A Replication of Are Machine Learning Cloud APIs Used Correctly

Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, Shan Lu
University of Chicago
{cwan, shicheng2000, hankhoffmann, mmaire, shanlu}@uchicago.edu

***Abstract*—This artifact aims to provide benchmark suite, data, and script used in our study "Are Machine Learning Cloud APIs Used Correctly?". We collected a suite of 360 non-trivial applications that use ML cloud APIs for manual study. We also developed checkers and tool to detect and fix API mis-uses. We hope this artifact can motivate and help future research to further tackle ML API mis-uses. All related data are available online.**

I. Introduction

Developing machine learning software systems also requires cross-domain knowledge. For non-experts, using a third-party API instead of building a network from scratch is a better solution: ML cloud APIs allow programmers to incorporate neural networks into software systems without designing and training the learning model themselves [2]. Unfortunately, there are still a number of challenges that must be addressed to ensure that the resulting applications are both correct and efficient. Unlike traditional APIs that are coded to perform well-defined algorithms, ML APIs are trained to perform cognitive tasks whose input are digitalized real-world visual, audio and text content, and semantics cannot be reduced to concise mathematical or logical specifications.

To understand the problems in real-world software using ML APIs and design appropriate solution, our work [7] performed the first empirical study of ML cloud API mis-uses. This artifact [6] aims to provide benchmark suite, data, and script used in our study, and thus motivate and help future research to further tackle ML API mis-uses.

In this artifact, we provide an organized database used for our manual studies: a suite of 360 non-trivial projects that use Google/Amazon ML APIs, including vision, speech, and language. It includes (1) the software project name, GitHub link, and exact release number; (2) the exact file and source-code line location of the 247 ML-API mis-uses in these projects; (3) A detailed explanation, including the anti-pattern category and a patch suggestion; (4) test cases to trigger the mis-use.

Also we provide code and data for the tools/checkers we developed for detecting and fixing some of the anti-patterns: (1) Output mis-interpretation checker; (2) Asynchronous API call checker; (3) Constant-parameter API call checker; (4) API wrappers.

II. Organized Database

The result of TABLE III (manual study part) in our paper [7] can be reproduced by this organized databse.

A. Applications

We collected a suite of 360 non-trivial applications that use Google/Amazon ML APIs, including 120 applications for each of the three major ML domains: vision, speech, and language.

They cover different programming languages, Python(80%), JS (13%), Java (3%), and others (4%). Around 80% of these applications use Google Cloud AI and around 20% use AWS AI, with 1% using both. We used fewer applications that use AWS AI service, as AWS Lambda [1], a serverless computing platform, sometimes makes it difficult for us to judge the exact application workflow. The sizes of these applications range from 46 to 3 millions lines of code, with 2228 lines of code being the median size and around 40% of them having more than 10 thousand lines of code. Most of these applications are young, created after 2018 (98% of them). They have a median age of around 18 months at the time of our study. This relatively young age distribution reflects the fact that the power of deep learning has only been recently recognized, and yet is being adopted with unprecedented pace and breadth.

We manually confirmed these applications each target a concrete real-world problem, integrate the ML API(s) in their workflow, and conduct some processing for the input or the output of the ML API, instead of simply feeding an external file into the ML API and directly printing out the API result. We do not have a way to accurately check how seriously these applications have been used in the real world, and it is possible that some of these 360 applications have not been widely used.

These applications can be found at "**All_benchmarks**" folder in our artifact.

B. Mis-uses

Because of the young ages of ML API services and hence the applications under study, we could *not* rely on known API mis-uses in their issue-tracking systems, which are very rare. Instead, we must discover API mis-uses *unknown to the developers* by ourselves.

Since there is no prior study on ML API mis-uses, our mis-use discovery can not rely on any existing list of anti-patterns. Instead, our team, including ML experts, carefully studies API manuals, intensively profiles the API functionality

and performance, and then manually examines every use of an ML API in each of the 360 applications for potential mis-uses. All the manual checking is conducted by two of the authors, with their results discussed and checked by all the co-authors.

We identify a wide variety of applications as containing ML API mis-uses including those both: small and large, young and old, AWS and Google-API based. This variety of mis-uses indicates that they are not rare mistakes by individual programmers and do not appear to diminish with software growth, age, or API provider.

All mis-uses can be found at "**Mis-uses**" folder in our artifact. Inputs for triggering these mis-uses are available at "**Trigger_misuse**" folder.

III. Scripts

The result of TABLE III (auto detection part) and Section VII in our paper can be reproduced by these scripts. All scripts and related materials can be found at "**Tools**" folder in our artifact.

The auto-detection tools are implemented with Jedi [3], AST [5] and PyGitHub [4] library. The API wrapper tool is implemented for Google ML APIs, but general for ML Cloud APIs from other services.

A. Output Misinterpretation Checker

We have built a static checker to automatically detect mis-uses of the `sentiment-detection` API's output. Our checker first identifies every call site of the API, and then examines the data-flow graph to see whether both the `score` field and the `magnitude` field of the API result are used in later execution. Our analysis is inter-procedural and path sensitive. The checkinging ends either when we have confirmed that both fields, have been used, or when we cannot see both of them being used after checking a threshold number of caller and callee functions. A bug is reported in the latter case.

To use this tool, one should execute

1) `ruby search_repo.rb` to get GitHub applications using `sentiment-detection` API.
2) `python check_usage_pre.py` to download related files from these applications.
3) `python check_usage.py` to do further checking.

B. Asynchronous API call checker

Many applications in our benchmark suite call asynchronous APIs in a synchronous, blocking way, and hence suffer reduced performance for no benefit. To automatically identify this problem, our checker first identifies all the places where an asynchronous API is called and then the application immediately waits on the result, following the common API usage patterns. The checker then looks for other concurrent execution. If not, this pattern is tagged as a place for performance optimization.

To use this tool, one should execute

- `python async_main_google.py [input_file] [output_file]`

where `[input_file]` is a list of GitHub applications using the corresponding Google Cloud / AWS service.

C. Constant-parameter API call checker

We have implemented a static checker to automatically identify speech synthesis API calls that use constant inputs, a type of performance mis-use. Our checker starts with every call site and tracks backward along the data dependency graph to see how the parameter of the API call is generated.

To use this tool, one should execute

- `python constant_input_main_google.py [input_file] [output_file]`
- `python constant_input_main_aws.py [input_file] [output_file]`

where `[input_file]` is a list of GitHub applications using the corresponding Google Cloud / AWS service.

D. API wrappers

We design API wrappers for all three domains of APIs.

In **vision** tasks, our wrapper down-samples large images to the suggested size of 640×480 pixels. It tackles the anti-patterns of missing input validation and unnecessarily high-resolution inputs. Its script is `vision_wrapper.py`.

In **language** tasks, the wrapper focuses on entity detection and syntax analysis, which allow input chunking with little impact to result accuracy. Our wrapper API takes in one or multiple text strings. It first concatenates all input strings together, which avoid the money wasting problem. If the combined string is not too long, a synchronous API is called; if it is too long, it will be chunked and get processed through batching API, avoiding the anti-patterns of forgetting parallel APIs and mis-use of asynchronous APIs. Its scripts are `nlp_wrapper.py` and `nlp_wrapper2.py`.

The wrapper for **speech** tasks is similar, but only takes one audio as input. The wrapper uses the synchronous API when the input size allows or streaming API otherwise. All these wrappers conduct an input validation and, in some cases, also transformation. Its script is `speech_to_text_wrapper.py`.

References

[1] Amazon. Aws lambda. Online document https://aws.amazon.com/lambda/, 2020.

[2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *ICSE-SEIP*, pages 291–300. IEEE, 2019.

[3] Dave Halter. Jedi: an awesome auto-completion, static analysis and refactoring library for python. Online document https://jedi.readthedocs.io.

[4] PyGithub. Pygithub: Typed interactions with the github api v3. https://pygithub.readthedocs.io/en/latest/introduction.html.

[5] Python. ast — abstract syntax trees. https://docs.python.org/3/library/ast.html.

[6] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. Project Webpage: Accurate Learning for EneRgy and Timeliness in Software System. https://alert.cs.uchicago.edu/#release.

[7] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. Are machine learning cloud apis used correctly? In *ICSE*, 2021.