Deriving Coding-Specific Sub-Models from LLMs using Resource-Efficient Pruning

Laura Puccioni[¶*], Alireza Farshin[†], Mariano Scazzariello[‡], Changjie Wang[§], Marco Chiesa[§], Dejan Kostić[§]
[¶]Spotify [†]NVIDIA [‡]RISE Research Institutes of Sweden [§]KTH Royal Institute of Technology

***Abstract*—Large Language Models (LLMs) have demonstrated their exceptional performance in various complex code generation tasks. However, their broader adoption is limited by significant computational demands and high resource requirements, particularly memory and processing power. To mitigate such requirements, model pruning techniques are used to create more compact models with significantly fewer parameters. However, current approaches do not focus on the efficient extraction of programming-language-specific sub-models. In this work, we explore the idea of efficiently deriving coding-specific sub-models through unstructured pruning (*i.e.,* Wanda). We investigate the impact of different domain-specific calibration datasets on pruning outcomes across three distinct domains and extend our analysis to extracting four language-specific sub-models: Python, Java, C++, and JavaScript. We demonstrate that it is possible to efficiently extract programming-language-specific sub-models using appropriate calibration datasets while maintaining acceptable accuracy w.r.t. full models. We are also the first to provide analytical evidence that domain-specific tasks activate distinct regions within LLMs, supporting the creation of specialized sub-models through unstructured pruning. We believe that this work has significant potential to enhance LLM accessibility for coding by reducing computational requirements to enable local execution on consumer-grade hardware, and supporting faster inference times critical for real-time development feedback.**

***Index Terms*—Large Language Models, LLMs, pruning, code**

I. Introduction

Large Language Models (LLMs) have become the forefront of technological advancements, making significant strides in various fields thanks to their exceptional performance across diverse and complex tasks. Recently, LLMs have been extensively used in code generation [1], [2], either acting as co-pilots (*i.e.,* assisting developers) [3], [4] or even automating parts of the development process [5], [6]. However, despite their growing use in code-related tasks, their broader adoption is limited by significant computational demands and high resource requirements of forefront models, especially in terms of memory and processing power [7]. Over the past year, model sizes have continuously increased, with Llama-3.1, the largest open-source model, reaching 405 billion parameters [8]. As the size of these models continues to grow, with GPT-5 allegedly exceeding 10 trillion parameters [9], the challenge of efficiently inferencing becomes significant, especially as LLMs deployment poses an ever-increasing energy demand that may soon become unsustainable [10].

One model does not fit all. General-purpose LLMs are trained on vast datasets spanning multiple domains, with their parameter count carefully set to optimize the balance in the loss function, largely influenced by the volume of training data. However, certain applications (*e.g.,* coding or math) do not require broad knowledge but instead benefit from expertise in a specific domain. For example, developers might find a model specialized in a particular programming language, such as Python, more helpful for coding tasks, without needing knowledge on unrelated subjects like physics or chemistry. In such instances, a domain-specific sub-model is ideal, as it can maintain essential human language understanding along with specialized domain expertise.

Efficiency via compression. In this context, model compression techniques have proven effective. These approaches aim to produce a compact/sparse model with significantly fewer parameters, enhancing deployment efficiency in terms of both cost and resource usage. One prominent approach in this domain is pruning, which involves removing unnecessary parameters from a model without significantly degrading its performance [11]. However, many existing pruning methods [12], [13] are not specifically devised for LLMs, causing significant inaccuracies, and often require retraining, making them less feasible for models of such scale and complexity. Such limitations have prompted the need for more tailored approaches that can address the specific challenges posed by these models. Wanda [14] presents a novel technique for pruning LLMs by leveraging a small set of calibration data to estimate activation norms. This method efficiently identify less critical weights/parameters in the model, allowing for pruning with lower computational/memory demands and minimal accuracy drops of the pruned LLM. Importantly, Wanda achieves this without the need to retrain or fine-tune the full model. Although Wanda offers several advantages, it is primarily designed to extract general-purpose sub-models that maintain strong performance across multiple tasks, rather than focusing on domain-specific sub-models. Furthermore, authors do not investigate the impact of calibration sample selection on pruning outcomes, relying exclusively on samples from the C4 dataset [15], a broad English-language text source.

Scope of the paper and contributions. In this paper, we aim to explore whether resource-efficient pruning techniques (*i.e.,* Wanda) can be successfully applied to extract domain-specific sub-models without requiring retraining. We also investigate how the selection of various domain-specific calibration samples impacts pruning results. Our primary focus is on sub-model extraction for code generation, specifically Python, Java,

[*]Work performed while at RISE Research Institutes of Sweden.

C++, and JavaScript. To further validate our approach, we also extract sub-models in the domains of math, common-sense reasoning (CSR), and language translation. Our results demonstrate that, to the best of our knowledge, we are the first ones to successfully extract programming-language-specific sub-models from foundational models using appropriate calibration sets, and that such sub-models retain acceptable accuracy w.r.t. full models. Furthermore, we validate that sub-models extracted using domain-specific datasets outperform those derived from unrelated domain datasets in specialized tasks (*e.g.,* code generation). Lastly, we provide the first analytical evidence that domain-specific tasks activate different regions within LLMs when applying resource-efficient unstructured pruning.

Impact. We believe that domain-specific sub-models offer several impactful advantages for coding-related tasks: (i) compact models reduce computational demands, making it feasible to run them locally on consumer-grade hardware, which broadens access to LLMs and democratizes their use for developers with limited resources; (ii) by employing local, domain-specific sub-models for critical domains (*e.g.,* military defense systems or cybersecurity), companies can ensure that proprietary or confidential data remains in-house, minimizing reliance on external APIs and enhancing data security; (iii) sub-models enable faster inference times, which is particularly valuable for tasks requiring real-time feedback, such as iterative coding/debugging; (iv) our approach supports an always-updated collection of specialized models that can be maintained independently, simplifying the rollout of improvements or domain-specific updates without compromising the accuracy of other models; and (v) our method allows individual sub-models to be fine-tuned or adjusted without altering an entire general-purpose LLM, offering flexibility to create highly customized sub-models that closely meet project-specific needs.

II. Related Work

LLMs are decoder-only models, meaning they only utilize the decoder part of the Transformer architecture [16]. In such models, input tokens are transformed into embeddings that are directly fed into the decoder, which processes them through a series of matrix multiplications involving a large number of parameters. Given the high computational demand of these multiplications, significant research efforts have focused on model compression, which aims to transform large, resource-intensive models into more compact versions [17]. These smaller models are suitable for deployment on devices with less sophisticated hardware and are optimized for faster execution with minimal latency [18].

Model compression approaches. In recent research efforts, five primary model compression approaches have emerged: (i) quantization, (ii) knowledge distillation, (iii) Neural Architecture Search (NAS), (iv) low-rank factorization, and (v) pruning. Each of these methods offers unique advantages and challenges, and are complementary. Quantization is a compression method that reduces model size by representing weights and activations with lower-bit representations, decreasing computational complexity and storage requirements [19]. The most common approach is post-training quantization, which converts pretrained weights to lower bit precision [20]. However, quantization may introduce errors that impact the model's ability to generalize or maintain performance across varied tasks, lowering overall accuracy. Low-rank factorization compresses models by decomposing weight matrices into lower-rank matrices, minimizing redundancy and facilitating faster computations by parallelizing memory access of dense matrices [21], [22]. Knowledge distillation transfers knowledge from a large model (teacher) to a smaller model (student), aiming to replicate the teacher's behavior with fewer resources [23]–[25]. NAS is similar to knowledge distillation and often involves training a super-network once, enabling the sampling of sub-networks through the weight-sharing principle [26], [27]. Low-rank factorization, knowledge distillation, and NAS all involve retraining or significant modifications to the model architecture, making them challenging to implement for LLMs. Among these approaches, pruning has proven to offer the best trade-off between computational gains and minimal loss in accuracy. It aims to reduce the size or complexity of a model systematically, eliminating redundant or less influential parameters, thereby reducing the model's computational and storage requirements [28]. In this work, we mainly focus on unstructured pruning, *i.e.,* induce sparsity at a finer granularity by eliminating individual weights rather than entire units or blocks (*i.e.,* neurons or layers) [29].

TABLE I
Qualitative comparison among pruning approaches.

	No Retraining?	Efficient?	Domain Specific?	w/ Calib. Dataset
Magnitude [28]	✗	✓	✗	✗
MaskLLM [30]	✗	✓	✗	✗
SparseGPT [31]	✓	✗	✗	✓
Wanda [14]	✓	✓	✗	✓
D-Pruner [32]	✗	✗	✓	✓

Existing pruning approaches. Several existing works, summarized in Table I, propose various methods for pruning. Magnitude pruning [28] removes low-magnitude weights, assuming these contribute minimally to the model's function, thus improving inference speed. While computationally efficient, it requires network fine-tuning and may prematurely remove weights that may become significant as pruning progresses [33]. This technique works well for smaller models, but it dramatically fails for LLMs even with relatively low levels of sparsity. MaskLLM [30] introduces an approach for pruning LLMs by incorporating mask learning directly into the training process. Unlike post-training pruning, MaskLLM learns structured sparsity patterns during model training, enabling the network to identify and retain essential weights while discarding redundant ones. While MaskLLM effectively reduces model size and computational demands, it requires modifications to the training pipeline. SparseGPT [31] uses a one-shot, layer-wise pruning strategy that avoids retraining by tackling a layer-wise reconstruction problem. This approach compresses each layer independently, aiming for sufficient accuracy by minimizing the error between original and pruned layers. Despite SparseGPT avoids retraining, it requires matrix inversion operations, which,

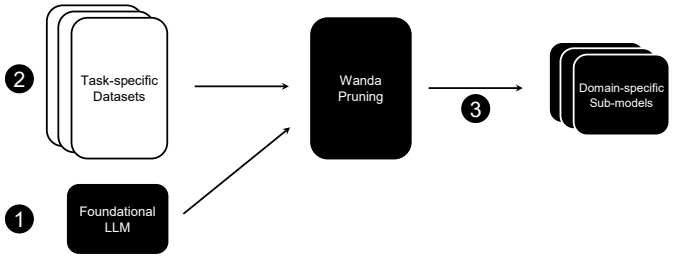


Fig. 1. Our methodology for domain-specific-LLMs extraction.

while contributing to its effectiveness, can pose computational challenges [34]. Wanda [14], which stands for "**W**eights **and** **a**ctivations" is one of the most innovative pruning methods designed for LLMs, balancing computational simplicity with the advantage of avoiding retraining. Wanda evaluates the significance of each weight by computing the product of its magnitude and the norm of the corresponding input activations, estimated using a small set of calibration data. It then "zeros-out" the weights with the lowest products within each layer. While Wanda is effective for creating general-purpose sub-models, authors only rely on the C4 dataset [15] for calibration, without exploring how different calibration samples might influence pruning results. D-Pruner [32] is a novel approach for creating domain-specific LLMs from a foundational model, leveraging full-parameter fine-tuning and gradient updates. Although effective, D-Pruner is memory-intensive due to the full-parameter fine-tuning and has higher computational complexity because of the gradient updates. In contrast, Wanda is more efficient, as it avoids gradient updates, weight reconstruction, and retraining, requiring only a single forward pass.

III. Methodology

Prior works examined how calibration data affect the effectiveness of model compression methods [34]–[37], offering two key insights: (i) no single dataset consistently performs best across all tasks, emphasizing the sensitivity of compression outcomes to the choice of calibration data, and (ii) the optimal size for calibration sets is relatively small, as larger datasets yield minimal performance improvements, suggesting that a carefully selected, compact set is sufficient for effective pruning. D-Pruner [32] is the first work to use task-specific calibration sets to extract domain-specific sub-models, focusing primarily on financial and medical domains, but not addressing programming or code generation tasks. Furthermore, as discussed in §II, D-Pruner relies on full-parameter fine-tuning and gradient updates, which results in high computational demands.

Following the approach shown in Fig. 1, we assess whether combining efficient pruning techniques (*i.e.,* Wanda) with task-specific calibration datasets can effectively generate domain-specific LLMs that maintain strong performance in specialized tasks while minimizing memory and computational demands. We first validate our approach by extracting sub-models in the domains of math, CSR, and language translation. Then, we focus on sub-model extraction for code generation, *i.e.,* Python, Java, C++, and JavaScript.

TABLE II
SELECTED DATASETS FOR PRUNING.

(a)

Domain	Datasets
Code Generation	[42] [43] [44]
Math	[45] [46] [47]
CSR	[44] [48] [49]
Translation	[50] [51] [52]

(b)

Language	Dataset
Python	[42]
Java	[53]
C++	[54]
JavaScript	[53]

Our methodology consists of three main steps, which will be detailed in the following.[1]

❶ **Model selection.** We begin by selecting one or more models for pruning. To ensure the study's validity and broad applicability, we chose a diverse range of models based on factors such as novelty, size, and purpose. First, we selected the latest versions of models (available at the time of writing) to enable fair and relevant comparisons. Second, our focus is on relatively compact models, as our goal is to create domain-specific sub-models that can be efficiently deployed on commodity hardware with limited resources. Finally, models were chosen based on their performance on widely recognized evaluation tasks, ensuring a trustable baseline for fair comparison between the original model and its pruned counterpart. While the original Wanda paper focused solely on Llama-2 [34], this study includes four diverse models, specifically: CodeLlama-7B and CodeLlama-13B [39], Mistral-7B-v0.1 [40], and Gemma-1.1-7B [41]. CodeLlama is fine-tuned specifically for code, while Mistral and Gemma are more general-purpose models. We include two CodeLlama versions to validate our findings on a larger model. All the LLMs were chosen in their "instruct" version, as it is designed to follow natural language instructions more seamlessly.

❷ **Datasets selection.** As mentioned in §II, Wanda utilizes a calibration set to estimate input activations essential for identifying weights to prune, making the quality and selection of these datasets critical for obtaining sub-models with high accuracy. For code generation, we employed: (i) three Python-specific datasets to validate the approach (see Table IIa), and (ii) four datasets covering different programming languages to generalize our findings to other coding tasks, reported in Table IIb. As for math, CSR, and language translation tasks, we chose the three datasets listed in Table IIa.

❸ **Pruning process.** We apply the Wanda pruning technique to the selected models and calibration datasets. Following prior best practices [34], we use 128 samples as calibration data for each dataset considered. Samples are randomly selected with a fixed seed. Given the diverse datasets and models presented in Step ❷, the pruning process is performed iteratively. This iterative approach systematically adjusts parameters related to the model, sparsity type, and datasets, allowing to assess how different configurations affect pruning results. Although our experiments primarily focuses on 50% unstructured sparsity, our takeaways are also applicable to structured pruning (*e.g.,* structured N:M sparsity [55]), as shown by Wanda [14]. After completing the pruning process, we obtain a total of

[1]The code of our methodology is publicly available on GitHub [38].

4 programming-language-specific sub-models for each base model (*i.e.,* from Table IIb) and 13 sub-models for each base model: 12 task-specific sub-models obtained using the datasets in Table IIa and one additional sub-model using the C4 dataset to compare against generic (non-domain-specific) pruning. This approach allows us to highlight similarities among models pruned with datasets from the same domain while also examining differences between models pruned with datasets from different domains.

Sub-models accuracy evaluation. The accuracy of the obtained sub-models is then evaluated using different types of benchmarks and metrics tailored to the considered domain. For the code generation task, we evaluate performance using the *pass@k* metric [56] on two standard benchmarks [56], [57]. The *pass@k* metric evaluates the performance of a model by measuring the probability that at least one of the top k generated solutions successfully meets the expected criteria. Thus, we chose this metric for its ability to assess the functional correctness of generated code, rather than only checking for semantic similarity to the reference solution. Specifically, we set $k = 10$ in our experiments. For CSR, following prior studies, we use a straightforward *zero-shot evaluation* across six datasets [44], [48], [58]–[60]. For the language translation task, we selected the *METEOR* metric [61] and evaluated sub-models on two French-English datasets [50], [52]. *METEOR* matches unigrams between machine-generated and human translations. For each test sample, we generate five translations, compare them to the reference, and retain the highest *METEOR* score among the five outputs. For the math task, we selected the GSM8K dataset [62]. To align with evaluation methods in other studies, we calculate accuracy using *8-shot accuracy*, where the model is given eight examples, each with a problem, solution, and a brief reasoning explaining the solution.

It is worth noticing that, although some evaluation datasets are also used as calibration datasets, we use separate splits to prevent overlap. In this way, we minimize the risk of overfitting, preserving the validity of the results.

Sub-models comparison. Our goal is to investigate how various calibration datasets influence pruning outcomes and to determine whether pruning techniques can effectively create domain-specific sub-models. To validate our hypothesis, we conduct a fine-grained comparison of the parameter masks of each layer of the pruned models. We first use the *penzai* tool [63] to visually represent the weight matrices of pruned sub-models. This visualization method allows us to clearly depict which weights were preserved and which were discarded across sub-models after pruning. By overlaying these matrices from sub-models belonging to the different domains, we aim to identify significant differences that could highlight the impact of the calibration datasets. Following a qualitative analysis, we validate our findings using a per-layer Jaccard distance measurement [64]. This metric allows us to quantify the similarity of retained weights across layers, allowing us to quantitatively evaluate the differences between sub-models within the same domain and those across different domains, highlighting differences based on domain specificity.

IV. Evaluation and Discussion

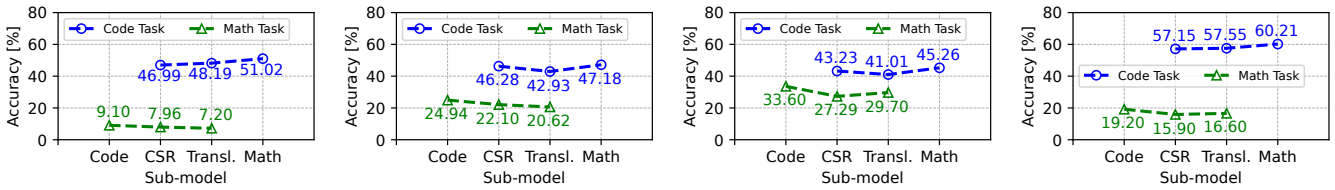
In this section, we discuss the results of the domain-specific sub-models created by pruning the foundational models described in §III with the chosen calibration datasets. We begin by reviewing the performance of sub-models generated for the four selected domains, followed by an analysis of the programming-language-specific sub-models (§IV-A). Additionally, we dig into the internal structure of the obtained sub-models using both visual and analytical metrics (§IV-B).

A. Task-Specific Pruning Evaluation

Pruning with domain-specific calibration sets brings clear benefits. The proposed approach shows an overall improvement over the original Wanda paper's results. Notably, the C4 dataset does not yield the best outcomes for any task across all models and domains assessed. In each case, the top-performing sub-models are those pruned using domain-specific calibration datasets. Table III summarizes the best results achieved for the selected models, showing the sub-model with the highest accuracy and the calibration dataset used. Although the sub-models generally exhibit lower accuracy than the original models, this is an expected trade-off to increase computational efficiency. Nonetheless, there is a consistent pattern across all models: the highest accuracy is obtained with calibration datasets tailored to each task, confirming our intuition that domain-specific datasets are essential for the creation of accurate, domain-specific sub-models.

Robust training yields strong post-pruning results. Looking at the 7B models results, an interesting finding emerges in the code generation task: the Gemma model shows a relatively smaller accuracy drop after pruning compared to code-specific models, *i.e.,* CodeLlama. Although CodeLlama, specifically designed for coding tasks, was expected to preserve most of its original performance, it is actually Gemma that retains a better accuracy, with a drop of ~5.83%, compared to a ~7.47% drop for CodeLlama. We speculate that this result may be due to Gemma's greater resilience to pruning, likely a result of its broader training data. Unlike CodeLlama, which is primarily fine-tuned on coding data [65], Gemma has been exposed to a diverse range of sources, including web documents, code, and math [41]. This wider training data could have helped Gemma develop a deeper understanding of language, enhancing its adaptability in the pruning process. This finding is consistent with previous research [66] suggesting that a wider pre-training knowledge base contributes to better model's robustness. Notably, since CodeLlama has higher accuracy in its base model, its pruned version also outperforms Gemma. We will further investigate whether this trend persists with larger model versions.

Pruning retains strong CSR for user query processing. Unlike other domains, the CSR task exhibits relatively consistent accuracy levels across different calibration datasets, even those from unrelated domains. This finding suggests that the



(a) CodeLlama-7B. (b) Mistral-7B. (c) Gemma-7B. (d) CodeLlama-13B.

Fig. 2. Comparison of code-task accuracy for non-coding sub-models and of math-task accuracy for non-math sub-models.

TABLE III
BEST PERFORMING SUB-MODELS.

(a) CodeLlama-7B.

Task	Best Calib. Dataset	Pre-Prune Accuracy	Post-Prune Accuracy	Accuracy Drop
Code	py_alpaca	60.50%	53.03%	7.47%
CSR	hellaswag	52.30%	49.30%	3.00%
Transl	wmt14	47.48%	44.40%	3.08%
Math	math_inst	17.81%	11.00%	6.81%

(b) Mistral-7B.

Task	Best Calib. Dataset	Pre-Prune Accuracy	Post-Prune Accuracy	Accuracy Drop
Code	py_alpaca	60.50%	47.28%	13.22%
CSR	ob_qa	61.40%	58.20%	3.20%
Transl	wmt14	52.95%	51.31%	1.64%
Math	math_qa	32.90%	27.82%	5.08%

(c) Gemma-7B.

Task	Best Calib. Dataset	Pre-Prune Accuracy	Post-Prune Accuracy	Accuracy Drop
Code	apps	52.90%	47.07%	5.83%
CSR	ob_qa	40.57%	38.90%	1.67%
Transl	o_books	50.41%	48.60%	1.81%
Math	math_o	46.40%	36.16%	10.24%

(d) CodeLlama-13B.

Task	Best Calib. Dataset	Pre-Prune Accuracy	Post-Prune Accuracy	Accuracy Drop
Code	py_alpaca	71.00%	61.00%	10.00%
CSR	hellaswag	54.70%	52.50%	2.20%
Transl	wmt14	51.47%	49.50%	1.97%
Math	math_qa	26.76%	21.68%	5.08%

sub-models may not gain significant benefits from the usage of CSR-specific calibration samples during pruning. The consistent accuracies across sub-models indicate that they possess a robust understanding of common knowledge and everyday reasoning principles, likely due to their training on web documents or datasets rich in general knowledge. Therefore, additional calibration with CSR-specific samples during pruning appears to offer limited benefits. Since each model requires a strong base of common knowledge to effectively process user queries, it is likely that essential weights for CSR are consistently retained across all sub-models.

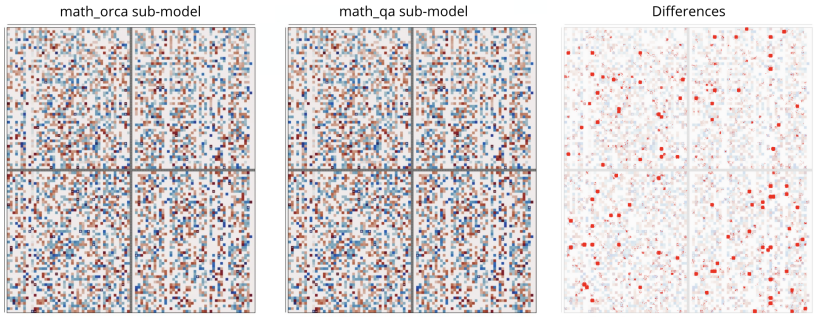
Task complexity influences post-pruning accuracy. In the language translation task, we observe a similar trend to that of code generation: translation-specific sub-models retain higher accuracy, confirming the effectiveness of our approach. Interestingly, for the selected tasks, translation sub-models experience a much smaller accuracy drop ($\leq$3%) than those focused on code generation tasks ($\geq$5%). We hypothesize that this notable difference arises from the fundamental nature of the tasks. Translation typically involves a more direct mapping between input and output sequences across languages, while code generation requires transforming natural language into executable code, which demands a deep understanding of programming logic, syntax, and semantics. Consequently, code-generation sub-models may lose essential information during pruning, leading to the observed larger accuracy drop. This observation also suggests that resulting sub-models could be utilized for performing specific sub-tasks within a domain (*e.g.,* line-by-line code translation between programming languages) rather than broader tasks (*e.g.,* generating complete code blocks). We will further explore whether this accuracy drop varies when applied to other coding tasks, *e.g.,* code completion.

Logical reasoning skills retain cross-task accuracy. We observe that code and math sub-models experience only a slight drop in accuracy when evaluated on tasks from the other domain (*i.e.,* math models on code tasks and vice versa). Fig. 2 shows the accuracy of code (blue line) and math (green line) tasks conducted with non-code and non-math sub-models for the selected 7B models. It is evident that code-specific sub-models perform slightly better on math tasks, and the same applies for math-specific sub-models on coding tasks. This interesting behavior can be explained by the shared skills and similarities between coding and mathematical reasoning. Both areas require logical thinking, problem-solving skills, and a solid understanding of numerical concepts. As a result, sub-models specialized in one domain may retain sub-areas of knowledge relevant to the other, enabling them to achieve higher accuracy levels.

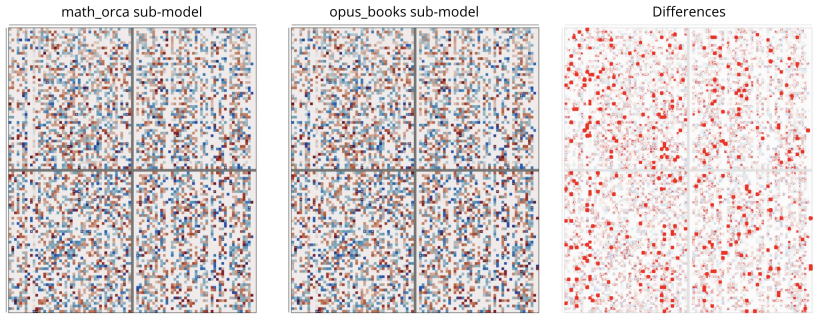
Proper calibration sets extract programming-language-specific sub-models. After evaluating the impact of task-specific calibration sets in pruning, which supports our approach for extracting domain-specific sub-models effectively, we

TABLE IV
LANGUAGE-SPECIFIC SUB-MODELS EVALUATION.

Language	Calib. Dataset	Evaluation Datasets HE (Python)	HE (Java)	HE (C++)	HE (JS)
-	None	71.00%	68.3%	64.1%	67.7%
Python	py_alpaca	**61.00%**	55.2%	54.2%	59.31%
Java	code-java	56.1%	**60.1%**	54.2%	55.6%
C++	cpp_dataset	56.01%	54.2%	**57.1%**	56.3%
Javascript	code-javascript	56.7%	55.7%	50.9%	**60.2%**



(a) Math vs. Math. The differences between the two sub-models' weights are minimal.



(b) Math vs. Translation. A good portion of the weights differs.

Fig. 3. The first two plots (left and center) represent the visualization of the `q_proj` weight matrices of layer 12 of two sub-models obtained by pruning Gemma on two different datasets. The last plot (right) represents the weights that differ between the two matrices (red dots).

explore whether similar outcomes could be achieved with programming-language-specific sub-models. In previous experiments, we solely focused on Python in the code generation task. Here, we expand our approach to extract three additional sub-models: Java, C++, and JavaScript. We select CodeLlama-13B as the base model, given the higher accuracy of its sub-models. Table IV presents the performance of each sub-model pruned with different programming language datasets as calibration sets. The evaluation spans four distinct versions of HumanEval, each tailored to a specific programming language. A significant observation is the strong language specificity of each sub-model: they achieve high accuracy within their own language but show an accuracy drop when applied to other languages. This finding demonstrates that it is possible to effectively extract programming-language-specific sub-models from foundational models with suitable calibration sets, and that such sub-models retain acceptable accuracy w.r.t. base models.

B. Sub-Models Structure Comparison

In order to determine whether our approach is effectively creating domain-specific sub-models, we conducted a comparison of the parameter masks of each layer of the pruned models.

Distinct weight patterns emerge in domain-specific sub-models. We analyzed the internal structure of the obtained domain-specific sub-models to determine whether the pruning method influences their specificity to certain domains. Due to space limitations, we include results only for Gemma-7B, which displays the most significant differences and has fewer layers, providing a clearer visualization. Fig. 3 presents visual representations of the post-pruning weights of the `q_proj` matrix[2] in the 12th layer of the model, generated using the *penzai* tool [63]. Specifically, Fig. 3a compares two math-specific sub-models, while Fig. 3b contrasts one math-specific sub-model with a translation-specific sub-model. In each figure, the left and center plots report weights as colored squares, where darker squares indicate higher values of the corresponding weights, while the right-most plot highlights the differences in weights between the two matrices (red dots). This visualization allows us to qualitatively assess that sub-models from distinct domains exhibit notable disparities compared to those within the same domain, as the number of red dots in Fig. 3a (math vs. math) is markedly lower than the ones in Fig. 3b (math vs. translation).

Quantitative analysis of sub-models weight patterns. We deepen the understanding of weight differences through an analytical per-layer comparison using the Jaccard distance. Fig. 4 depicts the per-layer Jaccard distance of the same Gemma sub-models shown in Fig. 3. Specifically, Fig. 4a reports the comparison between sub-models within the same domain (math), while Fig. 4b compares sub-models from different domains (math and translation). The clear contrast indicates that sub-models within the same domain tend to have more similar weight distributions, while those from different domains show more divergent weight distributions. These results suggest the existence of task-specific weights essential for particular domains, while other weights may be less relevant within those domains. However, these less relevant weights in one domain could be critical for others, explaining the significant

[2] `q_proj` refers to the Query projection matrix in the attention mechanism.

differences observed between models from different domains.

In summary, through our qualitative and quantitative analyses, we demonstrate that domain-specific tasks activate different regions within LLMs, resulting in distinct parameter masks, thereby extending and confirming similar findings from previous studies [32] but focusing on resource-efficient unstructured pruning. Therefore, we find that domain-specific sub-models can be effectively extracted from foundational LLMs using suitable calibration datasets.

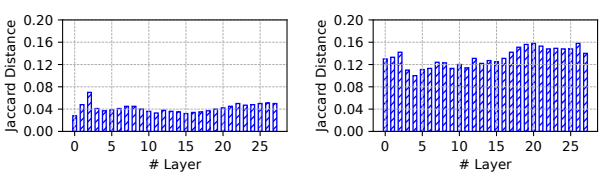
V. Conclusions

In this study, we demonstrate that domain-specific sub-models can be efficiently extracted from foundational LLMs using unstructured pruning (*i.e.,* Wanda). This approach was applied across three distinct domains: math, CSR, and language translation. Our findings reveal that the resulting sub-models maintain acceptable accuracy when compared to their full-model counterparts, and that sub-models derived using domain-specific calibration sets outperform those created with unrelated data, highlighting the importance of selecting appropriate calibration datasets. We are the first ones to extract four programming-language-specific sub-models, further confirming our findings. Furthermore, we provide the first analytical evidence that domain-specific tasks activate unique LLMs regions with unstructured pruning, indicating that certain weights are essential for maintaining domain-specific performance.

Future work could extend these experiments by incorporating additional models of different sizes and exploring a wider range of programming languages. Additionally, based on our insights from code-specific sub-models within the Gemma family, we will further explore how the training phase impacts post-pruning results. Specifically, we aim to determine whether factors such as dataset selection or specific training techniques lead to variations in the performance of sub-models. Finally, we also plan to develop a novel inference system that utilizes the extracted sub-models and dynamically selects the most appropriate according to user requirements (similarly to Composition of Experts [67]).

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and suggestions on this paper. This work has been partially supported by Knut and Alice Wallenberg Foundation (Wallenberg Scholar Grant for Prof. Dejan Kostić), Vinnova (the Sweden's Innovation Agency), the Swedish Research Council (agreement No. 2021-04212), and KTH Digital Futures. We acknowledge EuroHPC Joint Undertaking for awarding us access to the Leonardo supercomputer located at the CINECA data center in Bologna, Italy.



(a) (b)

Fig. 4. Jaccard distance of Gemma pruned on (a) math_qa (math) and math_orca (math), and (b) math_qa (math) and opus_books (translation).

References

[1] R. Ramler *et al.*, "Industrial experience report on ai-assisted coding in professional software development," ser. LLM4Code '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–7. [Online]. Available: https://doi.org/10.1145/3643795.3648377

[2] Fortune, "Over 25% of Google's code is now written by AI," 2024, https://fortune.com/2024/10/30/googles-code-ai-sundar-pichai/.

[3] GitHub, "GitHub Copilot - Your AI pair programmer," 2024, https://github.com/features/copilot.

[4] Tabby, "Tabby - Opensource, self-hosted AI coding assistant," 2024, https://tabby.tabbyml.com.

[5] Cognition, "Devin - Collaborative AI teammate," 2024, https://devin.ai.

[6] Anysphere, "Cursor - The AI Code Editor," 2024, https://www.cursor.com.

[7] M. U. Hadi *et al.*, "A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage," October 2023. [Online]. Available: https://d197for5662m48.cloudfront.net/documents/publicationstatus/170687/preprint_pdf/afaf5dabd52e8f44288e8e800a54f43d.pdf

[8] Meta AI, "Introducing Llama 3.1: Our most capable models to date," 2024, https://ai.meta.com/blog/meta-llama-3-1/.

[9] Twitter, "GPT-5 Model Size Leak," 2024, https://x.com/apples_jimmy/status/1831221448935100482/photo/1.

[10] Evan Halper and Caroline O'Donovan, "AI is exhausting the power grid. Tech firms are seeking a miracle solution." 2024, https://www.washingtonpost.com/business/2024/06/21/artificial-intelligence-nuclear-fusion-climate.

[11] T. Hoefler *et al.*, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, 2021. [Online]. Available: https://arxiv.org/pdf/2102.00554.pdf

[12] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, C. Cortes *et al.*, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf

[13] M. Xia *et al.*, "Structured pruning learns compact and accurate models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan *et al.*, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1513–1528. [Online]. Available: https://aclanthology.org/2022.acl-long.107

[14] M. Sun *et al.*, "A simple and effective pruning approach for large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=PxoFut3dWW

[15] C. Raffel *et al.*, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *arXiv e-prints*, 2019. [Online]. Available: https://arxiv.org/pdf/1910.10683

[16] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[17] Y. Cheng *et al.*, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *arXiv preprint arXiv:1710.09282*, June 2020. [Online]. Available: https://arxiv.org/pdf/1710.09282.pdf

[18] X. Zhu *et al.*, "A Survey on Model Compression for Large Language Models," *arXiv preprint arXiv:2308.07633*, September 2023. [Online]. Available: https://arxiv.org/pdf/2308.07633.pdf

[19] T. Choudhary *et al.*, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, vol. 53, pp. 5113–5155, February 2020. [Online]. Available: https://doi.org/10.1007/s10462-020-09816-7

[20] M. Nagel *et al.*, "A White Paper on Neural Network Quantization," *arXiv preprint arXiv:2106.08295*, June 2021. [Online]. Available: https://arxiv.org/pdf/2106.08295.pdf

[21] M. Jaderberg *et al.*, "Speeding up Convolutional Neural Networks with Low Rank Expansions," *arXiv preprint arXiv:1405.3866*, May 2014. [Online]. Available: https://arxiv.org/pdf/1405.3866.pdf

[22] C. Tai *et al.*, "Convolutional Neural Networks with Low-Rank Regularization," *arXiv preprint arXiv:1511.06067*, February 2016. [Online]. Available: https://arxiv.org/pdf/1511.06067.pdf

[23] J. Gou *et al.*, "Knowledge Distillation: A Survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021. [Online]. Available: https://arxiv.org/pdf/2006.05525

[24] L. Wang *et al.*, "Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 6, pp. 3048–3068, June 2021. [Online]. Available: https://arxiv.org/pdf/2004.05937.pdf

[25] R. Tang *et al.*, "Distilling Task-Specific Knowledge from BERT into Simple Neural Networks," *arXiv preprint arXiv:1903.12136*, March 2019. [Online]. Available: https://arxiv.org/pdf/1903.12136.pdf

[26] T. Elsken *et al.*, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[27] A. Sarah *et al.*, "Llama-nas: Efficient neural architecture search for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2405.18377

[28] S. Han *et al.*, "Learning both Weights and Connections for Efficient Neural Networks," *Advances in neural information processing systems*, October. [Online]. Available: https://arxiv.org/pdf/1506.02626.pdf

[29] E. Kurtic *et al.*, "The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models," *arXiv preprint arXiv:2203.07259*, October 2022. [Online]. Available: https://arxiv.org/pdf/2203.07259.pdf

[30] G. Fang *et al.*, "Maskllm: Learnable semi-structured sparsity for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2409.17481

[31] E. Frantar *et al.*, "SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot," pp. 10 323–10 337, March 2023. [Online]. Available: https://arxiv.org/pdf/2301.00774.pdf

[32] N. Zhang *et al.*, "Pruning as a domain-specific llm extractor," 2024.

[33] Y. Guo *et al.*, "Dynamic Network Surgery for Efficient DNNs," *Advances in neural information processing systems*, vol. 29, 2016. [Online]. Available: https://arxiv.org/pdf/1608.04493

[34] M. Sun *et al.*, "A Simple and Effective Pruning Approach for Large Language Models," *arXiv preprint arXiv:2306.11695*, October 2023. [Online]. Available: https://arxiv.org/pdf/2306.11695.pdf

[35] M. Williams *et al.*, "On the impact of calibration data in post-training quantization and pruning," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACM, 2024, p. 10100–10118. [Online]. Available: http://dx.doi.org/10.18653/v1/2024.acl-long.544

[36] I. Hubara *et al.*, "Accurate Post Training Quantization With Small Calibration Sets," in *Proceedings of the 38th International Conference on Machine Learning*, 2021. [Online]. Available: http://proceedings.mlr.press/v139/hubara21a/hubara21a.pdf

[37] ——, "Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming," December 2020. [Online]. Available: https://arxiv.org/pdf/2006.10518.pdf

[38] L. Puccioni, "Building Domain Specific Sub-Models from LLMs using Pruning," 2025, https://github.com/LaP19/Building-Domain-Specific-Sub-Models-from-Large-Language-Models-using-Pruning.

[39] B. Rozière *et al.*, "Code llama: Open foundation models for code," 2024. [Online]. Available: https://arxiv.org/abs/2308.12950

[40] A. Q. Jiang *et al.*, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[41] G. Team *et al.*, "Gemma: Open models based on gemini research and technology," 2024. [Online]. Available: https://arxiv.org/abs/2403.08295

[42] iamtarun, "python_code_instructions_18k_alpaca," 2023, (Last Accessed: April, 29th 2024). [Online]. Available: https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca

[43] thomwolf, "github-python," 2021. [Online]. Available: https://huggingface.co/datasets/thomwolf/github-python

[44] T. Mihaylov *et al.*, "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering," in *EMNLP*, 2018. [Online]. Available: https://arxiv.org/pdf/1809.02789.pdf

[45] A. Amini *et al.*, "MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms," in *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)*. Association for Computational Linguistics, 2019, pp. 2357–2367. [Online]. Available: https://aclanthology.org/N19-1245

[46] X. Yue *et al.*, "MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning," *arXiv preprint arXiv:2309.05653*, 2023. [Online]. Available: https://arxiv.org/pdf/2309.05653

[47] A. Mitra *et al.*, "Orca-Math: Unlocking the potential of SLMs in Grade School Math," 2024. [Online]. Available: https://arxiv.org/pdf/2402.14830

[48] R. Zellers *et al.*, "HellaSwag: Can a Machine Really Finish Your Sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. [Online]. Available: https://arxiv.org/pdf/1905.07830.pdf

[49] A. Talmor *et al.*, "COMMONSENSEQA: A Question Answering Challenge Targeting Commonsense Knowledge," in *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*. Association for Computational Linguistics, June 2019, pp. 4149–4158". [Online]. Available: https://aclanthology.org/N19-1421.pdf

[50] O. Bojar *et al.*, "Findings of the 2014 Workshop on Statistical Machine Translation," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, June 2014, pp. 12–58. [Online]. Available: http://www.aclweb.org/anthology/W/W14/W14-3302

[51] M. Faysse *et al.*, "CroissantLLM: A Truly Bilingual French-English Language Model," 2024.

[52] J. Tiedemann, "Parallel Data, Tools and Interfaces in OPUS," in *Proceedings of LREC'12*, 2012, pp. 2214–2218. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf

[53] H. Husain *et al.*, "CodeSearchNet Challenge: Evaluating the State of Semantic Code Search," *arXiv preprint arXiv:1909.09436*, 2019. [Online]. Available: https://arxiv.org/pdf/1909.09436

[54] nguyentruong-ins, "nhlcoding_cleaned_cpp_dataset," (Last Accessed: May, 1st 2024). [Online]. Available: https://huggingface.co/datasets/nguyentruong-ins/nhlcoding_cleaned_cpp_dataset

[55] A. Mishra *et al.*, "Accelerating Sparse Deep Neural Networks," 2021. [Online]. Available: https://arxiv.org/abs/2104.08378

[56] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," 2021. [Online]. Available: https://arxiv.org/pdf/2107.03374.pdf

[57] J. Austin *et al.*, "Program Synthesis with Large Language Models," *arXiv preprint arXiv:2108.07732*, August 2021. [Online]. Available: https://arxiv.org/pdf/2108.07732.pdf

[58] "WinoGrande: An Adversarial Winograd Schema Challenge at Scale," 2019. [Online]. Available: https://arxiv.org/pdf/1907.10641.pdf

[59] C. Clark *et al.*, "BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions," in *NAACL*, 2019. [Online]. Available: https://arxiv.org/pdf/1905.10044.pdf

[60] P. Clark *et al.*, "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," *arXiv preprint arXiv:1803.05457*, 2018. [Online]. Available: https://arxiv.org/pdf/1803.05457.pdf

[61] S. Banerjee *et al.*, "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Association for Computational Linguistics, June 2005, pp. 65–72. [Online]. Available: https://aclanthology.org/W05-0909

[62] K. Cobbe *et al.*, "Training Verifiers to Solve Math Word Problems," *arXiv preprint arXiv:2110.14168*, 2021.

[63] P. Authors, "Penzai," (Last Accessed: May, 15th 2024). [Online]. Available: https://penzai.readthedocs.io/en/stable/

[64] Uniqtech, "Understand Jaccard Index, Jaccard Similarity in Minutes," August 2020. [Online]. Available: https://medium.com/data-science-bootcamp/understand-jaccard-index-jaccard-similarity-in-minutes-25a703fbf9d7

[65] B. Rozière *et al.*, "Code Llama: Open Foundation Models for Code," *arXiv preprint arXiv:2308.12950*, 2023. [Online]. Available: https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code/

[66] J. Li *et al.*, "Towards Robust Pruning: An Adaptive Knowledge-Retention Pruning Strategy for Language Models," *arXiv preprint arXiv:2310.13191*, 2023. [Online]. Available: https://arxiv.org/pdf/2310.13191

[67] R. Raju *et al.*, "Samba-CoE v0.1 - Unlocking the power of routing to build a Composition of Experts," (Last Accessed: May, 15th 2024). [Online]. Available: https://sambanova.ai/blog/samba-coe-v01-composition-of-experts