Utilizing Large Language Models for Ablation Studies in Machine Learning and Deep Learning

Sina Sheikholeslami
KTH Royal Institute of Technology
sinash@kth.se

Hamid Ghasemirahni
KTH Royal Institute of Technology
hamidgr@kth.se

Amir H. Payberah
KTH Royal Institute of Technology
payberah@kth.se

Tianze Wang
KTH Royal Institute of Technology
tianzew@kth.se

Jim Dowling
Hopsworks AB
jim@hopsworks.ai

Vladimir Vlassov
KTH Royal Institute of Technology
vladv@kth.se

Abstract

In Machine Learning (ML) and Deep Learning (DL) research, ablation studies are typically performed to provide insights into the individual contribution of different building blocks and components of an ML/DL system (e.g., a deep neural network), as well as to justify that certain additions or modifications to an existing ML/DL system can result in the proposed improved performance. Although dedicated frameworks for performing ablation studies have been introduced in recent years, conducting such experiments is still associated with requiring tedious, redundant work, typically involving maintaining redundant and mostly-identical versions of code that correspond to different ablation trials. Inspired by the recent promising performance of Large Language Models (LLMs) in the generation and analysis of ML/DL code, in this paper we discuss the potential of LLMs as facilitators of ablation study experiments for scientific research projects that involve or deal with ML and DL models. We first discuss the different ways in which LLMs can be utilized for ablation studies and then present the prototype of a tool called AblationMage, that leverages LLMs to semi-automate the overall process of conducting ablation study experiments. We showcase the usability of AblationMage as a tool through three experiments, including one in which we reproduce the ablation studies from a recently published applied DL paper.

CCS Concepts: • **Computing methodologies** → **Model development and analysis**; **Machine learning**.

Keywords: Ablation Studies, Deep Learning, Feature Ablation, Model Ablation, Large Language Models

EuroMLSys '25, Rotterdam, The Netherlands

ACM ISBN 978-1-4503-XXXX-X/2025/05
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Sina Sheikholeslami, Hamid Ghasemirahni, Amir H. Payberah, Tianze Wang, Jim Dowling, and Vladimir Vlassov. 2025. Utilizing Large Language Models for Ablation Studies in Machine Learning and Deep Learning. In *Proceedings of The 5th Workshop on Machine Learning and Systems (EuroMLSys '25).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Deep neural networks (DNNs) are becoming increasingly larger in size and complexity. In just a decade, the size of practical and popular networks has grown from around 62 million parameters (AlexNet [16]) to hundreds of billions of parameters (e.g., Megatron-Turing NLG [23], PaLM [4], Falcon [21], Vision Transformer (ViT) [6], and Llama 3 [7]). Meanwhile, through the introduction of new architectures and training approaches, different modules and mechanisms for constructing and training DNNs have become widely adapted (e.g., inception modules [25], residual connections [12], and multi-head attention [27]).

A common practice in Machine Learning (ML) and Deep Learning (DL) research and creating ML/DL systems is to start with an established network architecture (e.g., a Transformer [27]) or training method (e.g., Adam optimizer [15]) a "baseline" and try to improve that architecture or training method with regard to specific downstream tasks. The new architecture or method will then become different from the baseline in terms of the number of "components" that may have been added to or removed from the baseline. After that, to evaluate the new architecture or method, its performance on a number of benchmarks will be compared to that of the baseline as well as the state-of-the-art.

While such experimental results can be enough to determine the performance of the new architecture/method "as a whole" compared to the baseline, they may not provide information about the contribution of the different components of the architecture/method to its performance. To that end, a simple technique is to perform a systematic experiment known as ablation study [20, 22]. Essentially, in an ablation study, the changed components are added/removed from the baseline one by one or in groups, and the performance of each of these different "configurations" is then compared to

the baseline as well as the final architecture/method. This way, one can reason about the individual contribution of the different added, removed, or modified components to the overall performance.

However, while performing an ablation study seems to be a straightforward task, it is still missing from many of the scientific publications in the various fields that deal with ML/DL. This is partly due to its added cost and manual effort, as it requires performing extra experiments and maintaining multiple versions of the code required for defining and training the different configurations. Although in recent years, dedicated frameworks for ablation studies in ML/DL have been introduced to address these challenges [8, 22], many practitioners still choose to either perform ablation studies manually (e.g., to avoid adding another framework or library to their project) or forgo them entirely.

Meanwhile, a recent trend in the ML/DL research community is to use Large Language Models (LLMs) and different prompting techniques to provide novel solutions or to enhance the existing solutions for various downstream tasks [26, 28, 31]. In particular, LLMs have shown promising results in neural architecture search [2, 13] and hyperparameter optimization [9, 18, 32]. Considering that the training data for the leading publicly available LLMs includes a large corpus of codes and documents related to DNN training available on the web, LLMs have shown good potential in generating sound and relevant code for different ML/DL stages and tasks [30].

Contributions: Motivated by these efforts, we (*i*) investigate how LLMs can be leveraged for performing ablation experiments, specifically by helping in the design of the ablation study, generating correct and coherent artifacts for performing the ablation trials, and analyzing and presenting the results. We then (*ii*) introduce AblationMage, a tool that leverages an LLM to semi-automate the process of conducting ablation studies in ML/DL. Finally, we (*iii*) evaluate the usability of AblationMage using three examples related to common scenarios related to ablation studies. To the best of our knowledge, AblationMage is the first LLM-based dedicated tool for automating and conducting ablation study experiments.

2 Using LLMs for Ablation Studies

We now discuss settings and use cases in which LLMs can enable or enhance ablation study experiments. To aid the discussion, we break down an ablation study into three steps: (*i*) study design, (*ii*) experimentation, and (*iii*) analysis & presentation.

Study Design. In the first step, the user has to decide how to design the ablation study experiment. This includes identifying the baseline and the changed (added, removed, or modified) components. It is worth noting that the components can relate to any part of the ML/DL data and training pipelines, e.g., dataset features can also be included in an ablation study (a.k.a. "feature ablation"). Here, the user can provide the code and/or a description of the setting to the LLM, and the LLM can come up with suggestions for different ablation trials. Ideally, an LLM may be able to identify the changed components compared to a baseline, given a final implementation and a specification (either in code or text) of the baseline. This is, in particular, very straightforward for ablation studies that deal with the training data, e.g., when a feature ablation study is desired.

Experimentation. Given a set of specifications for the different ablation trials, either as an output from an LLM or provided by the user, the second step deals with study execution. In this step, an LLM can provide executable code and artifacts that correspond to the different ablation trials in the ablation study. The user can, e.g., also ask for a "diff" for the code of each trial to be able to quickly verify the correctness of the provided codes.

Analysis & Presentation. In the final step, the LLM can be used to provide explanations about the results of the ablation study, and to quickly generate plots and figures. The user can also ask for scripts that automate the execution and presentation of results for the full ablation study, e.g., to include as part of the artifacts of a publication.

3 AblationMage

We now discuss the prototype implementation of AblationMage, a reusable and customizable tool for ablation experiments[1]. An overview of AblationMage is shown in Figure 1. AblationMage receives the original or annotated source code of the system alongside any other documentation, such as dataset properties or the related research paper, as the input. It can then provide suggestions for annotations related to trials (if the code does not have any annotations), and the user can modify and verify them. The annotated code is then analyzed to find and extract the associated code snippets. Then, a full prompt is generated by AblationMage, and an LLM will be queried. We refer to this as the *first call*.

The response of the LLM will include the executable source code(s), instructions for conducting the trials, description of added trials, and any code added for analysis and presentation. Once the user executes the provided code(s), in case there are any errors or issues related to the correctness, they can use AblationMage to perform a *follow-up call*, which prompts the LLM to fix the specific issues. The user can repeat this process until desired artifacts are created. Example

[1]The source code of AblationMage and results from the experiments will be released upon acceptance of the paper.

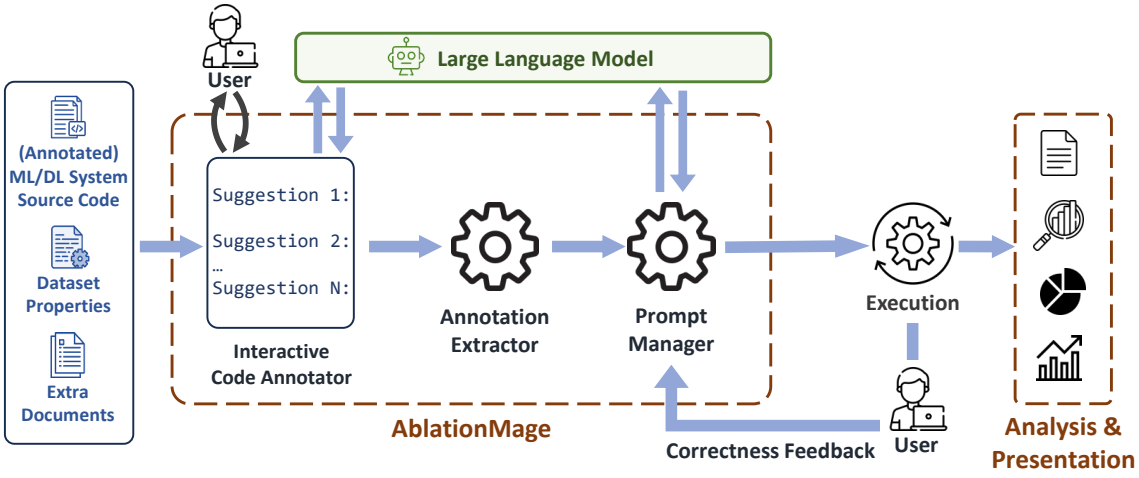

Figure 1. Overview of AblationMage.

prompts and LLM responses for each of these types of calls (first or follow-up call) can be found in the Appendix.

Currently, AblationMage supports two types of annotations: *explicit annotations* and *hint annotations*. Explicit annotations require the user to explicitly annotate the lines of code that may correspond to ablation trials. This is particularly useful for model/layer ablation trials, and the users can add "`#ABLATABLE_COMPONENT`" as comments to the lines of code that correspond to adding different layers to a model (see Listing 1 as an example). The users can also provide hint annotations, which are natural language descriptions of the desired ablation trials that are added in the form of comment blocks before related parts of the code (e.g., one might place a hint annotation before the definition of a function that creates a model, as shown in Listing 3).

Depending on how sophisticated the original implementation (base code) is, the LLM might still be able to "understand" the logic of the base code and help in both the design and implementation of the ablation trials without any explicit annotations or correct placement of hint annotations; however, the LLMs seem to benefit from documentation and code comments in code understanding [24].

AblationMage is implemented in Python and uses HuggingFace's Chat Templates to communicate with LLMs. The current version supports the OpenAI and Anthropic APIs, but it can be easily extended to support other standard LLM APIs. Next, we will evaluate the usability of AblationMage while using Claude Sonnet 3.5 as the LLM backend.

4 Evaluation & Discussion

To evaluate the usability of our prototype implementation of AblationMage, we conduct three experiments: in the first experiment, we use AblationMage to generate code for an ablation study of a few layers of a Convolutional Neural Network (CNN) trained on the CIFAR-10 dataset, where we annotate the lines of code that correspond to defining the layers of the CNN with `#ABLATABLE_COMPONENT` comments. In the second experiment, we try to reproduce the ablation studies mentioned in a recently published paper [5]. In the aforementioned paper, the authors discuss an ablation study in their manuscript and have made their code repository publicly available. However, the said repository does not contain the code related to their ablation studies. We are interested to see if we can use AblationMage to generate the code to perform the ablation studies given the original implementation by the authors. Finally, in the third experiment, we use AblationMage for a feature ablation study of the Higgs Boson Machine Learning Challenge dataset [1] when used for training an XGBoost [3] classifier.

4.1 Experiment 1: Layer Ablation of a CNN

For this experiment, we start with a typical PyTorch code that trains a simple CNN on the CIFAR-10 dataset. We are interested in performing a *layer ablation* experiment, in which the goal is to study the relative contribution of a convolutional layer (`conv2`) as well as a fully connected layer (`fc1`). To this end, we annotate the model creation code lines corresponding to those layers with `#ABLATABLE_COMPONENT` comments. Listing 1 shows the annotated code of the model creation function.

The next step is to pass this annotated code to AblationMage, and wait for it to initiate an API Call to an LLM and return the results (this corresponds to the *first call* as discussed in the previous section). Listing 1 shows how to initiate a first call. The user can expect to receive the response after a few seconds. The response is returned in the form of a text file containing both the full versions of the modified source file(s), as well as instructions on how to run the ablation study, an overview of the changes made

Listing 1: Annotating specific lines of the original code to specify ablation trials (explicit annotation).

```
...
class CIFAR10CNN(nn.Module):
    def __init__(self):
        super(CIFAR10CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1) #
              ABLATABLE_COMPONENT
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 4 * 4, 512) #
              ABLATABLE_COMPONENT
        self.fc2 = nn.Linear(512, 10)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
...
```

Listing 2: First call to an LLM using AblationMage.

```
python ablationmage.py first-call -a anthropic -m claude-3-5-
      sonnet-20241022
```

to the original source code, as well as additional notes, e.g., regarding the plots or how to interpret possible results.

We take the portion of the result that corresponds to the full source code for the ablation study to replace the original source code, and we run it. The baseline trial executes successfully, but the script fails when trying to execute the first layer ablation trial. Upon looking at the error logs and the stack trace, we realize this is due to the fact that when the layer is removed, the input of its following layer should be modified to match the output of the preceding layer. We paste the relevant parts of the stack trace in a file called `output_result.txt` and then use AblationMage to initiate a *follow-up call*.

The follow-up call is made similarly to the first call, but instead of the original source file(s), it requires the path to the new source file(s) - which we just executed - as well as the path to `output_result.txt`. The response to the follow-up call also comes in a similar format to the response to the first call. Upon replacing the source code(s) with the new version of the code and attempting the execution, the full ablation study experiment runs without a problem. The output of running the study includes several plots that can be used to compare the baseline and the different ablation trials, as well as a JSON file that includes the raw results of all the trials. We hence conclude that AblationMage can be used as an aid in conducting ablation study experiments.

4.2 Experiment 2: Reproducing the Ablation Studies of a Research Paper

We now consider a different scenario: conducting an ablation study on a paper or code authored by other researchers. In cases where authors include an ablation study in their paper, they may or may not provide the code needed to reproduce it. We aim to evaluate whether AblationMage can assist in such situations, specifically when an original implementation is available for a research paper that discusses an ablation study, but the code to reproduce the study is unavailable.

Listing 3: Example of a hint annotation for model ablation.

```
...
#ABLATION_HINT_START
# The ablation study should consist of the following layer
      ablation trials on GraphSAGE:
# - the second layer removed
# - the third layer removed
# - the fourth layer removed
# - the second, third, and fourth layers removed, while the number
       of output neurons of the first layer is 8
# - the second, third, and fourth layers removed, while the number
       of output neurons of the first layer is 64
#ABLATION_HINT_END

class GraphSAGE(torch.nn.Module):
  def __init__(self, dim_in, dim_out):
...
```

We take [5] as a relevant example of this case among the papers we had recently read. In the aforementioned paper, the authors propose an approach using Graph Representation Learning to automate the triage of emergency patients, i.e., to classify each patient's emergency treatment condition. As part of their system, they use a model based on GraphSAGE [11] as one of the possible classifiers. Since the model based on GraphSAGE showed the best performance among the different classifiers, they performed an ablation study on the relative contribution of the different layers of the model to the classification accuracy on the test set. They report their results and discuss the outcome of the different trials; however, their implementation on their publicly available repository does not contain the code for conducting the ablation study.

We add a description of the desired ablation trials in the form of a *hint annotation* and add it before the place in the code where the model is defined. Listing 3 shows the annotation we added to the original source code. Upon providing the source code to AblationMage, the response from the first call got the correct implementation of different trials, but it missed an `import` of a required module. This required a simple, one-liner fix and could be done by the user, but nevertheless, we made a follow-up call with AblationMage, and the output contained the missing `import` statement. The main part of the code was correct, and the trials were executed successfully, but there was an error related to the results visualization code that was added by the LLM. Again, this was something that could be fixed easily, but we used another follow-up call, and the final code was executed without any errors. We were able to replicate the study as described by the authors and witness mostly similar results. We further confirmed the correctness of the generated code by comparing it with the ablation study code provided by the authors of the original article upon our request.

Listing 4: Example of a hint annotation for feature ablation.

```
...
def main():

    data = pd.read_csv('training.csv')

    #ABLATION_HINT_START
    # The ablation study should consist of the following feature
        ablation trials:
    # - remove 'DER_mass_MMC'
    # - remove 'DER_mass_transverse_met_lep'
    # - remove 'DER_mass_vis'
    # - remove 'DER_pt_h'
    # - remove 'DER_deltaeta_jet_jet'
    # - remove 'DER_mass_jet_jet'
    # - remove pairs of the above features, one pair at a time
    #ABLATION_HINT_END

    data['Label'] = data['Label'].map({'s': 1, 'b': 0})
...
```

4.3 Experiment 3: Feature Ablation of the Higgs Boson Challenge Dataset

We now look at another component of an ML/DL system that may benefit from ablation study experiments: the dataset used for training the model. Depending on the type and modality of the data, the examples within a dataset can have several ablatable dimensions (and in a multi-modal datasets, modalities themselves can be ablated), but perhaps the most typical type of data ablation is *feature ablation*. In a feature ablation study, we remove individual (or groups of) features (e.g., channels in an image, or columns in a tabular dataset) from the training dataset, train the same model on each variation of the dataset, and examine the difference in the performance of the model. This simple examination can provide us with useful information on the importance of different dataset features.

To demonstrate the capability of AblationMage for data ablation studies, we perform a feature ablation study on the Higgs Boson ML Challenge Dataset [1], a well-known tabular dataset. The training data consists of 250000 particle collision events, and we want to train an XGBoost classifier to determine whether a collision event is a signal, or background noise. Each event has an ID column, 30 feature columns, a weight column, and a label column. The test set comprises 550000 events with an ID and 30 feature columns. We are interested to know about the individual and pairwise importance of 6 of the features to the performance of the classifier trained on the dataset.

We add a description of the desired ablation trials as a hint annotation and add it immediately after the line of code in which we load the training dataset. Listing 4 shows the annotation we added to the original source code. Upon providing the source code to AblationMage, the response from the first call gets the correct implementation required to execute all the trials.

4.4 Discussion

The results of these experiments verify the usability and potential of AblationMage as a tool to semi-automate the design and execution of ablation studies. Regarding the possibility of full automation, although LLMs have shown promising performance in generating ML/DL code, the users should still verify the correctness of the generated codes to make sure they correspond exactly to the target ablation study experiments. That being said, a natural next step for improving AblationMage would be to provide more automation for the correctness/verification loop, possibly eliminating or reducing the need for multiple executions of possibly faulty code, e.g., by detecting and fixing common errors through static code analysis [10, 19].

Another possible challenge stems from the limited maximum number of input and output tokens and the limited context length of current LLMs [14]. This, in particular, may make it challenging for LLMs to understand and/or modify multiple source files simultaneously, especially in cases where the ML/DL system implementation spans many files and possibly hundreds of thousands of lines of code. This challenge can be alleviated by providing more sophisticated annotations and asking the LLM to provide experimentation code for the ablation trials one at a time. This may also ensure that the context length growth stays within the context length of the current publicly available LLMs such as GPT, Claude, and DeepSeek. This is particularly possible for code generation and code understanding tasks related to ablation studies, since the trials in an ablation study are intrinsically independent of each other (i.e., each trial can be generated and performed independently of the others). Nevertheless, we should also note the rapid improvement of LLMs in this regard; e.g., OpenAI's recent `o3-mini` model has a context window of 200K tokens and can have 100K tokens in its outputs, showing a remarkable increase compared to `GPT-4`'s 8192-token context window and maximum output size. These improvements can enable LLMs to assist users with much more complex ablation studies. Our tests of AblationMage on code repositories of complex DL frameworks and systems, while using more than a hundred source and documentation files as input documents, show that LLMs can successfully analyze and generate the desired code.

Limitations of existing dedicated ablation study frameworks (AutoAblation [22] and ABLATOR [8]) compared to utilizing state-of-the-art LLMs for conducting ablation studies include their dependence on their developers for implementing and providing support for different ML and DL frameworks, and the initial investment and learning curve required to learn their workflows, which might not be justified for smaller or simpler projects. We predict that as more users adopt LLM-based tools for their coding tasks [33], using LLM-based tools such as AblationMage for conducting ablation studies will also become more common.

A natural extension to AblationMage would be to provide full automation using agentic frameworks such as LangGraph [17] and AutoGen [29]. By leveraging these frameworks, a complete ablation study workflow can be structured as interactions among multiple specialized agents (e.g., for code generation and optimization, experiment execution, and result analysis), each potentially utilizing different LLMs and iterating over generated artifacts and outputs (such as errors and stack traces) until a desired outcome is achieved. Meanwhile, a human-in-the-loop can validate artifacts and provide high-quality guidance for complex tasks, ensuring correctness and refining the workflow as needed.

5 Summary

In this paper, we investigated the potential of LLMs as aids for performing ablation studies in ML/DL. We highlighted a number of ways in which LLMs can be a facilitator for ablation studies, in the different steps of study design, experimentation, and analysis & presentation, and discussed the prototype implementation of AblationMage, an LLM-based tool that can semi-automate the process of conducting an ablation study by exploiting explicit and hint annotations provided by the users. Using three experiments, including one in which we reproduced the ablation studies of a recently published paper, we evaluated the usability of AblationMage and briefly reviewed its current challenges and possible extensions.

References

[1] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. 2015. The Higgs boson machine learning challenge. In *NIPS 2014 workshop on high-energy physics and machine learning*. PMLR, 19–55.

[2] Angelica Chen, David Dohan, and David So. 2023. EvoPrompting: language models for code-level neural architecture search. *Advances in Neural Information Processing Systems* (2023).

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[4] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[5] Annamaria Defilippo, Pierangelo Veltri, Pietro Lió, and Pietro Hiram Guzzi. 2024. Leveraging graph neural networks for supporting automatic triage of patients. *Scientific Reports* 14, 1 (2024), 12548.

[6] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. 2023. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*. PMLR, 7480–7512.

[7] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[8] Iordanis Fostiropoulos and Laurent Itti. 2023. ABLATOR: Robust Horizontal-Scaling of Machine Learning Ablation Experiments. In *International Conference on Automated Machine Learning*. PMLR, 19–1.

[9] Hamid Ghasemirahni, Alireza Farshin, Mariano Scazzariello, Marco Chiesa, and Dejan Kostić. 2024. Deploying Stateful Network Functions Efficiently using Large Language Models. In *Proceedings of the 4th Workshop on Machine Learning and Systems*. 28–38.

[10] Xueting Guan and Christoph Treude. 2024. Enhancing Source Code Representations for Deep Learning with Static Analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. 64–68.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Ganesh Jawahar, Muhammad Abdul-Mageed, Laks VS Lakshmanan, and Dujian Ding. 2023. LLM Performance Predictors are good initializers for Architecture Search. *arXiv preprint arXiv:2310.16712* (2023).

[14] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. 2024. LLM Maybe LongLM: Self-extend LLM context window without tuning. *arXiv preprint arXiv:2401.01325* (2024).

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv e-prints* (2014), arXiv–1412.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[17] LangChain-AI. 2025. LangGraph. https://github.com/langchain-ai/langgraph. Accessed: 2025-02-09.

[18] Siyi Liu, Chen Gao, and Yong Li. 2024. Large Language Model Agent for Hyper-Parameter Optimization. *arXiv preprint arXiv:2402.01881* (2024).

[19] Panagiotis Louridas. 2006. Static code analysis. *Ieee Software* 23, 4 (2006), 58–61.

[20] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. 2019. Ablation studies in artificial neural networks. *arXiv preprint arXiv:1901.08644* (2019).

[21] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116* (2023).

[22] Sina Sheikholeslami, Moritz Meister, Tianze Wang, Amir H Payberah, Vladimir Vlassov, and Jim Dowling. 2021. Autoablation: Automated parallel ablation studies for deep learning. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. 55–61.

[23] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990* (2022).

[24] Demin Song, Honglin Guo, Yunhua Zhou, Shuhao Xing, Yudong Wang, Zifan Song, Wenwei Zhang, Qipeng Guo, Hang Yan, Xipeng Qiu, and Dahua Lin. 2024. Code Needs Comments: Enhancing Code LLMs with Comment Augmentation. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 13640–13656. https://doi.org/10.18653/v1/2024.findings-acl.809

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[26] Kotaro Tanahashi, Yuichi Inoue, Yu Yamaguchi, Hidetatsu Yaginuma, Daiki Shiotsuka, Hiroyuki Shimatani, Kohei Iwamasa, Yoshiaki Inoue, Takafumi Yamaguchi, Koki Igari, et al. 2023. Evaluation of Large Language Models for Decision Making in Autonomous Driving. *arXiv preprint arXiv:2312.06351* (2023).

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[28] Jiaqi Wang, Zhengliang Liu, Lin Zhao, Zihao Wu, Chong Ma, Sigang Yu, Haixing Dai, Qiushi Yang, Yiheng Liu, Songyao Zhang, et al. 2023. Review of large vision models and visual prompt engineering. *Meta-Radiology* (2023), 100047.

[29] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155* (2023).

[30] Jinglue Xu, Jialong Li, Zhen Liu, NAV Suryanarayanan, Guoyuan Zhou, JIA GUO, Hitoshi Iba, and Kenji Tei. 2024. Large Language Models Synergize with Automated Machine Learning. *Transactions on Machine Learning Research* (2024). https://openreview.net/forum?id=RDEaIfOiJM

[31] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. 2023. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129* (2023).

[32] Michael R Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. 2023. Using Large Language Models for Hyperparameter Optimization. *arXiv e-prints* (2023), arXiv–2312.

[33] Dewu Zheng, Yanlin Wang, Ensheng Shi, Hongyu Zhang, and Zibin Zheng. 2024. How Well Do LLMs Generate Code for Different Application Domains? Benchmark and Evaluation. *arXiv preprint arXiv:2412.18573* (2024).

A Appendix: Example Prompts and LLM Responses

Here we provide examples of the prompts that AblationMage creates as well as the responses from Claude Sonnet 3.5, the LLM backend we used for our experimental evaluation. These prompts and responses correspond to the second experiment, in which we reproduced the ablation studies of a recently published paper by adding hint annotations to the original code. The prompts and snippets of the responses for the first call are shown in Figure 2, while Figure 3 shows the same for the first follow-up call AblationMage makes. In this case, the "Feedback/Error Output" was the stack trace of errors after the execution of the code provided by the LLM in response to the first call.

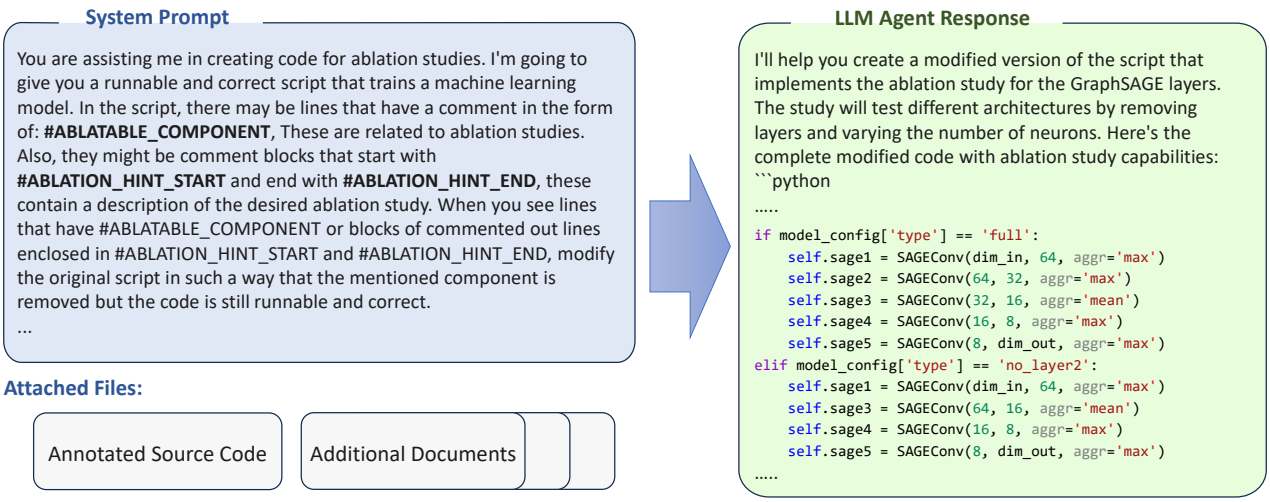

Figure 2. The prompt prepared by AblationMage for the "first call" to the LLM and part of the response. The user provides the path to the annotated source code and other documents.

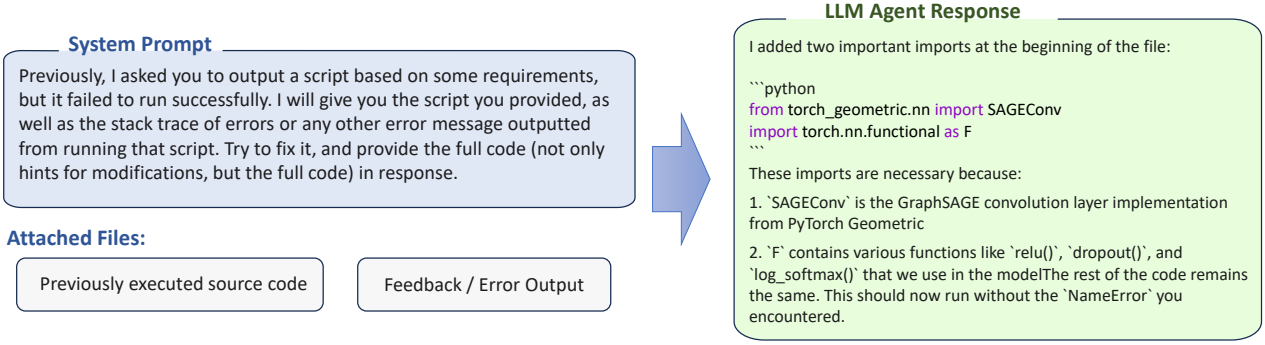

Figure 3. The prompt prepared by AblationMage for the "follow-up call" to the LLM and part of the response.