Exploring the Use of Software Product Lines for the Combination of Machine Learning Models

Marcos Gomez-Vazquez
marcos.gomez@list.lu
Luxembourg Institute of Science and Technology
Esch-sur-Alzette, Luxembourg

Jordi Cabot
jordi.cabot@list.lu
Luxembourg Institute of Science and Technology
Esch-sur-Alzette, Luxembourg
University of Luxembourg
Esch-sur-Alzette, Luxembourg

ABSTRACT

The size of Large Language Models (LLMs), and Machine Learning (ML) models in general, is a key factor of their capacity and quality of their responses. But it comes with a high cost, both during the training and the model execution phase. Recently, various model merging techniques and Mixture of Experts (MoE) architectures are gaining popularity as they enable the creation of large models by combining other existing ones (the "experts" in the MoE approach). Creating these combinations remains a deep technical task with many possible configurations to consider. In this sense, this paper aims to democratize the creation of combined ML models by presenting a product line approach to the specification and training of this type of ML architectures from an initial feature model that helps users define, among other aspects, the type of models they want to combine, the combination strategy and even, for the MoE approach, the tasks that should be associated to each expert.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Model-driven software engineering**; • **Computing methodologies** → *Machine learning approaches.*

KEYWORDS

Software Product Line, Feature Model, Machine Learning, Large Language Model, Model Merging, Mixture of Experts

ACM Reference Format:
Marcos Gomez-Vazquez and Jordi Cabot. 2024. Exploring the Use of Software Product Lines for the Combination of Machine Learning Models. In *28th ACM International Systems and Software Product Line Conference (SPLC '24), September 02–06, 2024, Dommeldange, Luxembourg*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3646548.3676599

1 INTRODUCTION

Many of the new Large Language Models (LLMs) topping the LLM leaderboards are not trained from scratch but created by combining other preexisting LLMs. This is not only cheaper (both, in terms of time and money) but it also allows to push the limits of state-of-the-art architectures by building models that excel at specific tasks and domains by combining the knowledge of the merged models.

We are now witnessing an explosion of model composition strategies and techniques but there is a lack of well-defined methods to guide users interested in creating them. As with any other kind of emerging software, it is necessary to define precise requirements and properties of composite AI models in order to abstract its development process from the underlying technology and to enable non-technical users to create it. This paper advocates for the use of a Software Product Lines (SPLs) to define and generate LLM compositions by means of different merging algorithms.

Indeed, Software Modeling and Generative Software Development is key to defining the properties, requirements, commonalities and variabilities in system families. Machine Learning (ML) is establishing a new paradigm in Computer Science but we still lack of modelling approaches, low-code tools and domain-specific languages (DSLs) for an easy, reusable and maintainable development of this new type of intelligent systems. Even worse, we are starting to see convoluted systems built on multiple smart components where more variables are being involved and increasing the complexity of the software development process [5].

Feature models capture all the possible products in an SPL [15]. We can therefore use feature models to define software families of intelligent systems. While the use of ML for SPL development has been widely explored, the other way around (using SPLs to characterize ML systems, such as, in this case, compositions of LLMs) not much.

In this sense, this paper proposes an SPL approach for the emerging field of combinations of LLMs. More specifically, we introduce a feature model to characterize the dimensions and variability aspects of this domain, together with a code generation approach able to transform a feature configuration into an actual merged model. We rely on Mergekit [11] for the generation phase and provide our own tool support for the whole process.

The rest of the paper is structured as follows. Section 2 gives some background on LLM composition techniques. Then, Section 3 presents our feature model for this domain while Section 4 discusses how then a feature configuration could be transformed into a combined and ready-to-use LLM. Section 5 presents our tool support, Section 6 compares our approach with related work and, finally, Section 7 draws some conclusions and further work.

SPLC '24, September 02–06, 2024, Dommeldange, Luxembourg

ACM ISBN 979-8-4007-0593-9/24/09
https://doi.org/10.1145/3646548.3676599

2 BACKGROUND

Large Language Models (LLMs) are showing impressive results in a large variety of tasks [21]. But training a LLM from scratch is a long process that requires massive amount of data and computing power. Running inferences on them also requires a significant computational cost. While the ML community is doing good work to improve the efficiency of training and running LLMs, this is still an open research challenge.

Nevertheless, over the last year, we have witnessed how a new type of LLMs has started to populate (and top) the benchmark leaderboards [7]. The common aspect of these LLMs is that they are not trained from scratch but built through the combination of other LLMs, proving that it is possible to combine multiple knowledge sources into a single model at a fraction of the training cost. In this section we briefly introduce the two major techniques in the area: Model Merging and the Mixture of Experts approaches.

2.1 Model Merging

Model Merging is a technique that combines two or more pre-trained LLMs into a single unified one. Currently, several merging methods have been successfully implemented. From simple methods like linear weighted average, to more advanced ones such as Task Arithmetic [13] or evolutionary optimization approaches [2], among others. These merging methods have been proven to generate new LLMs that outperform the original models alone. Furthermore, the merging process can be run entirely on CPU or with low GPU resources.

2.2 Mixture of Experts

In 1991, Jacobs et al. proposed a new supervised learning procedure for systems composed of many separate neural networks [14]. These networks would be trained to solve different problems and then, a special gating network would be in charge of deciding, for a given input, which networks (i.e., experts) would generate the output. This Mixture of Experts (MoE), depicted in Figure 1, has recently gained a lot of new attention thanks to its application to Transformer models (the foundational architecture of most of the current state-of-the-art LLMs [20]). The main advantage with respect to regular LLMs is their sparsity. These models only use a subset of all their parameters when running inference (i.e., some experts work while others "sleep"), which allows for a faster inference compared to a model with the same number of parameters.

While there are MoE models that are trained from scratch, there is a new trend of combining preexisting LLMs, trained to solve specific problems, and turn them into a MoE. These are being colloquially called *FrankenMoEs* to distinguish them from pre-trained MoEs. For simplicity, we will use the term MoE to refer to FrankenMoEs, and consider them as another technique of Model Merging.

3 THE FEATURE MODEL

This section introduces our feature model (FM) to characterize the building of composite LLMs. As there is not a single feature modelling notation [16], to express our complex FM, we combine several syntaxes to express modularity and compositionality in FMs [1], feature attributes consisting of name, domain and value [4] and cardinality-based features [8].

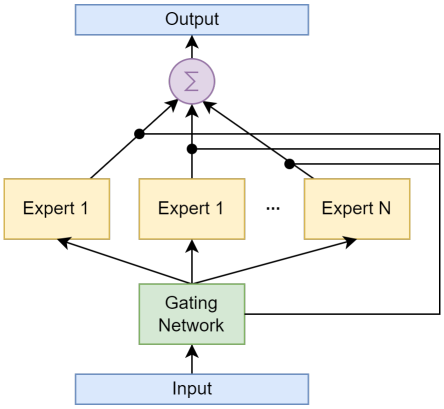

Figure 1: Example MoE architecture. The gating network generates weights for each expert, based on the given input. Then, the output generated by each of the selected experts is composed.

Figure 2 depicts the FM. The root feature is Composite. At the first depth level, there are 2 mandatory features: 1 - Composite tool, which, at the moment, only has Mergekit as children features but could be extended with new tools in the future, and 2 - Composite config, which is the starting point for all the features related to the LLM composition itself. From here, an alternative set of features, composed by MoE and Merge is used to distinguish between MoE or Merge composition strategies, since each of them will need a different set of features to be properly defined. In the rest of the section we comment on some of these properties but refer to the technical documentation of the underlying merging techniques for the full details.

A Merge may be created by combining a set of models or slices (see the alternative relationship with Merge). Slices are parts of models. These parts are defined by indicating the layers that we want to select from a model (see layer_ini and layer_end leaf features). Among other features, a Merge has a merge_method, which determines the algorithm used to combine the selected models or slices, and a tokenizer_source that defines the tokenizer to use in the generated LLM.

A MoE has a base_model, a gate_mode that defines how the MoE gates are initialized, the number of experts_per_token and the experts (a minimum of 2 experts is required). Each expert has a source_model, 1 or more positive_prompts (i.e., prompts that will favor the selection of the expert by the gating network) and an arbitrary number of negative_prompts (i.e., prompts that will disfavor the selection of the expert).

There are other, smaller FMs that are embedded in the Composite FM but that we define as separate sub-feature models linked to the main one to avoid duplications and facilitate their evolution (if they are updated, all embeddings in the core FM would be automatically updated as the sub FMs are linked via references). These sub FMs are model_reference (defining a model to be used within the merging process) , base_model (some merging techniques need to define explicitly what model plays the role of a base model, that simply contains a model_reference), dtype (the data type used for the

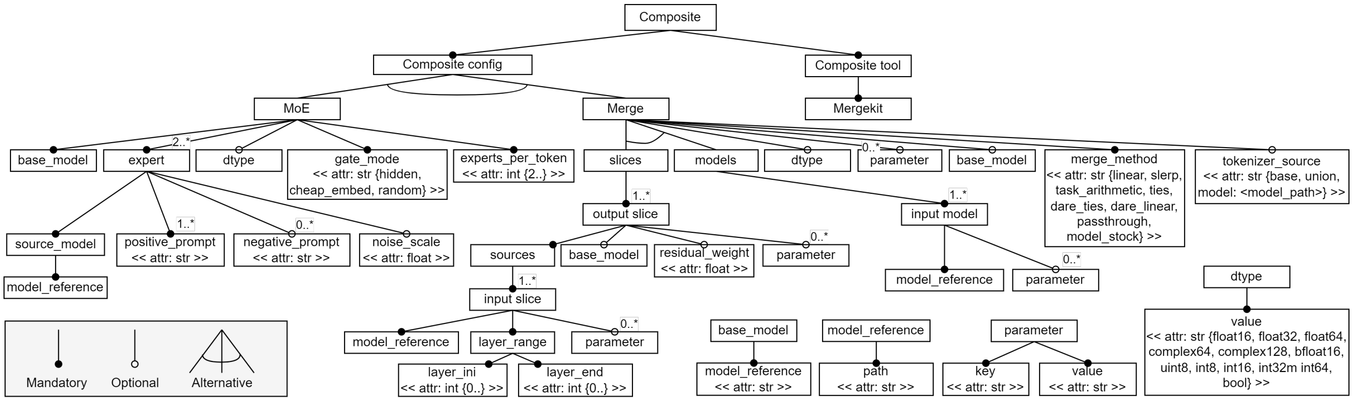

Figure 2: A Feature Model for configuring combinations of machine learning models

merging operation, e.g., `float32` or `int16`) and `parameter` (used to specify various parameters such as weights and densities, which can be present at different levels of the FM).

Some features have non-standard cardinalities, such as `expert`, that has a `2..*` cardinality. This gives enough freedom to multiply the number of sub-FMs with the same structure without restricting the number of pre-defined trees in a feature configuration (e.g., a MoE could be composed of 4, 8, 16, etc. experts). Additionally, there are features embedding an attribute that enables the definition of custom values when creating a configuration. Since there is only 1 attribute per feature at most, the attribute name is not present in the diagram. The attribute domains can be open (e.g., strings, positive integers, etc.) or closed (e.g., a predefined set of strings).

4 FROM A FEATURE CONFIGURATION TO A COMPOSITE LLM

Our feature model encompasses all possible combinations of LLMs that could be generated via the different merging techniques. Each of the unique valid feature selections defines a feature configuration, which precisely describes a composite LLM. This section discusses how we go from the LLM combination configuration to a brand new trained and running LLM deployed on premises or in a cloud environment.

4.1 Generate the configuration scripts

The first step is to generate the necessary configuration scripts representing the composite LLM in the appropriate input format expected by the selected `Composite tool`.

Currently, the dominant tool in the market is Mergekit. Therefore, only a Mergekit generator has been implemented so far. Mergekit requires a YAML configuration file indicating the LLMs to be merged, the merging technique and other required properties. Our generator creates this YAML file by traversing the feature configuration and transforming each feature selection into the corresponding YAML excerpt.

4.2 Run the composition

The next step is to actually generate the new LLM. Our tool is able to run Mergekit according to the previously defined YAML file and right Mergekit parameters. The duration of the process will vary depending on the available resources, the merging technique being used and the LLM sizes. Once the process finishes, the generated LLM will be stored in a local directory, containing all the model weights and other configuration files. At this point, the LLM is ready to be used and, optionally, deployed to a cloud environment (see next section).

5 TOOL SUPPORT

This section describes the tool support for this work. All the components are freely available as open source software[1] and are part of the BESSER low-code platform [3].

Firstly, we have developed a graphical front-end to facilitate the creation of feature configurations. Users can go through the feature model, select the features and then fill the configuration values in the feature configuration panel showing the list of selected features so far. This front-end has been built with Streamlit, a Python framework for frontend development.

This configuration is then internally stored as a set of Python objects that store the configuration values and the links with the selected features. The classes for these objects are a simplification, for the purpose of this specific LLM configuration feature model, of the typical metamodels for FMs available in the literature (e.g., see [18]). A set of Python validators has been created to check the stored configuration is valid.

Finally, our generator reads this information, produces the configuration file described in the previous section and runs Mergekit to build the composite LLM. Note that Mergekit allows the combination of locally stored LLMs. Nevertheless, it also supports HuggingFace's LLMs[2], allowing to reference their ids directly from the HuggingFace model hub. The LLM weights can be serialized with Safetensors (best alternative, faster) or Pickle, and loaded with many frameworks such as Tensorflowor PyTorch.

Our tool saves the generated LLM locally and it also allows to publish it to HuggingFace, so that it can be immediately used by the community. To enable this, the user must specify their HuggingFace username, the license under which the LLM will be published and

[1] https://github.com/BESSER-PEARL/spl-for-ai
[2] https://huggingface.co/

a valid HuggingFace access token (with write permissions). This step is recommended as it makes the LLM accessible for everyone, facilitates its installation and its reuse through the HuggingFace API and Transformers library.

6 RELATED WORK

AI techniques and, in particular, Generative AI approaches, have promising applications in the field of software variability [12]. Indeed, there are plenty of approaches using some type of ML approach for the creation, analysis and evolution of SPLs. For instance, Temple et al. use ML to find the best set of constraints for feature models [19]. and Saini et al. propose different approaches to test SPLs by means of ML solutions, optimizing the number of tests to be run to determine the validity of a product [17], among several other examples.

Nevertheless, employing SPLs to configure a family of ML/AI components has been much less explored. In [6], an SPL was put in place to define the different parts of an ML workflow. More similar to our proposal, other works aim to use SPLs to identify subsets or parts of artificial neural networks (ANNs) to extract sub-tasks that can be reused in other ANNs [9].

None of the existing approaches focus on LLMs and even less on how to combine them, which is a major challenge nowadays and the goal our proposal is aiming to facilitate it.

7 CONCLUSIONS AND FUTURE WORK

We have presented an SPL approach to create families of composite ML models, aiming to capture similarities and variability among the possible combinations available. We have considered two major kinds of compositions: merging algorithms and Mixtures of Experts and their corresponding variability dimensions. We see our approach as a new step towards the increasing use of SPL-driven approaches for the configuration and generation of intelligent systems.

As future work, we plan to improve the feature model by formalizing in more detail the valid configurations, aiming to find an optimal granularity level that allows us to model all relevant restrictions but without hampering the usability of our approach by making the feature model too complex. We would also like to build an AI assistant to help users to create and evolve feature configurations (e.g. based on previous "recipes" known by the assistant). Finally, we will extend the feature model to cover both other types of models (e.g. compositions of computer vision models) and other aspects of the LLM creation process (e.g. composition of datasets, including their metadata for trustworthiness AI results [10]). Extensions to the code generator to target new libraries will also be implemented when needed.

ACKNOWLEDGMENTS

This project is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. 2010. Comparing Approaches to Implement Feature Model Composition. In *Modelling Foundations and Applications*, Thomas Kühne, Bran Selic, Marie-Pierre Gervais, and François Terrier (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–19.

[2] Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. 2024. Evolutionary Optimization of Model Merging Recipes. (2024). arXiv:2403.13187 [cs.NE]

[3] Iván Alfonso, Aaron Conrardy, Armen Sulejmani, Atefeh Nirumand, Fitash Ul Haq, Marcos Gomez-Vazquez, Jean-Sébastien Sottet, and Jordi Cabot. 2024. Building BESSER: An Open-Source Low-Code Platform. In *Enterprise, Business-Process and Information Systems Modeling*, Han van der Aa, Dominik Bork, Rainer Schmidt, and Arnon Sturm (Eds.). Springer Nature Switzerland, Cham, 203–212.

[4] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering*, Oscar Pastor and João Falcão e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 491–503.

[5] Jordi Cabot and Robert Clarisó. 2023. Low Code for Smart Software Development. *IEEE Software* 40, 1 (2023), 89–93. https://doi.org/10.1109/MS.2022.3211352

[6] Cécile Camillieri, Luca Parisi, Mireille Blay-Fornarino, Frédéric Precioso, Michel Riveill, and Joël Cancela-Vaz. 2016. Towards a Software Product Line for Machine Learning Workflows: Focus on Supporting Evolution. In *10th Workshop on Models and Evolution co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016)*. Saint Malo, France. https://hal.science/hal-01484050

[7] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI]

[8] Krzysztof Czarnecki and Chang Hwan Peter Kim. 2005. Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*. ACM San Diego, California, USA, 16–20.

[9] Javad Ghofrani, Ehsan Kozegar, Anna Lena Fehlhaber, and Mohammad Divband Soorati. 2019. Applying Product Line Engineering Concepts to Deep Neural Networks. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 72–77. https://doi.org/10.1145/3336294.3336321

[10] Joan Giner-Miguelez, Abel Gómez, and Jordi Cabot. 2022. DescribeML: a tool for describing machine learning datasets. In *Companion of the 25th Int. Conf. on Model Driven Engineering MODELS 2022*. ACM, 22–26.

[11] Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. 2024. Arcee's MergeKit: A Toolkit for Merging Large Language Models. *arXiv preprint arXiv:2403.13257* (2024).

[12] Sandra Greiner, Klaus Schmid, Thorsten Berger, Sebastian Krieter, and Kristof Meixner. 2024. Generative AI And Software Variability - A Research Vision. In *Proceedings of the 18th International Working Conference on Variability Modelling of Software-Intensive Systems* (<conf-loc>, <city>Bern</city>, <country>Switzerland</country>, </conf-loc>) *(VaMoS '24)*. Association for Computing Machinery, New York, NY, USA, 71–76. https://doi.org/10.1145/3634713.3634722

[13] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing Models with Task Arithmetic. (2023). arXiv:2212.04089 [cs.LG]

[14] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Computation* 3, 1 (03 1991), 79–87. https://doi.org/10.1162/neco.1991.3.1.79 arXiv:https://direct.mit.edu/neco/article-pdf/3/1/79/812104/neco.1991.3.1.79.pdf

[15] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021. https://insights.sei.cmu.edu/library/feature-oriented-domain-analysis-foda-feasibility-study/ Accessed: 2024-Apr-4.

[16] Matthias Riebisch. 2003. Towards a more precise definition of feature models. *Modelling variability for object-oriented product lines* (2003), 64–76.

[17] Ashish Saini, Rajkumar, Amrita Kumari, and Satender Kumar. 2022. A Proposed Method of Machine Learning based Framework for Software Product Line Testing. In *2022 International Conference on Fourth Industrial Revolution Based Technology and Practices (ICFIRTP)*. 10–13. https://doi.org/10.1109/ICFIRTP56122.2022.10059409

[18] Samuel Sepúlveda, Carlos Cares, and Cristina Cachero. 2012. Towards a unified feature metamodel: A systematic comparison of feature languages. In *7th Iberian Conference on Information Systems and Technologies (CISTI 2012)*. 1–7.

[19] Paul Temple, José A. Galindo, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using machine learning to infer constraints for product lines. In *Proceedings of the 20th International Systems and Software Product Line Conference* (Beijing, China) *(SPLC '16)*. Association for Computing Machinery, New York, NY, USA, 209–218. https://doi.org/10.1145/2934466.2934472

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[21] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).