

Verbalized Uncertainty in Coding LLMs: A Contamination-Aware Study of Confidence Calibration

Hunter Tuerlings and Matias Valdenegro-Toro

Department of Artificial Intelligence

University of Groningen

`h.f.m.tuerlings@student.rug.nl, m.a.valdenegro.toro@rug.nl`

Abstract

Coding large language models (LLMs) can produce plausible solutions that fail hidden test cases, making it difficult to judge when to trust their output. This paper presents a contamination-aware comparison of established methods for eliciting verbalized confidence in generated code. Six instruction-tuned, coding-specialized LLMs from three model families are evaluated on 365 problems from a shared contamination-aware LIVE-CODEBENCH window. For each frozen code generation, confidence is elicited separately under three strategies, Direct numeric scoring, Chain-of-Thought reasoning followed by a score, and Linguistic confidence labels. Under strict scoring, all 18 model-strategy combinations have negative Net Calibration Error, showing a consistent tendency toward overconfidence. Linguistic elicitation achieves the lowest Expected Calibration Error for four of six models, but several favourable values result from collapsed confidence distributions. Eight of the 18 combinations are collapsed and five are unstable across repeated runs, including three Chain-of-Thought combinations. When outputs are ranked by confidence and the least confident are reviewed first, Direct elicitation gives the largest improvement over random selection in four of six models. Verbalized confidence cannot be reduced to a single trustworthy pass probability; calibration, ranking ability, distributional collapse, and stability must each be checked separately.

1 Introduction

Coding large language models (LLMs) are widely used and can produce plausible-looking code that appears correct at first glance. Generated code can still fail on hidden test cases or introduce subtle bugs through edge cases, so knowing when to trust an output matters in practice. In the 2025 Stack Overflow Developer Survey, 84% of respondents report using or planning to use AI-assisted tools,

yet 46% actively distrust the accuracy of the output they receive (Stack Overflow, 2025).

A natural response is to pair each generated solution with a confidence score, giving developers some insight into how much review the code might need (Spiess et al., 2025). A confidence score is a number, here on a 0 to 100 scale, expressing how likely the model judges its own output to be correct, and an uncertainty estimate is the same score read in reverse. Such a score is only useful if it is well calibrated, meaning predictions made with 70% confidence should be correct about 70% of the time (Geng et al., 2024).

The score studied here comes from verbalized confidence elicitation, in which the model states its confidence directly in natural language rather than reading it off internal token probabilities. The exact wording used to ask for that judgment is the elicitation strategy, and different strategies can produce different confidence values for the same code.

Figure 1 shows why the problem is not trivial. Both programs come from the same model under the same elicitation strategy, yet the reported confidence runs opposite to correctness. The program in (a) is rated 100 percent confident and fails the hidden tests, while the program in (b) is rated 0 percent confident and passes every one. A score that behaves like this is worse than no score at all, because it turns a developer toward trusting the wrong output and re-checking the right one. Both cases are worked through in Appendix P.

Prior work shows that LLMs are often poorly calibrated when asked to verbalize their uncertainty, with overconfidence observed across NLP and vision-language tasks (Xiong et al., 2024; Groot and Valdenegro-Toro, 2024). For code specifically, Spiess et al. (2025) find poor calibration across several code-related tasks, but their evaluation relies on static benchmarks whose problems may already have appeared in training data, which inflates performance and complicates the interpreta-

(a) Incorrect code, confidence 100
LeetCode 3417: zigzag traversal while skipping every other visited cell.

```
for i in range(rows):
    if i % 2 == 0:
        for j in range(0, cols, 2): ...
    else:
        for j in range(cols - 1, -1, -2): ...
FINAL_CONFIDENCE: 100
```

(b) Correct code, confidence 0
LeetCode 3502: minimum cost to reach every position in a line.

```
for i in range(1, n): # redundant loop
    answer[i] = cost[i]
    for j in range(i): ...
for i in range(n): # correct prefix minima
    min_cost = min(min_cost, cost[i])
    answer[i] = min_cost
FINAL_CONFIDENCE: 0
```

Figure 1: Two Qwen2.5-Coder-32B outputs under Direct elicitation. In (a), row-wise skipping breaks the required global traversal pattern, yet receives confidence 100. In (b), a redundant loop does not affect correctness, yet the passing program receives confidence 0. Full code and test-level explanations are in Appendix P.

tion of calibration results (Jain et al., 2025). LIVECODEBENCH addresses this by continuously collecting newly released competitive programming problems with release dates, so that evaluation can be restricted to problems that postdate a model’s training cutoff (Jain et al., 2025). That restriction is what makes the setting contamination-aware, and code generation is the LIVECODEBENCH task with enough post-cutoff problems to support the calibration analysis.

In code generation the model receives a problem specification and must produce a complete program, which is then executed against a suite of hidden test cases, so correctness is a binary pass or fail signal rather than a subjective judgment. The following research question guides this paper.

To what extent do verbalized confidence elicitation strategies produce calibrated uncertainty estimates for objectively scored code generation correctness in a contamination-aware setting, and how does calibration vary across elicitation strategies and models?

Three elicitation strategies are applied to six coding-specialized LLMs spanning three model families and two parameter scales each, with calibration quality, direction of miscalibration, ranking ability, and stability each assessed separately. Multiple models and strategies are compared because both the model and the elicitation wording can change the result, so a single pairing would not reveal whether an observed pattern generalizes. Our contribution is a contamination-aware comparison of established verbalized-confidence methods, using the same generated code and correctness labels across strategies within each model. Further,

we also assess how an additional call to repair incorrectly formatted confidence responses affects calibration and ranking.

2 Related Work

Calibration in language models: Calibration refers to the alignment between a model’s reported confidence and its empirical accuracy across many predictions (Geng et al., 2024). Two properties matter in practice and can come apart. Calibration quality is how closely stated confidence tracks observed accuracy on average. Ranking ability, also called discriminative ability, is whether the score separates correct predictions from incorrect ones regardless of whether the values are numerically calibrated. A related property, selective prediction, asks whether ordering outputs by confidence lets a developer review the least trustworthy ones first, which can hold even when the absolute values are miscalibrated.

Verbalized confidence in LLMs: Uncertainty methods are commonly divided into intrinsic approaches, which use model internals (e.g., log-token probabilities), and verbalized approaches, which ask for a number or label (Geng et al., 2024; Xiong et al., 2024). This work studies verbalized confidence because many deployed APIs expose only text and because token likelihood need not represent whole-program correctness. Recent intrinsic studies show that code uncertainty depends highly on structure; perplexity, how surprised an LLM is by a sequence of text, varies by programming language and model in code completion (Kotti et al., 2026), while syntax-aware aggregation improves uncertainty estimation for function calling (Ye et al., 2026). More broadly, a single scalar may conflate input ambiguity, knowledge gaps,

and decoding randomness (Taparia et al., 2026). Groot and Valdenegro-Toro (2024) find high calibration error and predominantly overconfident verbalized scores across several NLP tasks, while Xiong et al. (2024) show that values often cluster in the 80-100% range and on round multiples of five. Larger models can be well calibrated in structured multiple-choice and true/false settings (Kadavath et al., 2022), but Chain-of-Thought effects remain inconsistent across tasks and models (Xiong et al., 2024).

Calibration for code generation: Code is a useful calibration setting because hidden tests provide an objective binary correctness signal (Spiess et al., 2025). Spiess et al. (2025) evaluate intrinsic and reflective confidence across completion, synthesis, and repair, finding poor out-of-the-box calibration; however, their static benchmarks, including HumanEval and MBPP, may have appeared in training (Chen et al., 2021; Austin et al., 2021; Jain et al., 2025). Recent work has expanded uncertainty estimation for code generation through uncertainty-guided Chain-of-Thought prompting (Zhu et al., 2025), risk-controlling prediction sets (Xu et al., 2026), and multicalibration of likelihood-based confidence (Campos et al., 2026). These methods primarily focus on generation-time uncertainty or post-hoc calibration using intrinsic model signals. In contrast, no existing work compares black-box verbalized elicitation strategies across multiple coding-specialized LLMs on a shared temporally filtered window. This paper addresses that gap by combining the verbalized-confidence framework of Xiong et al. (2024) with the code-calibration framing of Spiess et al. (2025) and the contamination-aware design of LIVECODEBENCH (Jain et al., 2025).

3 Methods

The evaluation pipeline proceeds in five stages. Eligible problems are selected from LIVECODEBENCH using a shared post-cutoff window. Each model generates one solution per problem; these outputs are frozen. Frozen outputs are scored for correctness against hidden test cases. Confidence is elicited in a separate inference pass under three prompting strategies. Finally, the resulting confidence scores and correctness labels are paired to compute calibration metrics. Figure 4 in Appendix A summarizes the flow, and the subsections below describe each stage in the order it is run.

3.1 Benchmark and problem selection

LIVECODEBENCH is a continuously updated benchmark of competitive-programming problems collected from public contest platforms, with each problem tagged by its release date (Jain et al., 2025). Problems come in two input formats, that is function-completion starter code with a fixed signature, and standard input and output. Each problem has public example tests and a larger hidden test suite, and grading uses the hidden suite so a solution cannot be tuned to the visible cases.

Under strict scoring a solution counts as correct only when it produces the expected output on every hidden test, following the execution-based functional-correctness protocol of Chen et al. (2021). This all-or-nothing rule makes correctness a binary label and is the primary scoring used throughout. Lenient scoring is a secondary diagnostic that additionally counts a solution as correct when the expected output is a single number and the generated value falls within a 10% relative tolerance, with a full tolerance sweep in Appendix F.

All experiments use LIVECODEBENCH release_v6 (Jain et al., 2025), from the Hugging Face dataset `livecodebench/code_generation_lite` at revision `0fe84c3`, scored with the harness at commit `6729765`. Restricting evaluation to problems released only after a model’s training cutoff is central to the design, and to keep the comparison identical, all six models are evaluated on the same window from 2024-09-19 to 2025-04-30 inclusive. The window starts there because Qwen2.5-Coder is the newest family used and does not publish a formal training-data cutoff, so the first public arXiv submission date of its technical report, 2024-09-18, is used as a transparent proxy boundary (Hui et al., 2024). This does not prove that every earlier problem was in training nor that every later one could not have been seen, but it provides a reproducible boundary, mirroring the temporal split that defines LIVECODEBENCH itself (Jain et al., 2025). The heuristic is supported by Li and Flanigan (2024), who show that models perform better on tasks released before their training cutoff than on tasks released after. Within this window the code generation task yields exactly 365 eligible problems.

3.2 Models

Six instruction-tuned, coding-specialized LLMs are evaluated, drawn from three families at two parameter scales each, Qwen2.5-Coder (Hui et al.,

2024), CodeLlama (Rozière et al., 2023), and DeepSeek-Coder (Guo et al., 2024). Evaluating two scales within each family allows calibration to be examined as a function of capacity while holding the family fixed, a comparison of interest because larger models have been reported to be better calibrated in suitably formatted settings (Kadavath et al., 2022), and the three families together provide diversity in pretraining corpora and instruction tuning. All models are loaded without quantization in each checkpoint’s native BF16 precision, and each model’s bundled tokenizer chat template is applied to all prompts. Identifiers, parameter counts, and cutoff evidence are listed in Appendix Table 3. The DeepSeek-Coder dates come from a maintainer response on the official repository (DeepSeek-AI, 2023), the CodeLlama dates from the Hugging Face model cards (Meta AI, 2023b,a), and Qwen2.5-Coder has no formally disclosed corpus cutoff, so its rows are marked before September 2024 (Hui et al., 2024). All inference is run on the Hábrók cluster at the University of Groningen with all random seeds fixed, and library versions are listed in Appendix A.

3.3 Code generation and confidence elicitation

Code generation is performed once per model and the outputs are frozen before any confidence elicitation, so the model cannot adjust its code for the confidence prompt and correctness labels are determined independently of any confidence estimation. All primary generations use greedy decoding, so every result is deterministic and exactly reproducible for a fixed model and prompt. Each model generates a single solution per problem, the pass@1 setting of Chen et al. (2021), matching the deployment case in which a developer receives one committed output and must decide whether to trust it. Full generation parameters are in Appendix A.

Confidence is then elicited in a separate inference pass, following Lin et al. (2022) and Xiong et al. (2024), with the same model that generated the code assessing its own output, a self-evaluation setup following Kadavath et al. (2022). The confidence prompt contains the full problem statement and the extracted code. Three methods are compared. Direct elicitation asks for the probability that the code passes all hidden tests, without explanation, as an integer between 0 and 100. Chain-of-Thought (CoT) elicitation first asks the model to analyze whether the code handles edge cases and whether the algorithm is sound, then to output

the same final confidence line, applying chain-of-thought prompting (Wei et al., 2022) to the confidence judgment. Linguistic elicitation asks for one of five ordered labels, *very low* to *very high*, mapped to 10, 30, 50, 70, 90, testing whether a coarser categorical signal mitigates the tendency of numeric confidence to cluster at round high values (Xiong et al., 2024; Groot and Valdenegro-Toro, 2024). Each strategy is a single fixed prompt, reproduced in full in Appendix B, so a finding such as “CoT underperforms Direct” is a finding about that particular prompt.

A line-based parser handles all three strategies. If the primary parse fails, a single repair call presents the original prompt alongside the malformed response and requests only the correctly formatted output, and if repair also fails the confidence defaults to 50, which retains every problem. A repair call may elicit a new judgment rather than only changing formatting, therefore we compare the complete pipeline with immediate fallback to a confidence of 50 on the same problems (Appendix N). Across the full primary pipeline, 5,441 outputs parse cleanly, 1,123 are repaired, and 6 default to 50. Per-cell rates are in Appendix C.2.

3.4 Stability analysis

A confidence score is only useful if it is consistent across repeated elicitation on the same code. For each (problem, generated code) pair, five independent confidence elicitations are run with temperature 0.3 and sampling enabled, using seeds $42 + i$ for $i \in \{0, \dots, 4\}$. If a run falls back to the default confidence of 50, that value is kept so the reported variability covers the full pipeline. The sample standard deviation across the five values is computed for each pair, and pairs are classified on the 0–100 scale as stable (below 5), marginal (between 5 and 10), or unstable (10 or above). These thresholds are heuristic and descriptive and are not deployment-safety boundaries. Aggregate statistics per cell are in Appendix J.

3.5 Calibration metrics

Four properties are evaluated after normalizing confidence to $[0, 1]$. Expected Calibration Error (ECE) measures the count-weighted absolute gap between confidence and empirical accuracy over ten equal-width bins; lower is better (Naeini et al., 2015; Guo et al., 2017). Net Calibration Error (NCE) is the signed mean difference between correctness and confidence, so negative values indicate over-

confidence and positive values underconfidence (Groot and Valdenegro-Toro, 2024). Area Under the ROC Curve (AUROC) measures whether confidence ranks correct solutions above incorrect ones independently of numerical calibration, with 0.5 indicating random discrimination (Fawcett, 2006; Hendrycks and Gimpel, 2017).

Selective prediction retains the programs with the highest confidence scores. The fraction retained is called coverage; measuring accuracy as this fraction varies gives an accuracy-coverage curve. Its area is AUACC (Geifman and El-Yaniv, 2017; Kamath et al., 2020). The AUACC gaps in Appendix M subtract overall pass@1 accuracy from this area. For comparisons across models with different accuracies, we additionally report the prediction rejection ratio (PRR). PRR measures the improvement in AUACC over random ordering as a fraction of the improvement possible under an ideal ordering that places every correct solution before every incorrect one (Vashurin et al., 2025). PRR is zero for random ordering and one for ideal ordering. Separately, when programs receive the same confidence score, we average over their possible orderings when calculating PRR, so that the result does not depend on their order in the saved data. The calculation of PRR and the effect of tie handling can be found in Appendix O. ECE, AUROC, and NCE use 1,000 bootstrap resamples for 95% percentile confidence intervals. Reliability diagrams and 20-bin histograms diagnose patterns hidden by scalar metrics; a cell is labelled collapsed when one histogram bin contains at least 50% of its scores. Platt scaling is used only as a post-hoc diagnostic. Equations and full procedural details are in Appendix A.1, with recalibration results in Appendix E.

4 Results

4.1 Accuracy

Because code outputs are frozen before confidence elicitation, pass@1 accuracy is identical across strategies. Strict accuracy ranges from 8.22% for CodeLlama-34B to 29.86% for Qwen2.5-Coder-32B; CodeLlama is the only family with an inversion, as its 7B model reaches 9.86%. Lenient scoring adds only 0.55-3.56 percentage points. The complete accuracy table and tolerance sweep are in Appendix F.

Table 1: Strict calibration metrics for all 18 model-strategy cells. ECE: lower is better; AUROC: higher is better; negative NCE indicates overconfidence. Confidence intervals are in Appendix D.

Model	Strategy	ECE	AUROC	NCE
Qwen-32B	Direct	0.327	0.834	-0.294
Qwen-32B	CoT	0.330	0.810	-0.324
Qwen-32B	Linguistic	0.216	0.841	-0.216
Qwen-7B	Direct	0.624	0.766	-0.619
Qwen-7B	CoT	0.506	0.772	-0.495
Qwen-7B	Linguistic	0.507	0.709	-0.505
DS-33B	Direct	0.183	0.760	-0.057
DS-33B	CoT	0.553	0.675	-0.536
DS-33B	Linguistic	0.258	0.638	-0.197
DS-6.7B	Direct	0.849	0.518	-0.849
DS-6.7B	CoT	0.779	0.716	-0.779
DS-6.7B	Linguistic	0.713	0.498	-0.713
CL-34B	Direct	0.813	0.655	-0.813
CL-34B	CoT	0.814	0.620	-0.814
CL-34B	Linguistic	0.660	0.681	-0.660
CL-7B	Direct	0.862	0.639	-0.862
CL-7B	CoT	0.680	0.556	-0.663
CL-7B	Linguistic	0.013	0.491	-0.012

4.2 Calibration metrics

ECE, AUROC, and NCE for all eighteen cells (3 families $\times$ 2 sizes $\times$ 3 strategies) are listed in Table 1. Reliability diagrams for all cells are in Appendix K, and full bootstrap intervals in Appendix D.

All eighteen strict-scoring NCE point estimates are negative, and seventeen of the corresponding 95% bootstrap intervals remain below zero. The exception is CodeLlama-7B with Linguistic elicitation, where the NCE is -0.012 with a 95% CI of $[-0.04, 0.02]$ that includes zero. The magnitude of the NCE differs sharply across cells, from -0.012 for CodeLlama-7B Linguistic to -0.862 for CodeLlama-7B Direct with a 95% CI of $[-0.89, -0.83]$. Individual confidence outputs can diverge sharply from correctness in both directions, as illustrated in Figure 1.

ECE and AUROC measure different properties and do not correlate across cells. The lowest ECE in the table is 0.013 for CodeLlama-7B Linguistic, whose AUROC is 0.491, while the highest AUROC is 0.841 for Qwen2.5-Coder-32B Linguistic, whose ECE is 0.216. Qwen2.5-Coder-32B is the only model with AUROC above 0.80 for all three strategies. The bootstrap intervals separate some strategy comparisons but not others. For Qwen2.5-Coder-32B, Linguistic (ECE 0.216, [0.18, 0.25]) lies clearly below both Direct (0.327, [0.28, 0.37]) and CoT (0.330, [0.29, 0.37]), whereas for

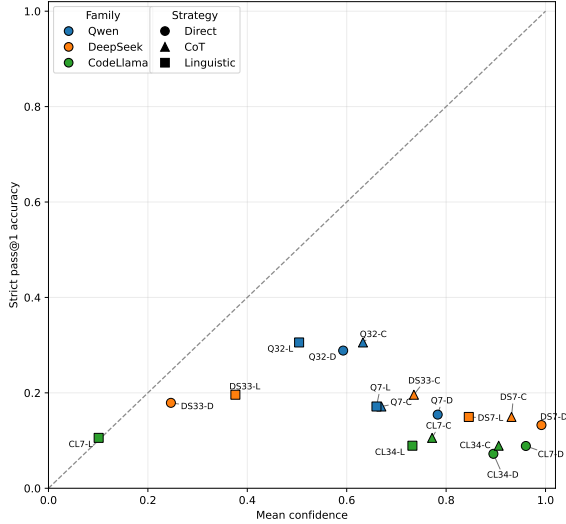


Figure 2: Mean elicited confidence versus strict pass@1 accuracy for all 18 model-strategy cells. Points below the dashed diagonal are overconfident. Color encodes model family; marker labels identify each combination.

Qwen2.5-Coder-7B the CoT and Linguistic estimates of 0.506 and 0.507 have overlapping intervals and are not meaningfully separated.

Figure 2 plots mean elicited confidence against strict pass@1 accuracy for all 18 cells. Almost all points sit where mean confidence exceeds accuracy. The three Qwen2.5-Coder-32B cells occupy the highest accuracy positions at approximately 0.30 and show lower mean confidence than the remaining cells, while several CodeLlama and DeepSeek-Coder-6.7B cells report mean confidence above 0.80 despite strict accuracy below 0.15. The reliability diagrams in Appendix K show the same split. Qwen2.5-Coder-32B spreads across the confidence range and sits mostly below the diagonal, whereas CodeLlama-7B Linguistic concentrates nearly all predictions at one value, so its diagram reduces to isolated points and its near-zero ECE reflects a constant confidence that happens to sit near the model’s empirical accuracy.

4.3 Strategy comparison

Linguistic elicitation has the lowest point-estimate ECE in four of six models, namely Qwen2.5-Coder-32B, DeepSeek-Coder-6.7B, CodeLlama-34B, and CodeLlama-7B. Three of these four lowest-ECE Linguistic cells are collapsed distributions, including CodeLlama-7B Linguistic (Section 4.6), so the low ECE there does not by itself indicate a useful confidence signal.

Table 2: Tie-averaged PRR for the saved repair pipeline. Zero represents random ordering; one represents ideal ordering, with every correct solution before every incorrect one.

Model	Direct	CoT	Ling.
Qwen-32B	0.633	0.525	0.705
Qwen-7B	0.428	0.413	0.257
DS-33B	0.474	0.239	0.291
DS-6.7B	0.016	0.257	-0.003
CL-34B	0.310	0.160	0.167
CL-7B	0.178	0.096	-0.029

4.4 Repair sensitivity and selective prediction

We compare the saved repair pipeline with an alternative that skips repair and assigns a confidence of 50 whenever the first response is unable to be successfully parsed. Both policies retain all the 365 problems for each model-strategy combination. For CodeLlama-7B CoT, ECE falls from 0.680 down to 0.445, while AUROC changes from 0.556 to 0.546; for DeepSeek-33B Direct, ECE rises from 0.183 up to 0.330. Following this, a lower calibration error under fallback confidence values does not mean better ranking. Eight combinations are unchanged because none of their responses required repair. The full results, along with confidence intervals, for the differences can be found in Appendix N. Importantly, any malformed original responses were overwritten with repair, this comparison is unable to establish if repair preserves the original judgment from the first response.

4.5 Within-family scaling

Mean ECE and AUROC averaged across the three strategies are visualized and tabulated in appendix H. For Qwen2.5-Coder, the 32B variant improves strict accuracy from 16.44% to 29.86%, reduces mean ECE from 0.546 to 0.291, and increases mean AUROC from 0.749 to 0.828 relative to the 7B variant. For DeepSeek-Coder, the 33B model improves the three quantities from 14.25% to 18.90%, from 0.780 to 0.331, and from 0.577 to 0.691 relative to the 6.7B model. CodeLlama does not follow this pattern. Its accuracy inverts, with CodeLlama-34B at 8.22% below CodeLlama-7B at 9.86%. Mean ECE is 0.518 for CodeLlama-7B and 0.762 for CodeLlama-34B, but excluding the degenerate CodeLlama-7B Linguistic cell raises the CodeLlama-7B mean ECE to 0.771. Mean AUROC is 0.652 for CodeLlama-34B and 0.562 for CodeLlama-7B.

4.6 Degenerate confidence distributions

A collapsed, or degenerate, confidence distribution is one in which the model outputs the same value for almost all problems, so the scores barely vary regardless of whether the underlying code is correct. A cell is flagged as collapsed when a single interval of the 20-bin confidence histogram contains at least 50% of all predictions. Eight of the eighteen cells meet this criterion, with histograms and per-cell statistics in Appendix I. CodeLlama-7B Linguistic places 359 of 365 predictions (98.4%) at confidence 10, producing an ECE of 0.013 and an AUROC of 0.491.

4.7 Stability

Five of the eighteen cells are unstable, with a mean sample standard deviation of at least 10 on the 0–100 scale across five temperature-0.3 runs. Three are CoT cells, namely Qwen2.5-Coder-7B (19.59), DeepSeek-Coder-33B (20.29), and CodeLlama-7B (17.32); the other two are DeepSeek-Coder-33B Linguistic (13.81) and CodeLlama-34B Linguistic (12.01). Every CoT cell has a higher mean standard deviation than the corresponding Direct cell for the same model, and in four of the six models CoT is the single most unstable strategy. The complete analysis is in Appendix J.

4.8 Selective prediction

A positive gap means that ordering outputs by confidence and reviewing the least confident first concentrates the failures better than a random ordering. Qwen2.5-Coder-32B has the largest gaps, with Linguistic at +0.264, Direct at +0.236, and CoT at +0.195, and Qwen2.5-Coder-7B and DeepSeek-Coder-33B are also positive for all three strategies. These gaps occur even though the same cells are overconfident in absolute terms, so a usable review ordering is present despite the miscalibration. Direct gives the highest gap in four of six models. Four cells fall below the random baseline, DeepSeek-Coder-6.7B Direct, DeepSeek-Coder-6.7B Linguistic, CodeLlama-34B Linguistic, and CodeLlama-7B Linguistic, all of which coincide with collapsed distributions. Direct produces one below-baseline gap, while Linguistic produces three.

We also report PRR to compare models with different accuracies. PRR expresses the gain in AUACC over random ordering as a fraction of the gain possible under ideal ordering. The AUACC

gaps reported above retain the saved-file order of programs with equal confidence scores; the PRR results average over possible orderings within each tied group. Table 2 shows that Direct achieves the highest PRR for four of the six models. Linguistic performs best for Qwen2.5-Coder-32B, and CoT performs best for DeepSeek-Coder-6.7B. PRR is negative only for DeepSeek-Coder-6.7B Linguistic (−0.003) and CodeLlama-7B Linguistic (−0.029). The four below-baseline results reported above thus apply to the original AUACC calculation. They do not all remain below baseline after averaging over tied orderings, a distinction examined separately in Appendix O.

5 Discussion

Verbalized confidence is not reliably calibrated for contamination-aware code generation. The variation across strategies and models is where the picture becomes more complicated, and eight of the eighteen cells have collapsed confidence distributions, so the scalar metrics cannot be read alone.

5.1 Systematic overconfidence

Every one of the eighteen strict-scoring NCE point estimates is negative, and seventeen of the corresponding intervals remain below zero, a systematic pattern of overconfidence. This extends the overconfidence trend documented for general NLP and vision-language tasks (Xiong et al., 2024; Groot and Valdenegro-Toro, 2024) and for verbalized confidence under distribution shift (Borszukovszki et al., 2025) into contamination-aware code generation, and it agrees with the finding that code models are poorly calibrated out of the box (Spiess et al., 2025). Restricting the window using documented cutoffs where available, and a transparent release-date proxy for Qwen2.5-Coder, reduces the risk that memorized benchmark problems inflate the observed accuracy. That cannot rule out contamination completely, but it shows the pattern persists under a temporally filtered evaluation setting (Jain et al., 2025; Li and Flanagan, 2024). The magnitude is far from uniform, and the two smallest values are misleading. The near-zero NCE of CodeLlama-7B Linguistic reflects a collapsed distribution rather than a useful signal, and DeepSeek-Coder-33B Direct is a different special case where a small NCE reflects a bimodal distribution rather than good calibration.

5.2 Calibration versus discrimination

ECE and AUROC do not co-vary across cells, and this separation is central to interpreting the results. The lowest ECE (0.013, CodeLlama-7B Linguistic) sits at an AUROC of 0.491, which is random discrimination, while the highest AUROC (0.841, Qwen2.5-Coder-32B Linguistic) sits at an ECE of 0.216. A confidence score can rank correct solutions above incorrect ones while remaining numerically overconfident, and a low ECE can carry no ranking signal at all, which is why the two properties were measured separately (Geng et al., 2024; Fawcett, 2006; Hendrycks and Gimpel, 2017). The practical consequence is visible in the AUACC gaps. Qwen2.5-Coder-7B and DeepSeek-Coder-33B show positive AUACC gaps for all three strategies despite being overconfident in absolute terms, so a developer who reviews the least confident outputs first will catch more failures earlier than a random sweep would. Confidence can remain useful for prioritizing manual review even when miscalibrated, the same observation that motivates selective prediction under distribution shift (Kamath et al., 2020; Geifman and El-Yaniv, 2017). The PRR results also support this finding when averaging over possible program orderings with equal confidence.

5.3 Behaviour of the elicitation strategies

The ranking of the three strategies varies across metrics, so the best choice depends on how confidence scores will be used. Linguistic has the lowest ECE in four of the six models, with scores closer to the empirical pass rate on average. Direct has the largest AUACC gap in four models and the highest PRR in four after averaging over possible orderings of tied scores. The number of below-baseline results also depends on how ties are handled. Keeping tied programs in their saved-file order gives one negative AUACC gap for Direct and three for Linguistic. Averaging over tied orderings leaves Direct with no negative PRR values and Linguistic with two. These findings favour Direct for several tested models, but they do not establish that it is consistently more reliable for reviewing programs in confidence order.

CoT does not consistently improve calibration, although it achieves the highest PRR for DeepSeek-Coder-6.7B. It is the least stable strategy overall, with three of the five unstable cells being CoT cells. Prompting the model to reason about code quality

```
Since the algorithm logic is incorrect, the
confidence score should reflect
that the code does not meet the problem's
requirements.
FINAL_CONFIDENCE: 0
```

Figure 3: Closing lines of the Qwen2.5-Coder-32B Chain-of-Thought reasoning trace on a solution that passes all hidden tests. The trace concludes that the logic is incorrect and outputs confidence 0. The full example is in Appendix P.

before stating a score does not consistently improve the confidence signal and adds variability relative to Direct. One example makes the failure concrete. Qwen2.5-Coder-32B under CoT correctly transcribed the required condition, then claimed the submitted code checked a different condition and concluded it was wrong, giving a final confidence of 0. The two conditions were logically identical and the code passed all hidden tests. Figure 3 shows the closing lines. CoT can improve task accuracy (Wei et al., 2022), but here it does not reliably improve the confidence judgment, consistent with prior reports that reasoning-style prompting has inconsistent effects on calibration (Xiong et al., 2024).

5.4 Stability, scaling, and metric limits

Stability matters for deployment in a way the calibration metrics alone do not capture. A confidence value that varies widely on repeated elicitation of the same code cannot be trusted as a fixed property of that code, because the number a developer sees depends on the run rather than on the program. CoT is the least stable strategy across the full procedure, including repair calls. The saved data cannot separate variability in the initial responses from variability introduced by repair.

The larger Qwen2.5-Coder and DeepSeek-Coder variants improve on all three metrics, consistent with findings that larger models can be better calibrated in suitably formatted settings (Kadavath et al., 2022). CodeLlama does not follow this pattern, and its apparent ECE improvement at 7B is an artifact of the degenerate CodeLlama-7B Linguistic cell, since removing that cell leaves the two scales essentially tied. These comparisons are descriptive rather than causal, because the variants may differ in pretraining data and instruction tuning, not just parameter count.

The degenerate cells expose a limit in the metrics. A low ECE can result from distribution collapse rather than from alignment between confi-

dence and accuracy, so it must be read alongside the histogram and AUROC. CodeLlama-7B Linguistic is the clearest example, its near-zero ECE of 0.013 arising because a nearly constant confidence value happens to sit close to the model’s empirical accuracy, while its AUROC of 0.491 shows the score carries almost no information about which outputs are correct (Naeini et al., 2015; Guo et al., 2017). DeepSeek-Coder-33B Direct is a different degeneracy, where a bimodal distribution lets overconfidence in the high bin and underconfidence in the low bin partially cancel, explaining an NCE of only -0.057 despite an ECE of 0.183, while the cell keeps useful ranking ability. No single number is enough. NCE shows the direction of miscalibration, ECE its average magnitude, AUROC whether the scores rank correct outputs above incorrect ones, and the histogram whether the distribution has collapsed.

Limitations

The evaluation is contamination-aware rather than contamination-free. Qwen2.5-Coder has no disclosed corpus cutoff, so its technical-report date is used as a reproducible proxy, and competitive-programming problems do not capture the full complexity of production software. Only six open-weight coding models are evaluated, since closed assistants do not expose stable versions and cutoff evidence needed for the shared temporal window. Correctness also depends on the hidden test suite, which may miss semantically wrong programs, while lenient scoring is meaningful only for single-number outputs.

The strategy comparison covers one fixed prompt and, for Linguistic, one fixed label mapping. The collapse and stability thresholds describe the observed distributions and variation across runs. They have not been validated as cutoffs for reliable confidence. Because any malformed original responses were not saved, we cannot compare them directly with their repaired versions. Comparing repair with an immediate default of 50 measures the effect of using repair, but does not show whether it preserves the original confidence judgment that a model gave. The stability measurements cover the full procedure, including any repair calls. In-sample Platt scaling is an optimistic bound rather than deployable performance, although five-fold results remain close for most cells (Appendix E). The findings concern frozen pass@1 outputs and do

not establish behaviour under repeated sampling, reranking, iterative refinement, or developer use. Future work should cover more programming languages and prompt variants, and examine how confidence differs across error categories. Newer or closed models could also be included where the available cutoff evidence allows. Comparing intrinsic and verbalized measures over the same evaluation window would help assess their differences. Testing confidence ordering in real code review would show whether it improves the review process.

6 Conclusion

Verbalized confidence is not reliably calibrated in this setting, since all eighteen strict-scoring NCE point estimates indicate overconfidence and seventeen of the eighteen bootstrap intervals remain below zero. Calibration also depends strongly on the model, the elicitation strategy, and the metric. Calibration quality and ranking ability do not co-vary, so a low ECE can coincide with random discrimination and a strong ranking signal can coincide with substantial overconfidence. Among strategies, Linguistic minimizes ECE most often, Direct gives the highest AUACC gap and the highest PRR in four of six models, and CoT is the least stable overall. Among models, the larger variants show better accuracy, calibration, and ranking within Qwen2.5-Coder and DeepSeek-Coder, while CodeLlama does not follow this pattern. The repair comparison also shows that lower calibration error does not necessarily imply better ranking.

The overall takeaway is that verbalized confidence in coding LLMs should be treated as a multi-dimensional signal rather than a single calibration score. It can still support selective review in some model-strategy combinations, but it should not be interpreted as a trustworthy pass probability unless calibration, ranking ability, distributional collapse, and stability have all been checked.

Data Availability

Retained confidence responses, confidence scores, correctness labels, and repair-status records are provided in the accompanying [Zenodo dataset](#) (Tuerlings and Valdenegro-Toro, 2026).

References

- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. [Program synthesis with large language models](#). *CoRR*, abs/2108.07732.
- Mirko Borsukovszki, Ivo Pascal De Jong, and Matias Valdenegro-Toro. 2025. [Know what you do not know: Verbalized uncertainty estimation robustness on corrupted images in vision-language models](#). In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pages 247–265, Albuquerque, New Mexico. Association for Computational Linguistics.
- Viola Campos, Robin Kuschner, and Adrian Ulges. 2026. [Multicalibration for llm-based code generation](#). In *Proceedings of the 1st International Workshop on AI for Software Quality Evaluation - Judgment, Metrics, Benchmarks, and Beyond*, AI-SQE '26, page 61–71, New York, NY, USA. Association for Computing Machinery.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *CoRR*, abs/2107.03374.
- DeepSeek-AI. 2023. [Cutoff dates](#). GitHub issue #89 in the DeepSeek-Coder repository. GitHub issue opened 30 December 2023. Accessed 5 May 2026.
- Tom Fawcett. 2006. [An introduction to ROC analysis](#). *Pattern Recognition Letters*, 27(8):861–874.
- Yonatan Geifman and Ran El-Yaniv. 2017. [Selective classification for deep neural networks](#). In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pages 4878–4887.
- Jiahui Geng, Fengyu Cai, Yuxia Wang, Heinz Koepl, Preslav Nakov, and Iryna Gurevych. 2024. [A survey of confidence estimation and calibration in large language models](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6577–6595, Mexico City, Mexico. Association for Computational Linguistics.
- Tobias Groot and Matias Valdenegro-Toro. 2024. [Overconfidence is key: Verbalized uncertainty evaluation in large language and vision-language models](#). In *Proceedings of the 4th Workshop on Trustworthy Natural Language Processing (TrustNLP 2024)*, pages 145–171, Mexico City, Mexico. Association for Computational Linguistics.
- 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*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.
- Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. [DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence](#). *CoRR*, abs/2401.14196.
- Dan Hendrycks and Kevin Gimpel. 2017. [A baseline for detecting misclassified and out-of-distribution examples in neural networks](#). In *The Fifth International Conference on Learning Representations (ICLR)*.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shangaoran Quan, and 5 others. 2024. [Qwen2.5-Coder technical report](#). *CoRR*, abs/2409.12186.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. [LiveCodeBench: Holistic and contamination free evaluation of large language models for code](#). In *The Thirteenth International Conference on Learning Representations (ICLR)*.
- Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, and 17 others. 2022. [Language models \(mostly\) know what they know](#). *CoRR*, abs/2207.05221.
- Amita Kamath, Robin Jia, and Percy Liang. 2020. [Selective question answering under domain shift](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5684–5696. Association for Computational Linguistics.
- Zoe Kotti, Konstantina Dritsa, Diomidis Spinellis, and Panos Louridas. 2026. [The fools are certain; the wise are doubtful: Exploring llm confidence in code completion](#). *Preprint*, arXiv:2508.16131.
- Changmao Li and Jeffrey Flanigan. 2024. [Task contamination: Language models may not be few-shot anymore](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18471–18480.
- Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. [Teaching models to express their uncertainty in words](#). *Transactions on Machine Learning Research*.
- Meta AI. 2023a. [CodeLlama-34b-Instruct-hf model card](#). Hugging Face model hub. Accessed 5 May 2026.

- Meta AI. 2023b. [CodeLlama-7b-Instruct-hf model card](#). Hugging Face model hub. Accessed 5 May 2026.
- 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*, pages 2901–2907.
- Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, J’er’emy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre D’efossez, and 7 others. 2023. [Code Llama: Open foundation models for code](#). *CoRR*, abs/2308.12950.
- Claudio Spiess, David Gros, Kunal Suresh Pai, Michael Pradel, Md Rafiqul Islam Rabin, Amin Alipour, Sumit Jha, Prem Devanbu, and Toufique Ahmed. 2025. [Calibration and correctness of language models for code](#). In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 540–552. IEEE Press.
- Stack Overflow. 2025. [Ai | 2025 stack overflow developer survey](#). Accessed 5 May 2026.
- Aditya Taparia, Ransalu Senanayake, Kowshik Thopalli, and Vivek Narayanaswamy. 2026. [The anatomy of uncertainty in llms](#). *Preprint*, arXiv:2603.24967.
- Hunter Tuerlings and Matias Valdenegro-Toro. 2026. [Verbalized uncertainty in coding llms: A contamination-aware confidence calibration dataset](#).
- Roman Vashurin, Ekaterina Fadeeva, Artem Vazhentsev, Lyudmila Rvanova, Daniil Vasilev, Akim Tsvigun, Sergey Petrakov, Rui Xing, Abdelrahman Sadallah, Kirill Grishchenkov, Alexander Panchenko, Timothy Baldwin, Preslav Nakov, Maxim Panov, and Artem Shelmanov. 2025. [Benchmarking uncertainty quantification methods for large language models with lm-polygraph](#). *Transactions of the Association for Computational Linguistics*, 13:220–248.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 24824–24837.
- Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2024. [Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Senrong Xu, Yuhao Tan, Yanke Zhou, Guangyuan Wu, Zenan Li, Yuan Yao, Taolue Chen, Feng Xu, and Xiaoxing Ma. 2026. [Uncertainty quantification for llm-based code generation](#). *Preprint*, arXiv:2605.12201.
- Zihuiwen Ye, Lukas Aichberger, Michael Kirchhof, Sinead Williamson, Luca Zappella, Yarin Gal, Arno Blaas, and Adam Golinski. 2026. [Uncertainty quantification for llm function-calling](#). *Preprint*, arXiv:2604.22985.
- Yuqi Zhu, Ge Li, Xue Jiang, Jia Li, Hong Mei, Zhi Jin, and Yihong Dong. 2025. [Uncertainty-guided chain-of-thought for code generation with llms](#). *Preprint*, arXiv:2503.15341.

A Evaluation Pipeline and Generation Parameters

Figure 4 summarizes the five stages of the evaluation pipeline described in Section 3. Table 4 lists the generation parameters used at each stage.

All inference was run on the Hábrók high-performance computing cluster at the University of Groningen using Transformers 4.57.6, PyTorch 2.10.0, Datasets 3.6.0, and Accelerate 1.13.0, with all random seeds fixed for reproducibility.

A.1 Metric Definitions

Let $y_i \in \{0, 1\}$ denote execution-based correctness and $\hat{c}_i \in [0, 1]$ confidence for $N = 365$ predictions. With ten equal-width bins B_m , Expected Calibration Error is

$$\text{ECE} = \sum_{m=1}^{10} \frac{|B_m|}{N} |\bar{c}(B_m) - \bar{a}(B_m)|, \quad (1)$$

where $\bar{c}(B_m)$ and $\bar{a}(B_m)$ are mean confidence and empirical accuracy. Net Calibration Error preserves direction,

$$\text{NCE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{c}_i), \quad (2)$$

so negative values indicate overconfidence. AUROC is the probability that a randomly chosen correct program receives higher confidence than a randomly chosen incorrect program. For selective prediction, predictions are sorted by confidence, accuracy is computed at each retained coverage, and trapezoidal integration yields AUACC; subtracting unconditional pass@1 accuracy gives the reported gap. ECE, NCE, and AUROC are additionally computed over 1,000 bootstrap resamples, using percentile 95% confidence intervals.

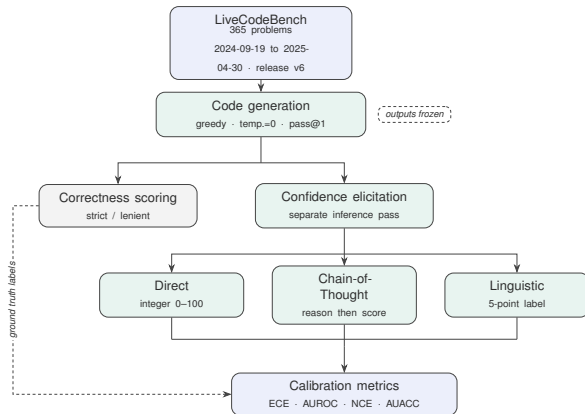


Figure 4: Overview of the evaluation pipeline. Code generation outputs are frozen before any confidence elicitation pass, ensuring correctness labels are independent of the confidence prompts that follow.

Table 4: Generation parameters by stage. Sampling is enabled whenever the temperature is greater than zero. No top- k or repetition penalty is applied. For the stability runs, “as primary” means each run reuses the max-token budget of the strategy it repeats. The repair pass uses the temperature of its enclosing run.

Stage	Temp.	Samp.	top- p	Max tok.	Seed
Code generation	0.0	off	1.0	1024	42
Direct (primary)	0.0	off	1.0	64	42
CoT (primary)	0.0	off	1.0	512	42
Linguistic (primary)	0.0	off	1.0	64	42
Stability (5 runs)	0.3	on	1.0	as prim.	42+ i
Repair (on failure)	run	match	1.0	64	42+ i +1

B Prompt Templates

This appendix lists the prompt templates used for the final experimental outputs. Code generation used provider-specific chat wrappers for Qwen2.5-Coder, CodeLlama, and DeepSeek-Coder. Confidence elicitation and repair used shared strategy prompts across models, passed through each model’s tokenizer chat template where available.

Table 5: Prompt templates used in the final pipeline.

Pipeline stage	Appendix location
Qwen2.5-Coder code generation	Appendix B.1.1
CodeLlama code generation	Appendix B.1.2
Shared code-generation user body	Appendix B.1.3
DeepSeek-Coder code generation	Appendix B.1.4
Direct confidence	Appendix B.2.1
Chain-of-Thought confidence	Appendix B.2.2
Linguistic confidence	Appendix B.2.3
Repair	Appendix C.1

B.1 Code Generation Prompts

B.1.1 Qwen2.5-Coder

The Qwen2.5-Coder code-generation provider used the following system message, passed through the tokenizer chat template with `add_generation_prompt=True`.

You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests. Return only the final code solution.

B.1.2 CodeLlama

The CodeLlama code-generation provider used the same system message as Qwen2.5-Coder, passed through the tokenizer chat template with `add_generation_prompt=True`.

Table 3: Models evaluated in this study, all loaded unquantized in native BF16 precision from Hugging Face. DeepSeek-Coder and CodeLlama have documented cutoff evidence; Qwen2.5-Coder has no formal disclosed corpus cutoff, so the evaluation window is set after the first public arXiv submission of its technical report (Sep. 2024).

Hugging Face Identifier	Family	Params	Cutoff
deepseek-ai/deepseek-coder-6.7b-instruct	DeepSeek-Coder	6.7B	Mar. 2023
deepseek-ai/deepseek-coder-33b-instruct	DeepSeek-Coder	33B	Mar. 2023
codellama/CodeLlama-7b-Instruct-hf	CodeLlama	7B	Jul. 2023
codellama/CodeLlama-34b-Instruct-hf	CodeLlama	34B	Jul. 2023
Qwen/Qwen2.5-Coder-7B-Instruct	Qwen2.5-Coder	7B	Before Sep. 2024
Qwen/Qwen2.5-Coder-32B-Instruct	Qwen2.5-Coder	32B	Before Sep. 2024

You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests. Return only the final code solution.

B.1.3 Shared Qwen2.5-Coder and CodeLlama User Body

For Qwen2.5-Coder and CodeLlama, the shared user body began with the following text.

You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests. You will NOT return anything except for the program.

Question:
{question_content}

When starter code was available, the following block was appended.

You will use the following starter code to write the solution to the problem and enclose your code within delimiters.

```
```python
{starter_code}
```
```

When the problem used standard input and output, the following block was appended instead.

Read the inputs from stdin solve the problem and write the answer to stdout (do not directly test on the sample inputs). Enclose your code within delimiters as follows.

```
```python
YOUR CODE HERE
```
```

B.1.4 DeepSeek-Coder

The DeepSeek-Coder provider used a hardcoded prompt format rather than a tokenizer chat template.

You are an AI programming assistant, utilizing the DeepSeek Coder model, developed by DeepSeek Company, and you answer questions related to computer science.

Instruction:

You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests. You will NOT return anything except for the program.

Question:
{question_content}

{starter_or_stdin_branch}

Response:

The {starter_or_stdin_branch} field used the same starter-code or standard-input branch shown in Appendix B.1.3.

B.2 Confidence Elicitation Prompts

B.2.1 Direct

Direct

System. You are a code correctness assessor.

User. Programming problem:

{question_content}

Submitted code:

{answer}

What is the probability (integer 0-100) that this exact code passes all hidden test cases?

Output EXACTLY one line in the form:

FINAL_CONFIDENCE: <integer 0-100>

Do not add any explanation.

B.2.2 Chain-of-Thought

Chain-of-Thought

System. You are a code correctness assessor. First, briefly analyse whether the submitted code handles edge cases correctly and whether the algorithm is sound. Then, on the final line, output EXACTLY: FINAL_CONFIDENCE: <integer 0-100>. Do not write anything after that final line.

User. Programming problem:

{question_content}

Submitted code:

{answer}

B.2.3 Linguistic

Linguistic

System. You are a code correctness assessor.

User. Programming problem:

{question_content}

Submitted code:

{answer}

How confident are you that this code passes all hidden test cases?

Output EXACTLY one line in the form:

FINAL_LABEL: <very low|low|moderate|high|very high>

Do not add any explanation.

Confidence prompts were passed through the tokenizer chat template as a system-user conversation with `add_generation_prompt=True`.

C Parsing and Repair

This appendix documents the repair step and the rate at which it was triggered. The repair prompt is given first, the per-cell parse and repair counts second, and representative malformed outputs third. Together these show that the models did not always follow the requested output format and that a single repair call recovered most of the failures.

C.1 Repair Prompt

If primary parsing failed, a single repair call was made. For Direct and CoT, the repair system message was constructed by appending the following instruction to the original system message.

Your previous reply was not in the required format. Restate ONLY the final confidence line, using the same underlying judgment. Output EXACTLY one line in the form: FINAL_CONFIDENCE: <integer 0-100>. Do not add any explanation.

For Linguistic, the repair system message instead requested the categorical output format.

Your previous reply was not in the required format. Restate ONLY the final label, using the same underlying judgment. Output EXACTLY one line in the form: FINAL_LABEL: <very low|low|moderate|high|very high>. Do not add any explanation.

The repair user message was shared across strategies.

Original task:
{original_user}

Your previous (malformed) reply:
{previous_response}

Provide the reformatted final line now.

C.2 Parse and Repair Rates

Table 6 shows the parse outcome for every (model, strategy) cell across all of the 365 problems. Across the full primary pipeline, 5,441 outputs are parsed cleanly, 1,123 are repaired successfully, and 6 default to confidence 50. The repair rate is not uniform across all the cells and is divided amongst them. For example, CodeLlama-7B CoT is the most extreme case, with only 55 of 365 outputs

(15.1%) parsing on the first attempt, and DeepSeek-Coder-6.7B CoT and CodeLlama-34B CoT also required repair on more than 40% of problems. CoT is the format the models follow least reliably, which is consistent with the longer reasoning trace that gives the model more room to drift away from the required line at the end.

Table 6: Parse and repair outcome counts per (model, strategy) cell, out of 365 problems. n_{clean} is the number parsed on the first attempt; n_{repaired} is the number corrected after a failed primary parse; n_{default} is the number that fell back to confidence 50 after both parse and repair failed.

| Model | Strategy | n_{clean} | n_{repaired} | n_{default} |
|---------------------|------------|--------------------|-----------------------|----------------------|
| Qwen2.5-Coder-32B | Direct | 365 | 0 | 0 |
| Qwen2.5-Coder-32B | CoT | 359 | 6 | 0 |
| Qwen2.5-Coder-32B | Linguistic | 365 | 0 | 0 |
| Qwen2.5-Coder-7B | Direct | 365 | 0 | 0 |
| Qwen2.5-Coder-7B | CoT | 269 | 96 | 0 |
| Qwen2.5-Coder-7B | Linguistic | 365 | 0 | 0 |
| DeepSeek-Coder-33B | Direct | 226 | 139 | 0 |
| DeepSeek-Coder-33B | CoT | 259 | 105 | 1 |
| DeepSeek-Coder-33B | Linguistic | 271 | 94 | 0 |
| DeepSeek-Coder-6.7B | Direct | 354 | 11 | 0 |
| DeepSeek-Coder-6.7B | CoT | 212 | 153 | 0 |
| DeepSeek-Coder-6.7B | Linguistic | 365 | 0 | 0 |
| CodeLlama-34B | Direct | 333 | 32 | 0 |
| CodeLlama-34B | CoT | 183 | 177 | 5 |
| CodeLlama-34B | Linguistic | 365 | 0 | 0 |
| CodeLlama-7B | Direct | 365 | 0 | 0 |
| CodeLlama-7B | CoT | 55 | 310 | 0 |
| CodeLlama-7B | Linguistic | 365 | 0 | 0 |

C.3 Examples of Parse Failures

The following constructed examples illustrate response formats rejected by the parser described in Section 3.3; they are not model outputs from the main experiment. Table 6 reports observed parsing outcomes. Below is an example of a common Chain-of-Thought failure, a reasoning trace that never emits a final FINAL_CONFIDENCE line at all.

The algorithm looks correct and handles the main cases, so it should pass most of the hidden tests. I am fairly confident in this solution.

A Direct failure typically states the score inside a sentence rather than on the required line, so there is no FINAL_CONFIDENCE marker and no standalone integer for the parser to accept.

I would estimate the probability at around 80 percent.

A Linguistic failure embeds the label in a sentence, which the parser ignores because a label is accepted

only as a FINAL_LABEL line or as a standalone label.

My confidence that this code passes all hidden tests is high.

In the evaluation pipeline, the repair call presented the original prompt and the malformed reply and asked only for the correctly formatted final line, which recovered the large majority of failures as shown in Table 6.

D Bootstrap Confidence Intervals

Table 7 reports 95% bootstrap confidence intervals for ECE, AUROC, and NCE across all 18 model-strategy cells under strict scoring.

E Cross-Validated Platt Scaling Diagnostics

In order to check whether the in-sample Platt scaling estimates overfit the binned ECE metric, the logistic transformations were recomputed under five-fold stratified cross-validation. For each model-strategy pair, a two-parameter logistic regression was fitted on 80% of observations and evaluated on the held-out 20%, which produced a fully out-of-fold probability vector across all 365 problems. Table 8 reports the resulting ECE alongside the in-sample baseline.

For fifteen of the eighteen cells, the absolute difference between in-sample and out-of-fold ECE is below 0.015. The small differences suggest limited in-sample optimism for this metric on the evaluated distribution, but do not establish performance on other distributions.

The near-zero ECE values under Platt scaling mostly belong to the collapsed cells discussed in Section 4.6. CodeLlama-34B under Direct elicitation, and DeepSeek-Coder-6.7B under Direct and Linguistic elicitation, generate confidence outputs that cluster almost entirely near the maximum value, leaving limited variation for the regression to use. Predictions close to the base rate can produce near-zero local error without demonstrating useful discrimination. This does not mean that the original confidence scores contain no discriminative information.

The largest shifts observed are DeepSeek-Coder-33B Linguistic (+0.027) and CodeLlama-7B Direct (−0.030). At $n = 365$, these differences may

Table 7: 95% bootstrap confidence intervals for ECE, AUROC, and NCE under strict scoring, computed from 1000 resamples with replacement. Each interval is the [2.5th, 97.5th] percentile of the bootstrap distribution. Point estimates repeat Table 1 for reference.

| Model | Strategy | ECE | | AUROC | | NCE | |
|---------------------|------------|-------|--------------|-------|--------------|--------|----------------|
| | | Point | 95% CI | Point | 95% CI | Point | 95% CI |
| Qwen2.5-Coder-32B | Direct | 0.327 | [0.28, 0.37] | 0.834 | [0.79, 0.88] | -0.294 | [-0.34, -0.25] |
| Qwen2.5-Coder-32B | CoT | 0.330 | [0.29, 0.37] | 0.810 | [0.76, 0.85] | -0.324 | [-0.37, -0.28] |
| Qwen2.5-Coder-32B | Linguistic | 0.216 | [0.18, 0.25] | 0.841 | [0.80, 0.88] | -0.216 | [-0.25, -0.18] |
| Qwen2.5-Coder-7B | Direct | 0.624 | [0.59, 0.66] | 0.766 | [0.71, 0.82] | -0.619 | [-0.66, -0.57] |
| Qwen2.5-Coder-7B | CoT | 0.506 | [0.46, 0.55] | 0.772 | [0.71, 0.83] | -0.495 | [-0.54, -0.45] |
| Qwen2.5-Coder-7B | Linguistic | 0.507 | [0.47, 0.55] | 0.709 | [0.66, 0.76] | -0.505 | [-0.55, -0.46] |
| DeepSeek-Coder-33B | Direct | 0.183 | [0.15, 0.22] | 0.760 | [0.70, 0.82] | -0.057 | [-0.10, -0.01] |
| DeepSeek-Coder-33B | CoT | 0.553 | [0.51, 0.59] | 0.675 | [0.61, 0.74] | -0.536 | [-0.58, -0.49] |
| DeepSeek-Coder-33B | Linguistic | 0.258 | [0.22, 0.30] | 0.638 | [0.56, 0.72] | -0.197 | [-0.24, -0.16] |
| DeepSeek-Coder-6.7B | Direct | 0.849 | [0.82, 0.89] | 0.518 | [0.51, 0.53] | -0.849 | [-0.88, -0.82] |
| DeepSeek-Coder-6.7B | CoT | 0.779 | [0.75, 0.81] | 0.716 | [0.66, 0.77] | -0.779 | [-0.81, -0.75] |
| DeepSeek-Coder-6.7B | Linguistic | 0.713 | [0.68, 0.75] | 0.498 | [0.44, 0.55] | -0.713 | [-0.75, -0.68] |
| CodeLlama-34B | Direct | 0.813 | [0.78, 0.84] | 0.655 | [0.58, 0.75] | -0.813 | [-0.84, -0.78] |
| CodeLlama-34B | CoT | 0.814 | [0.78, 0.84] | 0.620 | [0.50, 0.73] | -0.814 | [-0.84, -0.78] |
| CodeLlama-34B | Linguistic | 0.660 | [0.63, 0.69] | 0.681 | [0.64, 0.72] | -0.660 | [-0.69, -0.63] |
| CodeLlama-7B | Direct | 0.862 | [0.83, 0.89] | 0.639 | [0.55, 0.73] | -0.862 | [-0.89, -0.83] |
| CodeLlama-7B | CoT | 0.680 | [0.65, 0.71] | 0.556 | [0.45, 0.66] | -0.663 | [-0.70, -0.62] |
| CodeLlama-7B | Linguistic | 0.013 | [0.01, 0.05] | 0.491 | [0.48, 0.50] | -0.012 | [-0.04, 0.02] |

reflect sample-to-fold variability and changes in bin placement.

All of this taken together, the cross-validation results support reading scalar ECE alongside the confidence histogram and AUROC rather than just as a number by itself, consistent with the broader interpretive point that was outlined in Section 5.4.

F Leniency Threshold Sensitivity

The main text evaluates correctness under strict scoring, under which a solution is correct if and only if the hidden-test evaluator determines that it passes the test case(s). Because some LIVE-CODEBENCH problems request a single numeric value, a strict-failure solution that produces a numerically close answer may still be useful as a secondary diagnostic. This is what is defined as lenient scoring, so essentially giving the model some benefit of doubt in its answer if it’s close enough. This appendix first compares strict scoring with the primary lenient threshold of $\tau = 0.10$, then reports how the three primary calibration metrics change across relative numeric leniency thresholds from $\tau = 0.1\%$ to $\tau = 50\%$.

F.1 Strict vs. Lenient Correctness

Table 9 compares strict and primary-lenient ($\tau = 0.10$) metrics for all 18 (model, strategy) cells. Under lenient scoring, all strict passes remain correct. Some strict failures are additionally reclassified as correct, but only when both the model’s output and the expected answer are actual numbers, since there is nothing to compare if either side is

a string, an error, or a parse failure. When both sides are numbers, the model’s answer is accepted if it falls within 10% of the expected value, that is $|y_{\text{pred}} - y_{\text{true}}|/|y_{\text{true}}| \leq \tau$. When the expected answer is 0, a percentage tolerance is effectively meaningless, so a separate absolute check is used when $|y_{\text{true}}| < 10^{-6}$, accepting the prediction if $|y_{\text{pred}} - y_{\text{true}}| \leq 10^{-6}$. Anything that cannot be parsed as a number, including non-finite values and parse failures, remains a strict failure and stays in the analysis denominator.

F.2 Relative-Tolerance Sweep

Design

The sweep evaluates $\tau \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5\}$ for all six models and three confidence-elicitation strategies (Direct, CoT, Linguistic), giving 126 metric triples in addition to the Strict baseline that is used throughout the main text. Confidence values are loaded from the frozen merged JSON files that were stored after generation, so without re-running any model inference. The primary threshold used throughout the main text is $\tau = 0.10$ (10% relative tolerance).

Pass@1 accuracy

Table 10 reports mean accuracy (pass@1) under each threshold. Because accuracy is a property of the model outputs rather than the confidence distribution, all three strategies for a given model share the same accuracy at every threshold.

Table 8: Comparison of in-sample and five-fold cross-validated (CV) Platt scaling ECE across all eighteen model-strategy combinations under strict scoring ($N = 365$). Positive Δ values indicate in-sample optimism; negative values indicate that the full-sample estimator inflated ECE relative to the out-of-fold estimate. Cells with near-zero values reflect distribution collapse rather than calibration.

| Model | Strategy | Pos. Rate | In-Sample ECE | CV ECE | Δ (CV – In-Sample) |
|---------------------|------------|-----------|---------------|--------|---------------------------|
| Qwen2.5-Coder-32B | Direct | 0.299 | 0.0366 | 0.0396 | +0.0030 |
| Qwen2.5-Coder-32B | CoT | 0.299 | 0.0395 | 0.0430 | +0.0035 |
| Qwen2.5-Coder-32B | Linguistic | 0.299 | 0.0756 | 0.0854 | +0.0098 |
| Qwen2.5-Coder-7B | Direct | 0.164 | 0.0978 | 0.0738 | −0.0240 |
| Qwen2.5-Coder-7B | CoT | 0.164 | 0.0531 | 0.0559 | +0.0029 |
| Qwen2.5-Coder-7B | Linguistic | 0.164 | 0.0409 | 0.0407 | −0.0002 |
| DeepSeek-Coder-33B | Direct | 0.189 | 0.0123 | 0.0267 | +0.0144 |
| DeepSeek-Coder-33B | CoT | 0.189 | 0.0432 | 0.0449 | +0.0017 |
| DeepSeek-Coder-33B | Linguistic | 0.189 | 0.0150 | 0.0419 | +0.0268 |
| DeepSeek-Coder-6.7B | Direct | 0.142 | 0.0000 | 0.0000 | +0.0000 |
| DeepSeek-Coder-6.7B | CoT | 0.142 | 0.0167 | 0.0028 | −0.0139 |
| DeepSeek-Coder-6.7B | Linguistic | 0.142 | 0.0000 | 0.0002 | +0.0001 |
| CodeLlama-34B | Direct | 0.082 | 0.0000 | 0.0001 | +0.0000 |
| CodeLlama-34B | CoT | 0.082 | 0.0000 | 0.0001 | +0.0001 |
| CodeLlama-34B | Linguistic | 0.082 | 0.0273 | 0.0199 | −0.0074 |
| CodeLlama-7B | Direct | 0.099 | 0.0572 | 0.0273 | −0.0299 |
| CodeLlama-7B | CoT | 0.099 | 0.0000 | 0.0126 | +0.0126 |
| CodeLlama-7B | Linguistic | 0.099 | 0.0001 | 0.0000 | −0.0001 |

Table 9: Strict vs. lenient ($\tau = 0.10$) calibration metrics for all 18 cells.

| Model | Strategy | Strict | | | Lenient ($\tau = 10\%$) | | |
|---------------------|------------|--------|-------|--------|---------------------------|-------|--------|
| | | ECE | AUROC | NCE | ECE | AUROC | NCE |
| Qwen2.5-Coder-32B | Direct | 0.327 | 0.834 | −0.294 | 0.319 | 0.804 | −0.259 |
| Qwen2.5-Coder-32B | CoT | 0.330 | 0.810 | −0.324 | 0.299 | 0.782 | −0.288 |
| Qwen2.5-Coder-32B | Linguistic | 0.216 | 0.841 | −0.216 | 0.180 | 0.813 | −0.180 |
| Qwen2.5-Coder-7B | Direct | 0.624 | 0.766 | −0.619 | 0.599 | 0.763 | −0.594 |
| Qwen2.5-Coder-7B | CoT | 0.506 | 0.772 | −0.495 | 0.487 | 0.772 | −0.471 |
| Qwen2.5-Coder-7B | Linguistic | 0.507 | 0.709 | −0.505 | 0.482 | 0.716 | −0.481 |
| DeepSeek-Coder-33B | Direct | 0.183 | 0.760 | −0.057 | 0.197 | 0.736 | −0.038 |
| DeepSeek-Coder-33B | CoT | 0.553 | 0.675 | −0.536 | 0.533 | 0.674 | −0.517 |
| DeepSeek-Coder-33B | Linguistic | 0.258 | 0.638 | −0.197 | 0.244 | 0.628 | −0.178 |
| DeepSeek-Coder-6.7B | Direct | 0.849 | 0.518 | −0.849 | 0.830 | 0.518 | −0.830 |
| DeepSeek-Coder-6.7B | CoT | 0.779 | 0.716 | −0.779 | 0.760 | 0.724 | −0.760 |
| DeepSeek-Coder-6.7B | Linguistic | 0.713 | 0.498 | −0.713 | 0.694 | 0.499 | −0.694 |
| CodeLlama-34B | Direct | 0.813 | 0.655 | −0.813 | 0.807 | 0.645 | −0.807 |
| CodeLlama-34B | CoT | 0.814 | 0.620 | −0.814 | 0.808 | 0.587 | −0.808 |
| CodeLlama-34B | Linguistic | 0.660 | 0.681 | −0.660 | 0.655 | 0.684 | −0.655 |
| CodeLlama-7B | Direct | 0.862 | 0.639 | −0.862 | 0.854 | 0.621 | −0.854 |
| CodeLlama-7B | CoT | 0.680 | 0.556 | −0.663 | 0.671 | 0.557 | −0.655 |
| CodeLlama-7B | Linguistic | 0.013 | 0.491 | −0.012 | 0.021 | 0.491 | −0.004 |

Net Calibration Error

Table 11 reports $NCE = \text{mean}(\text{accuracy} - \text{confidence})$. Figure 5 shows all 18 NCE curves on a categorical axis; the $NCE = 0$ line separates overconfident (negative) from underconfident (positive) cells.

At $\tau = 0.10$, all 18 (model, strategy) cells remain at negative NCE, confirming systematic overconfidence under strict and mildly lenient scoring. Two cells first cross zero between $\tau = 0.10$ and $\tau = 0.25$: CodeLlama-7B Linguistic (NCE rises from -0.004 to $+0.021$) and DeepSeek-Coder-33B Direct (NCE rises from -0.038 to $+0.017$). No cell crosses zero at or below $\tau = 0.10$.

Expected Calibration Error

Table 12 reports ECE. For most cells, ECE decreases or remains stable as τ increases because promoting strict failures to correct narrows the confidence–accuracy gap. There are some exceptions, DeepSeek-Coder-33B Direct rises from 0.183 under Strict to 0.315 at $\tau = 0.5$, while CodeLlama-7B Linguistic increases as the threshold becomes more permissive. Qwen2.5-Coder-32B Direct shows non-monotonicity. For Qwen2.5-Coder-32B, Linguistic remains the lowest-ECE strategy at every tested threshold, while Direct and CoT remain close and change order. ECE values and some AUROC strategy rankings also shift at higher thresholds, so strict scoring remains the pri-

Table 10: Pass@1 accuracy under relative numeric leniency. Strict uses strict scoring; τ columns use lenient scoring. All strategies within a model share the same accuracy.

| Model | Strict | 0.1% | 0.5% | 1% | 5% | 10% | 25% | 50% |
|---------------------|--------|-------|-------|-------|-------|-------|-------|-------|
| Qwen2.5-Coder-32B | 0.299 | 0.310 | 0.312 | 0.312 | 0.321 | 0.334 | 0.389 | 0.460 |
| Qwen2.5-Coder-7B | 0.164 | 0.170 | 0.173 | 0.175 | 0.181 | 0.189 | 0.225 | 0.337 |
| DeepSeek-Coder-33B | 0.189 | 0.195 | 0.195 | 0.195 | 0.203 | 0.208 | 0.263 | 0.359 |
| DeepSeek-Coder-6.7B | 0.142 | 0.145 | 0.148 | 0.151 | 0.156 | 0.162 | 0.200 | 0.293 |
| CodeLlama-34B | 0.082 | 0.082 | 0.082 | 0.082 | 0.085 | 0.088 | 0.096 | 0.126 |
| CodeLlama-7B | 0.099 | 0.099 | 0.101 | 0.101 | 0.104 | 0.107 | 0.132 | 0.244 |

Table 11: NCE under relative numeric leniency. Negative values indicate overconfidence; the Strict column matches Table 1.

| Model | Strategy | Strict | 0.1% | 0.5% | 1% | 5% | 10% | 25% | 50% |
|---------------------|------------|--------|--------|--------|--------|--------|--------|--------|--------|
| Qwen2.5-Coder-32B | Direct | -0.294 | -0.283 | -0.281 | -0.281 | -0.272 | -0.259 | -0.204 | -0.133 |
| Qwen2.5-Coder-32B | CoT | -0.324 | -0.313 | -0.310 | -0.310 | -0.302 | -0.288 | -0.234 | -0.162 |
| Qwen2.5-Coder-32B | Linguistic | -0.216 | -0.205 | -0.202 | -0.202 | -0.194 | -0.180 | -0.125 | -0.054 |
| Qwen2.5-Coder-7B | Direct | -0.619 | -0.613 | -0.610 | -0.608 | -0.602 | -0.594 | -0.558 | -0.446 |
| Qwen2.5-Coder-7B | CoT | -0.495 | -0.490 | -0.487 | -0.484 | -0.479 | -0.471 | -0.435 | -0.323 |
| Qwen2.5-Coder-7B | Linguistic | -0.505 | -0.500 | -0.497 | -0.495 | -0.489 | -0.481 | -0.445 | -0.333 |
| DeepSeek-Coder-33B | Direct | -0.057 | -0.052 | -0.052 | -0.052 | -0.044 | -0.038 | +0.017 | +0.113 |
| DeepSeek-Coder-33B | CoT | -0.536 | -0.531 | -0.531 | -0.531 | -0.522 | -0.517 | -0.462 | -0.366 |
| DeepSeek-Coder-33B | Linguistic | -0.197 | -0.192 | -0.192 | -0.192 | -0.183 | -0.178 | -0.123 | -0.027 |
| DeepSeek-Coder-6.7B | Direct | -0.849 | -0.847 | -0.844 | -0.841 | -0.836 | -0.830 | -0.792 | -0.699 |
| DeepSeek-Coder-6.7B | CoT | -0.779 | -0.776 | -0.774 | -0.771 | -0.765 | -0.760 | -0.722 | -0.628 |
| DeepSeek-Coder-6.7B | Linguistic | -0.713 | -0.710 | -0.708 | -0.705 | -0.699 | -0.694 | -0.656 | -0.562 |
| CodeLlama-34B | Direct | -0.813 | -0.813 | -0.813 | -0.813 | -0.810 | -0.807 | -0.799 | -0.769 |
| CodeLlama-34B | CoT | -0.814 | -0.814 | -0.814 | -0.814 | -0.811 | -0.808 | -0.800 | -0.770 |
| CodeLlama-34B | Linguistic | -0.660 | -0.660 | -0.660 | -0.660 | -0.657 | -0.655 | -0.646 | -0.616 |
| CodeLlama-7B | Direct | -0.862 | -0.862 | -0.859 | -0.859 | -0.856 | -0.854 | -0.829 | -0.717 |
| CodeLlama-7B | CoT | -0.663 | -0.663 | -0.660 | -0.660 | -0.658 | -0.655 | -0.630 | -0.518 |
| CodeLlama-7B | Linguistic | -0.012 | -0.012 | -0.010 | -0.010 | -0.007 | -0.004 | +0.021 | +0.133 |

mary correctness definition throughout.

Area Under the ROC Curve

Table 13 shows AUROC; AUROC is stable or slightly decreasing with τ for most cells; promoting strict failures can reduce class separation. CodeLlama-34B CoT shows non-monotone AUROC across thresholds; AUROC for CodeLlama-7B Linguistic remains approximately constant, changing from 0.491 to 0.489 at $\tau = 0.50$, with a nearly constant confidence distribution. Strategy rankings are mostly preserved across thresholds, but some change at higher tolerances.

F.3 Interpretation

Overall, the evidence across all three metrics clearly supports treating $\tau = 0.10$ as the primary threshold for sensitivity analysis, while strict scoring remains the primary correctness definition in the main text. At $\tau = 0.10$, the qualitative findings from the main text are unchanged. All 18 cells remain overconfident under NCE, Linguistic remains among the better-performing ECE strategies for Qwen2.5-Coder-32B, and the Qwen2.5-Coder cells retain comparatively strong AUROC discrimination. Anything higher to be reported in the main text would come at the cost to the developer who

has to accept worse and worse answers. This was purely reported to see if the models can get close to the answers it needs, and if so, what that threshold would be.

G Strategy Ranking by ECE

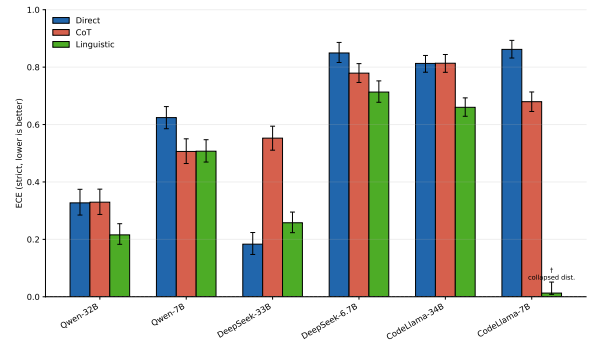


Figure 6: ECE under strict scoring per model grouped by elicitation strategy. Lower is better. Error bars show asymmetric 95% bootstrap confidence intervals. The near-zero CodeLlama-7B Linguistic bar reflects distribution collapse (Section 4.6).

Table 14 provides strategy rankings by strict ECE within each model.

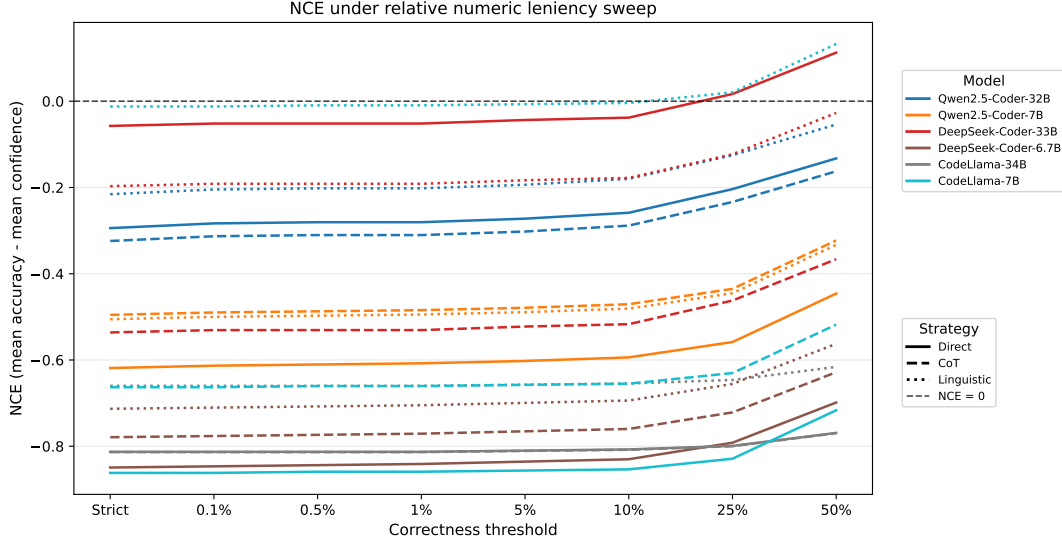


Figure 5: NCE for all 18 (model, strategy) cells across the leniency sweep. The leftmost position is strict. Subsequent positions are lenient correct scores at increasing τ . The dashed horizontal line marks NCE = 0. Line colour encodes model; linestyle encodes strategy (solid = Direct, dashed = CoT, dotted = Linguistic).

H Within-Family Scaling Statistics

Table 15 provides detailed scaling statistics for all six models. Mean ECE and mean AUROC are averaged over the three elicitation strategies under strict scoring. The CodeLlama-7B mean ECE of 0.518 is distorted by the degenerate Linguistic cell; excluding that cell raises the corrected mean to 0.771, as noted in Section 4.5 and Figure 7.

Table 15: Scaling comparison within each family under strict scoring. Mean ECE and AUROC are averaged over the three strategies. ‡ marks the CodeLlama accuracy inversion. CodeLlama-7B mean ECE excluding the degenerate Linguistic cell is 0.771.

| Family | Model | Acc. | Mean ECE | Mean AUROC |
|-----------|-------|--------|----------|------------|
| Qwen | 32B | 0.299 | 0.291 | 0.828 |
| Qwen | 7B | 0.164 | 0.546 | 0.749 |
| DeepSeek | 33B | 0.189 | 0.331 | 0.691 |
| DeepSeek | 6.7B | 0.142 | 0.780 | 0.577 |
| CodeLlama | 34B | 0.082‡ | 0.762 | 0.652 |
| CodeLlama | 7B | 0.099‡ | 0.518 | 0.562 |

I Collapsed Distribution Statistics

Table 16 provides distribution statistics for all eight collapsed model-strategy cells identified in Section 4.6.

J Full Stability Analysis

Table 17 reports mean sample standard deviation across five temperature-0.3 runs per cell for all 18 model-strategy combinations.

Table 17: Stability analysis across all 18 cells. Mean std dev is the average sample standard deviation of confidence over five temperature-0.3 runs on the 0–100 scale. Cells are classified as stable (mean std dev < 5), marginal ($5 \leq$ mean std dev < 10), or unstable (mean std dev $\geq$ 10). Unstable cells are shown in **bold**.

| Model | Strategy | Mean std dev | Category |
|---------------------|------------|--------------|-----------------|
| Qwen2.5-Coder-32B | Direct | 4.14 | stable |
| Qwen2.5-Coder-32B | CoT | 7.93 | marginal |
| Qwen2.5-Coder-32B | Linguistic | 1.67 | stable |
| Qwen2.5-Coder-7B | Direct | 8.96 | marginal |
| Qwen2.5-Coder-7B | CoT | 19.59 | unstable |
| Qwen2.5-Coder-7B | Linguistic | 8.78 | marginal |
| DeepSeek-Coder-33B | Direct | 9.89 | marginal |
| DeepSeek-Coder-33B | CoT | 20.29 | unstable |
| DeepSeek-Coder-33B | Linguistic | 13.81 | unstable |
| DeepSeek-Coder-6.7B | Direct | 2.64 | stable |
| DeepSeek-Coder-6.7B | CoT | 6.94 | marginal |
| DeepSeek-Coder-6.7B | Linguistic | 7.98 | marginal |
| CodeLlama-34B | Direct | 5.84 | marginal |
| CodeLlama-34B | CoT | 7.91 | marginal |
| CodeLlama-34B | Linguistic | 12.01 | unstable |
| CodeLlama-7B | Direct | 7.59 | marginal |
| CodeLlama-7B | CoT | 17.32 | unstable |
| CodeLlama-7B | Linguistic | 6.19 | marginal |

K Reliability Diagrams

Figure 11 shows reliability diagrams for all 18 model-strategy cells under strict scoring, arranged by model in rows and elicitation strategy in columns. Each panel plots mean elicited confidence (x-axis) against the empirical fraction correct (y-axis) for predictions falling in each of 10

Table 12: ECE under relative numeric leniency. The Strict column matches Table 1.

| Model | Strategy | Strict | 0.1% | 0.5% | 1% | 5% | 10% | 25% | 50% |
|---------------------|------------|--------|-------|-------|-------|-------|-------|-------|-------|
| Qwen2.5-Coder-32B | Direct | 0.327 | 0.316 | 0.313 | 0.313 | 0.316 | 0.319 | 0.313 | 0.297 |
| Qwen2.5-Coder-32B | CoT | 0.330 | 0.319 | 0.316 | 0.316 | 0.308 | 0.299 | 0.269 | 0.269 |
| Qwen2.5-Coder-32B | Linguistic | 0.216 | 0.205 | 0.202 | 0.202 | 0.194 | 0.180 | 0.125 | 0.073 |
| Qwen2.5-Coder-7B | Direct | 0.624 | 0.619 | 0.616 | 0.613 | 0.608 | 0.599 | 0.569 | 0.495 |
| Qwen2.5-Coder-7B | CoT | 0.506 | 0.501 | 0.498 | 0.495 | 0.490 | 0.487 | 0.457 | 0.392 |
| Qwen2.5-Coder-7B | Linguistic | 0.507 | 0.502 | 0.499 | 0.496 | 0.491 | 0.482 | 0.447 | 0.375 |
| DeepSeek-Coder-33B | Direct | 0.183 | 0.189 | 0.189 | 0.189 | 0.192 | 0.197 | 0.246 | 0.315 |
| DeepSeek-Coder-33B | CoT | 0.553 | 0.547 | 0.547 | 0.547 | 0.539 | 0.533 | 0.501 | 0.416 |
| DeepSeek-Coder-33B | Linguistic | 0.258 | 0.252 | 0.252 | 0.252 | 0.244 | 0.244 | 0.228 | 0.183 |
| DeepSeek-Coder-6.7B | Direct | 0.849 | 0.847 | 0.844 | 0.841 | 0.836 | 0.830 | 0.792 | 0.699 |
| DeepSeek-Coder-6.7B | CoT | 0.779 | 0.776 | 0.774 | 0.771 | 0.765 | 0.760 | 0.722 | 0.628 |
| DeepSeek-Coder-6.7B | Linguistic | 0.713 | 0.710 | 0.708 | 0.705 | 0.699 | 0.694 | 0.656 | 0.562 |
| CodeLlama-34B | Direct | 0.813 | 0.813 | 0.813 | 0.813 | 0.810 | 0.807 | 0.799 | 0.769 |
| CodeLlama-34B | CoT | 0.814 | 0.814 | 0.814 | 0.814 | 0.811 | 0.808 | 0.800 | 0.770 |
| CodeLlama-34B | Linguistic | 0.660 | 0.660 | 0.660 | 0.660 | 0.657 | 0.655 | 0.646 | 0.616 |
| CodeLlama-7B | Direct | 0.862 | 0.862 | 0.859 | 0.859 | 0.856 | 0.854 | 0.829 | 0.717 |
| CodeLlama-7B | CoT | 0.680 | 0.680 | 0.677 | 0.677 | 0.674 | 0.671 | 0.647 | 0.540 |
| CodeLlama-7B | Linguistic | 0.013 | 0.013 | 0.016 | 0.016 | 0.018 | 0.021 | 0.046 | 0.158 |

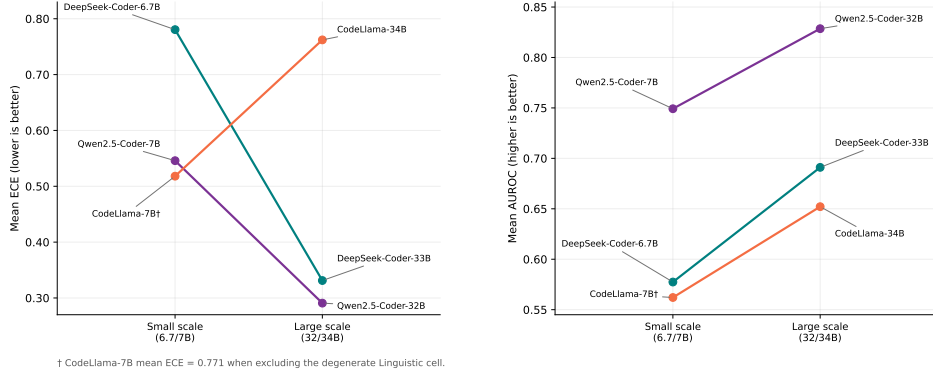


Figure 7: Mean ECE and mean AUROC averaged over three strategies, plotted against model scale within each family under strict scoring. Lines connect the small-scale and large-scale variants within each family. The CodeLlama ECE line is distorted by the degenerate CL-7B Linguistic cell (see Section 4.5).

equal-width confidence bins. The dashed diagonal represents perfect calibration; points below it indicate overconfidence.

L Confidence Score Distributions

Figure 12 displays the full confidence score distributions for all 18 model-strategy cells under strict scoring, arranged by model in rows and strategy in columns.

M Accuracy-Coverage Curves

Figure 10 shows accuracy-coverage curves for all 18 model-strategy cells under strict scoring.

N Repair Policy Ablation

The main pipeline makes one repair call after a failed primary parse, then defaults to a confidence of 50 if repair also fails. The alternative here skips

repair and instead immediately assigns 50. We use the saved parsing records to identify the responses from the models that were in need of repair. Across the 18 combinations, 1,129 responses needed repair, of which 1,123 yielded an output with a confidence score, while six received the fallback value of 50.

Both policies are evaluated on the same 365 generated programs for each model-strategy combination, with the same test outcomes. Responses whose confidence cannot be parsed are kept in the comparison, so any differences between the policies arise from the confidence scores assigned rather than from excluding programs. The responses that were malformed originally were overwritten by the repair output; thus, this comparison is unable to verify whether repair manages to preserve the original judgment.

Table 19 reports all cells. Table 20 gives confi-

Table 13: AUROC under relative numeric leniency. The Strict column matches Table 1.

| Model | Strategy | Strict | 0.1% | 0.5% | 1% | 5% | 10% | 25% | 50% |
|---------------------|------------|--------|-------|-------|-------|-------|-------|-------|-------|
| Qwen2.5-Coder-32B | Direct | 0.834 | 0.838 | 0.836 | 0.836 | 0.824 | 0.804 | 0.770 | 0.743 |
| Qwen2.5-Coder-32B | CoT | 0.810 | 0.810 | 0.809 | 0.809 | 0.802 | 0.782 | 0.759 | 0.730 |
| Qwen2.5-Coder-32B | Linguistic | 0.841 | 0.841 | 0.839 | 0.839 | 0.830 | 0.813 | 0.778 | 0.748 |
| Qwen2.5-Coder-7B | Direct | 0.766 | 0.770 | 0.765 | 0.771 | 0.775 | 0.763 | 0.749 | 0.647 |
| Qwen2.5-Coder-7B | CoT | 0.772 | 0.777 | 0.777 | 0.781 | 0.782 | 0.772 | 0.759 | 0.677 |
| Qwen2.5-Coder-7B | Linguistic | 0.709 | 0.713 | 0.714 | 0.716 | 0.719 | 0.716 | 0.698 | 0.603 |
| DeepSeek-Coder-33B | Direct | 0.760 | 0.750 | 0.750 | 0.750 | 0.745 | 0.736 | 0.673 | 0.624 |
| DeepSeek-Coder-33B | CoT | 0.675 | 0.679 | 0.679 | 0.679 | 0.677 | 0.674 | 0.636 | 0.591 |
| DeepSeek-Coder-33B | Linguistic | 0.638 | 0.641 | 0.641 | 0.641 | 0.639 | 0.628 | 0.571 | 0.546 |
| DeepSeek-Coder-6.7B | Direct | 0.518 | 0.518 | 0.518 | 0.518 | 0.518 | 0.518 | 0.519 | 0.515 |
| DeepSeek-Coder-6.7B | CoT | 0.716 | 0.719 | 0.721 | 0.723 | 0.728 | 0.724 | 0.674 | 0.669 |
| DeepSeek-Coder-6.7B | Linguistic | 0.498 | 0.500 | 0.489 | 0.491 | 0.495 | 0.499 | 0.518 | 0.525 |
| CodeLlama-34B | Direct | 0.655 | 0.655 | 0.655 | 0.655 | 0.649 | 0.645 | 0.631 | 0.621 |
| CodeLlama-34B | CoT | 0.620 | 0.620 | 0.620 | 0.620 | 0.603 | 0.587 | 0.599 | 0.602 |
| CodeLlama-34B | Linguistic | 0.681 | 0.681 | 0.681 | 0.681 | 0.683 | 0.684 | 0.672 | 0.634 |
| CodeLlama-7B | Direct | 0.639 | 0.639 | 0.624 | 0.624 | 0.615 | 0.621 | 0.611 | 0.554 |
| CodeLlama-7B | CoT | 0.556 | 0.556 | 0.555 | 0.555 | 0.546 | 0.557 | 0.551 | 0.573 |
| CodeLlama-7B | Linguistic | 0.491 | 0.491 | 0.491 | 0.491 | 0.491 | 0.491 | 0.491 | 0.489 |

Table 14: Strategy ranking by strict ECE within each model. Rankings are based on point estimates; near-ties should be interpreted with the bootstrap intervals in Appendix D. † marks cells with collapsed confidence distributions.

| Model | Best | Second | Worst |
|----------|--------------------------|---------------|---------------|
| Qwen-32B | Ling. (0.216) | Dir. (0.327) | CoT (0.330) |
| Qwen-7B | CoT/Ling.† (0.506/0.507) | – | Dir. (0.624) |
| DS-33B | Dir.† (0.183) | Ling. (0.258) | CoT (0.553) |
| DS-6.7B | Ling.† (0.713) | CoT (0.779) | Dir.† (0.849) |
| CL-34B | Ling.† (0.660) | Dir.† (0.813) | CoT (0.814) |
| CL-7B | Ling.† (0.013) | CoT (0.680) | Dir.† (0.862) |

dence intervals for the differences. We repeatedly sample problems with replacement, using the same sample for both policies to keep the comparisons paired. For each sample, we recalculate all metrics, including PRR and its random and ideal reference orderings. The fixed fallback can improve ECE simply by lowering overconfident scores. This improvement does not show that repair harmed an original judgment. Neither policy uniformly improves ranking.

O Prediction Rejection Ratio

For each model-strategy combination, we order programs from highest to lowest confidence and calculate accuracy among the top k programs. Coverage is the retained fraction k/N , where N is the total number of programs. Varying k from 1 to N gives an accuracy-coverage curve. We compare the area under this curve, A , with the areas under random ordering, A_r , and ideal (i.e., oracle) ordering, A_o . The ideal ordering places all correct programs before all incorrect programs. PRR expresses the improvement over random ordering as a fraction of the improvement possible under ideal ordering

(Vashurin et al., 2025):

$$\text{PRR} = \frac{A - A_r}{A_o - A_r} \quad (3)$$

We calculate all three areas by trapezoidal integration with the same coverage values k/N for $k = 1, \dots, N$. We exclude zero coverage because accuracy is undefined when no programs are retained. If p is the fraction of programs that pass all tests, random ordering has expected accuracy p at every retained fraction, so $A_r = p(N - 1)/N$. Ideal ordering has accuracy $\min(k, Np)/k$, since Np programs are correct. If a PRR of zero occurs, it corresponds to a random ordering, and a value of one corresponds to ideal ordering, along with potential negative values that indicate worse-than-random performance.

Furthermore, PRR normalization and the treatment of equal confidence scores are separate choices. Notably, confidence scores do not provide any ordering within a tied group, so we give every possible ordering within that group equal weight. Thus, suppose a tied group contains m programs, of which s are correct. The higher-

Table 16: Distribution statistics for all eight collapsed model-strategy cells. A distribution is classified as collapsed when a single 20-bin interval contains at least 50% of all predictions. The table reports the exact modal confidence value and the dominant 20-bin histogram interval separately; these coincide for all cells except CodeLlama-7B Direct. Confidence values are normalised to $[0, 1]$. AUROC is included to indicate whether the collapsed distribution retains discriminative ability.

| Model | Strategy | Exact modal value | | Dominant 20-bin | | AUROC |
|---------------------|------------|-------------------|--------------------------|-----------------|------------------------|-------|
| | | Value | $n_{\text{modal}} / 365$ | Interval | $n_{\text{bin}} / 365$ | |
| Qwen2.5-Coder-7B | Linguistic | 0.90 | 202 (55.3%) | [0.90, 0.95] | 202 (55.3%) | 0.709 |
| DeepSeek-Coder-33B | Direct | 0.00* | 275 (75.3%) | [0.00, 0.05] | 275 (75.3%) | 0.760 |
| DeepSeek-Coder-6.7B | Direct | 1.00 | 354 (97.0%) | [0.95, 1.00] | 354 (97.0%) | 0.518 |
| DeepSeek-Coder-6.7B | Linguistic | 0.90 | 305 (83.6%) | [0.90, 0.95] | 305 (83.6%) | 0.498 |
| CodeLlama-34B | Direct | 0.90 | 314 (86.0%) | [0.90, 0.95] | 314 (86.0%) | 0.655 |
| CodeLlama-34B | Linguistic | 0.90 | 232 (63.6%) | [0.90, 0.95] | 232 (63.6%) | 0.681 |
| CodeLlama-7B | Linguistic | 0.10 | 359 (98.4%) | [0.10, 0.15] | 359 (98.4%) | 0.491 |
| CodeLlama-7B | Direct | 1.00 | 155 (42.5%) | [0.95, 1.00] | 293 (80.3%) | 0.639 |

*DS-33B Direct is bimodal. A secondary mode at confidence 1.00 accounts for 89 predictions (24.4%); both modes are visible in Figure 8.

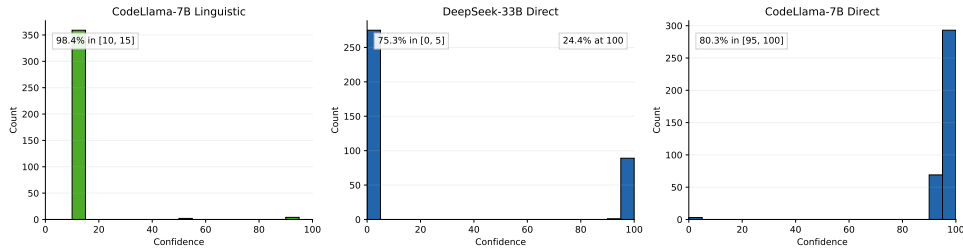


Figure 8: Confidence histograms for three extreme cells using 20 bins on the 0–100 confidence scale. CodeLlama-7B Linguistic concentrates 98.4% of predictions at confidence 10 (AUROC 0.491); DeepSeek-Coder-33B Direct is bimodal at 0 and 100 (AUROC 0.760); CodeLlama-7B Direct concentrates 80.3% of predictions in the top bin [95, 100], with 42.5% at the exact maximum value (AUROC 0.639).

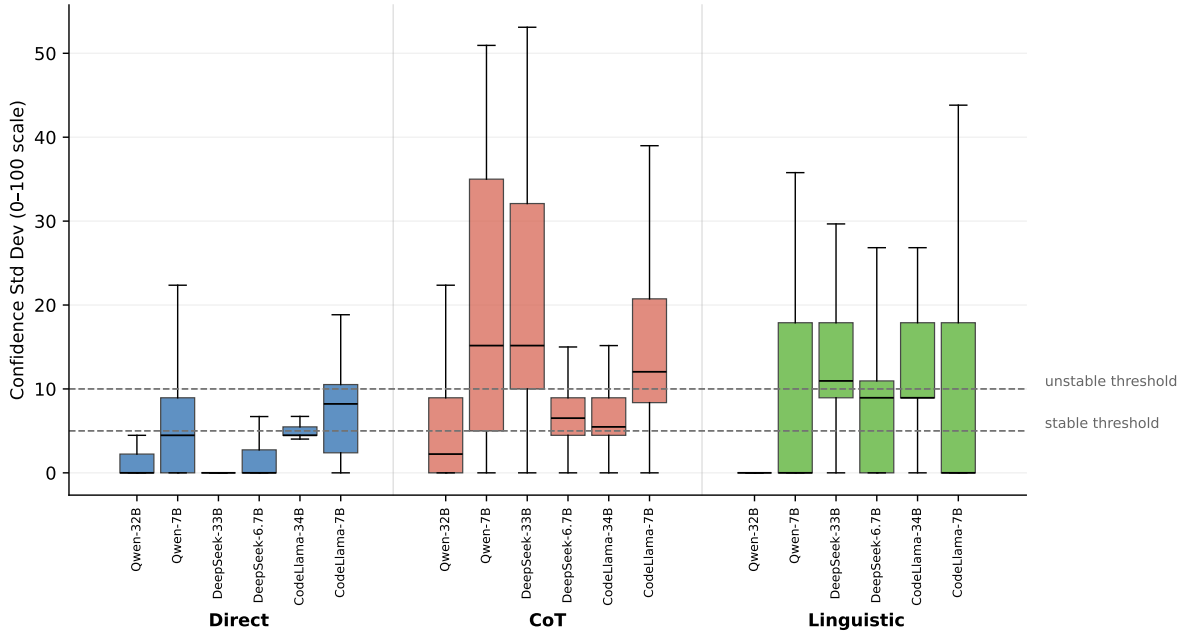


Figure 9: Distribution of per-problem confidence standard deviation across five temperature-0.3 runs, grouped by elicitation strategy across all six models. CoT cells show higher mean standard deviations than Direct cells for every model and than Linguistic cells for four of the six models.

confidence groups contain b programs, of which c is
are correct. Retaining the first j programs from the
tied group adds js/m correct programs on average.
For $j = 1, \dots, m$, the expected retained accuracy

$$\mathbb{E}[\text{accuracy}_{b+j}] = \frac{c + js/m}{b + j}. \quad (4)$$

Integrating the expected accuracies yields the
exact mean area across all possible orderings within

Table 18: AUACC gaps (area under the accuracy–coverage curve minus overall pass@1 accuracy) for all 18 model–strategy combinations under strict scoring. Programs with equal confidence retain their saved-file order; negative gaps are shown in **bold**. Appendix O reports PRR and examines the effect of averaging equal-score orderings. ‡ marks the CodeLlama accuracy inversion. Accuracy–coverage curves are in Appendix M.

| Model | Acc. (baseline) | Direct gap | CoT gap | Linguistic gap |
|---------------------|-----------------|---------------|---------|----------------|
| Qwen2.5-Coder-32B | 0.299 | +0.236 | +0.195 | +0.264 |
| Qwen2.5-Coder-7B | 0.164 | +0.130 | +0.116 | +0.087 |
| DeepSeek-Coder-33B | 0.189 | +0.145 | +0.072 | +0.090 |
| DeepSeek-Coder-6.7B | 0.142 | −0.026 | +0.044 | −0.024 |
| CodeLlama-34B | 0.082‡ | +0.044 | +0.007 | −0.004 |
| CodeLlama-7B | 0.099‡ | +0.036 | +0.015 | −0.025 |

Table 19: Full-set repair sensitivity ($N = 365$ per cell). R: saved repair pipeline; F: no repair, immediate fallback to 50 after an initial parse failure. PRR averages over possible orderings of programs with equal confidence. Accuracy is unchanged. The eight combinations without repair attempts are included.

| Model | Strategy | Repair attempts | ECE ↓ | | AUROC ↑ | | PRR ↑ | |
|----------|------------|-----------------|-------|-------|---------|-------|--------|--------|
| | | | R | F | R | F | R | F |
| Qwen-32B | Direct | 0 | 0.327 | 0.327 | 0.834 | 0.834 | 0.633 | 0.633 |
| Qwen-32B | CoT | 6 | 0.330 | 0.332 | 0.810 | 0.811 | 0.525 | 0.526 |
| Qwen-32B | Linguistic | 0 | 0.216 | 0.216 | 0.841 | 0.841 | 0.705 | 0.705 |
| Qwen-7B | Direct | 0 | 0.624 | 0.624 | 0.766 | 0.766 | 0.428 | 0.428 |
| Qwen-7B | CoT | 96 | 0.506 | 0.471 | 0.772 | 0.743 | 0.413 | 0.372 |
| Qwen-7B | Linguistic | 0 | 0.507 | 0.507 | 0.709 | 0.709 | 0.257 | 0.257 |
| DS-33B | Direct | 139 | 0.183 | 0.330 | 0.760 | 0.736 | 0.474 | 0.453 |
| DS-33B | CoT | 106 | 0.553 | 0.520 | 0.675 | 0.705 | 0.239 | 0.306 |
| DS-33B | Linguistic | 94 | 0.258 | 0.279 | 0.638 | 0.618 | 0.291 | 0.272 |
| DS-6.7B | Direct | 11 | 0.849 | 0.841 | 0.518 | 0.521 | 0.016 | 0.019 |
| DS-6.7B | CoT | 153 | 0.779 | 0.628 | 0.716 | 0.697 | 0.257 | 0.255 |
| DS-6.7B | Linguistic | 0 | 0.713 | 0.713 | 0.498 | 0.498 | -0.003 | -0.003 |
| CL-34B | Direct | 32 | 0.813 | 0.784 | 0.655 | 0.674 | 0.310 | 0.352 |
| CL-34B | CoT | 182 | 0.814 | 0.635 | 0.620 | 0.620 | 0.160 | 0.150 |
| CL-34B | Linguistic | 0 | 0.660 | 0.660 | 0.681 | 0.681 | 0.167 | 0.167 |
| CL-7B | Direct | 0 | 0.862 | 0.862 | 0.639 | 0.639 | 0.178 | 0.178 |
| CL-7B | CoT | 310 | 0.680 | 0.445 | 0.556 | 0.546 | 0.096 | 0.109 |
| CL-7B | Linguistic | 0 | 0.013 | 0.013 | 0.491 | 0.491 | -0.029 | -0.029 |

Table 20: Paired policy differences, F minus R, with 95% percentile intervals from 1,000 problem-level paired bootstrap resamples (seed 42). ECE below zero favors fallback; AUROC/PRR above zero favors fallback. Intervals are descriptive and unadjusted for multiple comparisons. Cells without attempted repair have exactly zero differences and are omitted.

| Model | Strategy | Δ ECE [95% CI] | Δ AUROC [95% CI] | Δ PRR [95% CI] |
|----------|------------|-------------------------|-------------------------|-------------------------|
| Qwen-32B | CoT | 0.003 [-0.002, 0.008] | 0.001 [-0.001, 0.005] | 0.001 [-0.001, 0.004] |
| Qwen-7B | CoT | -0.035 [-0.054, -0.015] | -0.030 [-0.069, 0.004] | -0.041 [-0.101, 0.007] |
| DS-33B | Direct | 0.147 [0.111, 0.178] | -0.024 [-0.056, 0.006] | -0.021 [-0.047, 0.005] |
| DS-33B | CoT | -0.032 [-0.054, -0.008] | 0.030 [-0.010, 0.065] | 0.066 [0.013, 0.121] |
| DS-33B | Linguistic | 0.021 [0.015, 0.027] | -0.020 [-0.030, -0.012] | -0.019 [-0.030, -0.011] |
| DS-6.7B | Direct | -0.008 [-0.015, -0.003] | 0.003 [0.000, 0.008] | 0.003 [0.000, 0.008] |
| DS-6.7B | CoT | -0.151 [-0.169, -0.133] | -0.020 [-0.053, 0.009] | -0.002 [-0.036, 0.029] |
| CL-34B | Direct | -0.029 [-0.042, -0.016] | 0.019 [-0.020, 0.048] | 0.042 [0.005, 0.077] |
| CL-34B | CoT | -0.179 [-0.200, -0.159] | -0.000 [-0.113, 0.110] | -0.010 [-0.121, 0.085] |
| CL-7B | CoT | -0.235 [-0.258, -0.213] | -0.011 [-0.115, 0.092] | 0.013 [-0.158, 0.178] |

tied groups. Table 21 compares PRR using saved-file order within ties with PRR averaged over all possible orderings within ties.

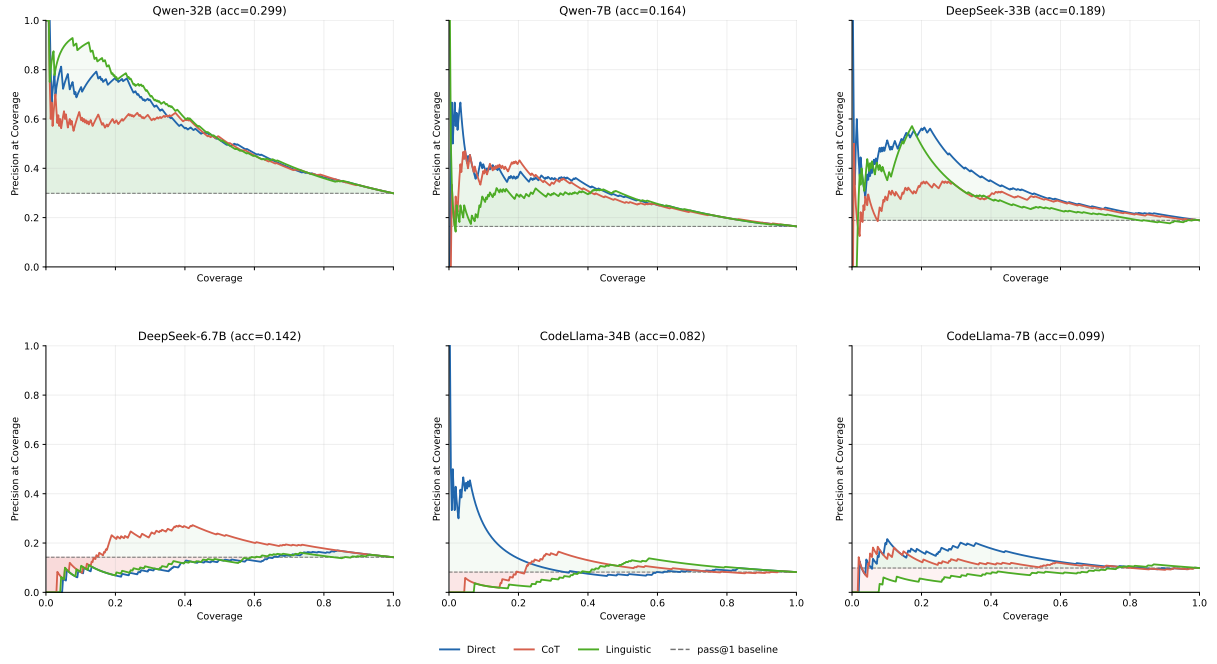


Figure 10: Accuracy-coverage curves for all 18 cells under strict scoring. Each panel shows one model; the three lines correspond to elicitation strategies. The horizontal dashed line marks the model’s pass@1 accuracy, which is the expected accuracy at coverage under random selection. A curve above this line indicates that confidence-ordered selection outperforms random selection at that coverage.

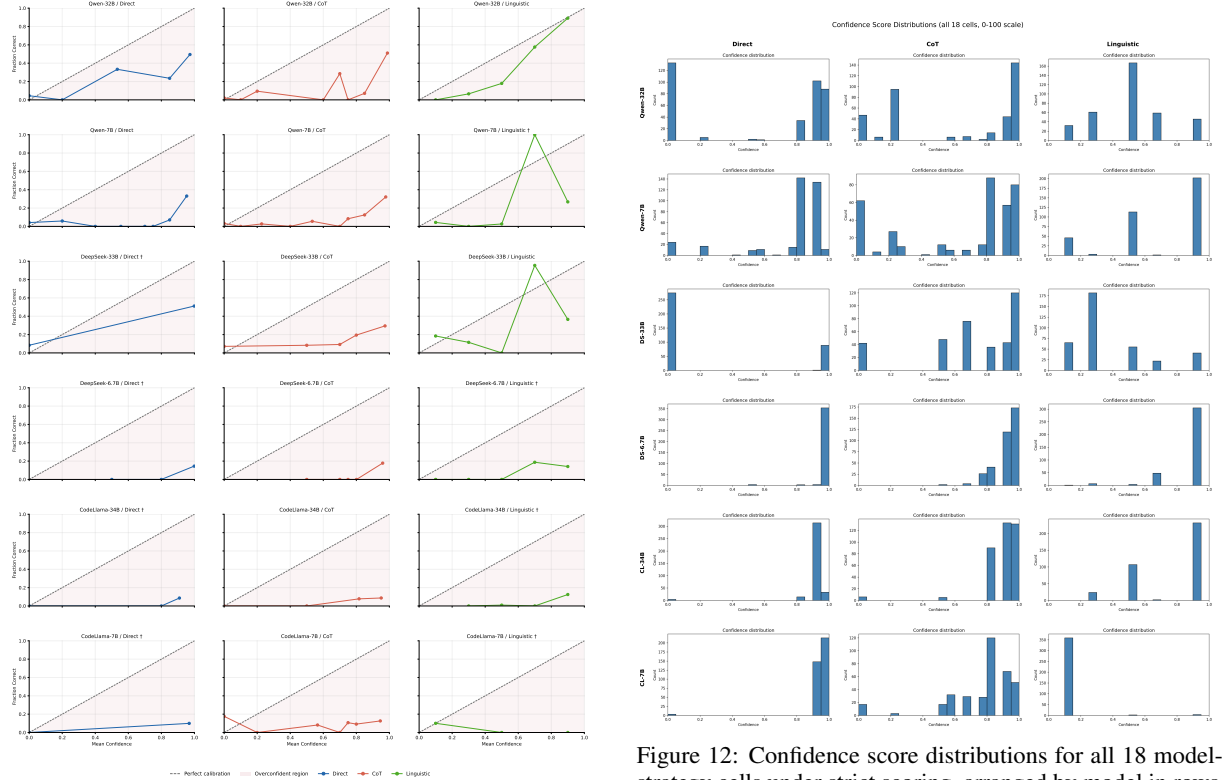


Figure 11: Reliability diagrams for all 18 model-strategy cells under strict scoring, arranged by model in rows and strategy in columns. The dashed diagonal represents perfect calibration. Points below the diagonal indicate overconfidence. Cells marked † have collapsed confidence distributions as described in Section 4.6.

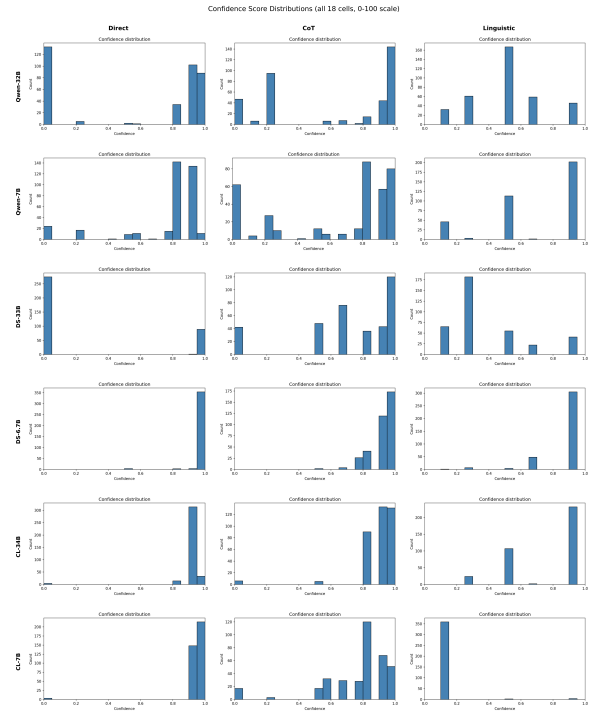


Figure 12: Confidence score distributions for all 18 model-strategy cells under strict scoring, arranged by model in rows (Qwen2.5-Coder-32B, Qwen2.5-Coder-7B, DeepSeek-Coder-33B, DeepSeek-Coder-6.7B, CodeLlama-34B, CodeLlama-7B from top to bottom) and strategy in columns (Direct, CoT, Linguistic from left to right). Each panel shows prediction counts across 20 equal-width bins on the 0–1 confidence scale.

Table 21: PRR tie sensitivity for the saved repair pipeline. File order keeps equally confident programs in their saved order; tie-averaged results average over all possible orderings within each equal-confidence group.

| Model | Strategy | File order | Tie-averaged |
|----------|------------|------------|--------------|
| Qwen-32B | Direct | 0.660 | 0.633 |
| Qwen-32B | CoT | 0.544 | 0.525 |
| Qwen-32B | Linguistic | 0.738 | 0.705 |
| Qwen-7B | Direct | 0.443 | 0.428 |
| Qwen-7B | CoT | 0.395 | 0.413 |
| Qwen-7B | Linguistic | 0.296 | 0.257 |
| DS-33B | Direct | 0.464 | 0.474 |
| DS-33B | CoT | 0.233 | 0.239 |
| DS-33B | Linguistic | 0.291 | 0.291 |
| DS-6.7B | Direct | -0.094 | 0.016 |
| DS-6.7B | CoT | 0.162 | 0.257 |
| DS-6.7B | Linguistic | -0.085 | -0.003 |
| CL-34B | Direct | 0.218 | 0.310 |
| CL-34B | CoT | 0.037 | 0.160 |
| CL-34B | Linguistic | -0.019 | 0.167 |
| CL-7B | Direct | 0.160 | 0.178 |
| CL-7B | CoT | 0.068 | 0.096 |
| CL-7B | Linguistic | -0.111 | -0.029 |

P Worked Examples of Confidence Elicitation

This appendix presents worked examples of verbalized confidence elicitation drawn from the per-problem output files. The examples are selected to illustrate the specific behaviors observed in the results.

Direct: The system message identifies the model as a code correctness assessor. The fixed instruction is:

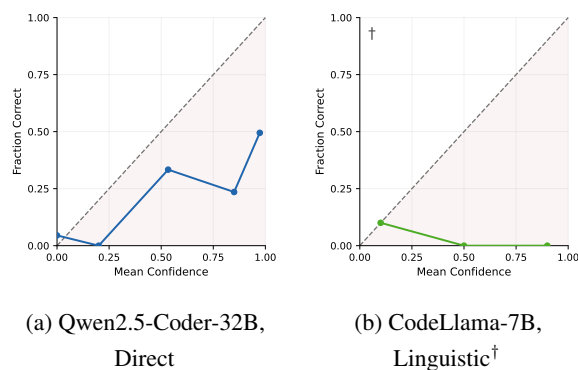


Figure 13: Reliability diagrams for two contrasting cells. The dashed diagonal is perfect calibration; the shaded region marks the overconfident half. Panel (a) spreads across the confidence range and sits mostly below the diagonal. Panel (b) is marked [†] for collapse and reduces to isolated points. All 18 diagrams are in Appendix K.

What is the probability (integer 0-100) that this exact code passes all hidden test cases?
Output EXACTLY one line in the form:
FINAL_CONFIDENCE: <integer 0-100>
Do not add any explanation.

Chain-of-Thought: The system message is:

You are a code correctness assessor. First, briefly analyse whether the submitted code handles edge cases correctly and whether the algorithm is sound. Then, on the final line, output EXACTLY: FINAL_CONFIDENCE: <integer 0-100>. Do not write anything after that final line.

Linguistic: The system message identifies the model as a code correctness assessor. The fixed instruction is:

How confident are you that this code passes all hidden test cases?
Output EXACTLY one line in the form:
FINAL_LABEL: <very low|low|moderate|high|very high>
Do not add any explanation.

Direct Failure Assigned 100% Confidence

Model and outcome: Qwen2.5-Coder-32B, Direct, question_id 3708. Parsed confidence: 100. Strict result: fail.

Problem excerpt. The model is asked to traverse a two-dimensional grid in a zigzag pattern while skipping every alternate cell. The traversal begins from the top-left cell, alternates direction between rows, and must return the visited values in order.

Generated code.

```
from typing import List

class Solution:
    def zigzagTraversal(self, grid:
        List[List[int]]) -> List[int]:
        result = []
        rows = len(grid)
        cols = len(grid[0])

        for i in range(rows):
            if i % 2 == 0:
                # Traverse right
                for j in range(0,
                    cols, 2):
                    result.append(
                        grid[i][j])
            else:
                # Traverse left
                for j in range(cols -
                    1, -1, -2):
                    result.append(
                        grid[i][j])

        return result
```

Raw confidence response.

FINAL_CONFIDENCE: 100

Interpretation: The model assigns complete confidence to a program that fails strict grading. For the available failing test, the expected output is [1, 3, 5, 7, 9] while the program returns [1, 3, 6, 4, 7, 9]. The row-local stride of two misaligns the global skip pattern when traversal direction reverses.

Direct Pass Assigned 0% Confidence

Model and outcome: Qwen2.5-Coder-32B, Direct, question_id 3832. Parsed confidence: 0. Strict result: pass.

Problem excerpt. The model is given an array of swap costs and must return the minimum total cost required to reach each position in a line. Swapping with a person in front has the listed cost, while later swaps with people behind are free.

Generated code.

```
from typing import List

class Solution:
    def minCosts(self, cost: List[int]
        ) -> List[int]:
        n = len(cost)
        answer = [float('inf')] * n
        answer[0] = cost[0]

        for i in range(1, n):
            answer[i] = cost[i]
            for j in range(i):
                answer[i] = min(
                    answer[i], answer[j])

        min_cost_so_far = float('inf')
        for i in range(n):
            min_cost_so_far = min(
                min_cost_so_far, cost[i])
            answer[i] =
                min_cost_so_far

        return answer
```

Raw confidence response.

FINAL_CONFIDENCE: 0

Interpretation: The model assigns zero confidence to a generated program that passes all strict hidden tests. Together with the previous example, this shows that extreme Direct scores can be wrong in both directions.

CoT Error on a Passing Program

Model and outcome: Qwen2.5-Coder-32B, Chain-of-Thought, question_id abc375_a. Parsed confidence: 0. Strict result: pass.

Problem excerpt. The task is to count patterns of three adjacent seats in which the two outer seats are occupied and the middle seat is unoccupied.

Generated code.

```
# YOUR CODE HERE
import sys

input = sys.stdin.read().split()
N = int(input[0])
S = input[1]

count = 0
for i in range(1, N - 1):
    if S[i] == '.' and S[i - 1] == '#' and S[i + 1] == '#':
        count += 1

print(count)
```

Raw confidence response.

```
The submitted code does not correctly solve the problem
as per the given requirements. The condition to
check should be that seats `i` and `i + 2` are
occupied (`#`), and seat `i + 1` is unoccupied
(`.`). However, the code checks if seat `i` is
unoccupied and seats `i - 1` and `i + 1` are
occupied, which is incorrect.
To fix this, the condition inside the loop should be
changed to:
```python
if S[i - 1] == '#' and S[i] == '.' and S[i + 1] == '#':
```
Additionally, the code handles edge cases correctly by
iterating from `1` to `N - 2`, ensuring that it
does not go out of bounds when checking `i + 1`
and `i - 1`.

Since the algorithm logic is incorrect, the confidence
score should reflect that the code does not meet
the problem's requirements.

FINAL_CONFIDENCE: 0
```

Interpretation: The generated program passes all strict hidden tests. The raw reasoning response nevertheless claims that the condition is wrong, then proposes the same condition with its terms reordered. This example makes the failure mode concrete. A reasoning trace can introduce an incorrect self-assessment rather than improve it, which goes against the expected intuition of reasoning actually helping an LLM.

Passing Submission

Model and outcome: CodeLlama-7B, Linguistic. This case is 3606. Linguistic label, which maps to confidence 10.

Problem excerpt. The model must replace each number in an integer array with the sum of its digits and return the minimum resulting value.

Generated code.

```
class Solution:
    def minElement(self, nums: List[
int]) -> int:
        for i in range(len(nums)):
            nums[i] = sum(int(digit)
for digit in str(nums[i]))
        return min(nums)
```

Raw confidence response.

FINAL_LABEL: very low

Interpretation: The identical label appears for a strict failure and a strict pass. Since 359 of 365 predictions in this cell have confidence 10, its AUROC of 0.491 reflects almost no ranking information.

Failing Submission

Model and outcome: CodeLlama-7B, Linguistic. This case is 3487. Linguistic label, which maps to confidence 10.

Problem excerpt. The model must determine the maximum number of removable characters from a source string while preserving a given pattern as a subsequence. Characters may be removed only at the provided target indices.

Generated code.

```
class Solution:
    def maxRemovals(self, source: str, pattern: str,
                    targetIndices: List[int]) -> int:
        # Initialize variables
        max_removals = 0
        current_removals = 0
        pattern_index = 0
        source_index = 0

        # Iterate through the target indices
        for target_index in targetIndices:
            # Check if the current character in the
            # source string matches the current character in
            # the pattern string
            if source[target_index] == pattern[
                pattern_index]:
                # Increment the pattern index
                pattern_index += 1

            # Check if the pattern index has reached
            # the end of the pattern string
            if pattern_index == len(pattern):
                # Increment the current removals
                current_removals += 1

                # Reset the pattern index
                pattern_index = 0

            else:
                # Check if the current removals is
                # greater than the maximum removals
                if current_removals > max_removals:
                    # Update the maximum removals
                    max_removals = current_removals

                # Reset the current removals
                current_removals = 0

        # Check if the current removals is greater than
        # the maximum removals
        if current_removals > max_removals:
            # Update the maximum removals
            max_removals = current_removals

        # Return the maximum removals
        return max_removals
```

Raw confidence response.

FINAL_LABEL: very low

Strict grading evidence. For the available failing test, the expected output is 1 while the program returns 0.