CodeClarity: A Framework and Benchmark for Evaluating Multilingual Code Summarization

Madhurima Chakraborty, Drishti Sharma, Eman Nissar, Maryam Sikander
Cohere Labs Community
mchak009@ucr.edu, drishtisharma96505@gmail.com,
emanisar3@gmail.com, maryamsikander66@gmail.com

Abstract

Large Language Models (LLMs) are increasingly used to summarize and document code, yet most research and training data remain limited to English. This creates barriers for developers working in other languages and leaves the multilingual capabilities of LLMs largely unexplored. We present CodeClarity, a framework for evaluating multilingual code summarization across six programming and six natural languages. It combines reference-based metrics, LLM-judge ratings, and faithfulness checks (identifiers and script) to capture surface similarity, semantic adequacy, and code-aware fidelity. Our experiments reveal that lexical metrics penalize morphologically rich languages, while judge-based evaluations provide more stable, semantically aligned assessments. This work establishes the first reproducible foundation for studying multilingual code summarization and points toward fairer, more inclusive evaluation of code intelligence systems. CodeClarity-Bench and the full evaluation pipeline are publicly available at huggingface.co/CodeClarity and github.com/MadhuNimmo/CodeClarity, enabling community-scale human validation and follow-up studies.

Keywords: Code understanding, Multilingual evaluation, LLMs, Multilingual code summarization, Software documentation

1. Introduction

Understanding source code remains a long-standing challenge as software systems grow larger and development teams become increasingly global. LLMs can now generate short natural language summaries of code, making code comprehension more accessible; however, most existing research and benchmarks still focus mostly on English. This narrow focus overlooks the diversity of the developer community where more than two-thirds of programmers are non-native English speakers (GitHub, 2022; JetBrains, 2024); leaving the multilingual capabilities of LLMs unexplored.

Programming languages themselves are English-centric, and most pretrained models inherit this bias from English-dominant documentation and comments. Consequently, models may produce fluent but semantically distorted summaries when generating in other languages. Further, current evaluation protocols rely mainly on surface-level overlap metrics whose reliability is debated even in monolingual settings (Haldar and Hockenmaier, 2024), offering limited insight into semantic fidelity or identifier preservation.

We introduce CodeClarity, the first framework for systematic evaluation of multilingual code summarization across multiple programming and natural languages. The framework unifies three complementary perspectives—reference-based metrics, reasoning-driven LLM-as-a-judge assessments, and code-aware fidelity diagnostics—to capture surface similarity, semantic adequacy, and identifier-level faithfulness. Alongside the framework, we release CodeClarity-Bench, a dataset of 7,344 multilingual summaries and an open-source evaluation pipeline to foster standardized, reproducible research on multilingual code understanding.

2. Evaluation Framework for Multilingual Code Summarization

CodeClarity is a framework for generating and evaluating multilingual code summaries using LLMs in two phases as outlined in Figure 1. First, we *generate* summaries across diverse programming and natural languages to create a controlled, reproducible multilingual benchmark. Then, we *evaluate* these summaries through complementary signals that capture surface similarity, semantic adequacy, and code–summary faithfulness. Next, we describe each component of the generation phase—dataset construction, prompt design, and model selection—followed by the evaluation design that integrates them.

2.1. Programming Languages Dataset

We use the CodeSearchNet validation split, which includes six programming languages—`Go`, `Java`, `JavaScript`, `PHP`, `Python`, and `Ruby`—representing imperative, functional, and object-oriented paradigms (Husain et al., 2019). This diversity allows us to assess whether

Proceedings of the Fifteenth Language Resources and Evaluation Conference (LREC 2026), pages 6439–6451
11–16 May 2026.

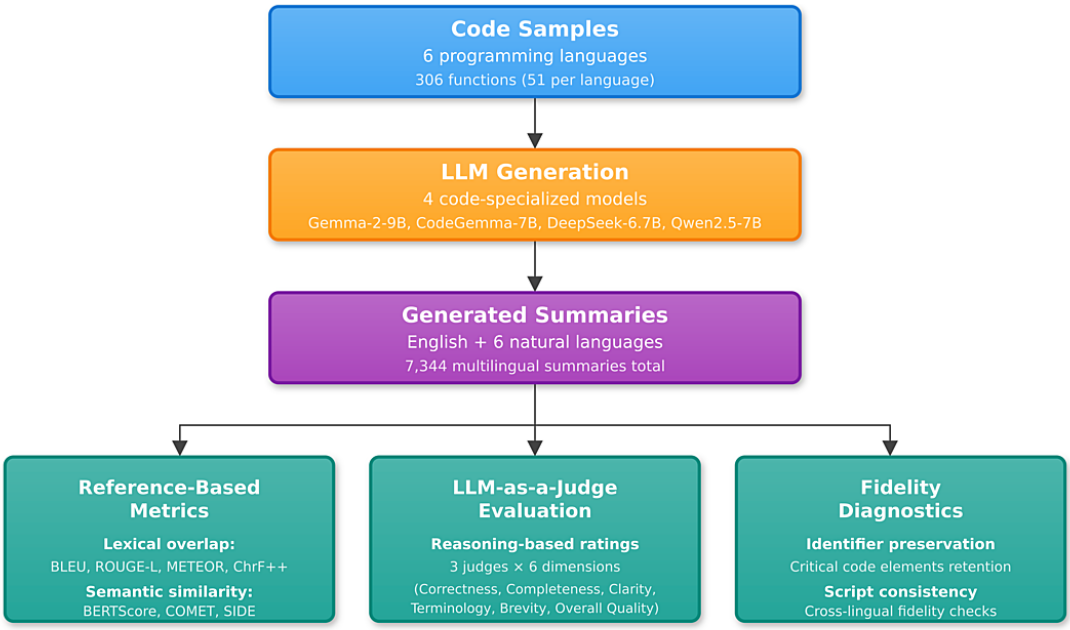

Figure 1: The CodeClarity Evaluation Framework.

LLMs can generalize across different syntactic conventions, indentation styles, and abstraction patterns inherent to each language.

We then applied a stratified sampling strategy to create the final dataset:

- **Bucketed by function length:** Short ($\leq$ 10 lines), medium (11–30 lines), and long ($>$ 30 lines);
- **Samples per bucket:** 17 samples per bucket yielding 51 functions per language;
- **Quality filters:** Removed empty or comment-only bodies;
- **Reproducibility:** Random seed was fixed to ensure consistent subsets across runs.

This yields $6 \times 51 = 306$ functions, which we summarize using four LLMs across six natural languages, producing **7,344 summaries** in total. This stratified design enables systematic analysis of how code length affects summarization quality across different languages and models.

2.2. Prompt Design

To generate summaries for these 306 functions, we designed a prompt that balances structural consistency with linguistic naturalness. The prompt instructs each LLM to produce a **natural language summary** of the input function by describing its **name**, **purpose**, **arguments**, and **core logic**, while allowing flexibility in phrasing. This light structure ensures that all summaries cover the same types of information across different languages, while still sounding natural. We also required the output to be in plain text without markdown, formatting, or headings, to avoid introducing stylistic biases in the evaluation process.

2.3. Natural Language Selection

We generate summaries in six target natural languages—Spanish, French, Portuguese (Brazil), Chinese, Hindi, and Arabic—chosen to balance linguistic diversity, prevalence among developer communities, and data availability. These languages span distinct scripts (Latin, Han, Devanagari, and Arabic) and exhibit varied morphological complexity, from analytic (Mandarin) to richly inflectional (Arabic, Hindi).

Spanish is spoken by over 460 million people, particularly across Latin America, yet remains underserved in technical documentation despite a strong developer community. Prior studies highlight its importance for multilingual programming and evaluation tasks (GitHub, 2022; Joshi and Martínez, 2020; García-Méndez et al., 2019). Mandarin Chinese, used by over nine million developers globally, poses unique challenges for idiomatic expression and word segmentation due to limited idiom-sensitive pretraining data (Cassano et al., 2024; Müller et al., 2021). Arabic is spoken by roughly 310 million users in the MENA region and introduces additional challenges due to right-to-left script and complex morphology (Attia, 2007; Hettiarachchi et al., 2025; Giagnorio et al., 2025). French serves a large developer base across Europe and Africa but often exhibits mixed-language code documentation (English–French), causing in-

consistency for multilingual models (Hettiarachchi et al., 2025; Müller et al., 2021). Portuguese represents a rapidly growing technology ecosystem (+18% year-over-year growth), motivating inclusion to support emerging developer communities and localized documentation growth (Gloroots, 2024). Finally, Hindi provides a valuable low-resource contrast with complex inflectional morphology and limited code-related corpora, testing LLM generalization in non-Latin scripts (Singh et al., 2024).

2.4. LLM Selection

Having established the target natural languages, we now select the LLMs used to generate summaries across these languages. Our model choices reflect three practical criteria: **(1)** multilingual coverage across our target languages, **(2)** strong instruction-following and code-reasoning capabilities, and **(3)** accessibility ($\leq 10B$ parameters) for reproducible research.

- Gemma-2-9B-IT : A multilingual, instruction-tuned model demonstrating high cross-lingual reasoning accuracy on XTREME-style tasks. Serves as a strong general baseline (Team et al., 2024).
- CodeGemma-7B-IT: A code-specialized variant within the Gemma family, enabling comparison between general and domain-focused models (CodeGemma Team, 2024).
- DeepSeek-Coder-6.7B-Instruct: A multilingual code model trained on English–Chinese corpora and fine-tuned for general code synthesis and explanation (Guo et al., 2024).
- Qwen2.5-Coder-7B-Instruct: A multilingual code model optimized for code understanding and generation. It shows strong performance on HumanEval and CodeXGLUE (Chen et al., 2021; Lu et al., 2021; Hui et al., 2024).

Our goal is not to find the single best model, but to build a reproducible and balanced evaluation pipeline. Therefore, we focus on publicly available models under 10B parameters that can run on modest hardware and support our target languages. We also include both general-purpose and code-specialized architectures to capture diverse model behaviors. This diversity prevents bias toward a specific model type and ensures that our analyses reflect framework-level insights rather than model-specific artifacts.

2.5. Evaluation Design

After generating summaries using four LLMs across six programming and natural languages, we assess their quality through three complementary components—*reference-based metrics*, *LLM-as-a-judge assessments*, and *fidelity diagnostics*. Together, these provide quantitative, reasoning-based, and code-aware perspectives on multilingual summarization quality.

Reference-based Metrics. We use seven complementary metrics grouped into three categories, each capturing a distinct aspect of summary quality. **(i) Lexical overlap.** BLEU (Papineni et al., 2002) measures n-gram precision, rewarding exact word matches between the model and human summaries. ROUGE-L (Lin, 2004) captures recall via the longest common subsequence, reflecting how much of the reference content is covered. METEOR (Banerjee and Lavie, 2005) extends these by recognizing stems and synonyms to give partial credit for meaningful variations, and CHRF++ (Popović, 2015) operates at the character level, handling morphological or spelling differences effectively. **(ii) Semantic similarity.** BERTSCORE (Zhang et al., 2020) compares contextual embeddings from a pretrained language model to assess semantic equivalence beyond surface wording. COMET (Rei et al., 2020) is a reference-based learned metric that encodes the source code, reference, and model summary to predict semantic adequacy and fluency. **(iii) Informativeness and brevity.** SIDE (Mastropaolo et al., 2024) (*Summary Information and Density Evaluation*) uses sentence embeddings to estimate how much key information from the reference is retained relative to length, balancing informativeness and conciseness. Although COMET, BERTSCORE, and SIDE rely on neural language models, they compute fixed similarity scores rather than engaging in reasoning or contextual judgment.

Since several of these metrics are English-centric (BLEU, ROUGE-L, METEOR, CHRF++, and SIDE), non-English summaries must first be translated to English for fair comparison. To ensure that translation noise does not bias evaluation, we benchmarked five multilingual back-translation systems—aya-expanse (Dang et al., 2024), llamax (Lu et al., 2024), gemmax2 (Cui et al., 2025), m2m (Fan et al., 2021), and towerinstruct (Alves et al., 2024)—representing diverse architectures (instruction-tuned generative, sequence-to-sequence, and large-scale multilingual translators). After empirical comparison, aya-expanse (Dang et al., 2024) was selected for all reported experiments because it consistently produced semantically faithful and stable translations across all language pairs.

LLM-as-a-judge Evaluation. To complement metric-based evaluation with reasoning-driven assessment, we employ three independent judges

from distinct model families: Command-A (Cohere Labs, 2025), GPT-5-nano (OpenAI, 2025), and Gemini-2-Flash-Lite (Google Cloud, 2025). Each LLM judge reads the source code, the human reference, and the generated summary, and rates the six dimensions—*correctness, completeness, clarity, terminology, brevity,* and *overall quality*—following a shared rubric. We assess reliability by analyzing inter-judge correlation and variance across natural languages.

Faithfulness Diagnostics. To evaluate the code–summary alignment beyond textual similarity, we compute two diagnostic indicators: **(a) identifier preservation**—the fraction of critical code identifiers (those appearing in both the code and reference summary) retained verbatim in the model output; and **(b) wrong-script leakage**—the proportion of tokens written in an unexpected script for the target language, excluding code spans. These diagnostics capture referential accuracy and linguistic consistency, highlighting fidelity aspects not reflected in standard metrics. Section 3 empirically validates these components and reports multilingual evaluation results.

3. Empirical analysis

We organize eight research questions into three modules that reflect a natural progression from *reliability* to *insight* to *action*:

- **Module A: Trustworthiness of Evaluation Signals** (**RQ1–RQ3**) – Validates the soundness of metrics, LLM judges and identifier/script fidelity.
- **Module B: Cross-Language & Complexity Variation** (**RQ4, RQ5**) – Examines how performance varies across programming and natural languages and by code length.
- **Module C: Generation Controls & Pathways** (**RQ6, RQ7**) – Quantifies controllable factors such as prompt design and pivoted generation.

Module A: Trustworthiness of Evaluation Signals

Module A evaluates the reliability of the three components introduced in Section 2.5. We analyze how reference-based metrics correlate (RQ1), how consistently LLM-as-a-judge scores align across languages (RQ2), and how faithfully summaries preserve identifiers and scripts (RQ3).

RQ1: How do reference-based metrics differ for multilingual code summarization?

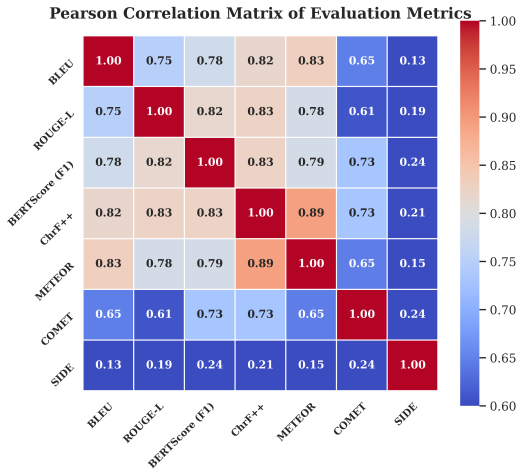

Figure 2: Correlation heatmap among seven automatic metrics.

Approach. To examine how the seven metrics introduced in Section 2.5 relate to one another, we computed a **Pearson correlation matrix** across all models and languages. Figure 2 shows the correlation matrix between the metrics over all evaluation samples, visualized as a heatmap where warmer colors indicate stronger positive association. This analysis reveals whether the metrics, which are intended to capture distinct quality aspects such as lexical overlap, semantic similarity, and informativeness, indeed behave independently in practice. For robustness, we also computed a **Spearman** (rank-based) correlation matrix, which showed a similar cluster structure.

Findings. Overlap-based metrics (BLEU, ROUGE-L, METEOR, CHRF++) correlate strongly ($r \in [0.75, 0.89]$), confirming their shared sensitivity to surface-level similarity. BERTSCORE aligns closely with this group ($r \approx 0.8$), reflecting its contextualized but still token-oriented comparison. COMET exhibits moderate correlation ($r \in [0.61, 0.73]$) with other metrics, capturing semantic adequacy even when phrasing diverges. SIDE correlates weakly with all others ($r \leq 0.25$), isolating its focus on information density and conciseness. Spearman correlations show the same pattern, confirming that these relationships extend from absolute scores to the relative ranking of samples.

Key Takeaway

Overlap-based metrics measure similar lexical patterns, while COMET and SIDE capture complementary semantic and informational aspects. No single metric fully reflects summary quality; combining diverse metrics provides a more balanced evaluation.

RQ2: How do LLM judges differ across languages, dimensions, and model families?

Approach. Three judges from distinct model families—Command-A (Cohere Labs, 2025), GPT-5-nano (OpenAI, 2025), and Gemini-2-Flash-Lite (Google Cloud, 2025)—scored all the summaries generated by four models in six natural languages. Each summary was rated on six dimensions—*correctness*, *completeness*, *clarity*, *terminology*, *brevity*, and *overall quality*—on a 1–5 scale. We computed (i) the average overall quality per language (averaged over models and judges), (ii) the standard deviation across judges to measure stability, and (iii) judge–judge Spearman correlations per dimension to evaluate ranking consistency. All LLM-as-a-judge evaluations were performed with deterministic decoding (temperature=0) using the same rubric across models. We do not report direct correlations with reference metrics because the signals target different dimensions—surface fidelity vs. holistic reasoning. Human judgment remains the most reliable arbiter of alignment between these dimensions, which we leave for future validation.

Findings. Across languages, we observe clear stratification in both average quality and evaluation stability. Romance languages (Portuguese, Spanish, French) and Chinese achieved the highest *mean overall quality scores* (4.5–4.6 on the 1–5 scale), computed as the average LLM-as-a-judge rating of overall quality across all models and samples in each language, followed by Hindi and Arabic with lower means (3.7–3.9). Inter-judge variability, measured as the standard deviation of scores across the three judges for the same samples was correspondingly low for Romance languages and Chinese (0.23–0.25) but higher for Hindi and Arabic (0.35–0.40). Pairwise correlations among judges (Figure 3) indicate high overall agreement in ranking behavior ($\rho = 0.72$–0.96). Command-A and Gemini-2-Flash-Lite agree most closely (median $\rho \geq 0.9$), while GPT-5-nano diverges slightly, indicating mild variation in scoring behavior. In terms of evaluation dimensions, judges are most consistent on *clarity* and *completeness* (mean $\rho \approx 0.9$) and least consistent on *terminology* and *brevity* (mean $\rho \approx 0.75$), where differences in linguistic style and conciseness preferences lead to greater scoring variability.

Overall, these results suggest that multilingual LLM-as-a-judge evaluations are both consistent and sensitive: high-resource languages yield stable, high-agreement judgments, while lower-resource ones show increased disagreement, amplifying disparities in evaluation reliability.

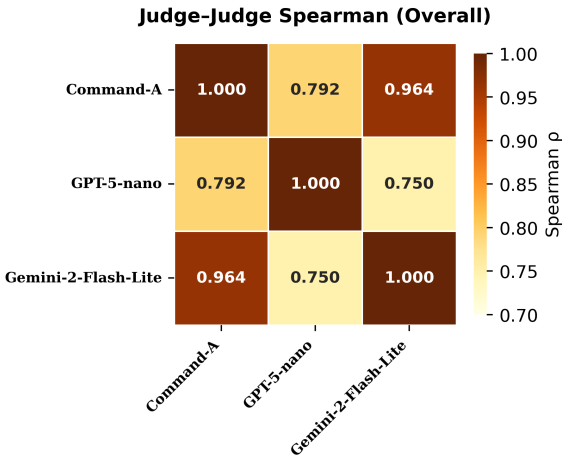

Figure 3: Judge–judge Spearman correlations.

Key Takeaway

LLM judges reach strong consensus overall, though agreement weakens for lower-resource languages and stylistically variable dimensions.

RQ3: How robustly do models preserve critical code elements (APIs, identifiers) when summarizing across languages?

Approach. A summary that reads fluently but omits or mistranslates key code elements—such as API names, variables, or class identifiers—can be misleading or incomplete. This issue is especially important in multilingual settings, where models must not only convey meaning but also preserve technical details consistently across scripts and languages. RQ3 therefore examines how reliably models retain these identifiers and whether losses correlate with broader semantic inaccuracies.

We assessed faithfulness using two automatic indicators and a manual audit. First, we computed **identifier preservation**, defined as the fraction of critical identifiers—those appearing in both the code and the model-generated English summary (used as the baseline reference) that were reproduced verbatim in the multilingual summary. This captures whether essential program entities are preserved across languages. Second, we measured **script consistency**. For each non-English summary, we automatically detected the Unicode script of every token and flagged words not matching the expected script for that target language, excluding identifiers and code spans. While this method is heuristic, it reliably identifies mixed-script cases such as untranslated English tokens in otherwise Arabic or Hindi text. Manual spot checks confirmed that these automatically flagged instances

aligned with visually apparent script mixing. Finally, we manually reviewed 3 random samples per programming–natural language pair to examine whether identifier loss coincided with semantic drift or hallucinated functionality.

Findings. Identifier preservation remained inconsistent across languages. On average, fewer than two-thirds of critical identifiers appearing in the English baseline summary were retained verbatim in the multilingual outputs, with the lowest retention observed for Hindi and Arabic. These models frequently replaced meaningful names with generic terms or partial transliterations, leading to summaries that felt plausible but were less informative.

Script inconsistency was also common: roughly 40–45% of non-English summaries contained tokens written in incorrect scripts, typically stray English words embedded in non-Latin text. This pattern was most pronounced in CodeGemma-7B-IT and Qwen2.5-Coder-7B-Instruct, suggesting incomplete target-language adaptation.

Manual inspection confirmed that identifier omissions often accompanied semantic inaccuracies. For instance, summaries that dropped or altered function names tended to simplify or misrepresent program behavior. Gemma-2-9B-IT showed the strongest overall fidelity, maintaining identifiers and semantics across most languages. By contrast, CodeGemma-7B-IT and DeepSeek-Coder-6.7B-Instruct often produced grammatically well-formed but semantically inaccurate summaries. Among target languages, Chinese, French, Spanish, and Portuguese remained most stable, while Hindi and Arabic continued to pose the greatest challenges.

Key Takeaway

Multilingual models frequently lose or alter key identifiers and mix scripts, especially in low-resource languages.

Module B: Cross-Language & Complexity Variation

With evaluation reliability established, we next examine how summarization quality differs across programming–natural language combinations and with increasing code length.

RQ4: How does summarization quality vary across combinations of programming languages and natural languages?

Approach. We compared summarization quality across all combinations of programming and natural languages—(`Python`, `Java`, `Go`, `Ruby`, `JavaScript`, `PHP`) paired with (Arabic, Chinese, French, Hindi, Portuguese, Spanish). Each pair was evaluated using six metrics (BLEU, ROUGE-L, BERTSCORE, CHRF++, METEOR, COMET), and scores were averaged across models to reveal systematic patterns.

Findings. Summarization quality varies along both linguistic axes. High-resource natural languages such as Spanish, Portuguese, and French consistently yield stronger scores across all metrics, followed by Chinese, while Hindi and Arabic lag behind. These differences likely stem from disparities in multilingual model training data: high-resource European languages benefit from better representation and tokenization, whereas morphologically rich or low-resource languages remain underrepresented.

Programming language choice also matters. `Python` and `Java` consistently produce higher-quality summaries, reflecting their extensive, well-documented open-source ecosystems and standardized naming practices. By contrast, languages such as `Go`, `Ruby`, `JavaScript`, and `PHP` show slightly weaker results, which may be attributed to more variable or informal documentation styles that reduce code–text alignment during pretraining.

Across metrics, these patterns remain stable: overlap-based measures (BLEU, ROUGE-L, METEOR, CHRF++) and embedding-based ones (BERTSCORE, COMET) produce consistent rankings. While lexical metrics are more sensitive to tokenization and morphology, semantic metrics smooth over such surface variation yet still reflect the same relative performance hierarchy.

Key Takeaway

High-resource natural languages paired with well-documented programming languages yield the strongest summaries. Lower-resource or morphologically complex languages underperform across all code languages.

RQ5: How does code length affect the quality of multilingual summarization?

Approach. To examine the effect of code length on summarization quality, we stratify code samples by number of lines as a proxy for complexity. Each programming language was stratified into three buckets—*short* (≤ 10 lines), *medium* (11–30), and *long* (> 30). We report (i) average reference-based metrics per bucket (Figure 4) and (ii) average LLM-as-a-judge scores per bucket (Figure 5) (pooling all judges), averaging across all summary models, programming languages, and target natural languages.

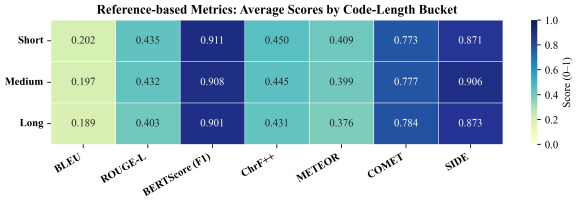

Figure 4: Reference-based metrics averaged by code-length bucket.

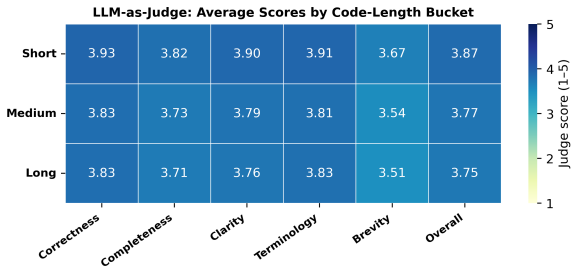

Figure 5: LLM-as-a-judge scores averaged by code-length bucket.

Findings. LLM-as-a-judge evaluations show that summary quality drops noticeably when moving from short to medium functions and then stabilizes for longer code. Short functions are self-contained and easy for models to summarize precisely, leading to high ratings across all dimensions. Medium functions, however, introduce more branching logic and parameter interactions, which models often compress or omit, reducing perceived clarity and completeness. Beyond this point, for longer code, models tend to default to higher-level abstractions that focus on the main purpose rather than line-by-line detail, causing scores to stabilize at a lower level. In contrast, reference-based metrics exhibit the opposite pattern: they remain stable between short and medium code but fall sharply for long functions, where lexical and structural divergence from the gold reference increases. Learned metrics such as BERTSCORE, COMET, and SIDE stay largely steady across all buckets, suggesting that they are more resilient to surface variation and better capture meaning at a coarse level of abstraction.

Key Takeaway

Surface-based metrics over-penalize longer code, while judge evaluations and learned metrics remain steady, capturing meaning beyond lexical overlap.

Module C: Generation Controls & Pathways

Finally, we study controllable factors affecting multilingual performance—specifically prompt structure and direct versus pivoted generation pathways.

RQ6: Does prompt design influence the quality and consistency of multilingual code summaries?

Approach. We compared three prompt styles—semi-rigid (prescribes content elements with flexible phrasing), rigid (enforces fixed structure with enumerated requirements), and relaxed (requests only natural description)—across code samples spanning all six programming languages for each model. A semi-automated script flagged anomalies (wrong language, mixed scripts, formatting issues), which we manually verified.

Findings. Prompt design significantly influences multilingual quality, with effects varying by model and language.

- **Semi-rigid prompts** proved most robust, producing stable summaries across all models and languages while minimizing garbled text and language leakage—even in weaker languages (Arabic, Hindi), most outputs remained usable.
- **Rigid prompts** successfully enforced structure for strong models (Gemma-2-9B-IT, Qwen2.5-Coder-7B-Instruct) but broke for weaker ones: CodeGemma-7B-IT often failed to output Arabic summaries, and DeepSeek-Coder-6.7B-Instruct occasionally reverted to English headings (e.g., "## Hindi Code Summary").
- **Relaxed prompts** generated broad coverage but produced verbose outputs, occasional English-only summaries, and garbled or mixed-script text in Arabic and Hindi.

Key Takeaway

Semi-rigid prompts offer the best balance: rigid ones work only for strong models but break for low-resource languages, while relaxed ones produce inconsistent quality.

RQ7: How does the quality of multilingual summaries differ between direct and pivoted generation pathways?

Approach. When producing summaries in multiple natural languages, it is not obvious whether generating them directly in the target language is always best. Many multilingual models are more fluent and semantically grounded in English, suggesting that generating an English summary first and then translating it might lead to higher-quality results.

To examine whether adding such an intermediate English step affects summary quality across languages, we compared two generation pathways for each model: (1) **Direct generation** (`code` → `target language`), and (2) **Pivoted generation** (`code` → `English` → `target language`). For both pathways, the same model was used to generate summaries. In the pivoted setup, the English summary was subsequently translated into the target language using a single translation model (aya-expanse) with fixed decoding settings. This design isolates the effect of the additional translation step by holding the summarization model and decoding parameters constant across both pathways.

Reference-based metrics such as BLEU, ROUGE-L, or COMET are not suitable for this setting because they rely on a gold-standard reference summary. In our case, the English summary itself serves as the gold reference, and the pivoted versions are direct translations of it. Evaluating these translations against the same reference would therefore conflate summarization with translation fidelity, offering little insight into the quality of the two-step pathway. To avoid this issue, we rely on LLM-as-a-judge evaluations, where the judge is given the code and the English summary and asked to determine which target-language summary (direct or pivoted) is more fluent, faithful, and informative.

Findings. As shown in Figure 6, the impact of pivoting depends strongly on language resources. For high-resource languages such as Chinese, French, Spanish, and Portuguese, direct and pivoted generations yield almost identical quality, with mean differences below 0.2 on the 1–5 scale. In contrast, low-resource languages—Arabic and Hindi—benefit substantially from pivoting, improving mean LLM-judge scores by ≈ 1 point. This suggests that English intermediates can mitigate fluency and completeness issues in under-represented languages.

Key Takeaway

Generating summaries through an English intermediate step substantially improves fluency and adequacy for lower-resource languages (Hindi, Arabic) while offering limited benefit for high-resource ones.

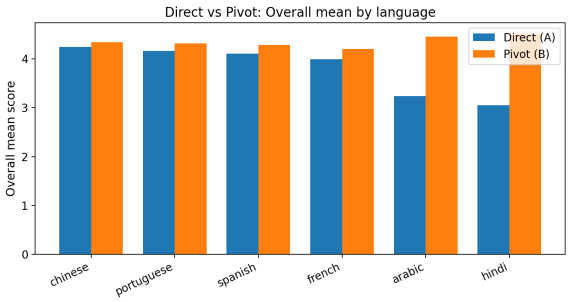

Figure 6: Comparison of overall mean LLM-judge scores (1–5 scale) between direct and pivoted generation.

4. Related Work

Early research in code summarization employed information-retrieval and template-based methods to extract salient terms and generate concise source code descriptions (Haiduc et al., 2010; Sridhara et al., 2010). The advent of pretrained models—including CodeBERT, GraphCodeBERT, and CodeT5—substantially improved performance on English-language code summarization, translation, and search tasks. Benchmarks such as CODESEARCHNET and CodeXGLUE provided standardized evaluation protocols for these English-centric code understanding and generation paradigms (Feng et al., 2020; Guo et al., 2021; Wang et al., 2021; Husain et al., 2019; Lu et al., 2021).

Concurrently, multilingual modeling has advanced considerably. Models such as mT5 have strengthened cross-lingual understanding across numerous *natural languages*, while benchmarks including MBXP and Multilingual HumanEval have extended code generation evaluation to diverse *programming languages* (Xue et al., 2021; Athiwaratkun et al., 2023). These efforts, however, predominantly assess how well models *generate code* rather than how effectively they *describe code* in different natural languages. Contemporary datasets such as XLCoST (Zhu et al., 2022) and MCoNaLa (Wang et al., 2023) establish mappings between code and natural language across multiple languages, yet prioritize text-to-code translation over code-to-text summarization.

Recent investigations have examined the capabilities of LLMs for English code summarization, analyzing summarization quality, prompt engineering, and evaluation methodologies (Szalontai et al., 2024; Fang et al., 2025; Sun et al., 2024). Though these studies yield valuable insights into how LLMs interpret and articulate code functionality, they remain confined to English and do not address multilingual or cross-linguistic dimensions.

This work addresses this gap by conducting the first systematic evaluation of *multilingual code sum-*

marization. We integrate reference-based metrics, LLM-as-a-judge assessment and fidelity checks across six natural and six programming languages. These contributions establish a reliability-aware framework for evaluating multilingual code summarization that extends beyond the predominantly English-focused literature.

5. Conclusion and Future Work

In this work, we introduced a unified framework for evaluating multilingual code summarization by combining reference-based metrics, LLM-as-a-judge scoring, and identifier/script-aware diagnostics. Our experiments revealed several interesting patterns, to highlight a few key findings: (i) summarization quality differs substantially across target languages; (ii) pivoting through English can improve fluency and content coverage for low-resource languages; and (iii) all models struggle to consistently preserve and interpret identifiers.

The last observation also reveals a core representational challenge in multilingual code summarization: identifiers (e.g., function names) must be interpreted both as syntactic symbols and as semantic units when generating natural language. Models need to understand their role in the code structure where English-like identifiers must align with non-English prose, and models often fail to preserve this link.

In future work, we will conduct stratified human evaluations with bilingual annotators to calibrate judge scores and expand our error typology. We will also extend the framework to related multilingual code tasks such as comment completion and code documentation.

6. Ethics Statement and Limitations

Linguistic Coverage. Our evaluation covers six natural languages but does not reflect the full typological or geographic diversity of global programming communities. Languages with complex morphology or distinct scripts—such as Japanese, Korean, Swahili, Yoruba, and Tamil—remain untested due to limited resources and model coverage, despite their linguistic and regional importance. (Chakravarthi et al., 2025; Hettiarachchi et al., 2025; Giagnorio et al., 2025).

Reference Quality and Dataset Bias. We use English summaries from CodeSearchNet as the reference ground truth. Although this provides a standardized evaluation benchmark, these automatically generated references may be imperfect or incomplete. Because CodeSearchNet primarily contains English code and documentation, some overlap with model pretraining data is possible. However, any such contamination is expected to be minimal, as our multilingual summaries are generated and evaluated in language spaces not present in the original dataset.

Back-translation Artifacts. For some reference-based metrics, outputs were translated into English for comparability. While this back-translation-based normalization ensures cross-language consistency, it can also introduce translation artifacts—particularly for morphologically rich or low-resource languages—thereby affecting metric reliability.

LLM-based Evaluation Biases. Using LLMs as judges enables scalable evaluation but may introduce biases from pretraining data and instruction-following behavior. We did not conduct human evaluations because of scale constraints. Instead, we report correlations between automatic metrics and LLM-judge ratings, with semantic metrics (COMET, BERTScore) showing the strongest alignment. Future work should validate these signals against human judgments to improve reliability calibration. Because trustworthy human assessment here requires multiple bilingual annotators per language and code domain, we defer a full human study and instead release resources to enable community evaluation at scale.

Computational Considerations. Our analysis does not examine latency or computational trade-offs across prompt styles and pivot strategies. Adding such comparisons and human–LLM alignment studies would improve the practical and ecological validity of the evaluation.

Overall Assessment. Despite these limitations, our study provides a systematic and extensible framework for multilingual code summarization and highlights directions for broader, fairer, and more inclusive future evaluations.

7. Resource Availability

All datasets, prompts, evaluation scripts, and model outputs are publicly available.
Dataset and benchmark can be downloaded from huggingface.co/CodeClarity
Code and evaluation pipeline is available at github.com/MadhuNimmo/CodeClarity
The repository includes scripts for dataset construction, multilingual generation, evaluation metrics, and LLM-as-a-judge evaluation.

References

Duarte M. Alves, José Pombal, Nuno M. Guerreiro, Pedro H. Martins, João Alves, Amin Farajian, Ben Peters, Ricardo Rei, Patrick Fernandes, Sweta Agrawal, Pierre Colombo, José G. C. de Souza, and André F. T. Martins. 2024. Tower: An open multilingual large language model for translation-related tasks.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Kumar Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, Arash Farahani, Siddhartha Jain, Robert Giaquinto, Haifeng Qian, Murali Krishna Ramanathan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, Sudipta Sengupta, Dan Roth, and Bing Xiang. 2023. Multi-lingual evaluation of code generation models. In *The Eleventh International Conference on Learning Representations*.

Mohammed Attia. 2007. Arabic tokenization system. In *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*, pages 65–72, Prague, Czech Republic. Association for Computational Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. 2024. Knowledge transfer from high-resource to low-resource programming languages for code llms. *Proc. ACM Program. Lang.*, 8(OOPSLA2).

Bharathi Raja Chakravarthi, Ruba Priyadharshini, Anand Kumar Madasamy, Sajeetha Thavareesan, Elizabeth Sherly, Saranya Rajiakodi, Balasubramanian Palani, Malliga Subramanian, Subalalitha Cn, and Dhivya Chinnappa, editors. 2025. *Proceedings of the Fifth Workshop on Speech, Vision, and Language Technologies for Dravidian Languages*. Association for Computational Linguistics, Acoma, The Albuquerque Convention Center, Albuquerque, New Mexico.

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

CodeGemma Team. 2024. CodeGemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*.

Cohere Labs. 2025. Command a. https://docs.cohere.com/docs/command-a. Accessed: 2025-10-12.

Menglong Cui, Pengzhi Gao, Wei Liu, Jian Luan, and Bin Wang. 2025. Multilingual machine translation with open large language models at practical scale: An empirical study. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5420–5443, Albuquerque, New Mexico. Association for Computational Linguistics.

John Dang, Shivalika Singh, Daniel D'souza, Arash Ahmadian, Alejandro Salamanca, Madeline Smith, Aidan Peppin, Sungjin Hong, Manoj Govindassamy, Terrence Zhao, Sandra Kublik, Meor Amer, Viraat Aryabumi, Jon Ander Campos, Yi-Chern Tan, Tom Kocmi, Florian Strub, Nathan Grinsztajn, Yannis Flet-Berliac, Acyr Locatelli, Hangyu Lin, Dwarak Talupuru, Bharat Venkitesh, David Cairuz, Bowen Yang, Tim Chung, Wei-Yin Ko, Sylvie Shang Shi, Amir Shukayev, Sammie Bae, Aleksandra Piktus, Roman Castagné, Felipe Cruz-Salinas, Eddie Kim, Lucas Crawhall-Stein, Adrien Morisot, Sudip Roy, Phil Blunsom, Ivan Zhang, Aidan Gomez, Nick Frosst, Marzieh Fadaee, Beyza Ermis, Ahmet Üstün, and Sara Hooker. 2024. Aya expanse: Combining research breakthroughs for a new multilingual frontier.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Edouard Grave, Michael Auli, and Armand Joulin. 2021. Beyond english-centric multilingual machine translation. *J. Mach. Learn. Res.*, 22(1).

Minying Fang, Xing Yuan, Yuying Li, Haojie Li, Chunrong Fang, and Junwei Du. 2025. Enhanced prompting framework for code summarization with large language models. *Proc. ACM Softw. Eng.*, 2(ISSTA).

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

Silvia García-Méndez, Milagros Fernández-Gavilanes, Enrique Costa-Montenegro, Jonathan Juncal-Martínez, and F. Javier González-Castaño. 2019. A library for automatic natural language generation of spanish texts. *Expert Systems with Applications*, 120:372–386.

Alessandro Giagnorio, Alberto Martin-Lopez, and Gabriele Bavota. 2025. Enhancing code generation for low-resource languages: No silver bullet. In *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*, pages 478–488.

GitHub. 2022. Global distribution of developers: The state of the octoverse. Accessed: 2025-05-01.

Gloroots. 2024. Best country to hire developers in 2024. Accessed: 2025-05-15.

Google Cloud. 2025. Gemini 2.0 flash-lite. `https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash-lite`. Accessed: 2025-10-17.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

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.

Sonia Haiduc, Andrian Marcus, Daniel Tatar, and Andrea DeLucia. 2010. On the use of automated text summarization techniques for summarizing software artifacts. In *2010 17th Working Conference on Reverse Engineering*, pages 35–44. IEEE.

Rajarshi Haldar and Julia Hockenmaier. 2024. Analyzing the performance of large language models on code summarization. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 995–1008, Torino, Italia. ELRA and ICCL.

Hansi Hettiarachchi, Tharindu Ranasinghe, Paul Rayson, Ruslan Mitkov, Mohamed Gaber, Damith Premasiri, Fiona Anting Tan, and Lasitha Uyangodage, editors. 2025. *Proceedings of the First Workshop on Language Models for Low-Resource Languages*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates.

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, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *CoRR*, abs/1909.09436.

JetBrains. 2024. How many developers are in the world? Accessed: 2025-04-29.

Priya Joshi and Carlos Martínez. 2020. Spanish code documentation: Challenges in NLP. *Journal of Software Engineering and Applications*, 13(7):305–316.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou,

Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Yinquan Lu, Wenhao Zhu, Lei Li, Yu Qiao, and Fei Yuan. 2024. LLaMAX: Scaling linguistic horizons of LLM by enhancing translation capabilities beyond 100 languages. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10748–10772, Miami, Florida, USA. Association for Computational Linguistics.

Antonio Mastropaolo, Matteo Ciniselli, Massimiliano Di Penta, and Gabriele Bavota. 2024. Evaluating code summarization techniques: A new metric and an empirical characterization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

Jan Müller, Anil Singh, and Wei Chen. 2021. MultiSumm: Multilingual summarization for programming languages. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4567–4579. ACL.

OpenAI. 2025. Gpt-5-nano model card. https://platform.openai.com/docs/models/gpt-5. Accessed: 2025-10-15.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395. ACL.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Harman Singh, Nitish Gupta, Shikhar Bharadwaj, Dinesh Tewari, and Partha Talukdar. 2024. Indicgenbench: A multilingual benchmark to evaluate generation capabilities of llms on indic languages. *arXiv preprint arXiv:2404.16816*.

Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. 2010. Towards automatically generating summary comments for java methods. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, page 43–52, New York, NY, USA. Association for Computing Machinery.

Weisong Sun, Yun Miao, Yuekang Li, Hongyu Zhang, Chunrong Fang, Yi Liu, Gelei Deng, Yang Liu, and Zhenyu Chen. 2024. Source code summarization in the era of large language models. Accepted to ICSE 2025.

Balázs Szalontai, Gergő Szalay, Tamás Márton, Anna Sike, Balázs Pintér, and Tibor Gregorics. 2024. Large language models for code summarization.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. 2024. Gemma: Open models based on gemini research and technology.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhiruo Wang, Grace Cuenca, Shuyan Zhou, Frank F. Xu, and Graham Neubig. 2023. MCoNaLa: A benchmark for code generation from multiple natural languages. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 265–273, Dubrovnik, Croatia. Association for Computational Linguistics.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni, and Chandan K. Reddy. 2022. XLCoST: A benchmark dataset for cross-lingual code intelligence.