COSMosFL: Ensemble of Small Language Models for Fault Localisation

Hyunjoon Cho
School of Computing
KAIST
Daejeon, Republic of Korea
hyunjoon.cho@kaist.ac.kr

Sungmin Kang
School of Computing
KAIST
Daejeon, Republic of Korea
sungmin.kang@kaist.ac.kr

Gabin An
School of Computing
KAIST
Daejeon, Republic of Korea
gabin.an@kaist.ac.kr

Shin Yoo
School of Computing
KAIST
Daejeon, Republic of Korea
shin.yoo@kaist.ac.kr

***Abstract*—LLMs are rapidly being adopted to build powerful tools and agents for software engineering, but most of them rely heavily on extremely large closed-source models. This, in turn, can hinder wider adoption due to security issues as well as financial cost and environmental impact. Recently, a number of open source Small Language Models (SLMs) are being released and gaining traction. While SLMs are smaller, more energy-efficient, and therefore easier to locally deploy, they tend to show worse performance when compared to larger closed LLMs. We present COSMos, a task-level LLM ensemble technique that uses voting mechanism, to provide a broader range of choice between SLMs and LLMs. We instantiate COSMos with an LLM-based Fault Localisation technique, AutoFL, and report the cost-benefit trade-off between LLM accuracy and various costs such as energy consumption, inference time, and the number of tokens used. An empirical evaluation using Defects4J shows that COSMos can build effective ensembles that can achieve Pareto-optimality in terms of FL accuracy and inference cost, when compared to individual models.**

***Index Terms*—Fault Localization, Ensemble Methods, Small Language Models, Evolutionary Algorithms**

I. Introduction

Large Language Models (LLMs) are rapidly being adopted by software engineers to automate various tasks across the software development lifecycle [1]. While LLMs are essentially autocompletion engines trained on a vast amount of data [2], they have exhibited many useful emergent behaviour, including their capability to perform in-context learning. This has led to many advanced prompting/inference techniques, such as Chain-of-Thought [3], self-consistency [4], and ReAct [5]. Increasingly, these techniques are being used to build LLM-based *agents* [6], [7], [8].

One challenge in broader adoption of these techniques is the dependence on commercial and closed LLMs. In addition to the financial cost of using those models, organisations may not want to reveal their source code as part of any prompt that are fed to external LLMs, for security reasons. Finally, while growing LLM sizes have so far been accompanied with improving performance, there are concerns about the environmental impact that these large models have [9], [10].

Recently, Small Language Models (SLMs) with open source licenses have started gaining traction due to their improving capabilities and ability to be hosted and served more locally, offering an alternative to the larger, closed models and their limitations [11], [12], [13]. However, individual SLMs typically fall short of the comprehensive performance delivered by larger language models like GPT-4 [14], creating a noticeable gap in their stand-alone effectiveness [15]. Consequently, a potential user of LLM-based software engineering technique is presented with two options: high performance with high cost, or low performance with low cost. It would be ideal to have a broader range of choices in the cost-benefit trade-off.

We propose COSMos (**CO**llection of **S**mall Language **Mo**dels), an ensemble of SLMs: it builds upon self-consistency, a prompting technique that states, for logical tasks, it is better to take multiple samples of LLM answers and marginalise them [4]. While self-consistency has been widely accepted as a simple yet effective validation technique that can improve the correctness of LLM-generated solutions [16], [15], [17], it exacerbates the issue of cost due to its need to sample multiple LLM-generated solutions. COSMos aims to both exploit and alleviate challenges that self-consistency introduces. Instead of taking multiple samples from a closed LLM, COSMos forms an ensemble of SLMs and aggregate their answers to form the final solution. In turn, we hope to lower the cost of the self-consistency based inferences by using SLMs, while maintain high FL accuracy.

In this paper, we concretely evaluate COSMos by instantiating it with ensembles of LLM-based Fault Localisation (FL) technique, AutoFL [17], to obtain COSMosFL. We first choose the membership of the ensemble based on the FL performance of individual SLMs. Subsequently, the ensemble of SLMs provide the multiple inference samples that result in the final ranking of the likely faulty methods via voting. We evaluate two ensemble schemes: one where each member SLM has the equal voting power, and another where we optimise the relative voting weights with the aim of improving the resulting FL accuracy. We report not only the FL accuracy of the ensemble, but also various cost measures including number of tokens, inference time, and the overall energy consumption. Our evaluation of COSMosFL using Defects4J [18] shows that ensembles can indeed outperform individual models when the FL task is constrained by energy consumption or token count. Detailed technical contributions are as follows:

- We introduce COSMos, an ensemble of multiple open source SLMs. While ensembles of LLMs have been

proposed for token level decoding, COSMos is the first to introduce task-level voting based ensembles.
- We compare two ensemble methods: a vanilla voting-based ensemble, and a weight-optimised ensemble. The weight optimisation uses Differential Evolution to optimise the weights that are applied to votes cast by membership SLMs.
- We instantiate COSMos with an LLM-based FL technique, AutoFL, to obtain COSMosFL. We evaluate its performance using the widely studied Defects4J benchmark. We report the trade-off between FL accuracy and various costs, such as power consumption, token size, and inference time.

The remainder of the paper is organised as follows. Section II presents AutoFL and the design of COSMos. Section III presents the settings of the empirical evaluation, the results of which are shown in Section IV. Section V discusses details of our findings. Section VI presents related work that are relevant to ours, and Section VII finally concludes.

II. Approach

A. Preliminaries

1) Self-Consistency: Self-consistency is a property of LLM-based reasoning that, for a complex reasoning task, taking multiple samples of LLM inferences and marginalising over them tend to yield more accurate answers when compared to greedy decoding [4]. This has been widely adopted by applications of LLMs for software engineering tasks [16], [17].

2) AutoFL: AutoFL [17] is a LLM agent for repository-level FL, incorporating four tools to gather project-related information: 1) class-level coverage of the failing test, 2) method-level coverage within a covered class, 3) the code snippet of the given method, and 4) comments associated with the method. The original AutoFL utilised GPT-3.5 and GPT-4.

To improve accuracy and reliability, AutoFL leverages self-consistency [4] by performing FL R times independently for a single bug. In each inference run, it predicts a set of likely buggy methods, and the results are aggregated using a voting-based mechanism. Specifically, each predicted method in a run is assigned a score equal to 1 divided by the total number of predicted methods in that run. These scores are averaged across all runs, ensuring that the sum of the scores for all methods equals 1, and are then used to derive the final FL ranking. For example, if the predicted methods from five runs ($R = 5$) are $\{m_1, m_2\}$, $\{m_2\}$, $\{m_2\}$, $\{m_2\}$, and $\{m_3\}$, the score of m_2 is $\left(\frac{1}{2} + 1 + 1 + 1 + 0\right)/5 = 0.7$, while the scores of m_1 and m_3 are 0.1 and 0.2, respectively. In AutoFL, the confidence in its result is defined by the maximum score of the methods, e.g., 0.7 in the previous example, since a higher maximum score indicates greater alignment of results across multiple runs.

AutoFL demonstrates that LLMs can be effectively applied to FL when augmented with specialised tools to extract contextual information. However, its original evaluation is limited to GPT models, which are proprietary and commercially available LLMs. While GPT models generally exhibit strong performance, their use may not always be feasible due to concerns such as code security and monetary costs.

TABLE I: Performance Comparison on Defects4J [18] (353 Bugs)

Model	Template	Accuracy (acc@k)				
		acc@1	*acc@2*	*acc@3*	*acc@4*	*acc@5*
Llama3 8B	original	80	115	131	140	148
Llama3 8B	modified	**108**	**147**	**168**	**180**	**190**
Gemma2 9B	original	105	134	150	160	166
Gemma2 9B	modified	**112**	**145**	**159**	**169**	**182**

B. Ensemble of Small Language Models

Intuitively, COSMos is a voting-based ensemble of SLMs that makes use of the multiplicity of reasoning samples required by self-consistency. Ensembles of SLMs can be constructed by collecting multiple reasoning samples from participating SLMs, instead of repeated sampling of a single LLM. Subsequently, COSMos marginalises over the samples using the voting-based ensemble mechanism. We posit that COSMos can be applied to any task for which self-consistency is shown to improve the performance of LLMs.

We instantiate COSMos with Fault Localisation (FL) problem and propose COSMosFL (**CO**llection of **S**mall Language **Mo**del**s** for FL), a novel approach that enables the application of diverse SLMs for FL (the overview of COSMosFL is shown in 1, along with AutoFL). We evaluate whether an ensemble of heterogeneous open-source SLMs, capable of running on local machines without network access (ensuring high security), can also effectively address the FL problem. Further, COSMosFL will allow us to investigate the cost-benefit trade-offs of our ensemble approach, COSMos, with respect to a concrete task.

To explore the effectiveness of SLMs in performing the FL task within the AutoFL framework, we first replaced GPT with open-source models—Llama3 8B [11] and Gemma2 9B [12]. Our initial experiments found that smaller models frequently called the first tool (class-level coverage) redundantly, diminishing performance. Based on this observation, we remove the class-level coverage tool for COSMosFL. Table I demonstrates an improvement in FL performance from the template modification.

The initial evaluation of two models also show that they rank distinct sets of bugs at the top, a characteristic we refer to as orthogonality. Comparing bugs ranked at the top by Llama3 8B and Gemma2 9B over five repetitions, we see that 71 bugs are ranked at the top by both models, while 37 exclusively by Llama3 8B and 41 exclusively by Gemma2 9B: approximately 35% of correctly localised bugs are uniquely identified by each model. This orthogonality suggests that individual models may excel at localizing certain types of faults missed by others, providing a foundation for an ensemble of heterogeneous SLMs to enhance FL performance by leveraging their complementary behaviour.

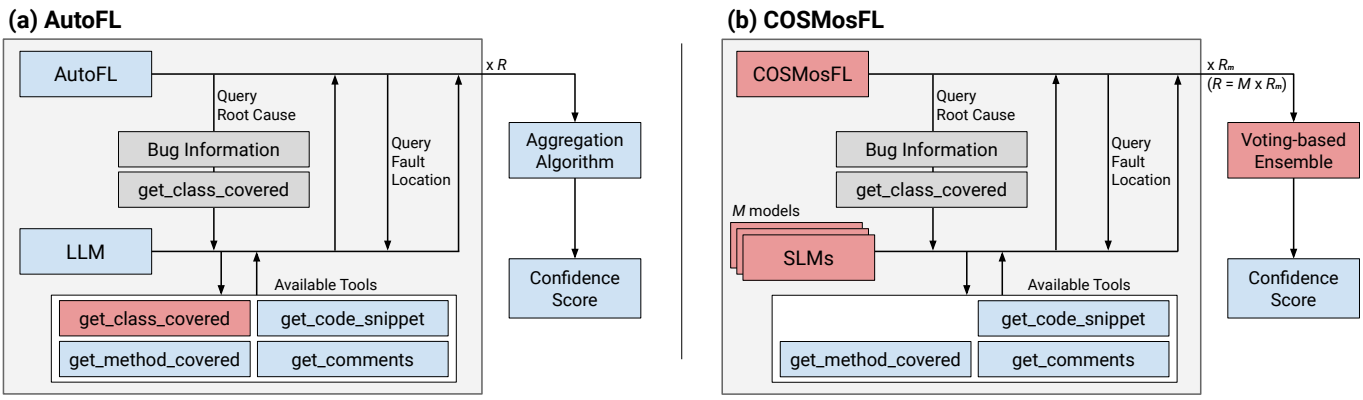


Fig. 1: Overview of our approach against AutoFL [17] with differences colored in red.

Building on the confidence scores generated by AutoFL, we develop a voting-based ensemble technique. Since the original AutoFL already uses voting as the aggregation mechanism, it is straightforward to implement a voting-based ensemble: instead of R inference runs from a single LLM, COSMosFL takes R_M inference runs per each of the M member models of the ensemble, such that $R_M \times M = R$. Subsequently, the same voting-based aggregation takes place, producing the final confidence score based on the ensemble of SLMs.

Algorithm 1: Differential Evolution

Input: Problem Dimension n, Fitness Evaluator **fitness**
Output: Best Agent $best$

1 $n_{pop} \leftarrow$ Population Size;
2 $n_{gen} \leftarrow$ Number of Generations;
3 $p_{cx} \leftarrow$ Crossover Probability;
4 $w_d \leftarrow$ Differential Weight;
5 $pop \leftarrow$ randomly generate n_{pop} agents;
6 $gen \leftarrow 0$;
7 **while** $gen < n_{gen}$ **do**
8 **for** $agent$ *in* pop **do**
9 $agent_{ref} \leftarrow$ **clone**$(agent)$;
10 $a, b, c \leftarrow$ **sample_three_from**(pop);
11 $R \leftarrow$ **random**$([1, ..., n])$;
12 **for** i *in [1, ..., n]* **do**
13 **if** $i = R$ *or* ***uniform***$(0, 1) < p_{cx}$ **then**
14 $agent_{ref}[i] \leftarrow a[i] + w_d * (b[i] - c[i])$;
15 **if** ***fitness***$(agent) <$ ***fitness***$(agent_{ref})$ **then**
16 $agent \leftarrow agent_{ref}$;
17 $best_{gen} \leftarrow$ **select_best**(pop);
18 **if** ***fitness***$(best) <$ ***fitness***$(best_{gen})$ **then**
19 $best \leftarrow best_{gen}$;
20 $gen \leftarrow gen + 1$;

We investigate two ensemble strategies to aggregate scores from individual models. The first, equal weighting, naively sums the scores from each model with uniform weights, providing a straightforward approach. The second, DE-optimised weighting, refines the voting weights using differential evolution (DE) algorithm [19]. A distinctive feature of DE, as shown in line 14 of Algorithm 1, is its use of scaled differences between agents to guide the generation of new candidates, enabling exploration of the search space. This characteristic has been shown to make DE particularly effective for problems in continuous search spaces [20], [21], making it an appropriate candidate for our weight optimisation task. This optimisation aims to maximise $acc@1$ – prioritising the buggy method in the top rank – while minimising wasted effort as a secondary objective in case of ties. With these objectives, we aim to rank the buggy method at top while minimising the number of irrelevant method inspections.

Considering the high inference cost of language models, we further hypothesise that the ensemble approach can better balance the cost-performance trade-off when utilizing multiple models. To enable further analysis, we integrate EnergyMeter [22] to track GPU energy consumption throughout experiments. In addition, we also report the number of tokens and the time taken for the inference as cost of COSMosFL.

III. Experimental Setup

A. Research Questions

Our primary objectives are to evaluate whether the ensemble of small language models enhances fault localisation effectiveness and to investigate the cost-benefit trade-off of ensembles.

- **RQ1. Effectiveness:** To what extent does our ensemble technique improve fault localisation performance? To address this, we conducted initial runs with seven open-source SLMs to assess model orthogonality, running each model five times. Based on these preliminary results, we selected four models that demonstrated the most complementary fault localisation performance in combination. For the final evaluation, we ran AutoFL on each selected model 30 times and sampled a varying number of runs to compare two ensemble weighting strategies: equal weighting and differential evolution (DE)-optimised weighting.

- **RQ2. Efficiency:** How does the ensemble technique perform in terms of the cost-performance trade-off, and does it contribute to balancing these factors? This analysis focuses on computational efficiency, considering model inference time, energy consumption, and number of tokens as the measure of cost. Note that we report the sum of input and output tokens as the number of tokens, as this reflects the pricing models of closed-source LLMs more closely.

B. Language Models

To align with the characteristics of smaller language models, we redesign the agent's task from a chat-completion to instruction-following.

Each model has been downloaded and served using Ollama [23], which is chosen for its convenient setup and support for multiple models. We focus on 4-bit quantised models for the sake of memory usage, and choose the following open-source SLMs: *CodeLlama 7B* [24], *Gemma2 9B* [12], *grantie3 8B* [25], *Llama3 8B* [11], *Llama3.1 8B* [11], *Mistral NeMo 12B* [26], and *Qwen2.5-Coder 7B* [13]. These models are all below the size of 8GB (when quantised for 4-bit): we expect them to be compatible with a wider range of machines without GPUs.

Techniques involving inherent randomness require sufficient repetitions to ensure reliable performance measurement [27]. We employ sampling to stabilise our performance metrics and account for stochastic variation of language models. We first run AutoFL with each selected model 30 times. Then, for a given number of runs (R) ranging from 4 to 24, we sample R runs from the 30 runs 20 times for each model. For the case of ensembles, we allocated an equal number of runs to each model, ranging from 1 to 6, resulting in R of multiples of 4.

C. Dataset

Our evaluation dataset is a subset of Defects4J [18] used by AutoFL, comprising a total of 353 bugs. We calculate *acc@k* as the number of bugs for which at least one buggy method is correctly ranked within the top k places, ensuring consistency with the prior work [17]. Table II summarises the number of bugs, Lines of Code (LOC) measured using `cloc` [28], and the number of methods and tests for each project.

TABLE II: Evaluation Dataset Details

Project	#Bugs	LOC	#Methods	#Tests
Chart	26	78,564–96,382	6,378–8,041	1,598–2,201
Closure	131	58,989–104,131	4,621–8,700	2,692–8,625
Lang	64	16,593–21,810	1,794–2,248	1,587–2,265
Math	106	9,471–84,317	1,174–6,015	686–3,548
Time	26	26,589–27,795	3,535–3,696	3,802–4,054

D. Hyperparameters & Environment

We utilise DEAP [29], a framework for evolutionary compuation, to implement the differential evolution algorithm. To reduce the over-fitting during the optimisation process, we apply 10-fold cross-validation. We select the DE parameters referring to Storn et al. [30], a population size of 40 and 30 generations. To foster exploratory behaviour during the optimisation process, we set high differential weight, 1.5, and also the crossover probability to 0.8.

EnergyMeter [22] is an open-source Python project that measures the energy consumption incurred by the hardware. As we focus on the language model inference cost, we only utilise the GPU energy consumption enabled by `nvidia-smi` [31].

We conduct all our experiments using the Docker image `nvidia/cuda:11.3.1-runtime-ubuntu20.04` on a Linux server equipped with 252 GB RAM and 40 Intel Xeon Silver 2.40GHz CPUs. A single NVIDIA GeForce RTX 3090 GPU is utilised to accelerate model inference.

IV. Results

A. RQ1. Effectiveness

Figure 2 shows the orthogonality result of four models, *Llama3 8B*, *Llama3.1 8B*, *Mistral NeMo 12B*, and *Qwen2.5-Coder 7B*, which are identified as ranking the highest number of bugs on top in combination. Collectively, these four models achieve an acc@1 of 180, i.e., 180 bugs are ranked at the top at least once by one of these models. Building on this, we examined the effectiveness of the ensemble approach for fault localisation.

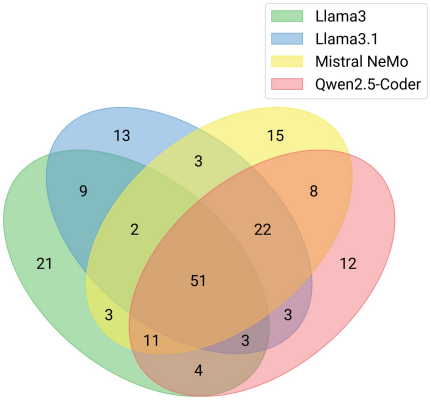


Fig. 2: Overlap of bugs ranked at first by Llama3, Llama3.1, Mistral NeMo, and Qwen2.5-Coder. Each model is run 5 times without applying ensemble.

Figure 3 presents the mean $acc@k$ for k ranging from 1 to 5 across individual models and two ensemble approaches. The results also include the reported performance of GPT-3.5 for $R = 5$. Both ensembles tend to outperform individual models as k increases. Since our experiments use more runs ($R = 20$) compared to the prior work ($R = 5$), a direct comparison is unfair. However, the observation that even the least effective model, Llama3.1, achieves higher accuracies at $k = 4$ and 5 suggests that increasing the number of runs could address the underperformance at higher k values relative to SBFL

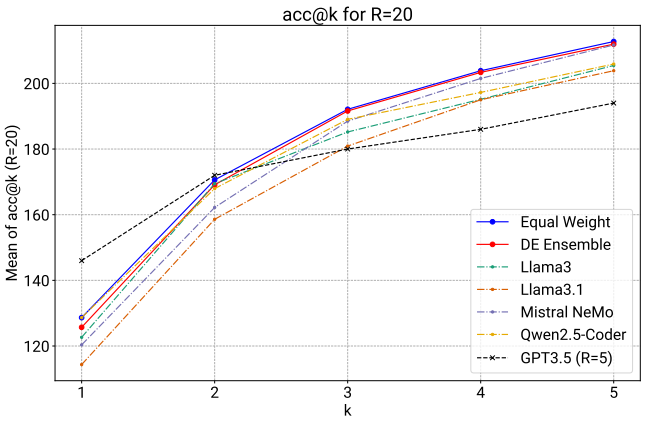


Fig. 3: acc@k for R=20 for each model and ensemble approaches, alongside AutoFL's reported GPT-3.5 performance.

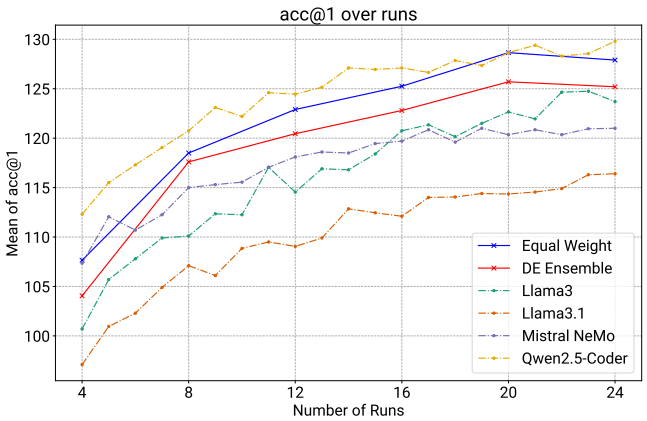


Fig. 5: Mean of $acc@1$ across runs for four single models and two ensemble approaches. Note that the ensemble techniques are only available at multiples of four runs.

methods, previously attributed to the inability to *dig deep* into a repository [17].

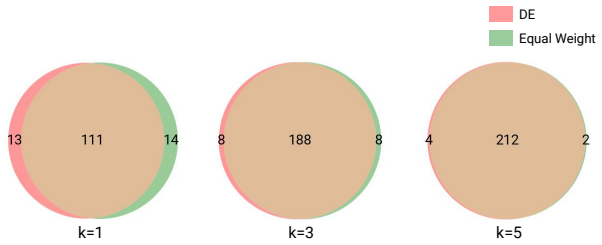


Fig. 4: Overlap of top-ranked bugs at each k for a single sample ($R = 20$) of DE and Equal Weight Ensembles.

Since DE-optimised weights and equal weights show similar performance, we further analyse overlap for a sample of five runs from each model, totalling $R = 20$. Figure 4 indicates that applying different weights for the same runs yields mild variation in bugs ranked at top, suggesting potential for further optimisations to harness model orthogonality. As k increases, however, the top-ranked bug sets become more consistent across weighting methods. SLMs list a limited number of methods as suspicious, allocating confidence score only for those. Thus, when we count the number of methods ranked at top five, they rather show higher consistency; in contrast, for the $acc@1$, they are more sensitive to the weight changes.

Figure 5 highlights $acc@1$, as high accuracy at the first rank is particularly desirable for FL tasks. The ensemble does not surpass the best single model's performance when given the same number of runs. This outcome can be attributed to how runs are distributed: assigning all weights to a single model effectively limits that model to R/4 runs in the ensemble setup, rather than the full R runs it would receive if evaluated independently. Consequently, mean optimised weights over samples and cross-validation folds, depicted in Figure 6, ranged from 0.15 to 0.37 for all four models. Although the

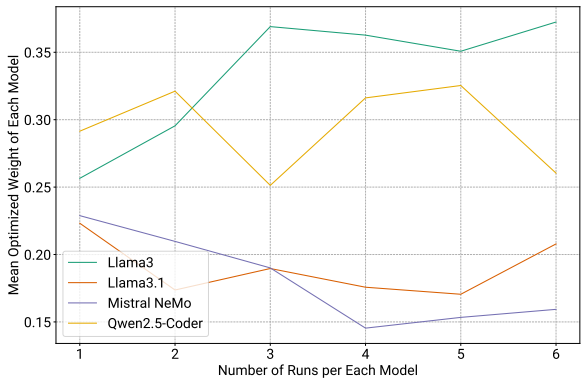


Fig. 6: Mean of optimised weights for each model over cross-validation folds and samples.

DE-based optimisation underperforms the equal weighting, resulting weights still utilize information from all runs.

Figure 7 illustrates the distribution of $acc@1$ over samples. While individual models tend to converge as R increases, ensembles tend to maintain a relatively stable variance level across runs, despite generally outperforming individual models. This highlights the need to investigate more advanced methods for building ensembles.

B. RQ2. Efficiency

Figures 8a to 8c show the average cost of each sample, composed of R runs, and its mean $acc@1$. The best performing model, Qwen2.5-Coder consumes the most amount of energy and time, while Llama3 requires the least. Ensemble cost and performance tend to lie between these two extremes. We note that Qwen2.5-Coder is less of an outlier for the number of tokens in Figure 8c, due to the imbalance in its performance. In our evaluation, Qwen2.5-Coder spends significant amounts of time and energy when generating tokens, but input tokens take

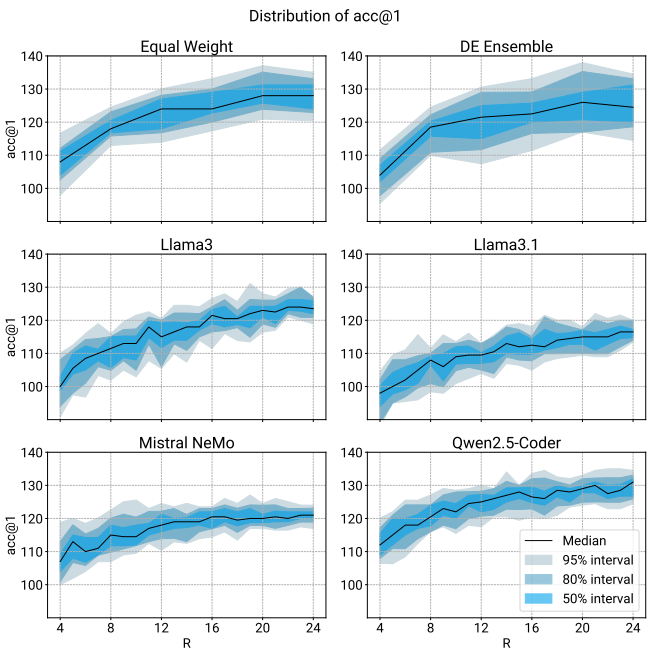


Fig. 7: acc@1 across runs for each model and ensemble method.

up the majority of the number of tokens used by an inference run, balancing out the energy and time cost.

To further understand the energy consumption pattern, we visualize the distribution of energy consumptions and execution time in Figure 9a and 9b. Unlike the per sample consumptions, all four models show similar median value. The median consumption of Qwen2.5-Coder is actually the second smallest among the four models. However, Llama3.1 and Qwen2.5-Coder have a group of outliers at around 1,000 times the median value, which we suspect is linked to an issue in Ollama, given that there is an issue report about some models generating tokens endlessly [32]. This erroneous behaviour may have resulted in the significant increase of overall energy consumption of Qwen2.5-Coder. Since the total energy consumption and execution time follow similar trends, the energy consumption per second remains relatively constant, as shown in Figure 9c.

V. Discussion

To better understand the performance characteristics of two ensemble schemes, we have conducted a grid search for weights for ensembles of every pair of models, increasing the number of runs (R_m) from one to five, as illustrated in Figure 10. Interestingly, the best weights are close to the equal weights in most scenarios, supporting the success of equal weight ensembles. As the number of runs increases, the $acc@1$ landscape becomes more rugged, implying a heightened risk of DE converging to local minima that do not generalise well to validation sets. We conjecture that introducing a more robust optimisation scheme could further harness the potential of the models.

This paper focuses on the quantitative aspect of the FL performance. However, the original work [17] also points that LLMs have the potential to provide rationales for their decisions. Since we now have a diverse set of runs, it would be interesting to see if we can rank generated explanations better based on the larger number of runs sampled from different SLMs, with more practical impact for the developers.

VI. Related Work

A. LLM-based Fault Localisation

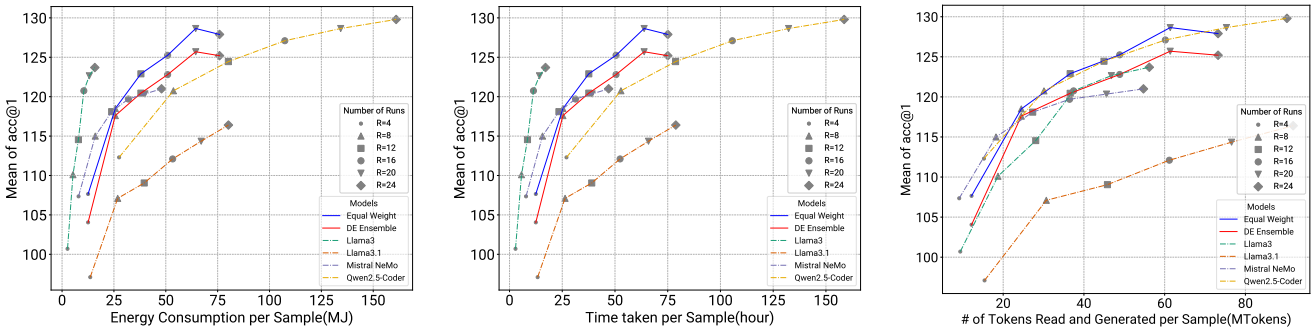
Fault localisation aims to automatically pinpoint software bugs, often by analysing program behaviour at different granularity levels. With large language models (LLMs) now available, researchers have explored using these models for fault localisation tasks. For instance, Wu et al. [33] achieved notable results in statement-level localisation given the buggy method using ChatGPT-4, though they identified challenges in extending the model's context handling to the class level. Addressing this issue, AutoFL [17] introduced an agent architecture that leverages the function-calling capabilities of OpenAI models, allowing LLMs to autonomously explore repositories. This approach also posits that LLMs can potentially offer explanations for root causes of bugs. Similarly, FuseFL [34] aimed to enhance explainability by integrating spectrum-based fault localisation results into prompts, though its scope was limited to student programming assignments.

Alongside these advancements, efforts to utilize open-source language models in fault localisation have gained traction. Yang et al. [35] proposed a test-free FL approach by fine-tuning large, open-source language models with datasets of buggy programs, with a focus on reducing programmer input. More recently, Liu et al. [36] conducted an empirical evaluation comparing FL performance between proprietary and open-source models on novice programming tasks. They found that while ChatGPT-3.5 and 4 outperformed other models, open-source models demonstrated complementary behaviors, successfully localizing bugs that proprietary models missed.

B. Ensemble Methods

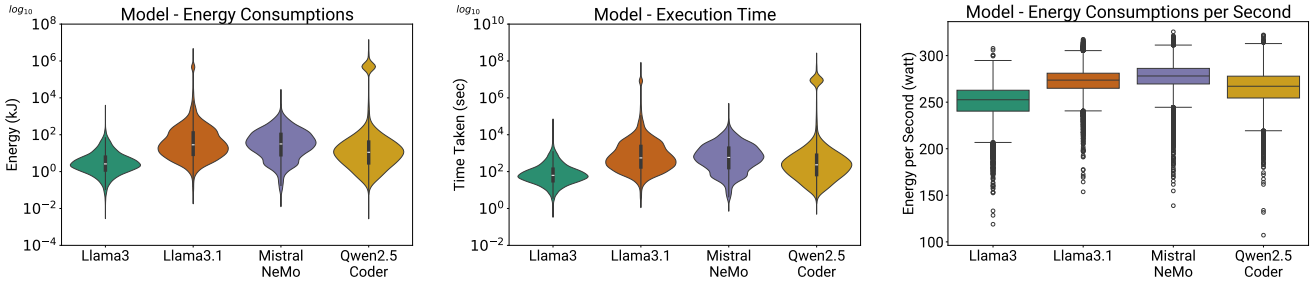
Ensemble learning combines predictions from multiple models to enhance overall performance by leveraging the diverse strengths of individual learners. Ensembles have been applied to fault localisation, based on the observation that no single FL technique is effective across all faults. Wang et al. [37] and Xuan et al. [38], combined FL outputs from multiple models to improve localisation accuracy. Sohn et al. [39] further explored ensemble learning across FL methods, demonstrating that combining diverse techniques could achieve better performance. COSMosFL is the first ensemble technique for LLM-based FL to the best of our knowledge.

There are recent works that have utilized multiple language models in ensembles. Zhang et al. [40] showed that combining outputs from different regex generators via self-consistency decoding contributed to improved performance. Kumar Dipongkor [41] employed ensemble methods for bug



(a) Energy Consumption vs. $acc@1$ (b) Execution Time vs. $acc@1$ (c) # of Tokens vs. $acc@1$

Fig. 8: Cost-Benefit Trade-Offs: energy consumption, inference time, and # of input and output tokens are mean values from 20 samples



(a) Energy consumption per run (b) Execution time (sec) per run (c) Energy consumption per second

Fig. 9: Boxplots of cost measures per model. Note that y-axis of (a) and (b) is log of the measure.

triaging using BERT variants, finding that the voting-based ensemble consistently outperformed the stacking-based approach (i.e., training an extra layer on top of the pre-trained language models' outputs). However, both ensembles aggregate lower level outputs, such as next regex tokens or next embedding vector. In comparison, COSMosFL implements voting-based ensemble at the task level by aggregating the FL scores.

VII. Conclusion

Through this paper, we present COSMos, an ensemble of small language models, and evaluate its instantiation on the fault localisation task, COSMosFL. We first examine the orthogonal behaviour of SLMs by adopting an LLM-based FL technique to evaluate the feasibility of our approach. Then, by implementing two versions of COSMosFL– equal weighting and DE-optimised weighting – we further assess its performance and cost compared to single-model repetitions. Our results show that COSMosFL has the potential to outperform a single SLM under cost constraints. Finally, we discuss the current limitations of COSMosFL, including the issues of the SLM serving platform, and outline future directions, such as improving the optimisation strategy and developing a ranking scheme for explanations.

References

[1] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: Future of Software Engineering*, ICSE-FoSE, pp. 31–53, IEEE Computer Society, May 2023.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, (Red Hook, NY, USA), Curran Associates Inc., 2024.

[4] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *Proceedings of the International Conference on Learning Representation*, ICLR 2023, 2023.

[6] R. Feldt, S. Kang, J. Yoon, and S. Yoo, "Towards autonomous testing agents via conversational large language models," in *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE 2023, pp. 1688–1693, 2023.

[7] I. Bouzenia, P. Devanbu, and M. Pradel, "Repairagent: An autonomous, llm-based agent for program repair," 2024.

[8] J. Yoon, R. Feldt, and S. Yoo, "Intent-driven mobile gui testing with autonomous large language model agents," in *Proceedings of the 16th IEEE International Conference on Software Testing, Verification and Validation*, ICST 2024, pp. 129–139, 2024.

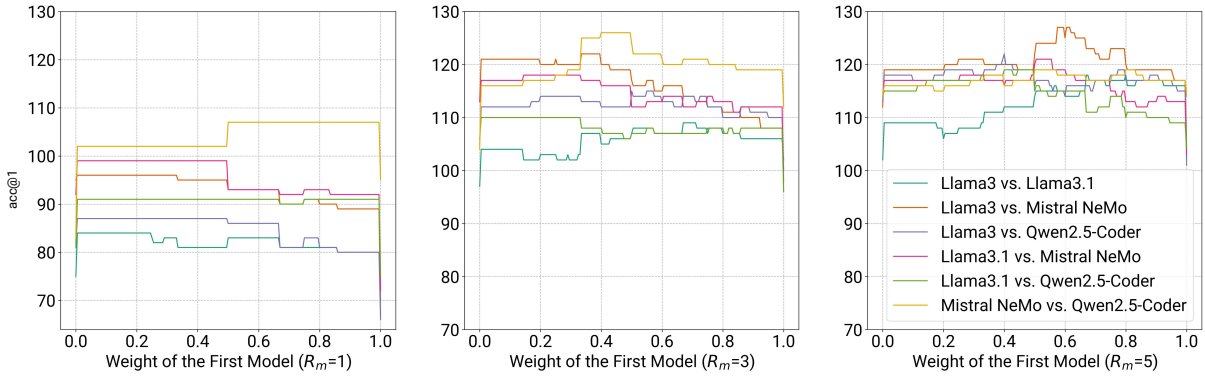


Fig. 10: Landscapes of Pairwise $acc@1$ explored by grid search

[9] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (A. Korhonen, D. Traum, and L. Màrquez, eds.), (Florence, Italy), pp. 3645–3650, Association for Computational Linguistics, July 2019.

[10] M. C. Rillig, M. Ågerstrand, M. Bi, K. A. Gould, and U. Sauerland, "Risks and benefits of large language models for the environment," *Environmental Science & Technology*, vol. 57, no. 9, pp. 3464–3466, 2023.

[11] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, and et al., "The Llama 3 Herd of Models," Aug. 2024.

[12] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, and et al., "Gemma 2: Improving Open Language Models at a Practical Size," Oct. 2024.

[13] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, and et al., "Qwen2.5-Coder Technical Report," Nov. 2024.

[14] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[15] S. Kang, J. Yoon, N. Askarbekkyzy, and S. Yoo, "Evaluating diverse large language models for automatic and general bug reproduction," *IEEE Transactions on Software Engineering*, vol. 50, no. 10, pp. 2677–2694, 2024.

[16] T. Ahmed and P. Devanbu, "Better patching using llm prompting, via self-consistency," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1742–1746, 2023.

[17] S. Kang, G. An, and S. Yoo, "A Quantitative and Qualitative Evaluation of LLM-Based Explainable Fault Localization," *Proc. ACM Softw. Eng.*, vol. 1, pp. 64:1424–64:1446, July 2024.

[18] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 international symposium on software testing and analysis*, pp. 437–440, 2014.

[19] R. Storn and K. Price, "Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces," *International computer science institute*, 1995.

[20] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4–31, 2010.

[21] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search-based repair of deep neural networks," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, pp. 1–26, 2023.

[22] M. F. Argerich and M. Patiño-Martínez, "Measuring and Improving the Energy Efficiency of Large Language Models Inference," *IEEE Access*, vol. 12, pp. 80194–80207, 2024.

[23] "Ollama." https://github.com/ollama/ollama, Nov. 2024.

[24] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, and et al., "Code Llama: Open Foundation Models for Code," Jan. 2024.

[25] Granite Team, IBM, "Granite 3.0 Language Models." https://github.com/ibm-granite/granite-3.0-language-models/blob/main/paper.pdf, 2024.

[26] Mistral AI, "Mistral NeMo." https://mistral.ai/news/mistral-nemo/, July 2024.

[27] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pp. 1–10, ACM, 2011.

[28] A. Danial, "cloc: v1.92," Dec. 2021.

[29] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[30] R. Storn, "On the usage of differential evolution for function optimization," in *Proceedings of North American Fuzzy Information Processing*, pp. 519–523, June 1996.

[31] N. Developer, "Nvidia system management interface," *NVIDIA System Management Interface*, 2021.

[32] "Ollama gets stuck in an infinite loop sometimes and has to be restarted · Issue #2805 · ollama/ollama." https://github.com/ollama/ollama/issues/2805.

[33] Y. Wu, Z. Li, J. M. Zhang, M. Papadakis, M. Harman, and Y. Liu, "Large Language Models in Fault Localisation," Oct. 2023.

[34] R. Widyasari, J. W. Ang, T. G. Nguyen, N. Sharma, and D. Lo, "Demystifying Faulty Code: Step-by-Step Reasoning for Explainable Fault Localization," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 568–579, Mar. 2024.

[35] A. Z. H. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large Language Models for Test-Free Fault Localization," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, (New York, NY, USA), pp. 1–12, Association for Computing Machinery, Feb. 2024.

[36] Y. Liu, H. Liu, Z. Yang, Z. Li, and Y. Liu, "Empirical Evaluation of Large Language Models for Novice Program Fault Localization," in *2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 180–191, July 2024.

[37] S. Wang, D. Lo, L. Jiang, Lucia, and H. C. Lau, "Search-based fault localization," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp. 556–559, Nov. 2011.

[38] J. Xuan and M. Monperrus, "Learning to Combine Multiple Ranking Metrics for Fault Localization," in *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 191–200, Sept. 2014.

[39] J. Sohn and S. Yoo, "Why train-and-select when you can use them all? ensemble model for fault localisation," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, (New York, NY, USA), pp. 1408–1416, Association for Computing Machinery, July 2019.

[40] S. Zhang, X. Gu, Y. Chen, and B. Shen, "InfeRE: Step-by-Step Regex Generation via Chain of Inference," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1505–1515, Sept. 2023.

[41] A. Kumar Dipongkor, "An Ensemble Method for Bug Triaging using Large Language Models," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, (New York, NY, USA), pp. 438–440, Association for Computing Machinery, May 2024.