

Automated Testing of Software that Uses Machine Learning APIs

Chengcheng Wan
University of Chicago
cwan@uchicago.edu

Henry Hoffmann
University of Chicago
hankhoffmann@uchicago.edu

Michael Maire
University of Chicago
mmaire@uchicago.edu

Shan Lu
University of Chicago
shanlu@uchicago.edu

Shicheng Liu
University of Chicago
shicheng2000@uchicago.edu

Sophie Xie
Whitney Young High School
sxie2@cps.edu

Yifan Liu
University of Chicago
liuyifan@uchicago.edu

発表者 花田

背景

- 人の行動を統計的に模倣する機械学習（ML）ソリューションを組み込んだソフトウェアアプリケーションが増加

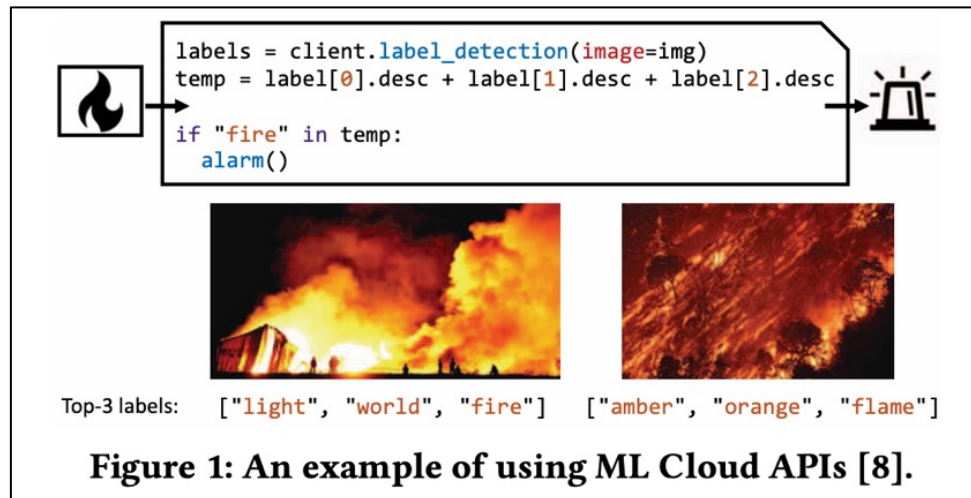
→以後,このようなアプリケーションは,MLソフトウェア

問題点

- ソフトウェアをテストするには
 - ソフトウェアに関連する画像・テキスト・音声の入力の設計
 - ソフトウェアがこれらの入力を多くの人間と同じように処理しているかどうかの判断
- この二点には多大な人間の労力がかかる
- 誤作動が発見された場合,その原因が
 - ML API内部にあるのか
 - APIを利用するコードにあるのか
- どちらなのか不明な場合が多い

問題点の例（画像認識の場合）

Phoenix



Phoenix(図1の上半分)

Google label detection APIを使って入力写真の画像分類
返ってくる分類ラベルの上位3つの中に「fire」が含まれる
→アラーム

このPhoenixに対するテストの問題点

- ①ランダムな画像入力ではうまく機能することができない.
 - そもそも大量の画像の中に、“fire”と出力される物があまり含まれない.→alarm()が実行されない.
- ②画像入力できたとしても,実行の精度評価
 - 手作業が必要な場合がある.
- ③テスト実行に失敗した場合
 - ・ label_detection の性質にあるのか
 - ・ アプリケーションによる API の不適切な使用にあるのか
 - その原因を知ることは難しい.

目的

- ML ソフトウェアのための新しいテストツールであるKeeperを紹介

- KeeperはML APIに制御フローが影響されるソフトウェアのためのテストツール

Keeperは、各ML APIに対して、

①関連する画像/テキスト/音声入力を自動的に生成（テスト入力生成）

②①を用いて、各ML APIに対して出力の正誤判定を行う（テスト出力評価）

③開発者が精度の失敗の根本原因を理解するため（どこにバグがあるのか）

KeeperはML APIの別の使用方法を調査

→精度の失敗を軽減するためのコード変更を開発者に通知

Keeper (テスト入力生成) ①

- Keeper の入力生成は既存の記号実行エンジンである DSE [47]を基に構築
 - 全ての分岐に対して記号的なパス制約が生成 (例 1)
 - 全てのパス制約を解くことでフルブランチカバレッジを提供するテストスイートが生成 (例 2)

例 1

```
1 def smart_can(img):  
2     labels = client.label_detection(image=img)  
3     classes = [x.desc for x in labels]  
4     for c in classes:  
5         if c == "food":  
6             return "organic"  
7         if c == "paper" or c == "aluminum":  
8             return "recyclable"  
9     return "non-recyclable"
```

Figure 4: A smart can application, Heap-Sort-Cypher [57]

例 2



Figure 5: Keeper-generated test cases for Figure 4

例えば, "A or B or C" という分岐条件に対して, (1) Aが真,
(2) Bが真,
(3) Cが真,
(4) A, B, Cのいずれも真でない,
という4つの制約が形成される.
これらの制約を解くと, これらの制約を個別に満
たす4つの入力または入力集合を得る.

Keeper（テスト出力評価）②

- ①で生成したテストスイート（画像・テキスト・音声）を擬似反転させ,擬似逆関数を生成

➤ 擬似反転

検索エンジン←キーワード⇒キーワードを反映した画像や文章を返す（擬似逆関数）

- 擬似逆関数を人間の判断の代用とし,関連するソフトウェア出力の正誤を自動判定

	ML Task	Main Output	Constraint Example	Pseudo-inverse Function
Vision	Image classification	image class	class=="fire" [8]	Search on internet, keyword: [image class]
	Object detection	object name	object=="tableware" [49]	Search on internet, keyword: [object name]
	Face detection	face emotion	emotion=="joy" [50]	Search on internet, keyword: [emotion] + "human face"
	Text detection	extracted text	text=="3923-6625" [51]	Print [extracted text] on an image
Language	Document classification	document class	class=="food" [52]	Search on internet, keyword: [document class]
	Sentiment detection	score, magnitude	score< 0 [53]	Select tweets from Sentiment140 dataset [54]
	Entity detection	entity name, type	type=="Person" [55]	Use text generation technique, seed: [name] or [type]
Speech	Speech recognition	transcript	text=="turn on the light" [56]	Use speech synthesize technique on [transcript]

Table 1: Different ML APIs handled by Keeper and their pseudo-inverse functions.

Keeper（失敗軽減） ③.1

- Keeper は 3 種類の失敗の症状を特定

（1）精度が低い

→開発者が分岐条件におけるいくつかの関連するラベルを見逃している可能性

➤ alarm（）が反応するのが“fire”のみ→ラベルが多いと望んでいる画像が出力されやすくなる

（2）デッドコード

→プログラム実行時に実行される可能性がない文 or 分岐の意味を成していない文

➤ 画像判定として“animal”をキーワードとすると,画像に何か写っているとTrueと返されてしまう

（3）クラッシュのような一般的な失敗

- ML API入力を必要とするパス上にあるバグが原因

➤ パスを実行するために必要なML API入力を生成

Keeper（失敗軽減） ③.2

Keeperの提案

- ラベルの変更

→3つの目標を元にラベルの拡張を提案

- (1) Recall目標： 条件を満たすテスト入力が増える
- (2) Precision目標： 条件を満たさない入力がほとんどである
- (3) Semantic 目標： 元のラベルと自然言語の意味的関連性がある

```
1 labels = client.label_detection(image=img)
2 temp = label[0].desc + label[1].desc + label[2].desc
3 if "fire" in temp or "flame" in temp or "ash" in temp:
4     alarm()
```

Figure 7: A fixed version of Figure 1 suggested by Keeper.

“flame”と“ash”のラベルを追加すれば、元のコードから40%改善されることがテキストで提案

Keeper評価(1)

(1) Keeperはテスト中のブランチカバレッジを向上させるのに役立ちますか？

➤ Keeperをベースラインとし,三つのテスト手法と比較

63 個のPythonアプリケーションに対し

→各テスト手法によって生成された 100 個の入力を実施

→累積ブランチカバレッジを計算

	Vision App.	Language App.	Speech App.
Keeper	91.9%	91.5%	89.7%
Random-Real	74.5%	85.0%	54.3%
Random-Real-Noise	73.0%	65.2%	54.3%
Fuzzing	44.4%	74.0%	24.9%

Table 3: Average branch coverage across 63 applications.

Keeper評価(2)

- (2) Keeperはテスト中にバグを発見することができますか？

- Keeper は生成した 100 個のテスト入力を実行

→25のアプリケーションの最新バージョンから35のバグを発見

これらのバグは様々な症状や根本的な原因を含む.

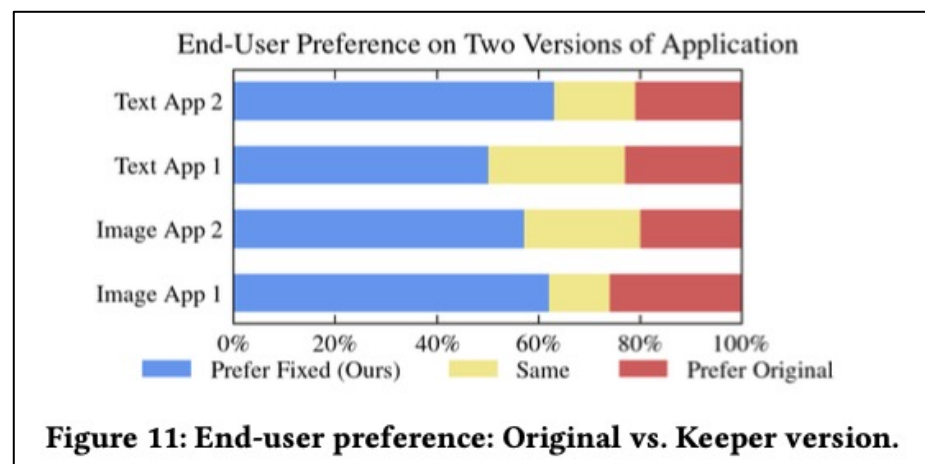
型変換の欠落による1つの失敗を除いて,すべて異なるタイプの認知的MLタスクに関連するもの

Failure type	Root Cause	Related ML Task	Keeper	RReal	RReal+Noise	Fuzz.
Crash failures	Out-of-bound accesses	Text detection, entity detection	6	5	5	4
	Missing input validation*	Document classification	1	-	-	-
	Missing type conversion	-	1	1	1	1
Accuracy failures	Improper labels	Image classi., object detect., document classi.	9	-	-	-
	API limitations	Image classification, object detection	6	-	-	-
	Improper threshold	Sentiment detection	9	-	-	-
Dead-code failures	Typos	Image classification, text detection	2	-	-	-
	Non-existing label	Image classification	1	-	-	-

Table 2: Unique failures exposed by Keeper. (*: This crash disappeared later with the most recent version of Google API.)

Keeper評価(3)

- (3) Keeperは精度不良の修正を提案することができますか？
 - Amazon Mechanical Turk (Mturk)で100人の参加者を募集し,ソフトウェアユーザー調査を実施
 - オリジナルのアプリケーションとKeeperからコードの変更を提案されたアプリケーションの出力を比較
- 精度不良に関するKeeperの判断と精度不良を修正するKeeperの試みが支持されている



結果

- Keeperはブランチカバレッジを大幅に向上させ,同時に多くの未知のバグを発見することができた
- 機械学習コンポーネントを含むソフトウェアを効率的かつ効果的にテストすることは困難
- 開発者がソフトウェアの実装のバグを検出し,修正案を提供するのを支援する自動化されたカバレッジガイド付きテストフレームワークであるKeeperを紹介
- Keeperはソフトウェアのクラッシュ,推論精度の低下,またはデッドコードにつながるバグを特定できる

所感

- 専門用語が多く,論文を辿る量がすごかった
- 相当論文を読んだが,完全に理解できなかった
- 所々,関数を用いた説明の部分がよくわからなかった