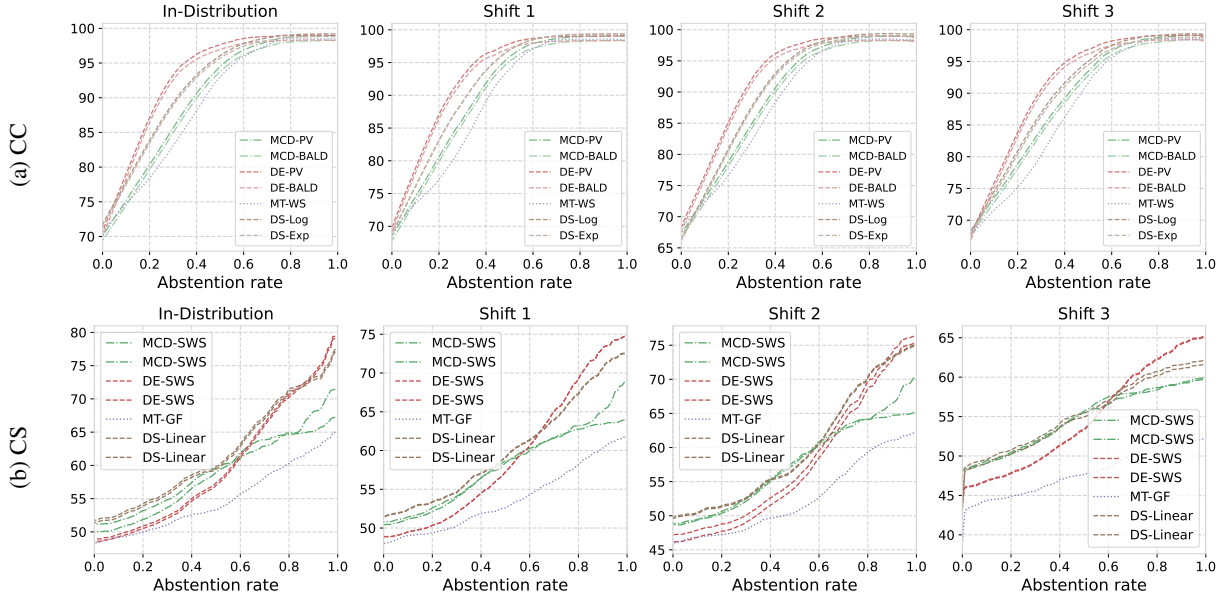
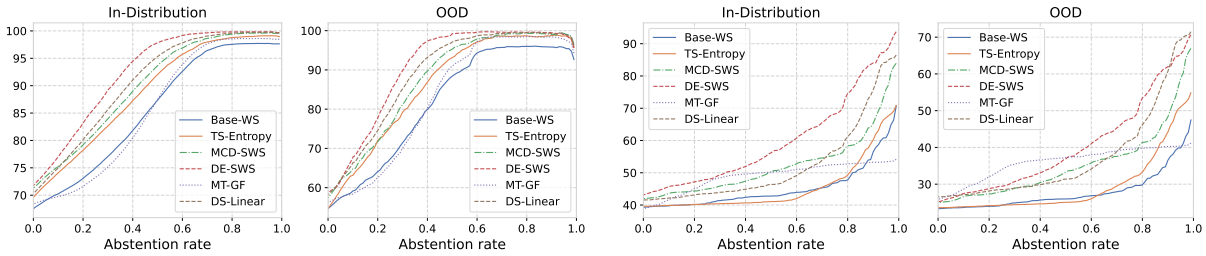


Figure 11: F-1 vs. Abstention curve for CodeLlama in (a) CC and (b) CS, using the remaining methods, with intensifying shifts. Each line is the average F-1 score over the three shift patterns.



(a) CC (b) CS

Figure 12: F-1 vs. Abstention curve for CodeLlama in (a) CC and (b) CS, using different methods in handling both in-distribution and OOD code samples.

Pattern	Model	Method	Code Completion						Code Summarization					
			Dev	Shift1	Shift2	Shift3	Avg↑	Rank↓	Dev	Shift1	Shift2	Shift3	Avg↑	Rank↓
TIMELINE	*Code2Vec*	Base / TS	45.00	44.28	43.80	42.15	43.81	5.00	30.80	30.69	30.51	30.37	30.59	4.75
		MCD	48.77	48.00	47.15	46.28	47.55	2.75	36.11	36.04	35.68	34.79	35.66	2.00
		DE	49.66	49.86	48.60	47.54	**48.92**	**1.00**	39.17	39.11	38.75	38.74	**38.66**	**1.00**
		MT	48.80	48.06	47.15	46.30	47.58	2.25	33.51	33.47	33.50	32.94	33.36	3.00
		DS	48.02	47.25	46.41	46.28	46.99	4.00	31.49	31.03	31.00	29.56	30.77	4.25
	CodeBERT	Base / TS	67.62	64.87	64.74	64.94	65.64	4.25	45.60	45.39	44.88	44.12	44.99	4.75
		MCD	67.80	65.03	64.82	64.99	65.66	2.75	49.39	47.63	47.12	46.56	47.68	**1.75**
		DE	69.75	67.50	67.21	67.18	**67.91**	**1.00**	51.21	48.58	46.37	45.28	**47.86**	**1.75**
		MT	67.63	65.01	64.89	64.79	65.58	3.50	45.80	45.39	44.90	44.13	45.06	4.25
		DS	68.61	66.53	64.74	64.25	66.03	3.50	48.85	48.84	48.26	47.55	47.13	2.50
	CodeGPT	Base / TS	70.03	66.84	66.72	66.66	67.56	5.00	48.85	48.83	48.23	47.59	48.38	4.50
		MCD	71.99	68.89	68.55	68.70	69.53	2.25	51.84	50.81	49.19	48.63	50.12	2.00
		DE	71.90	69.11	68.86	68.73	**69.65**	**1.75**	53.13	53.06	52.31	51.79	**52.57**	**1.00**
		MT	70.11	66.91	66.88	66.75	67.66	4.00	48.86	48.85	48.23	47.59	48.38	4.50
		DS	72.01	67.95	68.22	68.87	69.26	2.00	49.66	49.61	48.93	48.26	49.11	3.00
PROJECT	*Code2Vec*	Base / TS	46.43	46.27	45.14	42.11	44.99	5.00	34.61	32.58	30.04	26.53	30.94	5.00
		MCD	48.00	47.41	45.77	44.76	**46.49**	2.50	42.64	40.55	39.82	37.02	40.01	2.00
		DE	48.78	48.10	45.28	43.27	46.36	**2.00**	46.87	45.55	40.84	32.31	**41.39**	**1.75**
		MT	48.00	47.47	45.86	43.02	46.09	2.75	37.21	34.14	33.44	32.48	34.32	3.75
		DS	48.49	47.63	45.62	42.29	46.01	2.75	43.64	39.82	38.34	36.89	39.67	2.50
	CodeBERT	Base / TS	64.97	62.62	59.89	57.13	61.15	5.00	43.58	41.88	41.48	36.94	40.97	4.75
		MCD	67.41	66.70	65.89	64.13	66.03	1.75	49.88	48.94	47.71	46.18	48.18	**2.00**
		DE	67.62	65.88	63.12	60.23	64.96	2.75	50.96	48.95	47.08	45.59	48.15	**2.00**
		MT	65.08	63.12	60.15	57.55	61.47	4.00	44.52	43.58	41.89	36.91	41.73	4.25
		DS	69.49	66.94	67.75	63.98	**66.54**	**1.50**	52.40	49.33	46.87	44.45	**48.26**	**2.00**
	CodeGPT	Base / TS	67.60	66.83	63.75	61.25	64.86	4.00	51.69	48.89	47.28	45.54	48.35	3.25
		MCD	67.88	67.72	64.66	62.17	65.61	2.00	51.69	48.89	47.28	46.53	48.60	3.50
		DE	70.12	69.23	66.21	63.97	**67.38**	**1.00**	53.28	51.68	48.44	47.59	**50.25**	**1.00**
		MT	60.37	58.91	58.12	57.05	58.61	5.00	51.82	48.97	47.02	46.08	48.47	3.00
		DS	67.66	67.38	64.01	61.80	65.21	3.00	51.69	48.89	47.42	45.54	48.39	4.25
AUTHOR	*Code2Vec*	Base / TS	51.20	50.21	49.95	48.68	50.01	5.00	39.65	37.13	36.60	29.33	35.68	5.00
		MCD	59.64	59.17	58.14	54.90	57.96	3.25	44.24	42.43	41.46	34.10	40.56	3.00
		DE	60.28	59.85	58.74	55.56	**58.60**	**1.25**	50.29	49.93	49.05	40.29	**47.39**	**1.00**
		MT	59.68	59.20	58.21	54.95	58.01	2.25	41.52	39.74	38.93	31.47	37.92	4.00
		DS	59.18	57.14	56.19	55.98	57.12	3.25	44.83	44.50	44.08	36.11	42.38	2.00
	CodeBERT	Base / TS	72.40	72.11	71.37	69.27	71.29	4.00	48.57	46.00	45.26	41.98	45.45	4.50
		MCD	72.52	72.14	71.37	69.41	72.36	3.00	49.41	46.91	46.19	42.46	47.74	2.00
		DE	74.25	73.93	73.19	71.50	73.22	2.00	53.27	48.33	46.22	42.72	47.64	2.50
		MT	72.40	72.11	71.37	69.27	71.29	5.00	48.72	46.31	45.26	41.98	45.57	4.50
		DS	75.24	74.41	74.42	73.40	**74.37**	**1.00**	50.71	49.39	48.15	44.69	**48.24**	**1.50**
	CodeGPT	Base / TS	75.28	74.87	74.03	72.24	74.10	5.00	50.31	47.49	47.62	42.62	47.01	5.00
		MCD	75.29	74.93	74.15	72.40	76.44	**1.50**	49.89	48.73	47.46	43.58	48.92	2.25
		DE	76.62	76.13	75.21	73.86	75.46	3.00	56.55	53.92	53.68	48.94	**53.27**	**1.00**
		MT	75.31	74.96	74.04	72.28	74.15	4.00	50.67	47.94	47.78	43.10	47.37	4.00
		DS	77.44	76.90	76.52	75.15	**76.50**	**1.50**	50.98	49.43	48.27	44.74	48.36	2.75

Table 13: F-1 scores and ranking of different methods for Code2Vec, CodeBERT, and CodeGPT, across the three shift patterns. All methods consistently produce lower F-1 under intensifying shifts. Probabilistic methods (except TS) outperform the vanilla baseline.

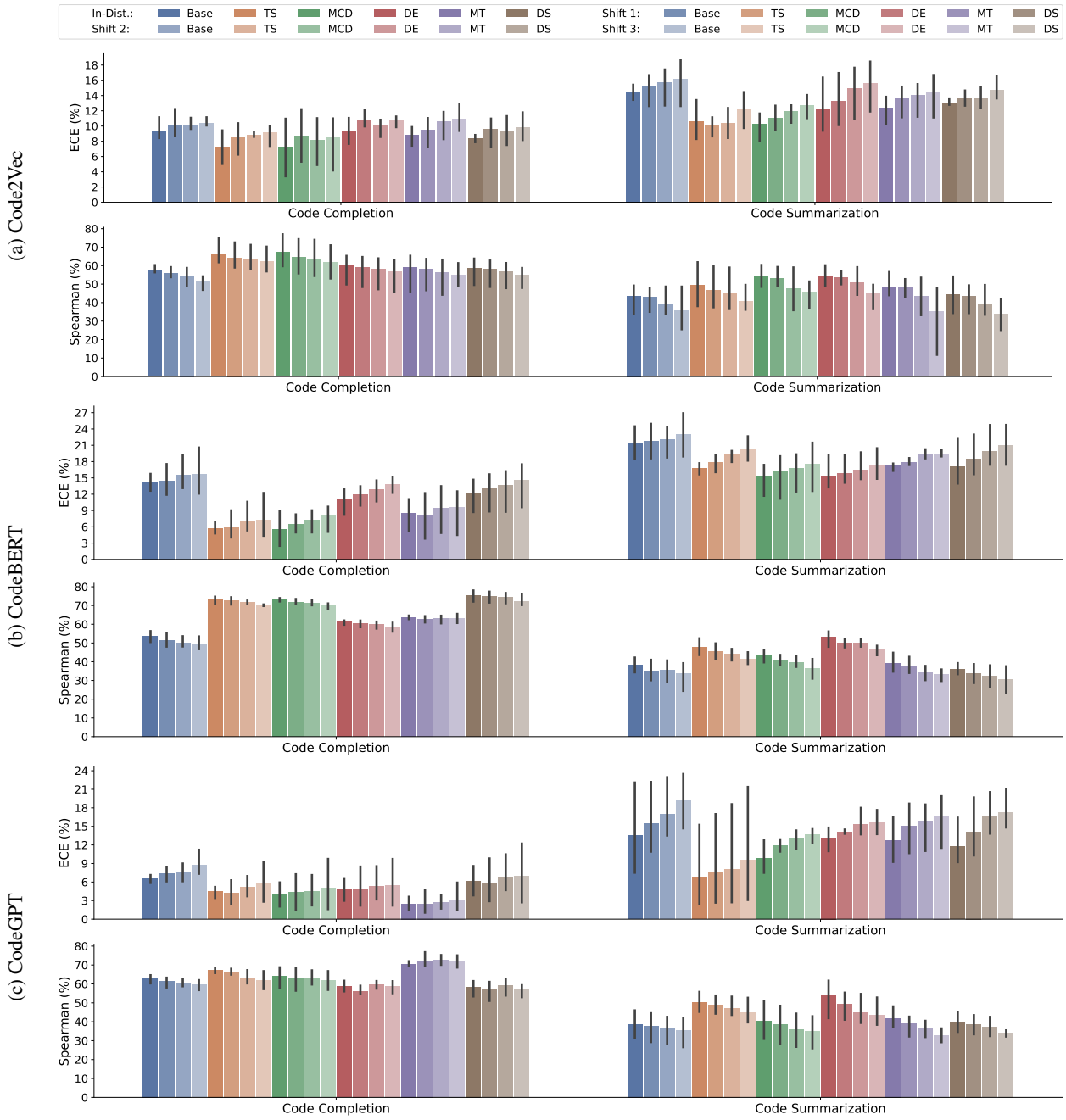


Figure 13: ECE↓ and Spearman's Rank Correlation↑ for (a) Code2Vec, (b) CodeBERT, and (c) CodeGPT of different methods in handling both in-distribution and shifted code examples. Error bars indicate variations across the three shift patterns.

rank correlation of probabilistic methods across the three models. Consistent with the CodeLlama findings, TS and MCD excel in lower ECE and higher correlation scores.