MoSE: Hierarchical Self-Distillation Enhances Early Layer Embeddings

Andrea Gurioli[1], **Federico Pennino**[1], **Joao Monteiro**[2*], **Maurizio Gabbrielli**[1]

[1]University of Bologna
[2]Apple MLR
andrea.gurioli5@unibo.it, federico.pennino2@unibo.it, joaomonteirof@apple.com, maurizio.gabbrielli@unibo.it

Abstract

Deploying language models often requires navigating accuracy vs. performance trade-offs to meet latency constraints while preserving utility. Traditional model distillation reduces size but incurs substantial costs through training separate models. We introduce MODULARSTARENCODER (MOSE), a 1-billion-parameter multi-exit encoder for code retrieval and classification that employs a novel Self-Distillation mechanism. This approach significantly enhances lower-layer representations, enabling flexible deployment of different model portions with favorable performance trade-offs. Our architecture improves text-to-code and code-to-code search by targeting specific encoder layers as exit heads, where higher layers guide earlier ones during training, thereby improving intermediate representations at minimal additional cost. We further enhance MOSE with a repository-level contextual loss that maximizes training context window utilization. Additionally, we release a new dataset created through code translation that extends text-to-code benchmarks with cross-language code-to-code pairs. Evaluations demonstrate the effectiveness of Self-Distillation as a principled approach to trading inference cost for accuracy across various code understanding tasks.

MoSE — https://huggingface.co/modularStarEncoder
SynthCoNL — https://huggingface.co/datasets/modularStarEncoder/SynthCoNL

1 Introduction

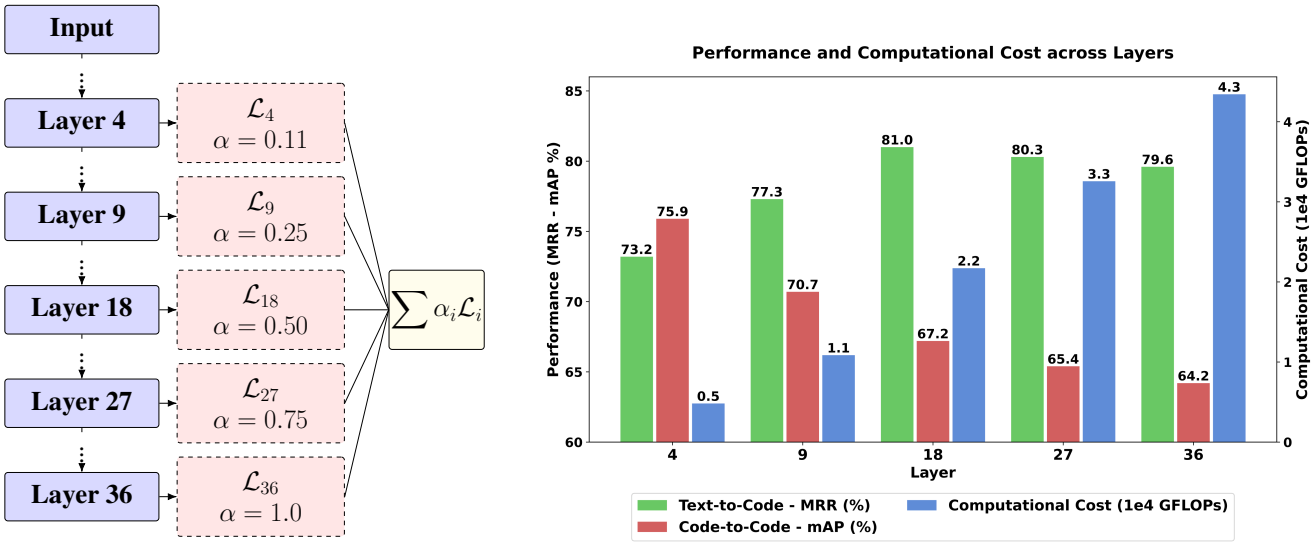
Large language models (LLMs) have significantly impacted natural language processing (Niu et al. 2023), but their computational demands pose substantial deployment challenges. The field has responded with various strategies aimed at improving efficiency: quantization techniques reduce numerical precision (Jacob et al. 2017; Lin et al. 2023; Egiazarian et al. 2024); knowledge distillation trains smaller "student" models to emulate larger "teacher" models (Sanh et al. 2019; Jiao et al. 2019); and pruning methods eliminate less influential parameters (Han et al. 2015). Concurrently, model families like LLAMA (Dubey et al. 2024), QWEN (Hui et al. 2024), MISTRAL (Jiang et al. 2023), and SMOLLM (Allal et al. 2025) exemplify a shift toward more efficient architectures at varying parameter scales.

Dynamic inference approaches further optimize efficiency through mechanisms like multi-exit networks. Early-exit architectures such as BRANCHYNET (Teerapittayanon, McDanel, and Kung 2017) balance computation and accuracy by allowing predictions at intermediate layers, while Matryoshka representation learning (Kusupati et al. 2022) enables adjustment of computational complexity through embedding dimensionality pruning. These approaches maintain performance while reducing computational demands in resource-constrained environments. Furthermore, insights from the SIMCLR framework (Chen et al. 2020) revealed that the final layer of a model does not necessarily produce the most useful representations. Valeriani et al. (2023) analyzed the geometric evolution of hidden representations within large transformer models across layers, outlining that intermediate layers are identified as holding the most semantically rich representations for downstream tasks.

Building on these principles, we introduce MODULARSTARENCODER (MOSE), a modular multi-exit encoder built on top of the STARCODER-2 (Lozhkov et al. 2024) architecture, with 1 billion parameters that incorporate intra-model Self-Distillation. As illustrated in Figure 1a, our architecture applies training objectives at multiple exit points throughout the network (layers 4, 9, 18, 27, and 36), with each exit contributing a weighted loss component to the overall objective. This approach encourages lower layers to learn better representations by mimicking higher-layer outputs. Figure 1b demonstrates the resulting accuracy-computation trade-off of encoders trained under this framework: despite a 90% reduction in floating point operations from layer 36 to layer 4, performance over text-to-code retrieval drops by only 6.4% in absolute terms, while the model maintains the best performances for code-to-code retrieval in very early layers.

Notably, the exit point with the best performance isn't necessarily the final layer, as noted in previous findings due to Chen et al. (2020); Valeriani et al. (2023). This offers both deployment flexibility and a natural model selection mechanism for specific tasks. Users can select which portion of the model to use during inference, starting from the first exit point at 160M parameters. By allowing users to select an earlier exit point, MOSE provides a direct mechanism to decrease computational demands. This flexibility is crucial as smaller models generally require less energy and memory for inference, leading to more sustainable deployment

*Work done before joining Apple



(a) (b)

Figure 1: **(a)** Overview of our multi-exit Self-Distillation encoder, shown here with exit heads at selected layers (e.g., Layers 4, 9, 18, 27, and 36). Each exit head predicts an output embedding and adds a layer loss, contribution weighted by a coefficient α_i, summed into the overall objective $\mathcal{L}$. **(b)** Computational cost (GFLOPs) vs. performances trade-off at different exit layers over CODESEARCHNET dataset (text-to-code in avg. MRR) and POJ104 (code-to-code in mAP). Despite a reduction of approximately 90% floating point operations from layer 36 to layer 4, MRR performance only drops by 6.4% in absolute terms. For POJ104, our best results are observed in the initial layers.

options (Samsi et al. 2023).

We fine-tuned MOSE on SYNTHCONL (cf. Sec. 5 for details) for text-to-code and code-to-code retrieval, and on BIGCLONEBENCH (Svajlenko and Roy 2015) for code clone detection, achieving competitive results among open models while maintaining deployment modularity.

Our contributions are as follows:

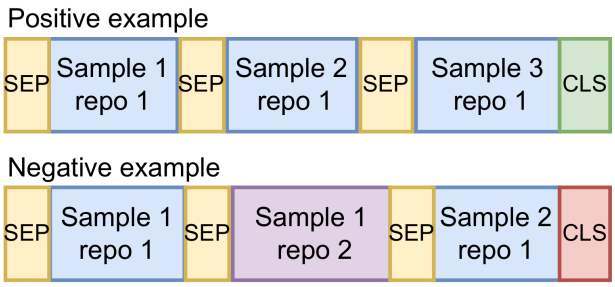
1. A Self-Distillation methodology that trains multiple models within a unified layer stack, reducing redundancy and improving scalability, an approach that could significantly impact LLM training pipelines dependent on multiple model distillations.
2. MOSE, pre-trained and fine-tuned, with up to 1 billion parameters and five exit points, allowing users to select model size based on memory and computational constraints.
3. SYNTHCONL, a new dataset of 1,071,367 natural language-code-code triplets constructed via code translation, expanding text-to-code benchmarks with cross-language code-to-code pairs.

2 Self-Distillation and In-Context Classification

As different layers of transformer models capture varying degrees of semantic richness, often with intermediate layers proving most effective for specific tasks (Bolya et al. 2025), we design our pre-training strategy to support modularity and ease downstream task adaptability. Rather than treating the deepest layer as the sole source of meaningful representations, we aim to enhance representations throughout the network.

We hypothesize that *applying supervision at multiple depths*, not just at the output, encourages more robust and semantically rich intermediate representations. To this end, we combine multiple training objectives, each contributing to more versatile and generalizable embeddings (Wang et al. 2023). While we retain the widely adopted Masked Language Modeling (MLM) loss (Feng et al. 2020), we discard the traditional Next Sentence Prediction (NSP) objective. NSP has been shown to offer minimal benefit after fine-tuning (Liu et al. 2019), and it imposes a rigid sentence-level structure that leads to inefficient use of context windows, especially so for long-form code inputs. In its place, we introduce an In-Context Classification (ICC) loss (see Section 2.1) that increases input density, reducing padding and fostering better chunk-level comprehension, essential for downstream tasks such as semantic retrieval or classification.

By integrating these two complementary objectives, MLM and ICC, and distributing their supervision across multiple layers, forming the basis of our *Self-Distillation* mechanism (see Section 2.2), our training approach improves internal representation capabilities and flexibility, ultimately yielding a model better suited for modular deployment and early exiting.




```
FUNCTION build_concatenation(repo_a, repo_b):
    is_negative ← random_choice([True, False])
    sequence ← ""

    WHILE NOT empty(repo_a):
        sequence ← sequence + "<sep_token>"
        # Sampling random positive snippet from repo_a
        pos_snip ← sample_snippet(repo_a)
        IF length(sequence + pos_snip) > max_len:BREAK
         sequence ← sequence + pos_snip

        # Interleaving with negative samples only if
        # the 'is_negative' label is positive
        IF is_negative AND NOT empty(repo_b):
            sequence ← sequence + "<sep_token>"
            # Sampling random negative snippet from repo_b
            neg_snip ← sample_snippet(repo_b)
            IF length(sequence + neg_snip) > max_len:BREAK
            sequence ← sequence + neg_snip

    sequence ← sequence + "<cls_token>"

    RETURN sequence
```

Figure 2: The illustration on the left depicts the in-context loss framework, where samples from various repositories are concatenated. Positive examples originate from the same repository context, whereas negative examples are sourced from different repositories. To enable the model's use of FLASHATTENTION V2, we applied left padding and positioned the CLS token at the end of the sentence. On the right side, you'll find the pseudocode for the in-context loss framework.

2.1 Masked Language Modeling and In-Context Classification

We revisited the NSP loss and introduced an In-Context Classification (ICC) objective. We hypothesize that predicting whether multiple code snippets belong to the same context (in our case, the same repository) can enhance semantic search performance while allowing efficient concatenation of multiple code fragments. Our final training objective is the summation of two losses: (1) MLM loss and (2) ICC loss: $\mathcal{L} = \mathcal{L}_{MLM} + \mathcal{L}_{ICC}$.

In $\mathcal{L}_{MLM}$, a certain percentage of tokens are randomly masked and predicted using a classification head. Following Zhang et al. (2024), we adopt a 15% masking rate with the standard 80-10-10 token replacement strategy (Devlin et al. 2019). The secondary objective, $\mathcal{L}_{ICC}$, classifies whether randomly concatenated inputs (separated by a [SEP] special token) originate from the same repository (see Figure 2). Each concatenated sample has a 50% probability of containing source code from different repositories. This approach increases input density, reducing padding by expanding the average input length from 630 to 1,300 tokens. Since repositories are inherently modular and often contain files written in multiple languages, learning from repository-level context may also improve inter-language understanding.

2.2 Self-Distillation: Multi-Layer loss

To achieve layer-wise modularity in transformer architectures, we apply the previously introduced loss (Section 2.1) across a selected set of layers, sharing classification heads (masked language modeling and in-context classification) *while incorporating a positional embedding of the layer index*. The total loss is computed as the sum of individual layer losses, weighted by a factor α to prioritize deeper layers: $\mathcal{L} = \sum_{i \in \iota} \mathcal{L}_i \cdot \alpha$ where $\alpha = i/|I|$ and $I = \{1, \ldots, 36\}$ represents all layers, and the selected subset $\iota = \{4, 9, 18, 27, 36\}$ defines the layers where the loss is applied. The selected subset was chosen to enable four model variants equally spaced in depth (9, 18, 27, 36). We also train a "tiny" single-exit baseline to enable isolating the Self-Distillation effect. The *Self-Distillation* mechanism illustrated in Figure 1 allows for flexible model deployment, enabling adaptive layer pruning with minimal impact on performance.

3 Pre-training

We built MOSE on top of STARCODER-2 (Lozhkov et al. 2024), applying several modifications to the model. We reduced its original size, resulting in a 1B-parameter model. Our architecture comprises 36 hidden layers and adopts Grouped Query Attention (GQA) (Ainslie et al. 2023) with 16 attention heads and 4 key-value heads. MOSE relies upon Rotary Positional Encoding (RoPE) (Su et al. 2021) with a base period $\theta = 10^6$ and features a hidden dimensionality of 1,024 with an intermediate size of 12,288.

To make our model class bidirectional, we removed the causal masking from the self-attention operations within STARCODER-2. Aiming for modularity, we also replaced sliding window attention with full attention. This step was taken to avoid the receptive field phenomenon of the sliding window mechanisms (Zhu et al. 2021). Finally, our implementation integrates FLASHATTENTION V2 (Dao 2023) for faster inference. We have included and summarized additional details about the model architecture in the Appendix.

	CodeSearchNet								CT	POJ104
Model	Ruby	JS	Go	Python	Java	PHP	avg. MRR	avg. NDCG	MRR	mAP
MoSE	74.1	**74.0**	82.5	**92.5**	**78.7**	**84.5**	**81.0**	**84.2**	**98.9**	**75.9**
CodeT5+	**78.0**	71.3	**92.7**	75.8	76.2	70.1	77.4	-	98.4	24.5
UniXcoder	74.0	68.4	91.5	72.0	72.6	67.6	74.4	-	97.6	41.0
ModernBERT-large	-	-	-	-	-	-	-	59.5	93.1	27.3
OpenAI Embedding	84.7	85.3	95.9	99.8	90.1	95.6	91.9	93.3	98.8	82.9

Table 1: Performance of different models on text-to-code and code-to-code tasks using the CodeXGLUE benchmark suite, specifically in terms of MRR (on CodeSearchNet and CodeTranslation CT dataset), NDCG, and mAP (on POJ104). On CodeSearchNet, MoSE surpasses all existing open-source models, achieving an improvement of 3.6% compared to the best-performing open model, CodeT5+. Furthermore, MoSE significantly reduces the performance gap between open-source models and closed-source alternatives, such as OpenAI text-embedding-3-large referred to as OpenAI Embedding, and performs comparably on the CT dataset. MoSE demonstrates competitive results with the CT benchmark and also outperforms all open-source models on the POJ104 benchmark.

3.1 Training Details

We pre-trained MoSE with a batch size of 4M tokens with a maximum context length of 2,048 tokens, for 245,000 training steps, processing approximately 1T tokens from TheStackV2 (Lozhkov et al. 2024) dataset.

We used the AdamW optimizer with β_1 set to 0.9, β_2 to 0.95, ϵ to $1e{-6}$, and a weight decay of $1e{-1}$. We initialized the learning rate at $6.24e{-4}$ and decreased it using a multi-step learning rate scheduler (Bi et al. 2024) with 4,000 warmup steps. The learning rate was reduced at 120,000, 185,000, 220,000, 230,000, and 240,000 training steps, applying a decay factor of 0.36, and from step 185,000 onward, further reduced by factors of 0.1, 0.031, 0.01, and 0.001. We conducted pre-training and fine-tuning on 512 NVIDIA Ampere (64GB) GPUs, requiring 450,000 GPU hours.

4 Downstream Tasks

4.1 Fine-tuning for Retrieval, text-to-code and code-to-code search

Using the SynthCoNL dataset detailed in Section 5, we leveraged an instruction prompting approach (Su et al. 2023) and trained a single model for both text-to-code and code-to-code retrieval tasks. The optimization objective uses CLIP-style loss (Radford et al. 2021) with the same multi-layer approach discussed in Section 2.2, modified to enhance representation learning. Accordingly, we replaced the single-head projection of the multi-layer loss with five distinct projection heads, applied at different exit points of the pre-trained model (layers 4, 9, 18, 27, and 36). We used a batch of 2,048 examples, ensuring that text-to-code and code-to-code were equally distributed across the batch.

We performed data augmentation by randomly replacing frequently occurring words (appearing more than twice and having at least three characters) with random strings. We applied the augmentation exclusively to code snippets in 30% of cases, leaving natural language descriptions unchanged. After conducting a grid search, we selected $1e{-5}$ as the learning rate, maintained throughout the fine-tuning process, and set the temperature parameter at 10.0.

	BigCloneBench		
Model	Recall	Precision	F1
MoSE (*L4*)	96.4	89.8	93.0
MoSE (*L9*)	**96.6**	90.4	93.4
MoSE (*L18*)	96.4	**92.1**	**94.2**
MoSE (*L27*)	96.5	91.8	94.1
MoSE (*L36*)	96.5	91.7	94.1
UniXcoder	92.9	**97.6**	**95.2**
CodeBERT	94.7	93.4	94.1
CodeT5+ (770M)	**96.7**	93.5	95.1

Table 2: BigCloneBench code clone detection performance in a classification setup. MoSE maintains good performance (%) throughout different layers (L) with results on par with open models.

4.2 Fine-tuning for classification, code clone detection

In addition to fine-tuning the model for multiple retrieval tasks as discussed in Section 4.1, we also explored how our methodology performs in a classification setup. We tested how the Multi-Layer and the In-Context classification losses adapt to the code clone detection task by: (1) fine-tuning multiple classification heads, one for each of the five exit points (layers 4, 9, 18, 27, and 36), and (2) leveraging the model's ability to understand relationships between code snippets by inputting pairs of (hypothetical clone) code segments and training it to classify whether they are clones. We followed the input pattern devised in Section 2.1, formatting the input as follows: [SEP] *snippet-1* [SEP] *snippet-2* [CLS], and then used the final [CLS] token representation as input to a classifier. We fine-tuned the model with a constant learning rate of $1e{-5}$ with 2,000 warmup steps, and used a batch size of 64 elements and trained for 14,000 steps.

5 SynthCoNL

We built SynthCoNL, a dataset that supports training and evaluation of encoders in text-to-code and code-to-code

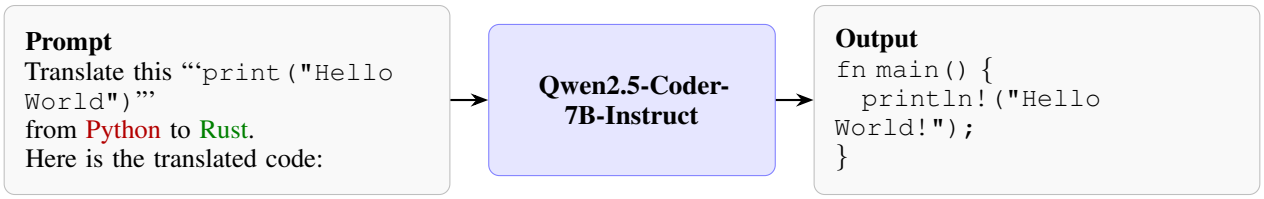


Figure 3: Prompt provided to QWEN2.5-CODER-7B-INSTRUCT for translating a given code snippet (`print("Hello World")` in the example) from a source programming language (`Python`) to a target one (`Rust`).

search, under multiple languages. Using the popular CODESEARCHNET (Husain et al. 2019) as a seed dataset and selecting popular programming languages (Python, Java, Go, and PHP), we augmented it by transpiling available code snippets onto other languages.

To generate semantically similar code snippets for code-to-code search, we translated each snippet into a different language randomly sampled from Go, Ruby, Python, Java, C++, PHP, C, JavaScript. We prompted the QWEN2.5-CODER-7B-INSTRUCT model with the source code, the name of the source language, and the name of the target language (see Figure 3). During code translation, we carried out a greedy sampling so as to prevent semantic discrepancies. This process yielded pairs of code snippets in distinct languages tied to the same natural language description. As a result, every sample in the fine-tuning dataset includes a natural language description and two code snippets from distinct languages. SYNTHCONL contains 1,071,367 triplets where, in the first code column, we directly sampled code snippets from CODESEARCHNET, including Python, Java, PHP, and Go. The second code column, artificially synthesized via code translation, includes Go, Ruby, JavaScript, Python, C++, PHP, C, and Java code snippets. After a manual inspection, we discovered that both columns contained code snippets that differed only in identifiers or function arguments. Several tasks were semantically identical but paraphrased with different parameter requirements (e.g., two identical paraphrased tasks asked for opening a socket on a different port). During the preprocessing phase of SYNTHCONL, motivated by the dataset's redundancy and preliminary experiments that show its effectiveness on the model's performance, we near-deduplicated the dataset using both the CODESEARCHNET code column and the synthesized code column. During the data near deduplication phase, we relied on Locality Sensitive Hashing (LSH) with a Jaccard similarity threshold of 0.7 and 256 permutations, analyzing character-level 5-grams.

6 Results and Discussion

6.1 Evaluation

We evaluated MOSE, after fine-tuned for retrieval, on diverse code understanding tasks, leveraging the CODEXGLUE (Lu et al. 2021) group of benchmarks.

On CODESEARCHNET dataset, we assessed cross-modal (natural language to code) learning by retrieving target code from 999 distractors using natural language queries. For code-to-code retrieval, again against 999 distractors, we utilized CODEXGLUE's Code Translation (CT) dataset to test cross-lingual retrieval (e.g., Java to C# snippets implementing the same functionality), and its POJ104 dataset (C++ snippets, semantically equivalent but syntactically different) for intra-lingual semantic search, evaluating generalization over structural differences while preserving semantics. In our retrieval performance evaluation, when comparing MOSE with other encoders, we included results from OPENAI'S TEXT-EMBEDDING-3-LARGE as a representative commercial, closed-source model.

Additionally, MOSE was fine-tuned and evaluated on the separate BIGCLONEBENCH (Svajlenko and Roy 2015) dataset for Java code clone detection (a binary classification task), assessing its classification performance.

6.2 Benchmarks

Table 1 presents the results for the CODESEARCHNET (text-to-code) task in terms of Mean Reciprocal Rank (MRR, higher is better) for each single language, average Normalized Discounted Cumulative Gain (NDCG, higher is better), and average MRR. Results for UNIXCODER, MODERNBERT, and CODET5+ are reported from the original papers (Guo et al. 2022; Warner et al. 2024; Wang et al. 2023). On CODESEARCHNET, MOSE achieves an MRR of 81.0 and a NDCG of 84.2, outperforming CODET5+ (770M), UNIXCODER, and MODERNBERT-LARGE. The only encoder that surpasses MOSE is OPENAI'S TEXT-EMBEDDING-3-LARGE.

In Table 1, we also present results from both POJ104 and CT datasets (code-to-code) reported respectively in terms of MRR for code translation (Java to C# retrieval) and mean average precision for POJ104 (C++ to C++ retrieval). MOSE reaches the best performance among the evaluated models. We further replicated the benchmarking for all models in a zero-shot setting for code-to-code tasks, as our model does not integrate POJ104 and the code translation datasets in the training set. We present a comprehensive and detailed analysis of code retrieval errors in the Appendix.

On the POJ104 dataset, in zero-shot, MOSE achieves a mean Average Precision (mAP, higher is better) of 75.9, which is the highest among open-source models to our knowledge. However, it is significantly behind OPENAI TEXT-EMBEDDING-3-LARGE. We underscore that a direct comparison with OPENAI TEXT-EMBEDDING-3-LARGE remains challenging because it is closed-source, and details such as model size, training methodology, or potential test data leakage are undisclosed. We observe a difference in performance profiles depending on whether the task is framed as text-to-code or code-to-code retrieval. As shown in Figure 1b,

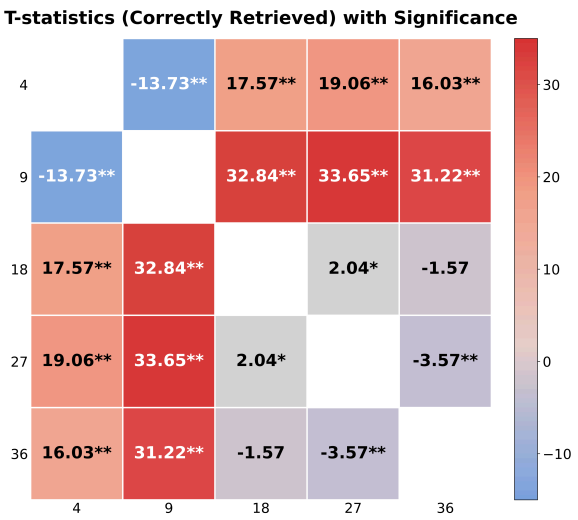


Figure 4: Similarity score Heatmap presenting results from permutation tests (10,000 permutations with $\alpha = 0.05$) across different exit points for Python-to-Java retrieval. * indicates p < 0.05 and ** indicates p < 0.001. Different layers, despite a common training objective, yield different similarity scores.

shallow layers yield better performance in the code-to-code setting, whereas for text-to-code retrieval, optimal performance emerges at layer 18.

This representation divergence suggests that the input modality influences where semantic alignment is most effectively captured within the model. This observation aligns with Valeriani et al. (2023) findings, underlining that *semantic information is distributed unevenly across layers* and is better expressed in the intermediate ones. MoSE leverages this property by supporting dynamic exit points, enabling users to balance computational cost and task performance based on where the most relevant representations emerge.

To further investigate this phenomenon, we compared similarity scores across layers to assess how each layer represents code snippets differently. Figure 4 shows permutation test results for a Python-to-Java retrieval task, highlighting significant variations in similarity scores that support our previous findings. A detailed analysis of these results is provided in the Appendix.

In Table 2, we report classification performance across different exit layers of MoSE for the BigCloneBench benchmark. Results show stable performance throughout layers, with F1-scores from Layer 18 onward (94.2–94.1) closely matching those of deeper exits. Earlier layers (e.g., Layers 4 and 9) exhibit slightly lower precision, reflecting the expected trade-off between early inference and representational depth.

Our method remains competitive compared to open models. While CodeT5+ (770M) and UniXcoder achieve peaks F1 of 95.1 and 95.2, MoSE delivers on-par results (-0.9) by Layer 18, with the added benefit of a modular architecture.

CoIR (NDCG@10)				
Model	CSN-CCR	CoSQA	CT-DL	CT-Contest
MoSE (*ICC*)	**10.1**	**0.4**	**32.0**	**12.1**
MoSE (*NSP*)	6.2	0.4	31.0	12.0

Table 3: Ablation study comparing pre-training objectives (In-Context Classification vs. Next Sentence Prediction), evaluated with NDCG@10 (%) on the CoIR suite. ICC demonstrates gains on the cross-context CSN-CCR task.

6.3 Ablation Study

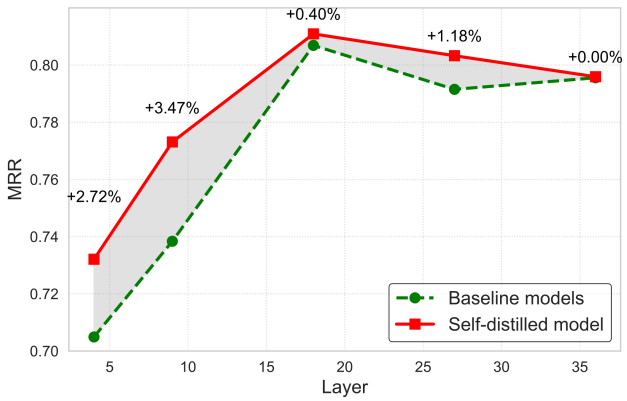
Effectiveness of the Multi Layer Loss We conducted an ablation study for the multi-layer loss, by fine-tuning each exit point independently on SynthCoNL, starting from the pre-trained MoSE. For example, for the baseline on layer 18, we retain the first 18 layers and fine-tune the model using just the projection head on that layer. Finally, we compared the baseline models with the corresponding results (Self-Distilled) of MoSE fine-tuned for retrieval with the multi-layer loss. All tests are conducted on the CodeSearchNet dataset. MoSE consistently outperforms the single-exit baselines (up to +4.36% averaged Recall@1 gain on layer 9), indicating that *lower-level layers benefit from training signals propagated from deeper layers*. This behavior is highlighted in Figure 5 where MoSE, indicated as *Self-Distilled*, outperforms all the single exit baselines consistently. This finding underscores a promising new direction in Self-Distillation for large-scale code and text models, enabling high performance even in more compact configurations. Moreover, Figure 5 also illustrates that MoSE maintains robust performance from layers 18 to 36, allowing users to scale down the network to match their memory, computational, or latency constraints while preserving strong retrieval performances.

Effectiveness of the In-Context Classification Objective We conducted an ablation study, detailed in Table 3, to assess whether our In-Context Classification (ICC) loss promotes inter-language understanding. For this study, we pre-trained two small versions of the model (each with 30M parameters) on 3B tokens. We compared the ICC loss with the Next Sentence Prediction (NSP) loss, commonly used in many state-of-the-art encoders (Li et al. 2023; Feng et al. 2020).

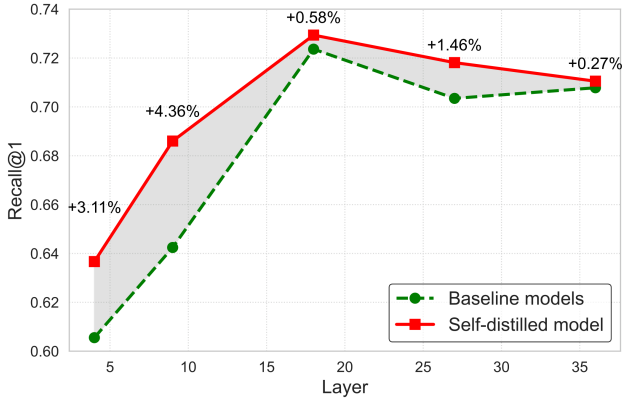
To assess the performance of our models, we utilized the inter-domain section of the CoIR evaluation suite (Li et al. 2025), which focuses on tasks across various languages and domains. Our results indicated that the In-Context Classification loss either outperformed or matched the effectiveness of the Next Sentence Prediction loss, while providing a more efficient and dense representation. These results suggest that ICC may serve as a stronger default pretraining signal than NSP for multilingual and cross-context information retrieval tasks.

7 Related Work

Since the introduction of ELMo (Peters et al. 2018), deep contextual information has enhanced generating embeddings for textual retrieval or classification, reaching state-of-the-art re-



(a) MRR × Embedding depth



(b) Recall@1 × Embedding depth

Figure 5: Performance Comparison between MoSE fine-tuned with multi-layer loss (self-distilled model) and baselines fine-tuned just on single exit points: The graph illustrates averaged MRR and Recall@1 results for different layers over the CodeSearchNet test set. Layers fine-tuned with Multi-Layer loss outperforms the baselines (up to +4.36% Recall@1).

sults in several tasks. BERT (Feng et al. 2020) followed those findings, adapting the Transformer architecture (Vaswani et al. 2017) to enable a bi-directional representation with two different training objectives, namely the masked language modeling and the next sentence prediction losses. (Lan et al. 2019) and (Liu et al. 2019) adapted the BERT architecture to obtain an enhanced pre-trained model by removing or modifying the NSP, focusing on pre-training data or hyperparameters optimization. More recently, ModernBERT (Warner et al. 2024) tied the gap between modern decoders (Jiang et al. 2023; Hui et al. 2024; Dubey et al. 2024; Touvron et al. 2023; Lozhkov et al. 2024) advancements that rely upon models with an increased number of parameters, trained upon more tokens, and being capable of handling longer contextual information.

In code representation, large language models must be adapted by training them on a curated corpus focused on software and by leveraging code's syntactic and semantic structures, which differ significantly from natural language. Feng et al. (2020) adapted the BERT architecture to produce semantically meaningful embeddings for source code, resulting in CodeBERT. This was accomplished by including more source code in the training set and focusing on a training loss that can leverage bimodal (natural language and code) contextual information (Clark et al. 2020). GraphCodeBERT enhanced CodeBERT (Feng et al. 2020) representations by incorporating data flow graphs, capturing dependencies between variables and operations, and improving tasks like code summarization and clone detection. UniXcoder (Guo et al. 2022) extended this by introducing a unified encoder-decoder framework, integrating abstract syntax trees (ASTs) and data flow information. Wang et al. (2023) expanded these findings with CodeT5+, stressing how multiple losses that leverage code semantics impact the model pertaining. The work incorporated text-code contrastive learning, text-code Matching, and text-code causal LM for better code understanding and generation. When trying to achieve better performance, research has shifted toward models with a high number of parameters. While this trend appears effective from a performance perspective, end users may face computational or memory limitations as LLMs vary from millions to billions of parameters. Sanh et al. (2019) pioneered the introduction of knowledge distillation, using a "teacher" model that guides a smaller model to emulate its behavior. This methodology has been widely adopted and improved upon recently (DeepSeek-AI et al. 2025; Hui et al. 2024), becoming a standard for obtaining high-performing smaller LLMs.

Our work differs from previous work by adapting a modern architecture (StarCoder-2 (Lozhkov et al. 2024)) to a code encoder-only based model and introducing a novel 'Self-Distillation' mechanism. We replace the next sentence prediction loss with an In-Context Classification focused on the repository level and expand the context to 2,048 tokens. Our novel Self-Distillation mechanism improves low-level layers, resulting in a modular transformer architecture without additional teacher models or further data for distillation.

8 Conclusion

We introduced MoSE, a modular, multi-exit encoder that uses Self-Distillation to enhance early-layer representations while achieving competitive results on code understanding benchmarks. By integrating multiple exit points, MoSE enables explicit efficiency-accuracy trade-offs tailored to deployment constraints. Our approach not only allows for modular use but also enhances the alignment between task requirements and representation depth. This improves task-specific layer specialization, providing the end-user with a more adaptable model. Future work will explore combining exit-points and further understanding task type and depth interactions. While our results are encouraging, we acknowledge that computational constraints limited extensive hyperparameter tuning and pre-training ablations, and that fine-tuning with synthetic code generated from code translation has an unclear impact.

Acknowledgments

We acknowledge ISCRA for awarding this project access to the LEONARDO supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA (Italy). This work was supported by Future AI Research (FAIR) PE01, SPOKE 8 on PERVASIVE AI funded by the National Recovery and Resilience Plan (NRRP).

References

Ainslie, J.; Lee-Thorp, J.; de Jong, M.; Zemlyanskiy, Y.; Lebrón, F.; and Sanghai, S. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *arXiv e-prints*, arXiv:2305.13245.

Allal, L. B.; Lozhkov, A.; Bakouch, E.; Blázquez, G. M.; Penedo, G.; Tunstall, L.; Marafioti, A.; Kydlíček, H.; Lajarín, A. P.; Srivastav, V.; et al. 2025. SmolLM2: When Smol Goes Big–Data-Centric Training of a Small Language Model. *arXiv preprint arXiv:2502.02737*.

Bi, X.; Chen, D.; Chen, G.; Chen, S.; Dai, D.; Deng, C.; Ding, H.; Dong, K.; Du, Q.; Fu, Z.; Gao, H.; Gao, K.; Gao, W.; Ge, R.; Guan, K.; Guo, D.; Guo, J.; Hao, G.; Hao, Z.; He, Y.; Hu, W.; Huang, P.; Li, E.; Li, G.; Li, J.; Li, Y.; Li, Y. K.; Liang, W.; Lin, F.; Liu, A. X.; Liu, B.; Liu, W.; Liu, X.; Liu, X.; Liu, Y.; Lu, H.; Lu, S.; Luo, F.; Ma, S.; Nie, X.; Pei, T.; Piao, Y.; Qiu, J.; Qu, H.; Ren, T.; Ren, Z.; Ruan, C.; Sha, Z.; Shao, Z.; Song, J.; Su, X.; Sun, J.; Sun, Y.; Tang, M.; Wang, B.; Wang, P.; Wang, S.; Wang, Y.; Wang, Y.; Wu, T.; Wu, Y.; Xie, X.; Xie, Z.; Xie, Z.; Xiong, Y.; Xu, H.; Xu, R. X.; Xu, Y.; Yang, D.; You, Y.; Yu, S.; Yu, X.; Zhang, B.; Zhang, H.; Zhang, L.; Zhang, L.; Zhang, M.; Zhang, M.; Zhang, W.; Zhang, Y.; Zhao, C.; Zhao, Y.; Zhou, S.; Zhou, S.; Zhu, Q.; and Zou, Y. 2024. DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. *CoRR*, abs/2401.02954.

Bolya, D.; Huang, P.-Y.; Sun, P.; Cho, J. H.; Madotto, A.; Wei, C.; Ma, T.; Zhi, J.; Rajasegaran, J.; Rasheed, H.; Wang, J.; Monteiro, M.; Xu, H.; Dong, S.; Ravi, N.; Li, D.; Dollár, P.; and Feichtenhofer, C. 2025. Perception Encoder: The best visual embeddings are not at the output of the network. arXiv:2504.13181.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. E. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 1597–1607. PMLR.

Clark, K.; Luong, M.; Le, Q. V.; and Manning, C. D. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Dao, T. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *arXiv e-prints*, arXiv:2307.08691.

DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; Zhang, X.; Yu, X.; Wu, Y.; Wu, Z. F.; Gou, Z.; Shao, Z.; Li, Z.; Gao, Z.; Liu, A.; Xue, B.; Wang, B.; Wu, B.; Feng, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; Dai, D.; Chen, D.; Ji, D.; Li, E.; Lin, F.; Dai, F.; Luo, F.; Hao, G.; Chen, G.; Li, G.; Zhang, H.; Bao, H.; Xu, H.; Wang, H.; Ding, H.; Xin, H.; Gao, H.; Qu, H.; Li, H.; Guo, J.; Li, J.; Wang, J.; Chen, J.; Yuan, J.; Qiu, J.; Li, J.; Cai, J. L.; Ni, J.; Liang, J.; Chen, J.; Dong, K.; Hu, K.; Gao, K.; Guan, K.; Huang, K.; Yu, K.; Wang, L.; Zhang, L.; Zhao, L.; Wang, L.; Zhang, L.; Xu, L.; Xia, L.; Zhang, M.; Zhang, M.; Tang, M.; Li, M.; Wang, M.; Li, M.; Tian, N.; Huang, P.; Zhang, P.; Wang, Q.; Chen, Q.; Du, Q.; Ge, R.; Zhang, R.; Pan, R.; Wang, R.; Chen, R. J.; Jin, R. L.; Chen, R.; Lu, S.; Zhou, S.; Chen, S.; Ye, S.; Wang, S.; Yu, S.; Zhou, S.; Pan, S.; Li, S. S.; Zhou, S.; Wu, S.; Ye, S.; Yun, T.; Pei, T.; Sun, T.; Wang, T.; Zeng, W.; Zhao, W.; Liu, W.; Liang, W.; Gao, W.; Yu, W.; Zhang, W.; Xiao, W. L.; An, W.; Liu, X.; Wang, X.; Chen, X.; Nie, X.; Cheng, X.; Liu, X.; Xie, X.; Liu, X.; Yang, X.; Li, X.; Su, X.; Lin, X.; Li, X. Q.; Jin, X.; Shen, X.; Chen, X.; Sun, X.; Wang, X.; Song, X.; Zhou, X.; Wang, X.; Shan, X.; Li, Y. K.; Wang, Y. Q.; Wei, Y. X.; Zhang, Y.; Xu, Y.; Li, Y.; Zhao, Y.; Sun, Y.; Wang, Y.; Yu, Y.; Zhang, Y.; Shi, Y.; Xiong, Y.; He, Y.; Piao, Y.; Wang, Y.; Tan, Y.; Ma, Y.; Liu, Y.; Guo, Y.; Ou, Y.; Wang, Y.; Gong, Y.; Zou, Y.; He, Y.; Xiong, Y.; Luo, Y.; You, Y.; Liu, Y.; Zhou, Y.; Zhu, Y. X.; Xu, Y.; Huang, Y.; Li, Y.; Zheng, Y.; Zhu, Y.; Ma, Y.; Tang, Y.; Zha, Y.; Yan, Y.; Ren, Z. Z.; Ren, Z.; Sha, Z.; Fu, Z.; Xu, Z.; Xie, Z.; Zhang, Z.; Hao, Z.; Ma, Z.; Yan, Z.; Wu, Z.; Gu, Z.; Zhu, Z.; Liu, Z.; Li, Z.; Xie, Z.; Song, Z.; Pan, Z.; Huang, Z.; Xu, Z.; Zhang, Z.; and Zhang, Z. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J.; Doran, C.; and Solorio, T., eds., *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 4171–4186. Association for Computational Linguistics.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; Goyal, A.; Hartshorn, A.; Yang, A.; Mitra, A.; Sravankumar, A.; Korenev, A.; Hinsvark, A.; Rao, A.; Zhang, A.; Rodriguez, A.; Gregerson, A.; Spataru, A.; Rozière, B.; Biron, B.; Tang, B.; Chern, B.; Caucheteux, C.; Nayak, C.; Bi, C.; Marra, C.; McConnell, C.; Keller, C.; Touret, C.; Wu, C.; Wong, C.; Ferrer, C. C.; Nikolaidis, C.; Allonsius, D.; Song, D.; Pintz, D.; Livshits, D.; Esiobu, D.; Choudhary, D.; Mahajan, D.; Garcia-Olano, D.; Perino, D.; Hupkes, D.; Lakomkin, E.; AlBadawy, E.; Lobanova, E.; Dinan, E.; Smith, E. M.; Radenovic, F.; Zhang, F.; Synnaeve, G.; Lee, G.; Anderson, G. L.; Nail, G.; Mialon, G.; Pang, G.; Cucurell, G.; Nguyen, H.; Korevaar, H.; Xu, H.; Touvron, H.; Zarov, I.; Ibarra, I. A.; Kloumann, I. M.; Misra, I.; Evtimov, I.; Copet, J.; Lee, J.; Geffert, J.; Vranes, J.; Park, J.; Mahadeokar, J.; Shah, J.; van der Linde, J.; Billock, J.; Hong, J.; Lee, J.; Fu, J.; Chi, J.; Huang, J.; Liu, J.; Wang, J.; Yu, J.; Bitton, J.; Spisak, J.; Park, J.; Rocca, J.; Johnstun, J.; Saxe, J.; Jia, J.; Alwala, K. V.; Upasani, K.; Plawiak, K.; Li, K.; Heafield, K.; Stone,

K.; and et al. 2024. The Llama 3 Herd of Models. *CoRR*, abs/2407.21783.

Egiazarian, V.; Panferov, A.; Kuznedelev, D.; Frantar, E.; Babenko, A.; and Alistarh, D. 2024. Extreme Compression of Large Language Models via Additive Quantization. *arXiv e-prints*, arXiv:2401.06118.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv e-prints*, arXiv:2002.08155.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Cohn, T.; He, Y.; and Liu, Y., eds., *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, 1536–1547. Association for Computational Linguistics.

Guo, D.; Lu, S.; Duan, N.; Wang, Y.; Zhou, M.; and Yin, J. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In Muresan, S.; Nakov, P.; and Villavicencio, A., eds., *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, 7212–7225. Association for Computational Linguistics.

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both Weights and Connections for Efficient Neural Networks. *arXiv e-prints*, arXiv:1506.02626.

Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Dang, K.; Yang, A.; Men, R.; Huang, F.; Ren, X.; Ren, X.; Zhou, J.; and Lin, J. 2024. Qwen2.5-Coder Technical Report. *CoRR*, abs/2409.12186.

Husain, H.; Wu, H.; Gazit, T.; Allamanis, M.; and Brockschmidt, M. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *CoRR*, abs/1909.09436.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv e-prints*, arXiv:1712.05877.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Singh Chaplot, D.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Renard Lavaud, L.; Lachaux, M.-A.; Stock, P.; Le Scao, T.; Lavril, T.; Wang, T.; Lacroix, T.; and El Sayed, W. 2023. Mistral 7B. *arXiv e-prints*, arXiv:2310.06825.

Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; and Liu, Q. 2019. TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv e-prints*, arXiv:1909.10351.

Kusupati, A.; Bhatt, G.; Rege, A.; Wallingford, M.; Sinha, A.; Ramanujan, V.; Howard-Snyder, W.; Chen, K.; Kakade, S.; Jain, P.; and Farhadi, A. 2022. Matryoshka Representation Learning. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 30233–30249. Curran Associates, Inc.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv e-prints*, arXiv:1909.11942.

Li, R.; Ben Allal, L.; 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.; Gontier, N.; Meade, N.; Zebaze, A.; Yee, M.-H.; Umapathi, L. K.; Zhu, J.; Lipkin, B.; Oblokulov, M.; Wang, Z.; Murthy, R.; Stillerman, J.; Sankalp Patel, S.; Abulkhanov, D.; Zocca, M.; Dey, M.; Zhang, Z.; Fahmy, N.; Bhattacharyya, U.; Yu, W.; Singh, S.; Luccioni, S.; Villegas, P.; Kunakov, M.; Zhdanov, F.; Romero, M.; Lee, T.; Timor, N.; Ding, J.; Schlesinger, C.; Schoelkopf, H.; Ebert, J.; Dao, T.; Mishra, M.; Gu, A.; Robinson, J.; Anderson, C. J.; Dolan-Gavitt, B.; Contractor, D.; Reddy, S.; Fried, D.; Bahdanau, D.; Jernite, Y.; Muñoz Ferrandis, C.; Hughes, S.; Wolf, T.; Guha, A.; von Werra, L.; and de Vries, H. 2023. StarCoder: may the source be with you! *arXiv e-prints*, arXiv:2305.06161.

Li, X.; Dong, K.; Lee, Y. Q.; Xia, W.; Zhang, H.; Dai, X.; Wang, Y.; and Tang, R. 2025. CoIR: A Comprehensive Benchmark for Code Information Retrieval Models. arXiv:2407.02883.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *arXiv e-prints*, arXiv:2306.00978.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Lozhkov, A.; Li, R.; Ben Allal, L.; Cassano, F.; Lamy-Poirier, J.; Tazi, N.; Tang, A.; Pykhtar, D.; Liu, J.; Wei, Y.; Liu, T.; Tian, M.; Kocetkov, D.; Zucker, A.; Belkada, Y.; Wang, Z.; Liu, Q.; Abulkhanov, D.; Paul, I.; Li, Z.; Li, W.-D.; Risdal, M.; Li, J.; Zhu, J.; Zhuo, T. Y.; Zheltonozhskii, E.; Osae Osae Dade, N.; Yu, W.; Krauß, L.; Jain, N.; Su, Y.; He, X.; Dey, M.; Abati, E.; Chai, Y.; Muennighoff, N.; Tang, X.; Oblokulov, M.; Akiki, C.; Marone, M.; Mou, C.; Mishra, M.; Gu, A.; Hui, B.; Dao, T.; Zebaze, A.; Dehaene, O.; Patry, N.; Xu, C.; McAuley, J.; Hu, H.; Scholak, T.; Paquet, S.; Robinson, J.; Anderson, C. J.; Chapados, N.; Patwary, M.; Tajbakhsh, N.; Jernite, Y.; Muñoz Ferrandis, C.; Zhang, L.; Hughes, S.; Wolf, T.; Guha, A.; von Werra, L.; and de Vries, H. 2024. StarCoder 2 and The Stack v2: The Next Generation. *arXiv e-prints*, arXiv:2402.19173.

Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C. B.; Drain, D.; Jiang, D.; Tang, D.; Li, G.; Zhou, L.; Shou, L.; Zhou, L.; Tufano, M.; Gong, M.; Zhou, M.; Duan, N.; Sundaresan, N.; Deng, S. K.; Fu, S.; and Liu, S. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In Vanschoren, J.; and Yeung, S., eds., *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Niu, C.; Li, C.; Ng, V.; Chen, D.; Ge, J.; and Luo, B. 2023. An empirical comparison of pre-trained models of source code. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2136–2148. IEEE.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep Contextualized Word Representations. In Walker, M.; Ji, H.; and Stent, A., eds., *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2227–2237. New Orleans, Louisiana: Association for Computational Linguistics.

Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; Krueger, G.; and Sutskever, I. 2021. Learning Transferable Visual Models From Natural Language Supervision. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, 8748–8763. PMLR.

Samsi, S.; Zhao, D.; McDonald, J.; Li, B.; Michaleas, A.; Jones, M.; Bergeron, W.; Kepner, J.; Tiwari, D.; and Gadepally, V. 2023. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference. In *IEEE High Performance Extreme Computing Conference, HPEC 2023, Boston, MA, USA, September 25-29, 2023*, 1–9. IEEE.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv e-prints*, arXiv:1910.01108.

Su, H.; Shi, W.; Kasai, J.; Wang, Y.; Hu, Y.; Ostendorf, M.; Yih, W.; Smith, N. A.; Zettlemoyer, L.; and Yu, T. 2023. One Embedder, Any Task: Instruction-Finetuned Text Embeddings. In Rogers, A.; Boyd-Graber, J. L.; and Okazaki, N., eds., *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, 1102–1121. Association for Computational Linguistics.

Su, J.; Lu, Y.; Pan, S.; Murtadha, A.; Wen, B.; and Liu, Y. 2021. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv e-prints*, arXiv:2104.09864.

Svajlenko, J.; and Roy, C. K. 2015. Evaluating clone detection tools with BigCloneBench. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 131–140.

Teerapittayanon, S.; McDanel, B.; and Kung, H. T. 2017. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. *arXiv e-prints*, arXiv:1709.01686.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Canton-Ferrer, C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR*, abs/2307.09288.

Valeriani, L.; Doimo, D.; Cuturello, F.; Laio, A.; Ansuini, A.; and Cazzaniga, A. 2023. The geometry of hidden representations of large transformer models. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *arXiv e-prints*, arXiv:1706.03762.

Wang, Y.; Le, H.; Gotmare, A.; Bui, N. D. Q.; Li, J.; and Hoi, S. C. H. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, 1069–1088. Association for Computational Linguistics.

Warner, B.; Chaffin, A.; Clavié, B.; Weller, O.; Hallström, O.; Taghadouini, S.; Gallagher, A.; Biswas, R.; Ladhak, F.; Aarsen, T.; Cooper, N.; Adams, G.; Howard, J.; and Poli, I. 2024. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. *CoRR*, abs/2412.13663.

Zhang, D.; Ahmad, W.; Tan, M.; Ding, H.; Nallapati, R.; Roth, D.; Ma, X.; and Xiang, B. 2024. Code Representation Learning At Scale. *arXiv e-prints*, arXiv:2402.01935.

Zhu, C.; Ping, W.; Xiao, C.; Shoeybi, M.; Goldstein, T.; Anandkumar, A.; and Catanzaro, B. 2021. Long-Short Transformer: Efficient Transformers for Language and Vision. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 17723–17736.