# Застосування методів машинного навчання в статичному аналізі програмних кодів

Виконав: Вохранов І.А., ДА-62

Керівник роботи: Булах Б.В.

# Мета

Метою дипломної роботи є дослідження можливостей використання машинного навчання в процесі статичного аналізу програмних кодів (виявленні помилок та ін.).

# Статичний аналіз коду

- Автоматизований процес перегляду коду (code review)
- Не потребує виконання коду
- Може виконуватись на дуже ранніх етапах розробки
- Може мати різні форми та виконувати різні задачі

# Машинне навчання в аналізі кодів

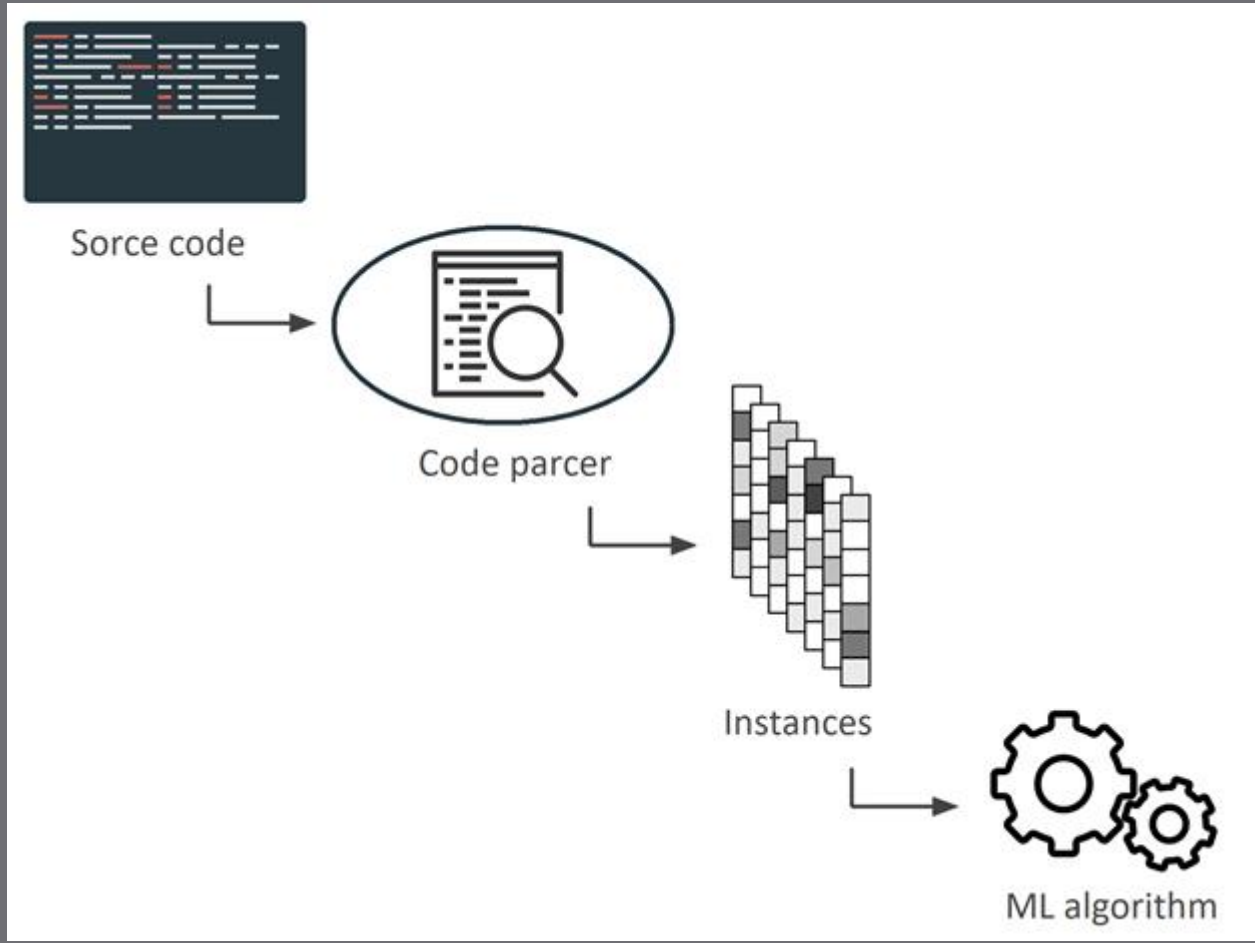
Використання машинного навчання в аналізі програмних кодів – це нова сфера досліджень.

Їх головна мета – розробка підходів для автоматичного виконання таких інженерних завдань, як:

- виявлення вразливостей у вихідному коді
- допоміжні огляди коду
- дедуплікація коду
- оптимізація
- і т.д.

# Основна проблема

Оскільки алгоритми ML не здатні працювати з вихідним кодом, постає питання, як програмний код може бути переведеним в набір екземплярів, на основі яких буде сформована вибірка для навчання.

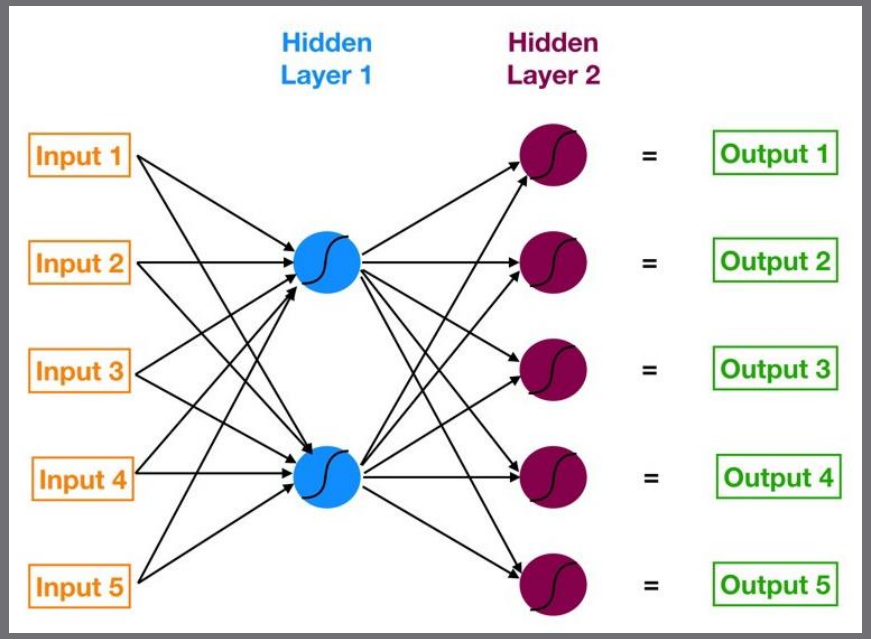


# Підходи до навчання на коді

- З ручним навчанням моделей, для якого навчальні вибірки генеруються штучно
- З самостійним навчанням моделей на великій кількості відкритого коду з репозиторіїв (GitHub, Bitbucket, GitLab, ...)

# Нейронні мережі

- Для навчання на великих об’ємах коду із репозиторіїв, потрібен гнучкий підхід, що дозволить моделі самостійно виявляти шаблони з вихідного коду проєктів.
- Одним із найкращих рішень є використання нейронних мереж.



# Навчання на послідовності символів

Найпростіший спосіб формування вибірки — представлення коду в вигляді послідовності символів.

Наприклад, вихідний код може бути переведеним в послідовність байтів

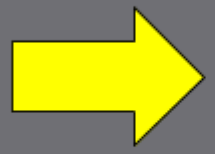
```
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

```
35 105 110 99 108 117 100 101 32 60 105 111 115 116 114 101
97 109 62 10 10 105 110 116 32 109 97 105 110 40 41 32 123
10 32 32 32 32 115 116 100 58 58 99 111 117 116 32 60 60 32
34 72 101 108 108 111 32 87 111 114 108 100 33 34 59 10 32
32 32 32 114 101 116 117 114 110 32 48 59 10 125
```

# Унітарний код (one hot encode)

Для отримання якісних результатів під час навчання нейронних мереж, звичайні послідовності краще перетворювати в матриці унітарного коду (код, в якому дозволеними комбінаціями значень є лише ті, в яких встановлено лише один біт — “1”, а всі інші вимкнено — “0”).

| Color |
|---|
| Red |
| Red |
| Yellow |
| Green |
| Yellow |



| Red | Yellow | Green |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

# Навчання нейронної мережі на послідовності символів

```
[ ]  def create_model_1():
         new_model = keras.Sequential([
             keras.layers.Dense(50, batch_input_shape=(None, 2*context, len(vocabulary)), activation='relu'),
             keras.layers.Dense(25, activation='relu'),
             keras.layers.Reshape((2*context*25,)),
             keras.layers.Dense(len(vocabulary), activation=tf.nn.softmax),
         ])

         new_model.compile(optimizer='adam',
                           loss='categorical_crossentropy',
                           metrics=['accuracy'])
         return new_model
```

```
[ ]  # model1 = create_model_1()

     model1 = keras.models.load_model(training_dir + '/my_model_1.h5')
```

# Навчання нейронної мережі на послідовності символів

```
[ ]  model1.fit(train_in, train_out, epochs=10)
     model1.save(training_dir + '/my_model_1.h5')
```

```
Epoch 1/10
5841/5841 [==============================] - 17s 3ms/step - loss: 1.1731 - accuracy: 0.6878
Epoch 2/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1710 - accuracy: 0.6877
Epoch 3/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1681 - accuracy: 0.6881
Epoch 4/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1670 - accuracy: 0.6888
Epoch 5/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1644 - accuracy: 0.6899
Epoch 6/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1629 - accuracy: 0.6899
Epoch 7/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1615 - accuracy: 0.6906
Epoch 8/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1598 - accuracy: 0.6911
Epoch 9/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1583 - accuracy: 0.6908
Epoch 10/10
5841/5841 [==============================] - 16s 3ms/step - loss: 1.1571 - accuracy: 0.6913
```

# Прогнозування

Прогнозування пропущеного символу “<” з послідовності:

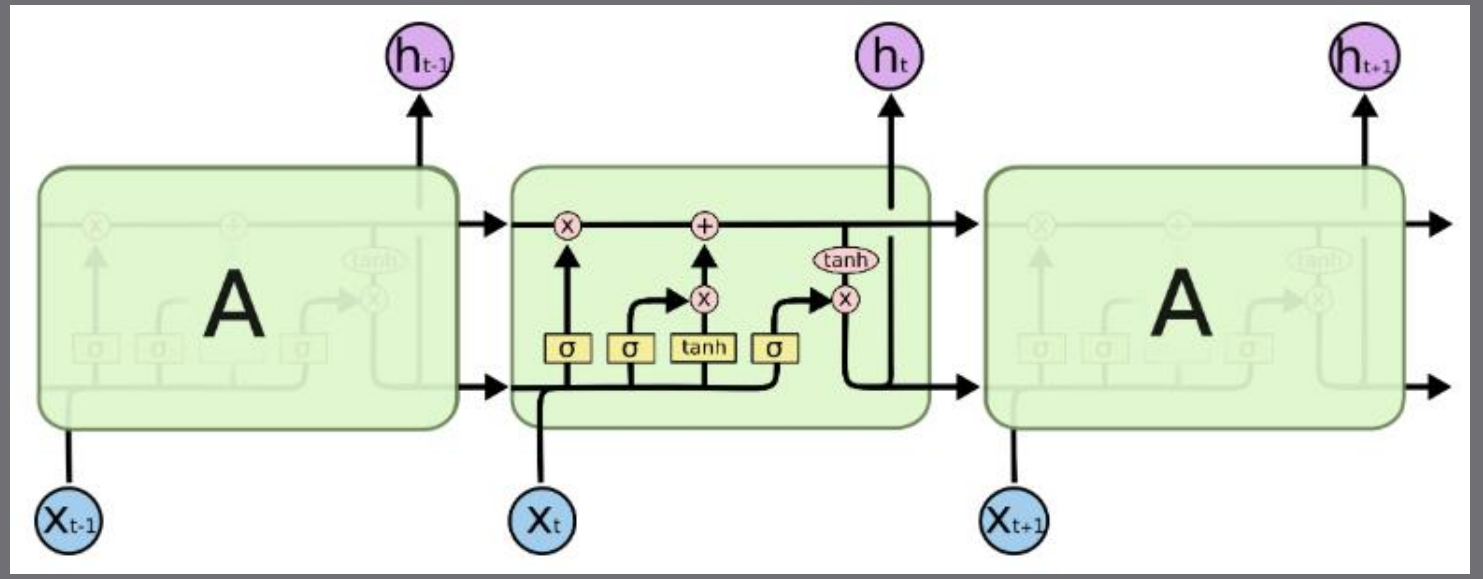
*“i := 0; i < 10; i++ {“*

```
[ ]  p = model1.predict(np.array([one_hot_encode('i := 0; i ' + ' 10; i++ {')]))
     for i in range(5):
         i = np.argmax(p)
         print(code_to_char[i], p[0][i])
         p[0][i] = 0
```

```
< 0.99703383
+ 0.0014873544
> 0.0005174677
& 0.00040776018
: 0.00025402667
```

# LSTM нейронна мережа

- особлива варіація рекурентних нейронних мереж
- здатна працювати з довгостроковими залежностями
- повторюваний модуль має складну структуру, що дозволяє контролювати стани



# Навчання LSTM мережі на послідовності символів

```
[ ]  def create_model_2():
         model = keras.Sequential()
         model.add(keras.layers.LSTM(512, return_sequences=True, batch_input_shape=(None, 2*context, len(vocabulary))))
         model.add(keras.layers.Dropout(0.2))
         model.add(keras.layers.LSTM(512, return_sequences=False))
         model.add(keras.layers.Dropout(0.2))
         model.add(keras.layers.Dense(len(vocabulary)))
         model.add(keras.layers.Activation('softmax'))

         model.compile(optimizer='adam',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
         return model
```

```
[ ]  # model2 = create_model_2()

     model2 = keras.models.load_model(training_dir + '/my_model_2.h5')
```

# Навчання LSTM мережі на послідовності символів

```
[ ]  model2.fit(train_in, train_out, epochs=5)
     model2.save(training_dir + '/my_model_2.h5')
```

```
Epoch 1/5
5841/5841 [==============================] - 61s 11ms/step - loss: 1.5468 - accuracy: 0.6300
Epoch 2/5
5841/5841 [==============================] - 61s 11ms/step - loss: 0.7626 - accuracy: 0.8044
Epoch 3/5
5841/5841 [==============================] - 62s 11ms/step - loss: 0.5507 - accuracy: 0.8529
Epoch 4/5
5841/5841 [==============================] - 62s 11ms/step - loss: 0.4304 - accuracy: 0.8817
Epoch 5/5
5841/5841 [==============================] - 62s 11ms/step - loss: 0.3488 - accuracy: 0.9018
```

# Прогнозування

Прогнозування пропущеного символу “<” з послідовності:

*“i := 0; i < 10; i++ {“*

```
[ ]  p = model2.predict(np.array([one_hot_encode('i := 0; i ' + ' 10; i++ {')]))
     for i in range(5):
         i = np.argmax(p)
         print(code_to_char[i], p[0][i])
         p[0][i] = 0
```

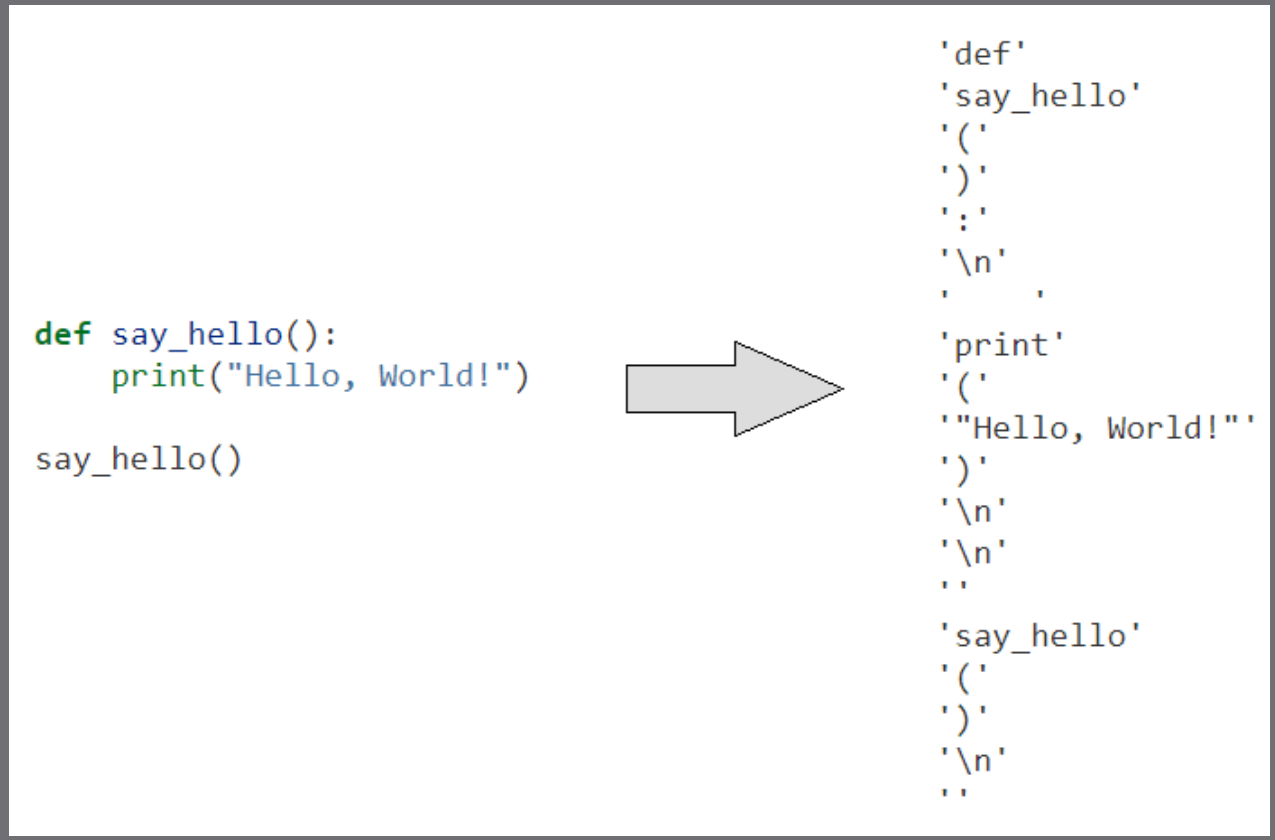
```
< 0.8427929
+ 0.024377882
> 0.018331857
1 0.01809166
= 0.015510654
```

# Проблеми прогнозування на символах

- Модель навчається визначати інформацію, яка вже відома зі структури мови програмування.
- Модель працює лише з фіксованою кількістю символів

# Навчання на послідовності токенів

Набагато кращим підходом буде виділення таких послідовностей в окремі лексеми (токени)



# Навчання LSTM моделі на послідовності токенів

Для навчання на токенах, використовувалась та ж модель LSTM мережі

```
Epoch 10/20
547/547 [==============================] - 15s 27ms/step - loss: 0.5328 - accuracy: 0.8566
Epoch 11/20
547/547 [==============================] - 15s 28ms/step - loss: 0.4903 - accuracy: 0.8663
Epoch 12/20
547/547 [==============================] - 15s 27ms/step - loss: 0.4478 - accuracy: 0.8793
Epoch 13/20
547/547 [==============================] - 15s 28ms/step - loss: 0.4233 - accuracy: 0.8832
Epoch 14/20
547/547 [==============================] - 15s 28ms/step - loss: 0.4040 - accuracy: 0.8908
Epoch 15/20
547/547 [==============================] - 15s 27ms/step - loss: 0.3795 - accuracy: 0.8952
Epoch 16/20
547/547 [==============================] - 15s 28ms/step - loss: 0.3670 - accuracy: 0.8966
Epoch 17/20
547/547 [==============================] - 15s 27ms/step - loss: 0.3579 - accuracy: 0.8982
Epoch 18/20
547/547 [==============================] - 15s 28ms/step - loss: 0.3445 - accuracy: 0.9000
Epoch 19/20
547/547 [==============================] - 15s 28ms/step - loss: 0.3348 - accuracy: 0.9038
Epoch 20/20
547/547 [==============================] - 15s 27ms/step - loss: 0.3246 - accuracy: 0.9026
```

# Прогнозування

Прогнозування пропущеного токену “in” з послідовності:

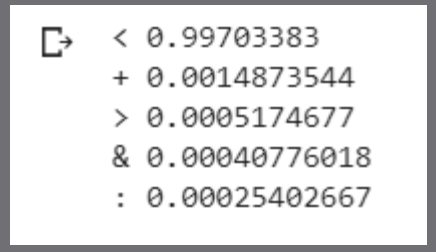
*“for k in keys:“*

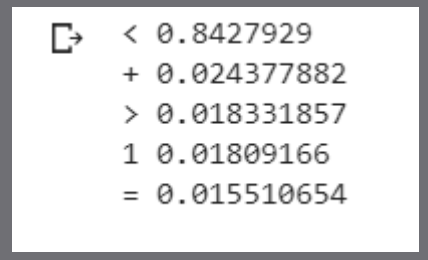
```
[ ]  tmp = ['for', 'k', 'keys', ':']     # for k in keys:
     p = model4.predict(np.array([one_hot_encode(tmp)]))
     for i in range(5):
         i = np.argmax(p)
         print(code_to_token[i], p[0][i])
         p[0][i] = 0
```

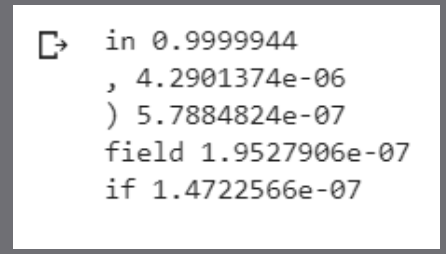
```
in 0.9999944
, 4.2901374e-06
) 5.7884824e-07
field 1.9527906e-07
if 1.4722566e-07
```

# Застосування отриманих результатів

- Один із найбільш перспективних способів застосування нейронних мереж в аналізі програмних кодів — це прогнозування елементів.
- Головна ідея полягає в тому, щоб замість того, щоб знаходити найбільш ймовірні елементи, використовувати ймовірності всіх інших, для виявлення аномалій.



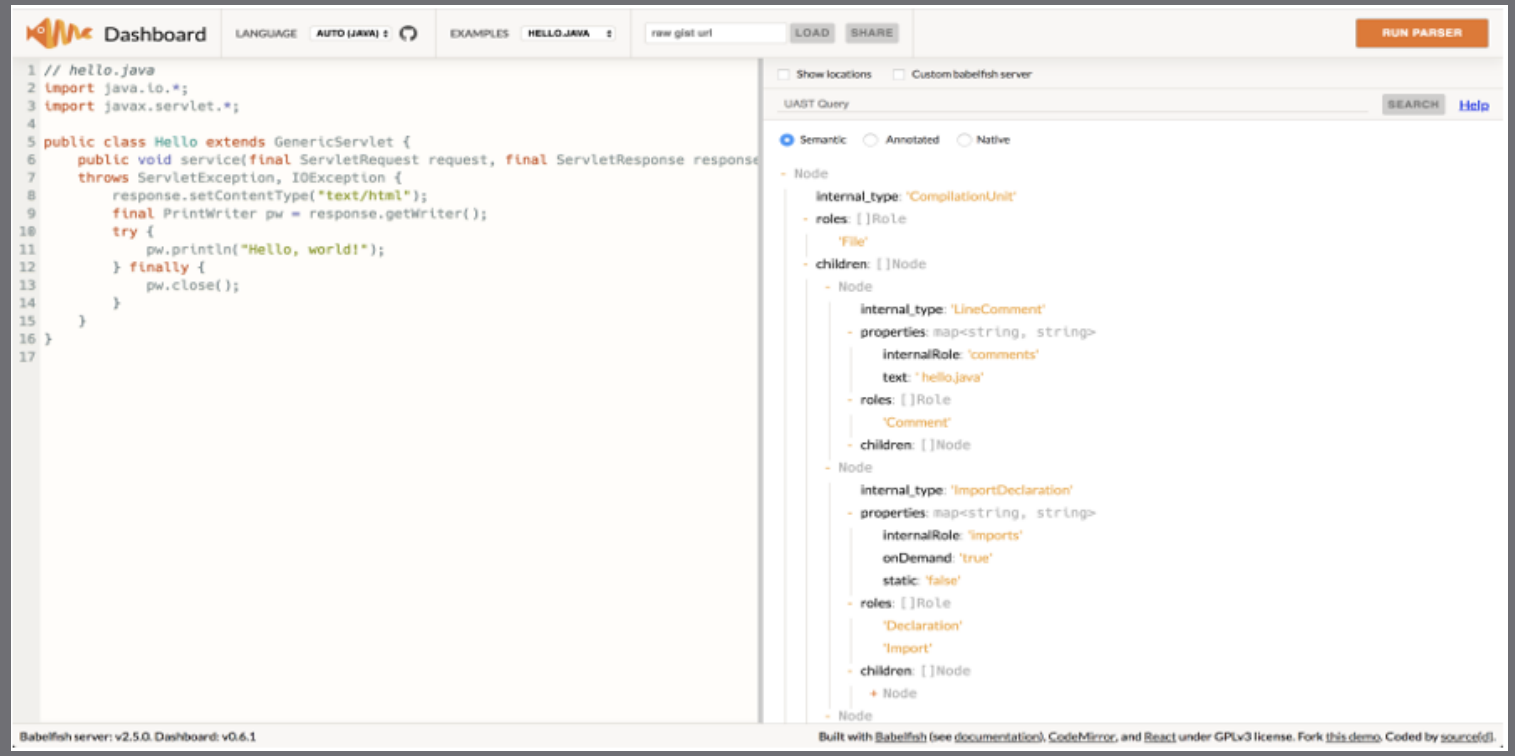



# Проблеми прогнозування на послідовності токенів

- Послідовності токенів все ще не містять великої к-ті зв’язків
- Необхідно працювати з дуже великою к-стю можливих значень. Адже, кількість символів, що можуть містити програмні мови — фіксована і відносно мала, а кількість можливих ідентифікаторів, що з них формуються, прямує до нескінченності.

# Подальші дослідження

- Навчання на деревах (Абстрактні синтаксичні дерева)
- Навчання на графах



# Висновки

- Проведене дослідження підтверджує придатність підходів машинного навчання до вирішення задач статичного аналізу програмних кодів
- Проведені експерименти демонструють особливості навчання на програмному коді та надають розуміння можливостей подальших досліджень
- Дана сфера тільки розпочинає розвиватись та має великі перспективи в найблищому майбутньому

# Дякую за увагу!