Roadmap of Machine Learning Issues

Yuxuan Jiang

Roadmap of ***Machine Learning Issues***

From a high-level perspective, this project aims to help developers detect silent/latent correctness issues (bugs) in machine learning pipelines before those issues result in unbearable costs. This file documents the storyline of the project, the milestones, and the goals of each milestone. It is a living document that will be updated as the project progresses.

1. Key Research Questions

The following are the key research questions that we will answer in this project:

1. What are the silent/latent correctness issues of machine learning pipelines?

   Reasons why this question is important:

   1. The insights from this question will help us understand the problem space and guide our design decisions.
   2. The insights might also help future researchers to design better abstractions to prevent those issues from happening in the first place.

2. How can we detect those silent/latent correctness issues early on?

   Reasons why this question is important:

   1. A tool that can detect those issues early on will help developers save a lot of time and money.

The meaning of "silent/latent" is that the issues are either completely silent (i.e., no error message is reported) or only raise exceptions when the pipeline is executed for a long time (e.g., hours).

For now, we are focusing on user level issues (i.e., issues that are caused by the user code or the user data). We are not focusing on issues that are caused by the environment (e.g., GPU drivers, CUDA, etc.) or issues that are caused by the ML library itself (e.g., PyTorch, TensorFlow, etc.).

Please see CONTRIBUTING.md#bug-selection for more details.

What are not (necessarily) the scope of this project?

1. **Hyperparameter tuning**: This is a very important yet heavily studied problem. As a result, we want to focus on other problems that are less studied but still matters a lot.
2. **Solving environment issues**: Engineering-wise, this is a very hard problem to solve due to the complexity of the Python ecosystem, lack of standardization in the ML community, and the lack of control over the user's environment. We feel like this problem is better solved by the ML community as a whole or by the Software Engineering community.
3. **Issues that are not silent/latent**: Issues that raise exceptions early on are usually easy to detect and fix. Though they still troubles developers a lot, the cost of such issues and effort to debug them are usually not as high as silent/latent issues.
4. **Finding issues in GPU Drivers, CUDA, or other non-user-level code**: Quality assurance of those components is very important, and our project should be able to help with that. **Our technique is indeed aimed at detecting general silent/latent correctness issues in ML pipelines due to various root causes despite their location no matter it is in user level code or the environment. However, we choose to focus on user issues for now as they are easier to control.** For example, a PyTorch API raising an exception might be caused by a bug in PyTorch, a bug in CUDA, or a bug in the user code or an environment installation issue. As a result, we feel like for now it is better to focus on user issues as they are easier to control. We might consider expanding our scope in the future.

2. Introduction & Expected Contribution

As machine learning (ML) gains popularity in both academia and industry, the correctness of ML pipelines becomes increasingly important. A silent or latent correctness issue in an ML pipeline often goes unnoticed, which adds to the challenges in timely detection and debugging of such issues. The cost of detecting and fixing such issues is further amplified as the scale of ML projects increases astronomically with the recent LLM arms race [3], usually involving hours or days of downtime and hours or days of wasted machine hours [4].

For example, a gradient clipping bug in Deepspeed's BF16Optmizer caused the certain parts of the model to silently diverge during training. Though the bug was detected before the divergence became too huge, the developers still had to spend 12 days to fix the bug [4]. On the other hand, if the bug was not detected early on, the developers might have to face the cost of retraining the model from scratch as the end model weights will be inconsistent.

Another example is that a model expecting input of certain batch size might raise an exception at the end of the training when the size of the training dataset is not a multiple of the batch size. This latent issue will cause the entire training to fail after hours of training [5].

Reproducing and fixing silent/latent issues imposes a huge burden on developers. Manifestation of such issues usually requires long execution time to surface in the form of exceptions or noticeable discrepancies in the metrics. Specific input might also be required for debugging, which is hardly available. As reported in CMU SEI's Interview with data scientists [6], "A typical thing that might happen is that in the production environment, something would happen. We would have a bad prediction, some sort of anomalous event. And we were asked to investigate that. Well, unless we have the same input data in our development environment, we can't reproduce that event.". In addition, even if the bug has been detected, it is also hard to estimate the time needed to fix it, as ML engineering still largely relies on trial and error [7]. Consequently, a tool that can detect such issues early on will help developers save a lot of time and money.

Silent or latent correctness issues are easy to make and hard to detect. Here we try to more formally define the underlying reasons:

1. Flexible and dynamic development process of ML pipelines. Thus, ML pipelines are less formally managed [8].

2. Complexity of the ML stack. ML libraries usually have a large number of APIs and complex configuration knobs. Thus, it is hard for developers to correctly understand API usage and to formally verify the correctness of their pipelines [9]. Dynamic types and the implicit broadcasting and type conversion rules also makes it harder to detect issues at compile time.

3. Data-dependent nature of ML pipelines. Thus, it is harder to apply traditional software engineering techniques (e.g., unit testing) to ML pipelines. For example, a dataset-model mismatch can lead to silent accuracy decrease (PyTorch#FORUM84911), Additionally, certain correctness issues might only be triggered by certain data [9].

4. No explicit manifestation of the issue (e.g., no error message or discrepancy needs time to accumulate).

Existing works fall short in the following ways:

1. Existing work tend to focuse on ensuring quality of ML models (DeepXplore [11]), DL frameworks (CRADLE [12] and DeepREL [13]), and compilers (NNSmith [Liu_2023]) rather than the quality of ML pipelines where bugs can reside in user-level code.
2. Existing work focusing on ML pipelines focuses on a very specific type of issues (e.g. PyTea [5], RANUM [10]) or are limited because they need prior runs of the same pipeline to pinpoint the issue (e.g. MLDebugger [14]).
3. Existing work struggle to scale beyond simple ML pipelines due to the enormous amount of Python syntax and API to support and the inherent limitation of static analysis [5].

We want to create a general-purpose tool that can detect a wide range of silent/latent correctness issues in ML pipelines. Our high-level approach is to leverage dynamic analysis to infer the likely invariants of ML pipelines and then use those invariants to detect correctness issues.

A few directions we are considering:

1. **Repository Mining + Static Analysis**: Infer library API specifications (pre-conditions and post conditions) from examples [15]. Then, use the inferred specifications to statically verify the user code connecting two API calls in the new ML pipeline.
   - Pro: The inferred specifications are likely to be correct and comprehensive as they are mined from a huge number well-maintained examples.
   - Con: The inferred invariants might be best for detecting issues that raises exceptions. It might not be as good for detecting issues that are completely silent as these are usually caused by the data or inappropriately configuration parameters.
2. **Predictive Analysis**: The hypothesis is that although ML pipelines are expensive to run, we can use a small number of executions to predict the correctness of future executions.
   - Pro: The cost of running ML pipelines is reduced.
   - Con: The correctness of the predictions is not guaranteed.
3. **Online Anomaly Detection**: The hypothesis is that the silent/latent correctness issues will cause the execution of the ML pipeline to violate certain invariants. We can use online anomaly detection to detect such violations.

We expect to make the following contributions:

1. In-depth understanding of the silent/latent correctness issues of ML pipelines.

2. A tool that can detect a wide range of correctness issues in ML pipelines at compile time or early on during runtime.

3. Stages

1. Understanding Real-World Silent/Latent Correctness Issues [*Current Stage*]
   - ☒ Collecting real-world issues
     - ☒ Collecting issues from GitHub issues
     - ☒ Collecting issues from StackOverflow
     - ☒ Collecting issues from other sources (e.g., Reddit, Twitter, etc.)
   - ☒ Analyzing the issues
   - ☒ Summarizing the issues
2. Testing the Hypothesis [*Current Stage*]
   - ☐ Testing all the hypothesis we have
     - ☐ Implementing prototypes
     - ☐ Evaluating prototypes

3. Iterating on the Hypothesis
    - ☐ Iterating on the hypothesis
        - ☐ Desiding on one potential direction to focus on
        - ☐ Finish iterating on the direction with data collected from Stage 1
        - ☐ Repeating Stage 2 and 3
    - ☐ Large-Scale Evaluation
        - ☐ Collecting more real-world issues (probably from industry collaborators)
        - ☐ Evaluating the tool on the issues
4. Large-Scale Evaluation
    - ☐ Collecting more real-world issues (probably from industry collaborators)
    - ☐ Evaluating the tool on the issues
5. Writing Paper
    - ☐ Writing the paper
    - ☐ Submitting the paper

4. Milestones

4.1. Milestone 1: Understanding Real-World Silent/Latent Correctness Issues

- **Outcome**: A list of 10 real-world silent/latent correctness issues that 1) meet our bug-choosing criteria, 2) are reproducible, and 3) are analyzed. We will also analyze the issues and summarize the findings, to guide our design decisions.
- **Satisfying Criteria**: This milestone will be considered as satisfied if we have several issues that meet our bug-choosing criteria, are reproducible, and are analyzed.
- **Deadline**: No deadline. We will keep collecting issues even after this milestone is satisfied.

4.2. Milestone 2: Have a working prototype

- **Outcome**: A working prototype built from our hypothesis that can detect a wide range of silent/latent correctness issues in ML pipelines. The minimum requirement is being able to infer certain types of constraints from the pipeline.
- **Satisfying Criteria**:
    - This milestone will be considered as satisfied if the working prototype shows promising results on the issues collected in Milestone 1.
    - A working prototype is considered as "working" if it can detect some silent/latent correctness issues in ML pipelines at a satisfying practicality (e.g., the cost of running the pipeline is not too high).
- **Deadline**: 2024 Winter

4.3. Milestone 3: Reaching a satisfying utility

- **Outcome**: A tool that can detect a wide range of silent/latent correctness issues in ML pipelines with a satisfying utility.
- **Satisfying Criteria**: This milestone will be considered as satisfied if the tool can detect a wide range of silent/latent correctness issues in ML pipelines with a satisfying utility. We also need a very large set of real-world issues to evaluate the tool.
- **Deadline**: 2024 Summer

4.4. Milestone 4: Writing Paper

- **Outcome**: A paper that describes the tool and the findings.
- **Satisfying Criteria**: This milestone will be considered as satisfied if the paper is accepted by a top-tier conference.
- **Deadline**: 2024 Fall (Before the OSDI deadline)

5. Goals by the end of this semester (2024 Winter)

- Finish Milestone 1 and 2.
- Have a plan for Milestone 3.
- Have a revised introduction, background, and related work section for the paper (Milestone 4).

6. Existing Flaws

Since this project aims to leverage dynamic invariant inference to detect silent/latent correctness issues in ML pipelines, the following are the existing flaws that we need to address threats to validity related to this approach:

1. Input:
    - Whether the input (e.g. collected trace, ml pipelines) is representative of real-world ML pipelines?
    - Whether the input itself is correct? If not, the invariants inferred from the input might be incorrect.
    - Noise in the input (e.g. irrelevant variables, or indeterminism, etc.) might affect the quality of the inferred invariants.
2. Performance:
    - Whether the performance of the tool is acceptable? If not, the tool might not be practical.
3. Accuracy:
    - Whether the inferred invariants are accurate? If not, the tool might not be able to detect the issues.

References

[1] S. Bekman, “BLOOM: Megatron-DeepSpeed.” Hugging Face Blog, 2022. Available: https://huggingface.co/blog/bloom-megatron-deepspeed

[2] Anonymous, “Introducing gemini: Google’s AI for understanding video.” Google AI Blog, 2023. Available: https://blog.google/technology/ai/google-gemini-ai/

[3] Anonymous, “Building the next generation of AI with large language model LLAMA.” Meta AI Blog, 2023. Available: https://ai.meta.com/blog/large-language-model-llama-meta-ai/

[4] BigScience Workshop, “Chronicles of BigScience TR11-176B-ML training.” GitHub repository, 2022. Available: https://github.com/bigscience-workshop/bigscience/blob/master/train/tr11-176B-ml/chronicles.md#2022-04-30-hanging-at-eval

[5] H. Y. Jhoo, S. Kim, W. Song, K. Park, D. Lee, and K. Yi, “A static analyzer for detecting tensor shape errors in deep neural network training code.” 2021. Available: https://arxiv.org/abs/2112.09037

[6] G. A. Lewis, S. Bellomo, and I. Ozkaya, “Characterizing and detecting mismatch in machine-learning-enabled systems.” 2021. Available: https://arxiv.org/abs/2103.14101

[7] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, “An empirical study of common challenges in developing deep learning applications,” in *2019 IEEE 30th international symposium on software reliability engineering (ISSRE)*, 2019, pp. 104–115. doi: 10.1109/ISSRE.2019.00020.

[8] D. Sculley *et al.*, “Hidden technical debt in machine learning systems,” in *Proceedings of the 28th international conference on neural information processing systems - volume 2*, in NIPS’15. Cambridge, MA, USA: MIT Press, 2015, pp. 2503–2511.

[9] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, “Taxonomy of real faults in deep learning systems.” 2019. Available: https://arxiv.org/abs/1910.11015

[10] L. Li, Y. Zhang, L. Ren, Y. Xiong, and T. Xie, “Reliability assurance for deep neural network architectures against numerical defects.” 2023. Available: https://arxiv.org/abs/2302.06086

[11] K. Pei, Y. Cao, J. Yang, and S. Jana, “DeepXplore: Automated whitebox testing of deep learning systems,” in *Proceedings of the 26th symposium on operating systems principles*, in SOSP ’17. ACM, Oct. 2017. doi: 10.1145/3132747.3132785.

[12] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, “CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries,” in *2019 IEEE/ACM 41st international conference on software engineering (ICSE)*, 2019, pp. 1027–1038. doi: 10.1109/ICSE.2019.00107.

[13] Y. Deng, C. Yang, A. Wei, and L. Zhang, “Fuzzing deep-learning libraries via automated relational API inference.” 2022. Available: https://arxiv.org/abs/2207.05531

[14] R. Lourenço, J. Freire, and D. Shasha, “Debugging machine learning pipelines,” in *Proceedings of the 3rd international workshop on data management for end-to-end machine learning*, in SIGMOD/PODS ’19. ACM, Jun. 2019. doi: 10.1145/3329486.3329489.

[15] "PyTorch/examples." Available: https://github.com/pytorch/examples