Санкт-Петербургский государственный университет

Кафедра Системного Программирования

Группа 21.Б09-мм

Разработка инфраструктуры для обучения с учителем искусственной нейронной сети, выбирающей пути при символьном исполнении

Чистякова Анна Артуровна

Отчёт по учебной практике
в форме «Эксперимент»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., С. В. Григорьев

Санкт-Петербург
2024

Оглавление

Введение

В современном мире с ростом актуальности автоматизации различных процессов обработки данных и вычислений растет и необходимость в тестировании таких систем с целью обнаружить ситуации с нежелательным, непредсказуемым или даже опасным исходом в зависимости от входных данных. Для решения этой задачи применяются разные подходы. Например, можно тестировать программу на случайных входных данных, используя фаззинг или ручное тестирование, но у таких подходов есть недостаток в том, что вариантов входных данных зачастую очень много и перебрать все невозможно. Из-за этого можно упустить входы, на которых программа ведет себя не так, как ожидалось.

Эту проблему решает символьное исполнение, в котором рассматриваются не конкретные значения параметров, а состояния, которые могут порождаться входным значением в зависимости от некоторых условий в коде. С его применением, обойдя все возможные пути графа потока управления, можно учесть все состояния, в которых может оказаться программа.

Из-за того, что состояний может быть очень много, а путей достижения максимального тестового покрытия может быть несколько, подбор оптимальной последовательности шагов — нетривиальная задача. Так же в процессе такого анализа кода может возникнуть проблема хранения всех возможных состояний различных переменных. Существует множество подходов, оптимизирующих выбор очередного шага при символьном исполнении [7]. Среди них использование различных алгоритмов на графах, например, DFS, который позволяет хранить в памяти немного информации, но на больших графах может работать медленно, BFS, позволяющий исследовать пути параллельно, но требующий больших затрат по памяти. Также в некоторых работах [2, 3, 5, 1] были использованы эвристики для максимизации тестового покрытия. В [3] используется информация о расстоянии до ближайшей неисследованной вершины в вычислении вероятности, с которой шаг будет выбран.

Кроме того, для оптимизации символьного исполнения применяют-

ся методы машинного обучения [4, 6]. В них для предсказания используются различные свойства состояния, но не учитывается структура их связей между собой.

Графовые нейронные сети позволяют учитывать не только свойства состояний и вершин графа потока управления, но и связи между ними. Исследованию их применимости в этой области и посвящена данная работа.

Листинг 1: Пример кода для символьного исполнения

```
1  def function(x: int):
2      if x > 0:
3          y = x + 1
4          return y
5      else:
6          s = other_function(x)
7          s += 1
8          return s
9
10 def other_function(x: int):
11     if x == 5:
12         return 0
13     else:
14         return x
```

1. Символьное исполнение

В этом разделе символьное исполнение будет описано исходя из его реализации в VSharp[1], на основе которого выполнена эта работа. Для лучшего понимания рассмотрим простой пример кода (листинг 1).

Весь код, который нужно протестировать, разбивается на базовые блоки, и по ним строится граф потока управления. Так же к графу добавляются вершины, описывающие возможные состояния. Формально полученный граф можно описать так:

$G = (V, E), V = V_1 \cup V_2, E = E_{V_1} \cup E_{in} \cup E_{parent\_of} \cup E_{history} \cup E_{call} \cup E_{return}$, где

- V_1, E_{V_1} — множества вершин и ребер графа потока управления;
- V_2 — вершины, описывающие состояния;
- $E_{in} = \{(u, v) \mid (u \in V_2, v \in V_1),$ состояние u находится в базовом блоке $v\}$;
- $E_{parent\_of} = \{(u, v) \mid (u \in V_2, v \in V_2),$ состояние u породило состояние $v\}$;

[1] https://github.com/VSharp-team/VSharp (дата доступа: 7 января 2024 г.).

- $E_{history} = \{(u, v, w) \mid (u \in V_2, v \in V_1),$ состояние u находилось в вершине v, w — количество посещений вершины v состоянием $u\}$;
- $E_{call} = \{(u, v) \mid (u, v \in V_1),$ блок u закончился инструкцией вызова метода, точкой входа которого является блок $v\}$;
- $E_{return} = \{(u, v) \mid (u, v \in V_1),$ блок u закончился инструкцией возврата, и управление должно передаться в блок $v\}$.

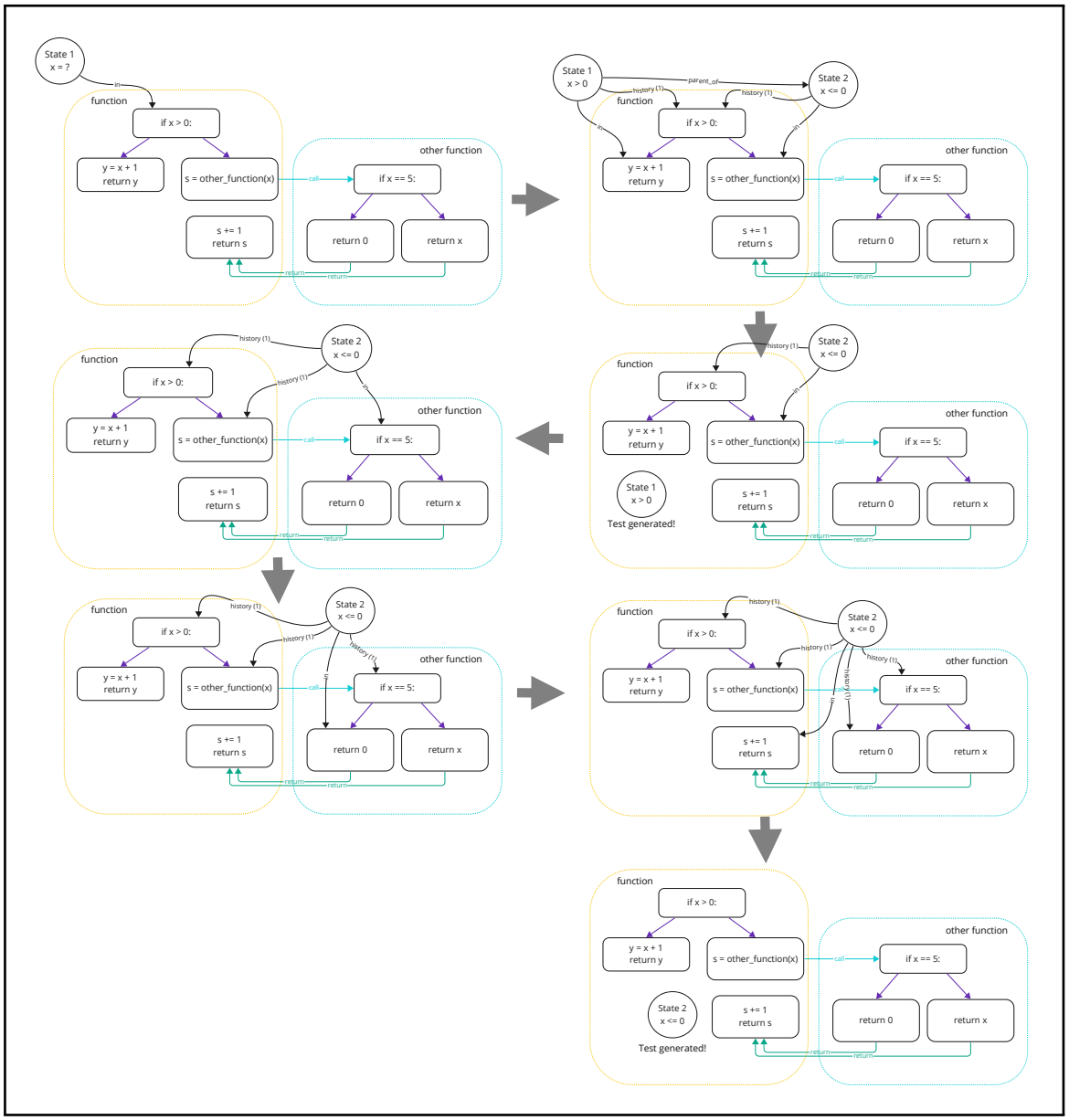

Рис. 1: Пример символьного исполнения кода в листинге 1

На рис. 1 представлен процесс символьного исполнения рассматриваемого кода. Обе функции были разделены на базовые блоки. Так как

из функции $function$ происходит вызов метода $other\_function$, между блоками этих функций есть ребра $call$ и $return$.

Весь процесс символьного исполнения можно представить в виде игры, где граф потока управления — игровое поле или карта, а состояние — фишки, которыми можно ходить. Цель игры — сгенерировать тесты с максимальным процентом покрытия за минимальное число шагов.

На изображении 1 видно, что сначала существует только одно состояние. Им мы и делаем первый шаг, который порождает еще одно состояние, так как было пройдено условие на переменную x. В этой ситуации выгодно сделать шаг первым состоянием, так как оно находится в блоке, где программа завершается, а значит, после такого шага будет сгенерирован тест и получен некоторый процент покрытия. И так далее, пока не будут рассмотрены все возможные состояния.

При дальнейшем описании проделанной работы будет использоваться приведенная игровая аналогия и граф потока управления в представленном выше виде.

2. Существующие решения

Так как задача выбора путей в символьном исполнении является актуальной уже длительное время, существует множество решений, оптимизирующих полный перебор возможных путей и максимизирующих при этом тестовое покрытие.

К оптимизации символьного исполнения были применены разные подходы. Среди них есть несколько основанных на машинном обучении. В Learch [4] для определения наиболее удачного состояния на очередном шаге используется линейная регрессия. В [6] применяется техника обучения с подкреплением.

3. Используемые технологии

Существует множество библиотек для работы с нейронными сетями. Так как в задаче подразумевается использование графовых моделей, было принято решение использовать фреймворк PyTorch[2] совместно с библиотекой PyG[3], содержащей в себе большое количество реализаций различных обучаемых операторов. Так же при выборе инструмента учитывалась подробность документации.

В обучении нейронных сетей большую роль играет подбор гиперпараметров. Для их подбора есть различные алгоритмы, а так же библиотеки с их реализациями. В ходе выбора основное внимание уделялось совместимости с доступным для обучения оборудованием и разнообразию реализованных алгоритмов. Этим требованиям удовлетворяла библиотека optuna[4].

[2] https://pytorch.org/ (дата доступа: 7 января 2024 г.).

[3] https://pytorch-geometric.readthedocs.io/en/latest/index.html (дата доступа: 7 января 2024 г.).

[4] https://optuna.readthedocs.io/en/stable/index.html (дата доступа: 7 января 2024 г.).

4. Описание проекта

Проект, на основе которого реализовывалось описываемое решение, состоит из двух частей: сервер, который непосредственно производит шаги символьного исполнения кода, и клиент, который по ситуации, полученной от сервера, вычисляет следующий шаг и передает эту информацию обратно (рис. 2).

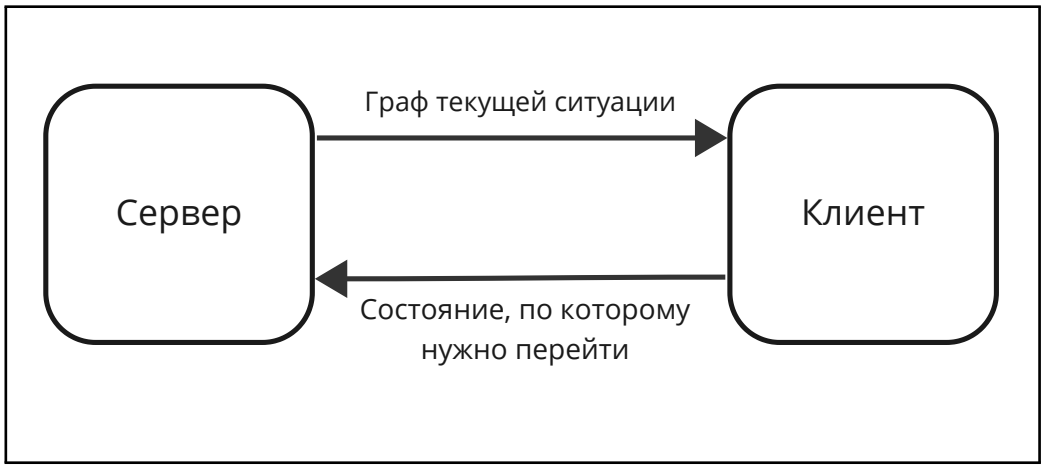

Рис. 2: Схема взаимодействия клиента с сервером

Клиентская часть состоит из трех частей, которые представляют собой три разных этапа обучения нейронной сети (рис. 3). Первый этап представляет собой предобучение нейронной сети предсказанию некоторых свойств состояния, которые полезны для определения следующего шага. Второй этап — обучение с помощью генетических алгоритмов, используя несколько предобученных моделей в начальном поколении. Третий этап можно назвать постобучением модели, полученной на первом этапе с помощью результатов, полученных на втором.

Последняя стадия обучения состоит из трех частей: генерация датасета, обучение на полученных данных с помощью градиентного спуска и валидация. Этому этапу посвящена данная работа, поэтому он будет рассмотрен более подробно.

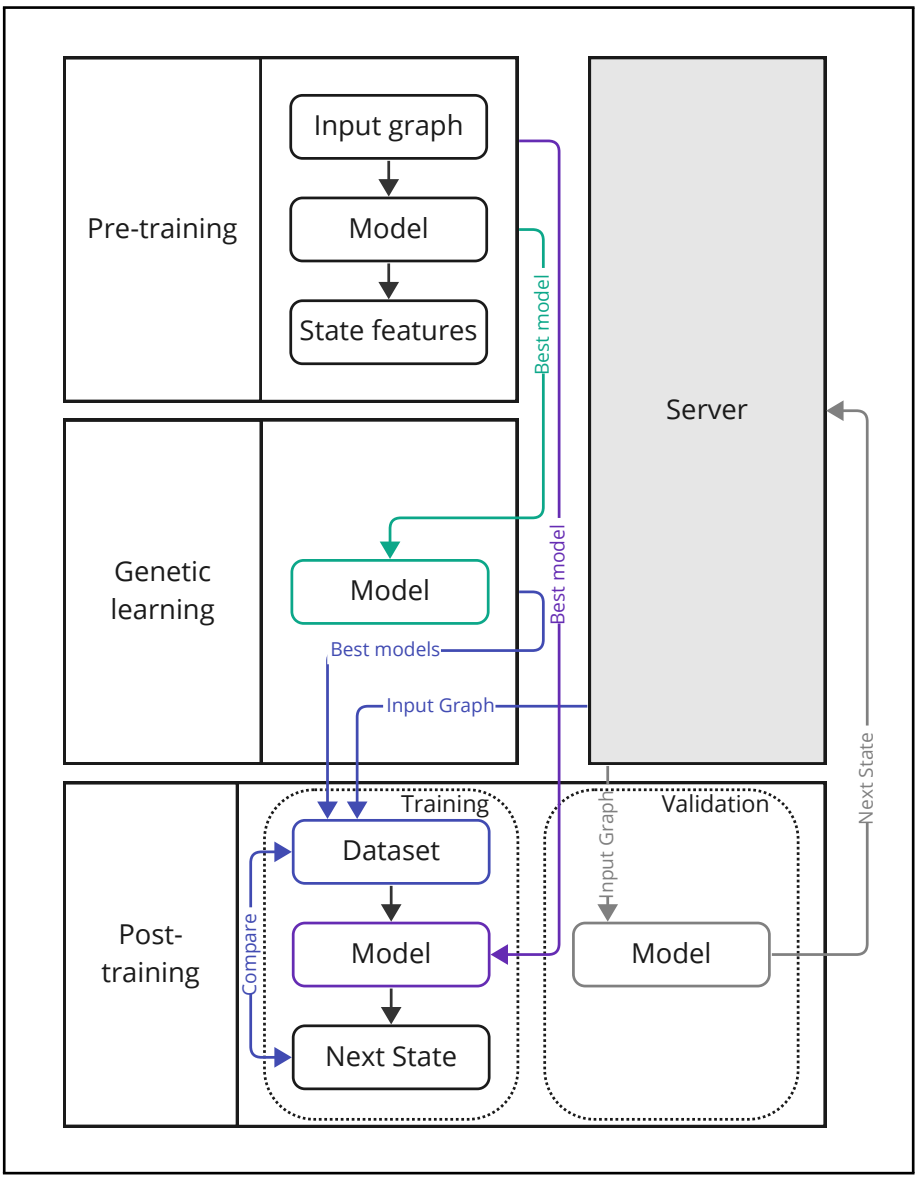

Рис. 3: Схема этапов обучения

5. Постановка задачи

Целью данной работы является исследование применимости графовых нейронных сетей к нахождению путей при символьном исполнении. Для её достижения были поставлены следующие задачи:

1. Разработка инфраструктуры для обучения с учителем графовой нейронной сети, включающую в себя подготовку датасета, процесс обучения и валидации. Обеспечение ее взаимодействия с другими компонентами проекта.

2. Проведение экспериментов с различными гиперпараметрами с це-

лью достижения максимально возможного процента тестового покрытия, оценка результатов и их сравнение с аналогами.

6. Реализация

Этот раздел посвящен подробному описанию всех частей третьего этапа обучения модели.

6.1. Подготовка датасета и обучение

Набор данных состоит из 128 реализации алгоритмов и методов различного рода сложности и размера в плане количества возможных состояний. Среди них есть как простые рекуррентные алгоритмы на графах, сортировки, так и методы, эмулирующие работу процессоров и других устройств, с огромным количеством состояний и вершин графа. Также в выборке содержатся синтетические методы с большой концентрацией условий и циклов.

На этих данных учатся модели на втором этапе, и каждую эпоху для каждой карты и набора обучаемых на ней моделей сохраняется результат в виде кортежа из четырех чисел: процент покрытия, полученный данной моделью, количество сгенерированных тестов, число найденных ошибок и количество шагов, сделанных в процессе символьного исполнения данной карты. Такие метрики были выбраны, исходя из того, что целью является максимизация процента покрытия и найденных исключений, с помощью минимального количества шагов и сгенерированных тестов. По этим данным для каждой карты выбирается модель с лучшим результатом, и в датасет в качестве эталона сохраняется полученная ей последовательность шагов.

Затем из набора данных удаляются выбросы, содержащие в эталонном значении NaN, и графы, имеющие одну вершину состояния, и, следовательно, не несущие в себе полезной информации для обучения. Так же набор данных для каждой карты проверяется на одинаковые шаги и если таковые нашлись, то они удаляются из тренировочной выборки с некоторой заранее заданной вероятностью. Шаги считаются одинаковыми, если в ходе символьного исполнения они были сделаны друг за другом, при этом у входных графов было одинаковое число вершин и в результате было выбрано одно и то же состояние.

После этого все шаги перемешиваются и разделяются на батчи для обучения модели.

6.2. Валидация

После изменения весов модели оценивается ее качество. Для этого с ее помощью исполняются все карты датасета, а затем оценивается результат по среднему проценту покрытия по всему набору данных.

Процесс валидации занимает большую часть времени описываемого этапа обучения, поэтому была добавлена возможность запускать его параллельно на разных наборах карт (рис. 4). То есть перед запуском обучения можно задать количество процессов, между которыми делится весь датасет, и таким образом разные карты исполняются параллельно.

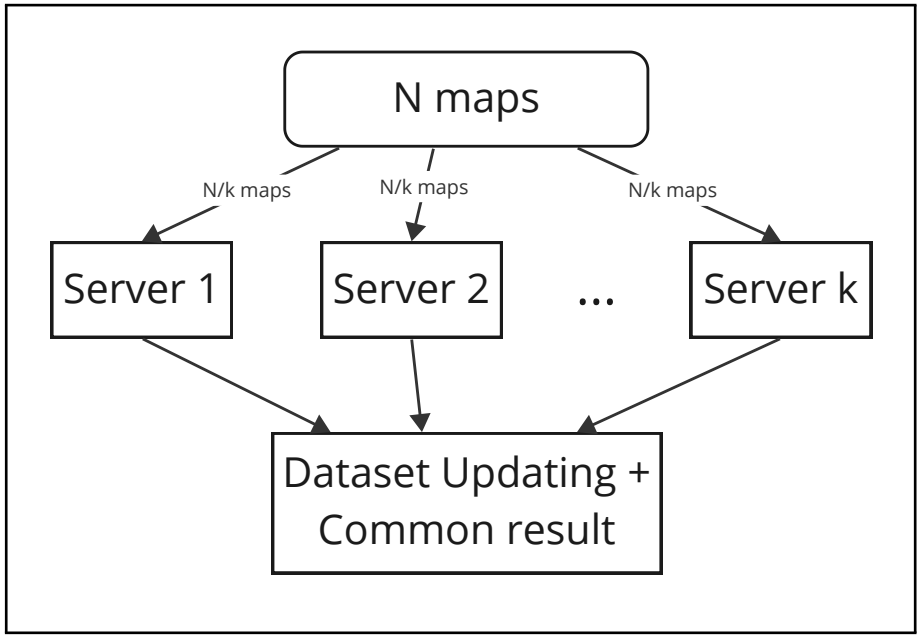

Рис. 4: Параллельность валидации

Так же стоит отметить, что в датасете не хранится оптимальный набор шагов, так как он просто не известен, и бывают ситуации, когда обучаемая нейронная сеть показывает лучший результат, чем был получен с помощью уже сохраненных шагов. Тогда новая последовательность шагов сохраняется вместо старого в наборе данных.

7. Эксперименты

В ходе работы были проведены эксперименты с различными архитектурами предобученных моделей, с нейронной сетью, инициализированной случайными параметрами, а также с отдельным обучением последнего слоя предобученной нейронной сети. Использовался оптимизатор Adam. Разница между полученным и эталонным распределениями оценивалась расстоянием Кульбака-Лейблера. Размер батча и величина шага подбиралась автоматически с помощью алгоритма Tree-structured Parzen Estimator, который работал на основе максимизации среднего процента покрытия, полученного в результате символьного исполнения.

Полноценный анализ результатов и остаток экспериментов будут проведены в следующем семестре.

Заключение

В данной работе было проведено исследование применимости графовых нейронных сетей к нахождению путей при символьном исполнении. Были получены следующие результаты.

- Разработана инфраструктура для обучения с учителем графовой нейронной сети.
- Проведены эксперименты с некоторыми архитектурами моделей.

Исходный код доступен по ссылке.

Развитие исследования

Поскольку работа была больше нацелена на достижение стабильности процесса обучения моделей, не все возможные эксперименты были поставлены. Таким образом, в дальнейшем планируется запуск серии экспериментов и сравнение описанного подхода с другими методами символьного исполнения.

Список литературы

[1] Chipounov Vitaly, Kuznetsov Volodymyr, Candea George. The S2E platform: Design, implementation, and applications // ACM Transactions on Computer Systems (TOCS). — 2012. — Vol. 30, no. 1. — P. 1–49.

[2] EXE: Automatically generating inputs of death / Cristian Cadar, Vijay Ganesh, Peter M Pawlowski et al. // ACM Transactions on Information and System Security (TISSEC). — 2008. — Vol. 12, no. 2. — P. 1–38.

[3] Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. / Cristian Cadar, Daniel Dunbar, Dawson R Engler et al. // OSDI. — Vol. 8. — 2008. — P. 209–224.

[4] Learning to explore paths for symbolic execution / Jingxuan He, Gishor Sivanrupan, Petar Tsankov, Martin Vechev // Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. — 2021. — P. 2526–2540.

[5] Unleashing mayhem on binary code / Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, David Brumley // 2012 IEEE Symposium on Security and Privacy / IEEE. — 2012. — P. 380–394.

[6] Wu Jie, Zhang Chengyu, Pu Geguang. Reinforcement learning guided symbolic execution // 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER) / IEEE. — 2020. — P. 662–663.

[7] A survey of symbolic execution techniques / Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia et al. // ACM Computing Surveys (CSUR). — 2018. — Vol. 51, no. 3. — P. 1–39.