Title: **Run-Time Prevention of Software Integration Failures of Machine Learning APIs**
Authors: Chengcheng Wan, Yuhan Liu, Kuntai Du, Henry Hoffmann, Junchen Jiang, Michael Maire, **Shan Lu**
Published in: OOPSLA 2023

Alex Qin
Engineering Research Paper
Question–Answer Form
Last Edit at 2024-09-20

1) What is your take-away message from this paper?

- Importance of Addressing Integration Failures in ML APIs
- Run-Time Adaptation Methods
  - symbolic analysis (we can use LLM to do that https://x.com/AndreasZeller/status/1837066182509166923)
- Knowledge Graphs for verification
- not only traditional ML

2) What is the motivation for this work (both people problem and technical problem), and its distillation into a researchquestion? *Why doesn't the people problem have a trivial solution? What are the previous solutions and why are they inadequate?*

The Machine Learning (ML) APIs are indeterminate and unpredictable. For example,

- Every task may have more than 1 correct answers.
- ML APIs documents didn't include all the possible outputs.
- The ML model is inherently probabilistic and may return different results for the same input.

This paper did testing over 55 apps from github, and perform ~5000 unit tests. Here are the key root cause of integration failures:

- Mismatch
  - Hierarchy mismatch
    - e.g. Expect *Light* but got *Roof Lantern*
  - Perspective mismatch
    - e.g. Expect *Wall socket* but got *Plastic*
  - Focus mismatch
    - e.g. Expect *Plumbing* but got *Hand* (another object from the figure)
- Incorrectness of ML API output
  - Neural Network Vulnerability
    - Noise …
  - Low Input Quality
    - e.g. Low resolution image

"While much prior work focuses on improving neural network accuracy, our study shows that software provides additional information to tackle the incorrectness of ML API"

The research question is: How to converts ML API output at runtime to better fit the software component.

Related Work

- ML Services
  - Security
  - Evaluate the effectiveness of ML cloud systems
  - ML Serser side services
- Software Engineering
  - Develop ML programs

    - Optimize ML models ot improve overall software accuracy
- WebDev
    - Intergrate WEB APIs can cause crashes or error
- Reliability
    - Runtime-error patching

3) What is the proposed solution (hypothesis, idea, design)? *Why is it believed it will work? How does it represent an improvement? How is the solution achieved?*

- Static analysis to the API handler
    - Identify the desired API outputs as *focal values*
- Runtime verification
    - Check the output of the ML API at runtime
    - focal or not
- Incompatibility checker
    - Hierarchy checker, Perspective checker, Focus checker

Methodology

- Obtaining Focal values
    - dynamic symbolic execution approach [Irlbeck et al. 2015]
- Understanding Cognitive Relationships
    - Knowledge graph database for vision tasks
    - Google Natural Language AI for NLP tasks
- Tackling Hierachy, Perspective and Focus mismatch
- API output
    - use two overlapped ML API and combine them
- Check API consistency

Still Suffering from false positives

4) What is the author's evaluation of the solution? *What logic, argument, evidence, artifacts (e.g., a proof-of-concept system), or experiments are presented in support of the idea?*

1. **Experimental Setup**: The evaluation was conducted using a set of 65 real-world open-source applications that integrate ML APIs. These applications covered a wide range of ML tasks, including vision, language, and speech recognition. Each application was tested with 100 inputs designed to exercise different control-flow paths influenced by ML API outputs, totaling 6,500 test runs.
2. **Metrics Used**: The authors measured SmartGear's effectiveness using two main metrics:
    - **Detection Coverage**: The ability of SmartGear to detect errors in ML API outputs, including mismatches and incorrectness.
    - **Prevention Success**: The rate at which SmartGear successfully prevents failures by converting incompatible ML API outputs into compatible ones.
3. **Baseline Comparisons**: SmartGear was compared against four baseline techniques:
    - **Crash-Only**: Detects only fail-stop symptoms such as crashes.
    - **ML-API-Only**: Uses confidence scores from the ML APIs to detect errors.
    - **Software-Only**: Detects errors based on software execution paths that do not align with expected focal values.
    - **Software-Segment**: Detects errors using a segmentation technique to aggregate ML API outputs.
4. **Key Findings**:

- **High Detection Rate**: SmartGear detected 70% of all ML API output errors during testing (718 out of 1,026 errors), which was significantly higher than all other baselines.
- **Low False Positives**: SmartGear reported only 26 false positives, demonstrating high accuracy in distinguishing between correct and incorrect or mismatched outputs.
- **Prevention Effectiveness**: SmartGear successfully prevented 67% of potential integration failures, increasing the average correct execution rate of the tested applications from 84% to 95%.

5. **Evidence and Artifacts**:
   - **Proof-of-Concept Implementation**: SmartGear was implemented as a runtime tool with Python function decorator interfaces, which allowed easy integration into existing applications.
   - **Use of Knowledge Graphs**: The tool leveraged a public knowledge graph (Wikidata) to understand the cognitive relationships between API outputs and software expectations, a key feature that contributed to its superior performance.
   - **Empirical Study Data**: The initial empirical study of 55 applications provided a solid foundation for understanding the types of ML API integration failures, directly informing the design of SmartGear.
6. **Experiments and Results**:
   - The experiments included both quantitative (performance metrics) and qualitative (analysis of error types and conversion success) evaluations, providing robust evidence of SmartGear's capabilities.
   - SmartGear's ability to dynamically detect and fix errors was demonstrated through real-world scenarios, such as a flood detection application and a voice assistant system, illustrating its practical impact.

5) What is your analysis of the identified problem, idea and evaluation? *Is this a good idea? What flaws do you perceive in the work? What are the most interesting or controversial ideas? For work that has practical implications, ask whether this will work, who would want it, what it will take to give it to them, and when might it become a reality?*

I think the ML API integration error is a very realistic problem. Just like the LLM issue I encountered during my internship at InsightFinder, it can affect the accuracy of the business, sometimes requiring an engineer to spend months constantly adjusting it to achieve the best results.

I think the issues I encounter with the LLM API and the integration issues with the ML API have similar aspects. However, the uncertainty of LLM illusions is more severe than that of the ML API. For example, when I ask LLM to output content in markdown format, it is not encoded in standard markdown format, and I need to use an additional markdown linter to adjust it, otherwise it will display errors on the frontend.

6) What are the paper's contributions (author's and your opinion)? *Ideas, methods, software, experimental results, experimental techniques...?*

- **Ideas**: ML API integration failure
  - The authors conducted an empirical study on 55 real-world applications to identify common causes of integration failures when using ML APIs.
- **Methods**: Use of Knowledge Graphs for Mismatch Detection
  - By using structured knowledge about real-world entities, SmartGear can dynamically reason about output compatibility in ways that traditional static checks cannot achieve.
- **Evaluation**: The paper provides a thorough evaluation of SmartGear using 65 real-world applications, comparing it against multiple baseline approaches. The results show that SmartGear significantly improves error detection and failure prevention, with low false-positive rates.

7) What are future directions for this research (author's and yours, perhaps driven by shortcomings or other critiques)?

LLM, LLM and LLM.

Actually they have another work *Keeper: Automated Testing and Fixing of Machine Learning Software* published on TOSEM'24 after that.

8) What questions are you left with? ***What questions would you like to raise in an open discussion of the work (review interesting and controversial points, above)? What do you find difficult to understand? List as many as you can, at least three, not including questions that can be answered quickly by searching the internet.***

It seems that the main issue for Machine Learning Reliability lies in the fact that traditional machine languages are unable to express high-level languages.

We cannot say that the Machine Learning API returns incorrect results. As long as the model is in practical use, its accuracy is generally high (detecting model drift is another part of the work). So, the content returned by the ML API is mostly resonable.

This work uses Knowledge Graph to understand the content of "natural language" and its inner rationability. However it still looks like hard coded rules.

I know there are tons of works on "use LLM to correct ML". However I also do not believe it's an ultimate solution.