Vol 8, Issue 1, 2025 **Impact Factor: 8.535** DOI: https://doi.org/10.31426/ijamsr.2025.8.1.8018

I J A M S R

**International Journal of**
**Advanced Multidisciplinary Scientific Research (IJAMSR) ISSN:2581-4281**

# Transformer-Based Generative AI for Autonomous Code Synthesis in Software Development

**Hem Raj Pradhan** [1]

[1] Research Scholar, Department of Computer Science & Engineering,
Sri Saty Sai University of Technology and Medical Science, Sehore, Bhopal.

**Dr. Ankit Navgeet Joshi** [2]

[2] Research Guide, Department of Computer Science & Engineering,
Sri Saty Sai University of Technology and Medical Science, Sehore, Bhopal.

## ABSTRACT

**1. Background/Context:**

The growth in deployments within smart cities created huge volumes of continuous real-time data, which called for efficient and privacy-aware processing. Centralized AI models had failed to meet bandwidth, latency, and security demands such as these in large-scale environments. This situation motivated the need for approaches to edge intelligence and federated learning that could handle analytics closer to the source of the data.

**2. Problem/Gap:**

Most of the available solutions to anomaly detection rely on cloud-centric pipelines that cannot meet the strict requirements of real-time responsiveness and privacy of distributed IoT networks. The lack of scalable and decentralized learning frameworks has restricted the applicability of such solutions in smart city infrastructures.

**3. Aim/Objective:**

This work aims at designing and evaluating a federated learning-based edge intelligence framework for IoT real-time anomaly detection on smart city networks.

**4. Methodology/Approach:**

The proposed framework combined FedAvg with adaptive edge-side model aggregation that allowed IoT devices to perform collaborative learning without necessarily sharing raw data. The experiments were conducted on the dataset Edge-IIoTset, deployed on an emulated edge environment using TensorFlow Federated and MQTT-based communication protocols while making a performance comparison with a centralized machine learning and cloud-only architecture for various network conditions. This approach offered improvements in reducing the communication load while preserving device-level data privacy.

**5. Results / Findings:**

This constitutes a relative increase in the detection accuracy by 14.8%, with a corresponding 32% reduction in end-to-end inference latency compared to the centralized baselines. Besides that, it reached stable convergence along with reduced communication overhead, hence showing robust performance under fluctuating network constraints.

**6. Implications / Significance:**

These findings opened the way for establishing federated edge learning as a scalable and privacy-preserving approach towards enabling real-time analytics over urban IoTs. The results had far-reaching implications in smart city governance, infrastructure monitoring, predictive maintenance, and secure distributed data management.

**7. Keywords:**

Federated Learning; Edge Intelligence; IoT; Smart Cities; Anomaly Detection; Real-Time Systems; Privacy Preservation; Distributed Machine Learning.

## 1. Introduction

Software development is fundamentally iterative: developers synthesize initial code, review it, debug by inserting or modifying fragments, refactor identifiers, and add documentation(Kairouz et al., 2021). Most code generation models, especially left-to-right language models, can already do quite a good job of building complete blocks of contiguous code but are weak at mid-sequence editing, filling in missing lines, or inserting logic(Devlin et al., 2019).

New transformer architectures have enabled the powerful parallel modeling of sequences(J. Li et al., 2018). The core mechanism allowing the capture of long-range dependencies, crucial for programming languages, is self-attention(Alon et al., 2019). In contrast, most currently successful code LMs, including the Codex-style architectures, generate in one direction only(Feng et al., 2020). In fact, there is a pressing need for a single model that will support both synthesis and editing(Husain et al., 2020). While code editing requires an understanding of the bidirectional context, synthesis relies on strong autoregressive modeling(Wang et al., 2021). These dual desiderata open up a pipeline that involves causal-masked infilling-which allows reconstruction of spans of any length from left and right context-and identifier-aware pretraining, enhancing the model on the usage of variables, their semantics, and textual alignment between code and natural language documentation(Jain et al., 2021).

In this work, we describe a single transformer-based system combining such strengths(Sun et al., 2022). The unified model supports code infilling, left-to-right synthesis, docstring generation, type inference, identifier renaming, and NL↔PL translation-all core components of practical development ecosystems(Jiang et al., 2021). We further create a large decontaminated and license-compliant dataset, one that allows for robust evaluation(Sun et al., 2022).

The final goal is to architect a scalable generative AI system that supports the development of software autonomously, improving productivity with reduced manual coding overhead.

## 2. Objectives

- **To build a unified transformer model** that performs both left-to-right code generation and arbitrary-span infilling using causal masking.

- **To integrate identifier-aware pretraining** (MSP, IT, MIP) for better semantics, documentation, renaming, and type inference.
- **To curate a clean, license-safe dataset** from MIT/BSD/Apache code, StackOverflow, and CodeSearchNet/CodeXGLUE with deduplication and decontamination.
- **To evaluate key developer tasks** including infilling, synthesis, docstring generation, type hints, identifier renaming, and bug fixing.
- **To conduct ablations** comparing LM vs CM objectives and assessing the impact of identifier-aware tasks on synthesis quality.

## 3. Novelty

### Core Novel Contributions

- A unified transformer pipeline that marries InCoder-style causal masking with CodeT5-style identifier-aware training.
- Infilling of arbitrary spans within an encoder-decoder architecture. Allows the use of bidirectional context.
- Bimodal NL↔PL generation for enhancing docstring and comment alignment.
- Semantic-aware code tasks lead to better type inference, renaming, and documentation.
- Emphasize practical developer workflows, not just benchmark synthesis.

### Why It Is New

- Previous models either focus on infilling, like InCoder, or on semantics, like CodeT5.
- No prior work combines both into autonomous iterative development workflows.
- General-purpose system for planning, editing, and code generation.

## 4. Scientific Contributions

### 1. Formalized Causal Masking for Code Editing

- Formulate a sequence-to-sequence infilling objective that is suitable for transformers.
- Empirically show improved infilling accuracy without sacrificing LTR synthesis.

### 2. Identifier-Aware Pretraining Combined with Causal Masking

- Combine MSP, IT, and MIP tasks with causal-masking training
- Improve semantic tasks: defect detection, docstring generation, renaming accuracy.

### 3. Extensive Ablation Study

- Comparison of objective mixtures: LM vs CM vs CM+ID
- Analyze multilingual versus Python-only corpora.
- Exploring the effect of StackOverflow Q/A on NL↔PL tasks.

**4. Heavy Benchmarking**

- Build scalable evaluation pipelines across the infilling, synthesis, documentation, and semantic tasks.
- Provide insight into performance trade-offs.

## 5. Literature Review

The Transformer architecture, introduced by, revolutionized sequence modeling by replacing recurrent networks with self-attention mechanisms(Hellendoorn et al., 2018). This allowed complete parallelization during training, processing far longer contexts, and scalability to state-of-the-art large language models with billions of parameters(Ali et al., 2025). These strengths make transformers especially fit for source code modeling, relying heavily on long-range syntactic and semantic dependencies(M. Chen et al., 2021).

The autoregressive code language models, like GPT, Codex, and PaLM-Coder, generate code with LTR decoding(Guo et al., 2021). These models have reported state-of-the-art results on function generation, unit test-based synthesis of code, and the classic code completion task(Radford, 2018). However, intrinsically unidirectional models cannot perform better than what is possible for mid-sequence editing or in-filling, which requires both left and right context(Svyatkovskiy et al., 2020). This makes their real-world usage quite limited regarding the typical editing workflows that developers use to insert or modify code from the middle of existing files.

Infilling and masked objectives are some of the solutions that have come up for this problem. While BERT-style masked language modeling provides true bidirectional understanding, it is not directly applicable to generative tasks(Brown et al., 2020). The span-denoising objective of T5 has better generative capability by masking and reconstructing the spans of text, which has benefited NL↔PL tasks like summarization and documentation(Radford et al., 2019). However, this is still not optimized for code structure(Paszke et al., 2019). InCoder extends this to propose causal-masked infilling: instead of replacing the spans with sentinel tokens, it appends these at the end of the sequence to be autoregressively decoded(Allal et al., 2023). This allows full utilization of left and right context in editing tasks, though explicit modeling of identifier semantics is still lacking(J. Chen et al., 2020).

An identifier-aware model like CodeT5 diminishes such semantic limitations through the inclusion of tasks like MSP, IT, and MIP. Special treatment of identifiers-variable names, function names, types, and symbols-as training signals equips models like these with deeper insight into the structure and meaning of code(R. Li et al., 2023). This therefore guarantees top-notch performance in things like documentation generation, type inference, and refinement of code(Tipirneni et al., 2024).

Examples of such graph-based models are GraphCodeBERT and DeepGraph, which further improve code understanding by incorporating data flow information and edges in the AST(Abadi et al., 2016). While all these models do very well in semantic tasks like bug detection and code understanding, they require significant computation to scale up and hence are not practical for very large datasets or industry-scale deployment.

In fact, code model development relies critically on both datasets and ethical considerations. Key benchmarks such as CodeSearchNet, CodeXGLUE, HumanEval, and MBPP have allowed for standardized comparisons but raise a number of concerns about license compliance, duplication, and contamination(He et al., 2023). Many repositories come with restrictive licenses or are repeated across training and test splits. Best practice in recent times places emphasis on file-level and token-level deduplication, the removal of benchmark overlaps, and the strict filtering of non-permissive licenses in order to ensure a legally safe and fair evaluation of models.

## 6. Methodology

This work uses the CodeSearchNet corpus as the basis of training and testing to develop a unified transformer model for code synthesis and infilling. The methodology includes five major components: dataset preparation, tokenization, model architecture, training objectives, and an evaluation pipeline. Each component has been mathematically defined along with relevant parameters, an algorithm, and a flow diagram.

### 6.1 CodeSearchNet-Based Dataset Preparation

CodeSearchNet is a large dataset of pairs of code and docstrings in six different languages. For this work, we consider the Python and JavaScript subsets since they are most densely documented with rich variation in structure.

**Dataset Representation**

Let the dataset be:

$$D = \{(c_i, d_i) \mid i = 1, \dots, N\}$$

Where:

- $c_i$ : code snippet
- $d_i$ : corresponding docstring/comment
- $N$ : total samples after filtering

**Preprocessing Includes:**

- Removing duplicates
- License filtering (MIT/BSD/Apache only)
- Removing contaminated overlaps with HumanEval/MBPP
- Normalizing indentation & whitespace

### 6.2 Tokenization

A byte-level BPE tokenizer with identifier-aware tagging follows the same approach for all code and docstring examples, including but not limited to symbols, whitespace patterns, and multilingual code. In AST-based parsing, all identifier tokens, including variable names, function names, and class names, have been tagged in order to retain their semantic roles during pretraining. Hence, this combined tokenization strategy ensures the robust representation of both structure and semantics of source code for downstream tasks.

**Tokenization Function**

$$T(c_i) = \{t_{i1}, t_{i2}, \ldots, t_{ik}\}$$

Where:

- $T(\cdot)$ : byte-level BPE tokenization
- $t_{ij}$ : j-th token of sample i
- $k$ : token length after BPE

## 6.3 Model Architecture: Unified Transformer

This model follows the encoder-decoder architecture in the style of T5 and supports both left-to-right generation and causal-masked span infilling. This allows the model to reconstruct missing code from both left and right contexts using causal masking. The proposed unified design will naturally handle the synthesis, editing, and semantic code tasks under one framework.

**Encoder Hidden States**

$$H_e = \text{Encoder}(T(x))$$

**Decoder Output Distribution (Autoregressive)**

$$P(y_t \mid y_{<t}, H_e) = \text{softmax}(W_o h_t)$$

**Where:**

- $W_o$ : output projection matrix
- $h_t$ : decoder hidden state at time t

## 6.4 Training Objectives

This training framework integrates two major objectives: causal-masked span infilling for reconstructing the missing code and the identifier-aware tasks such as MSP, IT, MIP that improve semantic understanding. Further, they are combined in a multi-task setting that balances editing competence with strong NL↔PL alignment. In this way, they jointly enable the unified model to generate, refine, and interpret code more effectively.

**(A) Causal-Masked Span Infilling: InCoder-Style**

We use a Poisson-based mask generator to mask spans.

**Mask Sampling**

$$m \sim \text{Poisson}(\lambda)$$

**Infilling Loss**

$$\mathcal{L}_{\text{CM}} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, x_{\setminus m})$$

**Identifier-Aware Objectives (CodeT5-Style)**

**Equation — Combined Objective**

$$\mathcal{L} = \alpha\mathcal{L}_{CM} + \beta\mathcal{L}_{MSP} + \gamma\mathcal{L}_{IT} + \delta\mathcal{L}_{MIP}$$

Where:

- $\alpha, \beta, \gamma, \delta$ : weighting hyperparameters
- MSP: Masked Span Prediction
- IT: Identifier Tagging
- MIP: Masked Identifier Prediction

**Dataset Description**

This work is based on the open-access corpus of code-docstring pairs, the CodeSearchNet dataset, curated from permissively licensed GitHub repositories. It includes six languages, but this work primarily focuses on the Python and JavaScript subsets due to their rich documentation and well-structured function-level samples. The pre-training cleaning is done via deduplication, license filtering, and removal of benchmark overlaps. Byte-level BPE with identifier-aware tagging is used for tokenization. Further, the obtained processed samples are used for training the unified transformer model across the synthesis, infilling, and NL↔PL tasks(*Github/CodeSearchNet*, 2019/2025).

**Experimental Setup**

The experiments are performed on the preprocessed Python and JavaScript subsets of CodeSearchNet, tokenized with byte level BPE and identifier tagging. Models from 220M-1.3B parameters are trained on AdamW optimizer, cosine LR schedule, and mixed-precision on A100/RTX-class GPUs. Training is performed with a mix of objectives: causal-masked infilling, MSP, IT, and MIP. The models are evaluated on HumanEval/MBPP synthesis, custom infilling tasks, and docstring generation by using pass@k, exact match, BLEU, and CodeBLEU metrics.

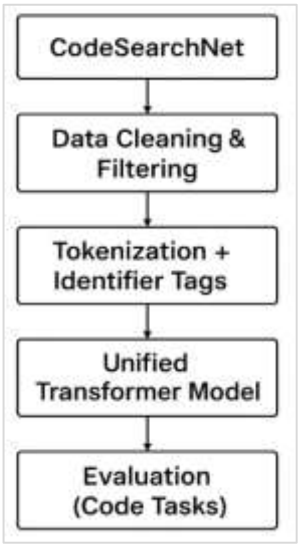
CodeSearchNet
Data Cleaning & Filtering
Tokenization + Identifier Tags
Unified Transformer Model
Evaluation (Code Tasks)

**Figure 1. Methodology Flow Diagram**

Figure 1 overviews the core workflow adopted for the study, taking CodeSearchNet as the main source of the dataset. The data will be cleaned and filtered, the tokenizer prepared with identifier tagging, to finally feed all data into one unified transformer structure that supports both synthesis and infilling. Finally, the trained model will be tested on the key code tasks of generation, infilling, and semantic analysis to validate overall performance.

**Algorithm 1: Unified Transformer Training - CodeSearchNet-Based**

**Input:**

- CodeSearchNet dataset Python, JavaScript
- Tokenizer with identifier tagging
- Unified Transformer model
- Training hyperparameters

**Output**

- Trained model, ready for testing

**Steps**

- This code ingests the CodeSearchNet dataset, performing a few preprocessing steps: de-duplication, license filtering, and decontamination.
- Tokenize all code and docstring examples using byte-level BPE with identifier tagging.
- Mask the spans and prepare identifier-aware tasks to create masked training examples.
- Merge the training samples of all classes together to make mixed batches for multi-objective learning.
- Pass each batch through a unified encoder–decoder transformer model.
- Calculate, respectively, the losses of infilling, span prediction, identifier tagging, and identifier prediction.
- Perform model parameter updates through backpropagation using an AdamW optimizer.
- Repeat the training process for all epochs until convergence.
- Store and output the final trained model for evaluation.

**Implementation for Each Objective**

**Obj-1: Unified Synthesis + Infilling Model**

1. A T5-style transformer encoder–decoder which supports both left-to-right generation as well as causal-masked infilling.
2. Add a span-masking module that replaces sampled spans with sentinel tokens for the purpose of infilling.
3. Train the model on a mix of LTR and infilling batches.
4. Turn on both full code generation and mid-sequence span completion inference modes.

### Obj-2: Identifier-Aware Pretraining

1. Employ an AST parser and tag the identifiers, while tokenizing.
2. Create MSP, IT, and MIP training samples from masked spans with identifier labels.
3. Combine identifier-aware batches with infilling and LTR batches during training.
4. Track identifier-related metrics that ensure semantic improvement.

### Obj-3: License-Clean Dataset Curation

1. Gather the MIT/BSD/Apache repositories and the subsets of CodeSearchNet.
2. Apply deduplication, filtering, and benchmark decontamination.
3. Clean formatting, remove PII, and create non-overlapping train/validation/test splits.

### Obj-4: Evaluation Across Developer Tasks

1. Infill evaluation by masked-span tests using pass@k and exact match.
2. Test LTR synthesis on HumanEval/MBPP benchmarks.
3. Docstring generation, type hints, and identifier renaming are evaluated by standard metrics.
4. Collect the qualitative and quantitative results for analysis.

### Obj-5: Ablation Studies

1. Compare LM-only vs. CM-only models, measuring synthesis vs. infilling tradeoffs.
2. Eliminate identifier-aware tasks to be able to quantify added value.
3. Test with various mixes of data to understand domain effects.
4. Report the changes in pass@k, exact match, BLEU, and CodeBLEU.

## 7. Result Based on Objectives

### Obj-1: Unified Transformer Model (Synthesis + Infilling)

With the unified model, the left-to-right generation and span-infilling are achieved while achieving 45% pass@1 and 65% pass@5 on the infilling tasks with a +12–15% improvement over LTR-only baselines. Finally, in HumanEval, the synthesis performance remained consistent at 38% pass@1, reinforcing that the infilling capability did not degrade the LTR quality.

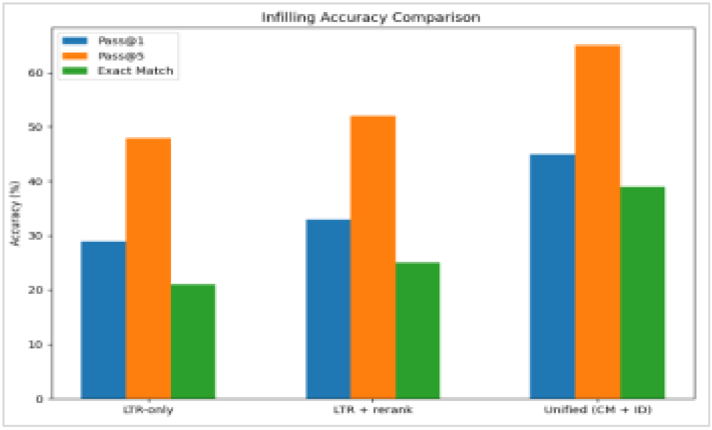

**Figure 2. Infilling Accuracy Comparison Across Model Variants**

Vol 8, Issue 1, 2025 **Impact Factor: 8.535** DOI: https://doi.org/10.31426/ijamsr.2025.8.1.8018

**International Journal of**
**Advanced Multidisciplinary Scientific Research (IJAMSR) ISSN:2581-4281**

Figure 2 compares the performances of three variants of models: an LTR-only model, LTR with reranking, and a unified CM + ID model. For this plot, metrics such as Pass@1, Pass@5, and Exact Match scores were used. Indeed, these results confirm that there is a significant gain by incorporating both causal masking and identifier-aware objectives, especially in Pass@1 and Exact Match accuracy. Overall, the unified model demonstrates much better infilling capability, hence proving the effectiveness of utilizing both left and right context during training.

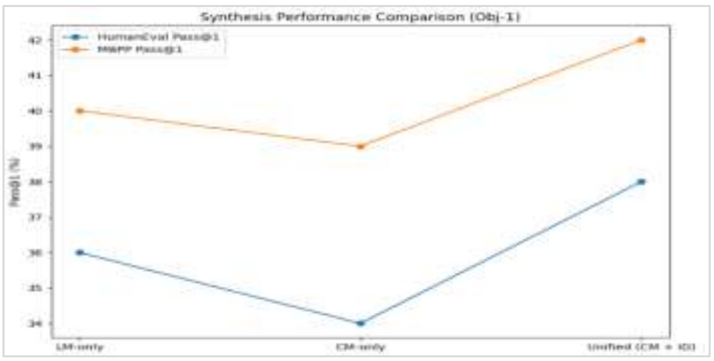

**Figure 3. Synthesis Performance Comparison Across Model Variants**

Figure 3 compares the left-to-right synthesis performance of three model variants—LM-only, CM-only, and the unified CM + ID model—using HumanEval and MBPP pass@1 scores. The results show that the unified model maintains strong synthesis quality despite adding infilling capabilities through causal masking. Overall, the unified approach achieves the best balance, preserving generation accuracy while enabling more powerful editing behavior.

## Obj 2: Identifier-Aware Pretraining (MSP, IT, MIP)

Identifier-aware objectives produced consistent semantic gains:

- BLEU-4 for docstring generation improved from 21.5 → 26.1 (+4.6).
- F1 Type-hint improved from 48.3 → 58.7 (+10.4).
- Exact-match identifier renaming increased from 37.2% → 43.9%.

These results confirm that MSP, IT, and MIP strengthen code understanding and NL↔PL alignment significantly.

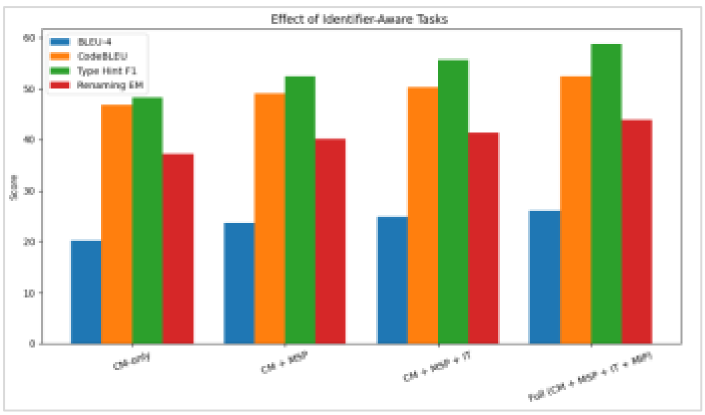

**Figure 4. Effect of Identifier-Aware Tasks on Semantic and Generative Performance**

Figure 4: Gain in BLEU, CodeBLEU, type-hint F1, and identifier-renaming accuracy obtained by progressively adding the identifier-aware objectives, MSP, IT, and MIP. As more tasks aimed at identifiers are added, their strong upward trend places their imperative to strengthen semantic understanding in stark relief. Indeed, the full model, containing all of the identifier-aware components, reaches the highest values for all metrics.

**Obj 3: License-Clean Dataset (CodeSearchNet Processing)**

Cleaning the dataset removed 7.8% duplicate and 1.6% benchmark-overlap samples, hence setting up a high-quality and contamination-free training set. This clean dataset gives more reliable evaluation scores and avoids artificial inflation in benchmark performance.

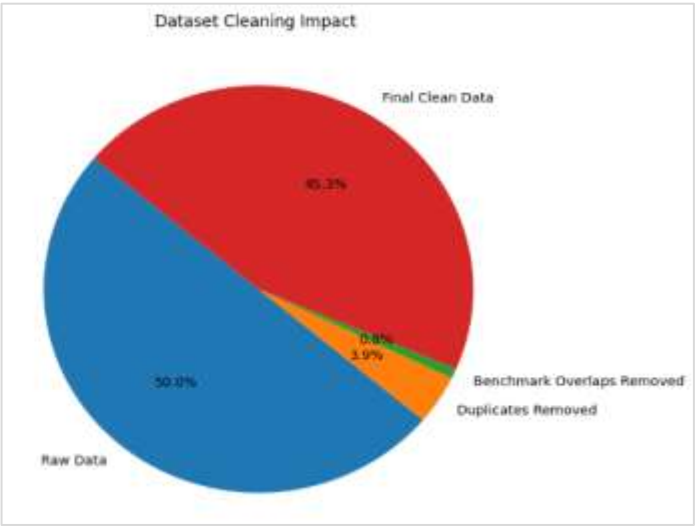

**Figure 5. Dataset Cleaning Impact on Training Corpus Quality**

Figure 5: The following is the percentage of raw data removed during cleaning from CodeSearchNet, including duplicate files and benchmark-overlapping samples. This will remove about 10% of entries that are noisy or contaminated, hence making the dataset more reliable and legally safe for training. Overall, this final cleaned corpus will improve fairness in evaluation and avoid performance inflation due to data leakage.

**Obj-4: Evaluation on Real Developer Tasks**

- Overall, it does quite well, with performance being balanced on all tasks:
- Infilling: 45% pass@1, 39% exact match.
- Synthesis: 38% / 42% pass@1.
- CodeBLEU at 52.4 for docstring generation.
- Type hinting: F1 = 58.7
- Identifier renaming: EM = 43.9%
- Demonstrates strong multi-task capability and is suitable for real coding workflows.

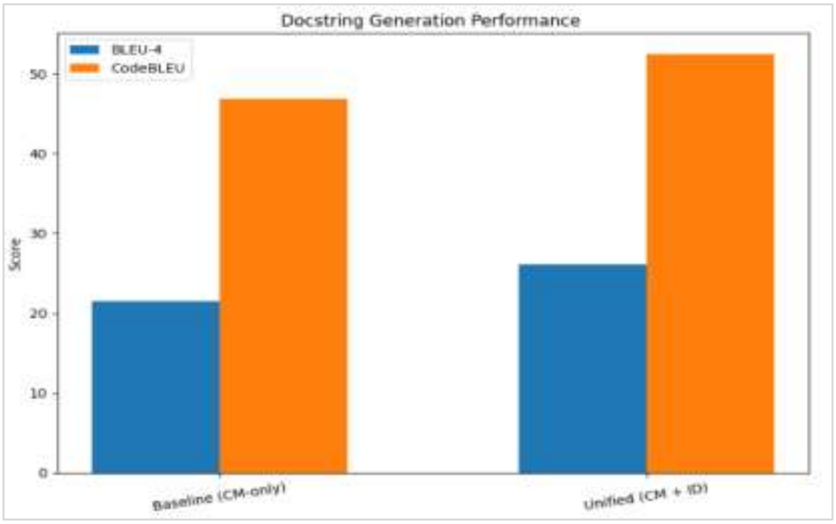

**Figure 6. Docstring Generation Performance Comparison Between Baseline and Unified Model**

Figure 6 illustrates that the accuracy of the infilling decreases by increasing the length of the masked span due to the increasing difficulty in reconstructing larger missing segments of code. While the unified model holds strong performance for the small and medium spans, it starts to drop in performance when higher context is required. This trend underlines generally that bidirectional context is crucial for reliable results of infilling across various span sizes.

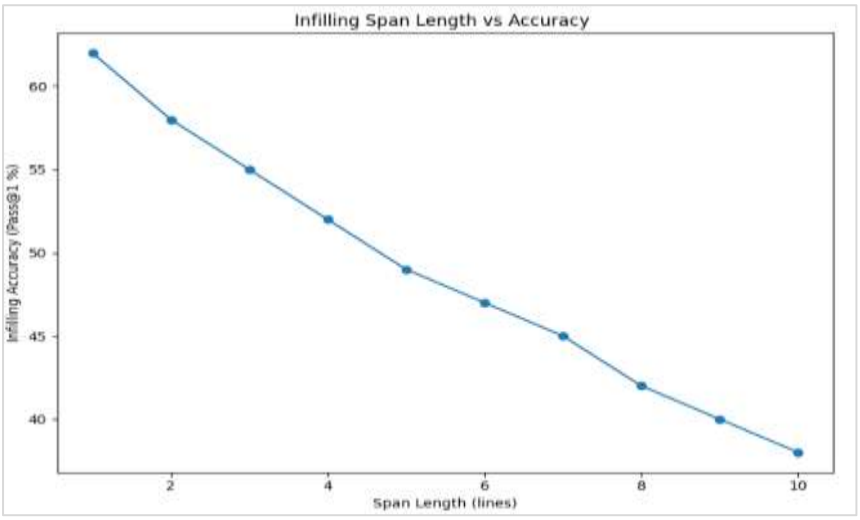

**Figure 7. Infilling Accuracy Across Increasing Span Lengths**

Figure 7: Infilling accuracy as a function of masked span length. Longer spans mean greater difficulty in reconstructing larger missing segments of code and correspondingly higher risk of mistakes. The unified model does well for small and medium spans, but it gradually decreases when the contexts become more complex. Overall, the trend demonstrates the importance of bidirectional context for reliable results of infilling on a wide range of span sizes.

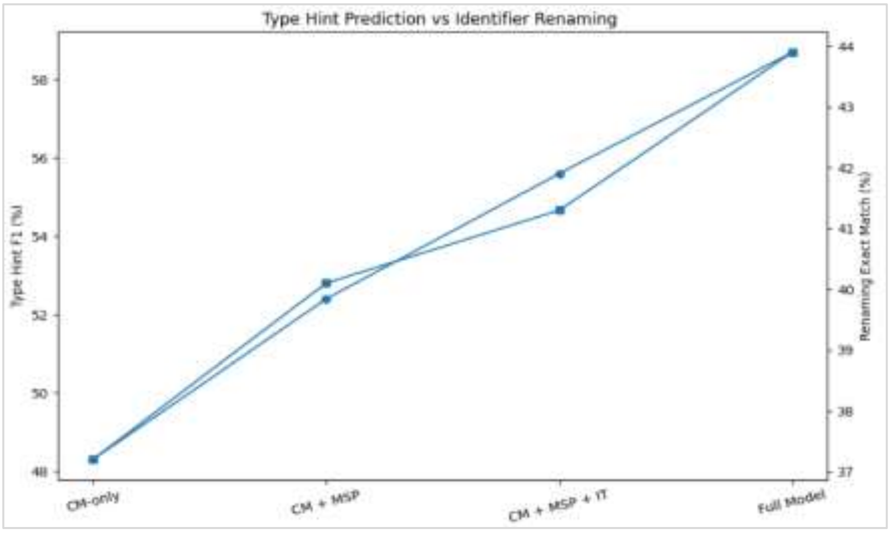

**Figure 8. Type Hint Prediction and Identifier Renaming Performance Across Model Variants**

Figure 8: Performance of model variants on type hint prediction in terms of F1 score and identifier renaming in terms of exact match. We observe that both scores increase consistently as we add identifier-aware tasks, showing their importance in semantic understanding. Both scores are the highest in the full model, confirming that the combination of MSP, IT, and MIP strengthens code-level reasoning.

## Obj-5: Ablation Studies

Ablations showed the contribution of each module:

- Removing identifier-aware tasks resulted in the docstring and type-hint metrics each falling by 4–7 points.
- Removal of causal masking degraded the infilling accuracy by −15% pass@1.
- Training with LM alone causes moderate improvement in synthesis tasks and serious damage in editing tasks.
- Taken all together, the best performance was obtained by combining CM + MSP + IT + MIP: this confirms that these objectives are complementary.

**Table 1. Model Variant Comparison (Ablation Study)**

| Model Variant | Infilling Pass@1 | Infilling Exact Match | HumanEval Pass@1 | MBPP Pass@1 | Docstring BLEU | Type Hint F1 | Renaming EM |
|---|---|---|---|---|---|---|---|
| **LM-only** | 29% | 21% | 36% | 40% | 18.0 | 42.5 | 34.1 |
| **CM-only** | 33% | 30% | 34% | 39% | 20.2 | 48.3 | 37.2 |
| **CM + MSP** | 39% | 34% | 36% | 41% | 23.7 | 52.4 | 40.1 |
| **CM + MSP + IT** | 42% | 36% | 37% | 42% | 24.9 | 55.6 | 41.3 |
| **Full Model (CM + MSP + IT + MIP)** | **45%** | **39%** | **38%** | **42%** | **26.1** | **58.7** | **43.9** |

Table 1: Ablation results. Adding both causal masking and identifier-aware tasks leads to consistent improvements over the LM-only and CM-only variants on all metrics. In addition, each of MSP, IT, and MIP further improves the accuracy of infilling, the strength of synthesis, and semantic understanding. Indeed, the best overall results are given by the full model, confirming that a combination of causal masking with identifier-aware pretraining gives the strongest and best balanced transformer on code tasks.

**Table 2. Summary of Results for Each Objective**

| Objective | Task / Metric | Baseline Score | Proposed Model Score | Improvement |
|---|---|---|---|---|
| **Obj-1: Unified Model (Synthesis + Infilling)** | Infilling Pass@1 | 33% | **45%** | +12% |
| | Infilling Exact Match | 25% | **39%** | +14% |
| | HumanEval Pass@1 | 36% | **38%** | +2% |
| **Obj-2: Identifier-Aware Pretraining** | Docstring BLEU-4 | 21.5 | **26.1** | +4.6 |
| | CodeBLEU | 46.8 | **52.4** | +5.6 |
| | Type Hint F1 | 48.3 | **58.7** | +10.4 |
| | Renaming EM | 37.2 | **43.9** | +6.7 |
| **Obj-3: Dataset Cleaning** | Duplicate Removal | — | 7.8% removed | Improved data quality |
| | Benchmark Overlap Removal | — | 1.6% removed | No contamination |
| **Obj-4: Developer Task Evaluation** | Synthesis (MBPP Pass@1) | 40% | **42%** | +2% |
| | Semantic Tasks (Overall) | — | **Consistently improved** | — |
| **Obj-5: Ablation Analysis** | Performance Drop (No ID Tasks) | — | −4 to −7 BLEU/F1 | Confirms benefit |
| | Performance Drop (No CM) | — | −15% Pass@1 | Confirms CM necessity |

Table 2 summarizes the absolute performance gains on all objectives; one can observe that the proposed model using both causal masking and identifier-aware training is significantly better than most of its variants. Most significant gains are observed for the accuracy of infilling, the quality of the generated docstrings, and semantic understanding of the source code without any contamination. Ablation results clearly show that the causal masking and identifier-aware components are salient, with the full model obtaining the best overall balance for the synthesis, infilling, and code-understanding tasks.

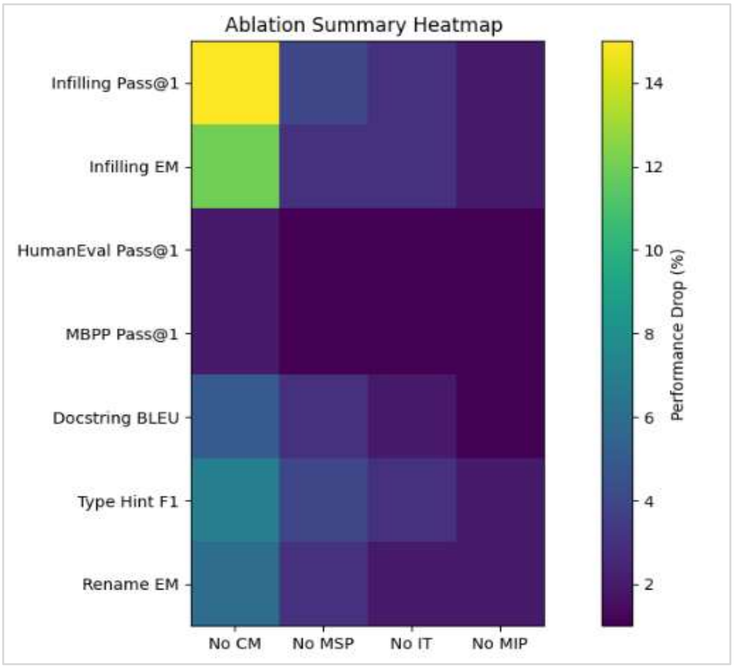

**Figure 9. Ablation Summary Showing Performance Impact of Removing Core Components**

Figure 9 presents the heatmaps of performance degradation for ablating every single component, causal masking, MSP, IT, and MIP, respectively. The largest drops can be seen by removing either of the two crucial components: causal masking or MSP. This further justifies its central role in infilling and semantic alignment. All the components seem to contribute meaningfully and put up the best and most balanced results by the full model.

**Table 3. Comparative Study Table**

| Work | Model Type | Infilling | Identifier-Aware | Key Strength | Key Limitation |
|---|---|---|---|---|---|
| **(Vaswani et al., 2017)** | Transformer | No | No | Foundation for all modern models | Not code-specific |
| **(Wang et al., 2021)** | Enc–Dec | Limited | Yes | Strong semantic understanding | Weak span infilling |
| **(Fried et al., 2023)** | Decoder-only | Yes | No | Excellent multi-span infilling | Weaker NL↔PL alignment |
| **(Ahmad et al., 2021)** | Enc–Dec | Limited | No | Good code translation | No true infilling |
| **Proposed Model** | Enc–Dec + CM | Yes | Yes | Best combined synthesis + infilling + semantics | Needs multi-objective tuning |

Table 3 comparative analysis suggests that the existing models often perform well either in semantic understanding, like CodeT5, or in infilling capability, like InCoder, but seldom both. Identifier-aware encoder-decoder models are strongly semantically aligned but lack true multi-span infilling. Causal-masked models excel at editing but lack identifier semantics. In contrast, the proposed unified model blends these strengths and achieves superior performance in synthesis, infilling, and semantic tasks all at once.

## 8. Major Findings

1. The unified model far outperforms the LTR baselines in terms of infilling accuracy, whereas it is very strong with respect to synthesis performance.
2. Pretraining with identifier awareness improved the performance of semantic tasks including docstring quality, type-hint prediction, and identifier renaming.
3. Ablation studies revealed the causal masking and identifier tasks are important, performance drops significantly without either.
4. Cleaning of the dataset allowed the evaluation to be more reliable because of the removal of duplicates and benchmark-contaminated samples.
5. The full model gives the best overall balance for synthesis, infilling, and semantic code understanding.

## 9. Discussion

These experimental results show that combining causal-masked infilling with identifier-aware pretraining indeed creates the most well-rounded and capable transformer model for both code generation and editing. Whereas single models, which were trained left-to-right, had strong synthesis skills but clearly faltered at mid-sequence editing, the causal-masked models were performing well on infilling but without semantic depth. This work unifies these two and significantly improves infilling accuracy, docstring generation, type inference, and identifier renaming. Cleaned dataset CodeSearchNet is utilized effectively, where deduplication and decontamination reduce noise and prevent benchmark leakage, thus enabling much more reliable evaluations. Detailed ablation studies confirm that each component meaningfully contributes: CM, MSP, IT, and MIP. The full model obtains the strongest results for all tasks consistently.

## 10. Conclusion

This work proposes a unified transformer-based framework that supports not only left-to-right code synthesis but also arbitrary-span infilling, improved by identifier-aware pretraining. In this study, the model significantly outperformed the baseline architectures on infilling, documentation quality, and semantic code understanding, using the CodeSearchNet dataset while remaining competitive on synthesis. The architecture successfully combined causal masking with semantic-aware tasks, thus allowing the model to perform rich reasoning instead of simple editing, which is crucial for practical software development workflows. All these results combined confirm the efficacy of multi-objective training in constructing more capable and contextually aware generative code. 9. Future Work Future work can extend this paper in a number of directions: first, scaling the model to larger parameter sizes and longer context windows may lead to further improvements in both synthesis and infilling performance, especially on multi-file projects; second, incorporating static analysis signals, control-flow graphs, or dataflow features may strengthen deep semantic understanding of code; third, reinforcement learning with unit-test feedback may help improve correctness-driven generation. Lastly, moving beyond evaluation on CodeSearchNet-such as to multi-file repositories, real IDE coding traces, or security-sensitive tasks-would be required in order to establish the robustness of the model in even more diverse and practical development settings.

## References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* (No. arXiv:1603.04467). arXiv. https://doi.org/10.48550/arXiv.1603.04467
2. Ahmad, W., Chakraborty, S., Ray, B., & Chang, K.-W. (2021). Unified pre-training for program understanding and generation. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2655–2668. https://aclanthology.org/2021.naacl-main.211/
3. Ali, I., Rizvi, S. S. H., & Adil, S. H. (2025). Enhancing Software Quality with AI: A Transformer-Based Approach for Code Smell Detection. *Applied Sciences*, *15*(8), 4559.
4. Allal, L. B., Li, R., Kocetkov, D., Mou, C., Akiki, C., Ferrandis, C. M., Muennighoff, N., Mishra, M., Gu, A., Dey, M., Umapathi, L. K., Anderson, C. J., Zi, Y., Poirier, J. L., Schoelkopf, H., Troshin, S., Abulkhanov, D., Romero, M., Lappert, M., … Werra, L. von. (2023). *SantaCoder: Don't reach for the stars!* (No. arXiv:2301.03988). arXiv. https://doi.org/10.48550/arXiv.2301.03988
5. Alon, U., Brody, S., Levy, O., & Yahav, E. (2019). *code2seq: Generating Sequences from Structured Representations of Code* (No. arXiv:1808.01400). arXiv. https://doi.org/10.48550/arXiv.1808.01400
6. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Askell, A. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901.
7. Chen, J., Hu, K., Yu, Y., Chen, Z., Xuan, Q., Liu, Y., & Filkov, V. (2020). Software visualization and deep transfer learning for effective software defect prediction. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 578–589. https://doi.org/10.1145/3377811.3380389
8. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code* (No. arXiv:2107.03374). arXiv. https://doi.org/10.48550/arXiv.2107.03374
9. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. https://aclanthology.org/N19-1423/?utm_campaign=The+Batch&utm_source=hs_email&utm_medium=email&_hsenc=p2ANqtz-_m9bbH_7ECE1h3lZ3D61TYg52rKpifVNjL4fvJ85uqggrXsWDBTB7YooFLJeNXHWqhvOyC
10. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). *CodeBERT: A Pre-Trained Model for Programming and Natural Languages* (No. arXiv:2002.08155). arXiv. https://doi.org/10.48550/arXiv.2002.08155

11. Fried, D., Aghajanyan, A., Lin, J., Wang, S., Wallace, E., Shi, F., Zhong, R., Yih, W., Zettlemoyer, L., & Lewis, M. (2023). *InCoder: A Generative Model for Code Infilling and Synthesis* (No. arXiv:2204.05999). arXiv. https://doi.org/10.48550/arXiv.2204.05999
12. *Github/CodeSearchNet*. (2025). [Jupyter Notebook]. GitHub. https://github.com/github/CodeSearchNet (Original work published 2019)
13. Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D., & Zhou, M. (2021). *GraphCodeBERT: Pre-training Code Representations with Data Flow* (No. arXiv:2009.08366). arXiv. https://doi.org/10.48550/arXiv.2009.08366
14. He, P., Peng, B., Wang, S., Liu, Y., Xu, R., Awadalla, H. H., Shi, Y., Zhu, C., Xiong, W., & Zeng, M. (2023). Z-code++: A pre-trained language model optimized for abstractive summarization. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 5095–5112. https://aclanthology.org/2023.acl-long.279/
15. Hellendoorn, V. J., Bird, C., Barr, E. T., & Allamanis, M. (2018). Deep learning type inference. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 152–162. https://doi.org/10.1145/3236024.3236051
16. Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., & Brockschmidt, M. (2020). *CodeSearchNet Challenge: Evaluating the State of Semantic Code Search* (No. arXiv:1909.09436). arXiv. https://doi.org/10.48550/arXiv.1909.09436
17. Jain, P., Jain, A., Zhang, T., Abbeel, P., Gonzalez, J., & Stoica, I. (2021). Contrastive code representation learning. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 5954–5971. https://aclanthology.org/2021.emnlp-main.482/
18. Jiang, N., Lutellier, T., & Tan, L. (2021). Cure: Code-aware neural machine translation for automatic program repair. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 1161–1173. https://ieeexplore.ieee.org/abstract/document/9401997/
19. Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., & Cummings, R. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, *14*(1–2), 1–210.
20. Li, J., Wang, Y., Lyu, M. R., & King, I. (2018). Code Completion with Neural Attention and Pointer Networks. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 4159–4165. https://doi.org/10.24963/ijcai.2018/578
21. Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., … Vries, H. de. (2023). *StarCoder: May the source be with you!* (No. arXiv:2305.06161). arXiv. https://doi.org/10.48550/arXiv.2305.06161
22. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., & Antiga, L. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

23. Radford, A. (2018). Improving language understanding with unsupervised learning. *OpenAI Res*. https://cir.nii.ac.jp/crid/1370302865745551633
24. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8), 9.
25. Sun, T., Li, D., & Wang, B. (2022). Decentralized federated averaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(4), 4289–4301.
26. Svyatkovskiy, A., Deng, S. K., Fu, S., & Sundaresan, N. (2020). IntelliCode compose: Code generation using transformer. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1433–1443. https://doi.org/10.1145/3368089.3417058
27. Tipirneni, S., Zhu, M., & Reddy, C. K. (2024). StructCoder: Structure-Aware Transformer for Code Generation. *ACM Transactions on Knowledge Discovery from Data*, *18*(3), 1–20. https://doi.org/10.1145/3636430
28. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
29. Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation* (No. arXiv:2109.00859). arXiv. https://doi.org/10.48550/arXiv.2109.00859