Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский ядерный университет «МИФИ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к индивидуальному проекту в рамках образовательной программы трека «Искусственный интеллект» проекта «Samsung Innovation Campus»

«ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ ФАЙН-ТЮНИНГА LLM ДЛЯ РЕШЕНИЯ ЗАДАЧИ ПОВЫШЕНИЯ ЧИТАБЕЛЬНОСТИ ДЕКОМПИЛИРОВАННОГО КОДА НА ЯЗЫКЕ СИ»

Автор проекта:
студент группы С23-712
Кислов К.А.

Руководитель:
Егоров А.Д.

Москва 2024

Оглавление

Введение

Классическая проблема в сфере реверс-инжиниринга – это проблема восстановления исходного кода из машинного для последующего анализа: определение принципов работы программы, используемых данных, нахождение слабых мест и изучение реакции на аварийные ситуации. Для выполнения данной задачи используются промышленные программы-декомпиляторы. Однако они генерируют код, который, во-первых, нельзя повторно компилировать (лишь небольшое число декомпиляторов обладают данной функцией), во-вторых, содержащаяся в нем информация требует от человека значительного времени для анализа: названия переменных и функций лишены изначального заложенного смысла и трудно прослеживается логика работы программы (рисунок 1).

```
#include <windows.h>
#include <defs.h>


//-------------------------------------------------------------------------
// Function declarations

int printf(const char *const Format, ...);
static time_t __cdecl time(time_t *const Time);
int __fastcall main(int argc, const char **argv, const char **envp);
__int64 __fastcall _main(_QWORD, _QWORD, _QWORD); // weak
// void __cdecl srand(unsigned int Seed);
// int __cdecl rand();


//----- (0000000140001580) ----------------------------------------------------
int __fastcall main(int argc, const char **argv, const char **envp)
{
  unsigned int v3; // eax
  __int64 v5; // [rsp+20h] [rbp-30h]
  __int64 v6; // [rsp+28h] [rbp-28h]
  int v7; // [rsp+40h] [rbp-10h]
  unsigned int v8; // [rsp+44h] [rbp-Ch]
  unsigned int v9; // [rsp+48h] [rbp-8h]
  int i; // [rsp+4Ch] [rbp-4h]

  _main(argc, argv, envp);
  v3 = time(0i64);
  srand(v3);
  printf("Weather Simulation:\n\n");
  for ( i = 0; i <= 6; ++i )
  {
    v9 = rand() % 40;
    v8 = rand() % 100;
    v7 = rand() % 20;
    LODWORD(v6) = rand() % 100;
    LODWORD(v5) = v7;
    printf(
      "Day %d:\nTemperature: %d degC\nHumidity: %d%%\nWind Speed: %d m/s\nRainfall: %d mm\n\n",
      (unsigned int)(i + 1),
      v9,
      v8,
      v5,
      v6);
  }
  return 0;
}
// 1400016D4: variable 'v5' is possibly undefined
// 1400016D4: variable 'v6' is possibly undefined
// 1400017B0: using guessed type __int64 __fastcall _main(_QWORD, _QWORD, _QWORD);

// nfuncs=141 queued=1 decompiled=1 lumina nreq=0 worse=0 better=0
// ALL OK, 1 function(s) have been successfully decompiled
```

Рис. 1.1: Декомпилированный код.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int temperature, humidity, windSpeed, rainfall;
    srand(time(NULL)); // initialize random seed based on time
    printf("Weather Simulation:\n\n");
    for(int i = 0; i < 7; i++) { // simulate 7 days of weather
        temperature = rand() % 40; // generate random temperature between 0 and 39 degrees Celsius
        humidity = rand() % 100; // generate random humidity between 0% and 99%
        windSpeed = rand() % 20; // generate random wind speed between 0 and 19 meters per second
        rainfall = rand() % 100; // generate random rainfall between 0mm and 99mm
        printf("Day %d:\nTemperature: %d degC\nHumidity: %d%%\nWind Speed: %d m/s\nRainfall: %d mm\n\n", i+1, temperature, humidity, windSpeed, r
ainfall);
    }
    return 0;
}
```

Рис. 1.2: Исходный код.

Стремительно развивающиеся технологии в сфере NLP позволяют приблизится к решению данных двух проблем. Из существующих эффективных вариантов для обработки вывода декомпиляторов можно выделить: 1) DIRTY [1] – seq2seq модель на базе трансформера для восстановления изначальных имен и типов переменных; 2) LmPa [2] – система для отправки запросов в ChatGPT с задачей изменить названия переменных и функций, 3) DecGPT [3] – проект, использующий в основе GPT-3.5 для исправления в коде ошибок, возникающих при повторной компиляции. Также существует ряд решений, которые обрабатывают изначальный машинный код (Beyond The C [4]) и анализируют не только код, но и иные структуры данных, например абстрактные синтаксические деревья и графы потоков управления (DIRE [5], SymLM [6], NFRE [7]).

Приведенные выше и многие другие решения для упрощения анализа кода направлены на внесение небольших изменений в результат работы декомпилятора (например, новых имен переменных и функций) и слабо затрагивают возможности дообучения существующих мощных LLM для более кардинального его улучшения, вплоть до написания программы со структурой кода, близкой к исходной.

В связи с этим была выдвинута гипотеза: предобученная LLM при относительно небольших затратах на ее тонкую настройку сможет продемонстрировать хорошие результаты при решении задачи повышения читабельности декомпилированного кода. И определена цель проекта: разработка адаптера на основе предобученной языковой модели для решения задачи интеллектуальной обработки (в данном проекте – повышения читабельности) декомпилированного кода, а также оценка его работоспособности.

Исследовательская составляющая проекта:

- Объект исследования – декомпиляция программного кода.

- Предмет исследования – применение языковых моделей для анализа программного кода.

-Методы исследования: поиск, анализ, сравнение, моделирование, программирование, тестирование, измерение.

Для достижения поставленной цели необходимо решить ряд задач (которые отражают ход работы над проектом):

1) Поиск и рассмотрение аналогов;

2) Выбор языковой модели;

3) Сбор первой части датасета (исходный код на Си);

4) Компиляция и декомпиляция – подготовка второй части датасета (входные данные модели);

5) Файн-тюнинг модели;

6) Проведение тестов.

Стек технологий

1. Языковая модель. В качестве LLM для последующей точной настройки была выбрана модель CodeLlama-7b [8] – дообученная модель Llama 2 [9] для написания, завершения и исправления кода. Семейство моделей Llama 2 на данный момент является довольно популярным вариантом для файн-тюнинга, так как адаптеры на их основе демонстрируют впечатляющие результаты при решении узких задач, несмотря на необходимость гораздо меньшего объема ресурсов по сравнению с первичным обучением LLM. Кроме того, выбор в пользу данного варианта обусловлен следующими особенностями модели: 1) Возможность выбора модели с оптимальным количеством обучаемых параметров – это позволит проводить точную настройку, используя доступную вычислительную мощность (у рассматриваемой модели около 7 млрд. обучаемых параметров); 2) Большой размер контекстуального окна – 4096 токенов (достаточно для большинства декомпилированных программ); 3) Токенизатор SentencePiece, обученный на программном коде; 4) Модель CodeLlama обучалась работать с кодом, поэтому для ее файн-тюнинга для решения рассматриваемой задачи требуется гораздо меньше времени и ресурсов, чем при точной настройке обычной Llama 2.
2. Датасет (исходный код). Первая часть обучающей выборки (95 тыс. примеров из исходного кода на Си) была сформирована из датасета FormAI Dataset [10] – большой коллекции созданных с помощью GPT-3.5-turbo компилируемых и независимых программ на языке Си. Примеры из данной части являются ground truth для модели в ходе обучения.
3. Датасет (декомпилированный код). Вторая часть обучающей выборки была сформирована посредством компиляции примеров из ground truth (компилятор – GCC

11.4.0) и последующей декомпиляции при помощи программы Hex-Rays [11] (8.3.0.230608) – одного из самых популярных промышленных декомпиляторов.

4. Тестовая выборка. Для тестирования модели была собрана выборка из 100 примеров вывода Hex-Rays, которые отсутствовали в датасете для обучения. Кроме того, аналогично сформирована выборка из 100 примеров кода, сгенерированного декомпилятором, с которым модель не была ознакомлена в ходе точной настройки, - RetDec [12], примеры работы которого сложнее для восприятия человеком, чем вывод Hex-Rays.

Fine tuning

Процесс точной настройки планировалось осуществить, используя доступные вычислительные ресурсы (предоставляемые бесплатно на платформе Kaggle – GPU T4 x2). В следствие чего модель перед точной настройкой была инициализирована с квантизацией Normal Float 4 (потребление видеопамяти при таком варианте составляет 15.6 Гб). Все обучаемые параметры модели в ходе обучения адаптера оставались замороженными.

Обучаемый LoRA адаптер был инициализирован со следующими параметрами: LoRA R = 1; LoRA alpha = 16; LoRA dropout = 0.05; целевые модули: query, key, value, output. Всего адаптер имеет 1.05 млн. обучаемых параметров (0.015 % от всех параметров модели). В ходе тестирования были получены результаты, позволяющие сделать вывод, что и такого относительно небольшого количества параметров достаточно для явного повышения эффективности исходной модели при решении рассматриваемой задачи. На рисунке 2 детально представлена архитектура модели с адаптером.

Модель с адаптером обучалась в режиме float16 (FP16) с использованием оптимизатора AdamW. Метод обучения – Self-Supervised Learning: на модель подавался пример по шаблону, состоящего из краткой постановки задачи (чтобы уменьшить loss на первых итерациях обучения), декомпилированного кода и соответствующего ему исходного кода. Примеры в таком формате являлись как входными данными, так и ожидаемыми выходными. При этом вход модели токенизировался стандартным образом, а в целевых данных декомпилированный код заменялся на специальные токены, по которым не протекал градиент (чтобы модель обучалась предсказывать только исходный код).

После обучения модели с адаптером в течении около 90 часов последний лучший loss составил 0.1435 (данный показатель после обработки нескольких первых сотен примеров был равен 0.2624).

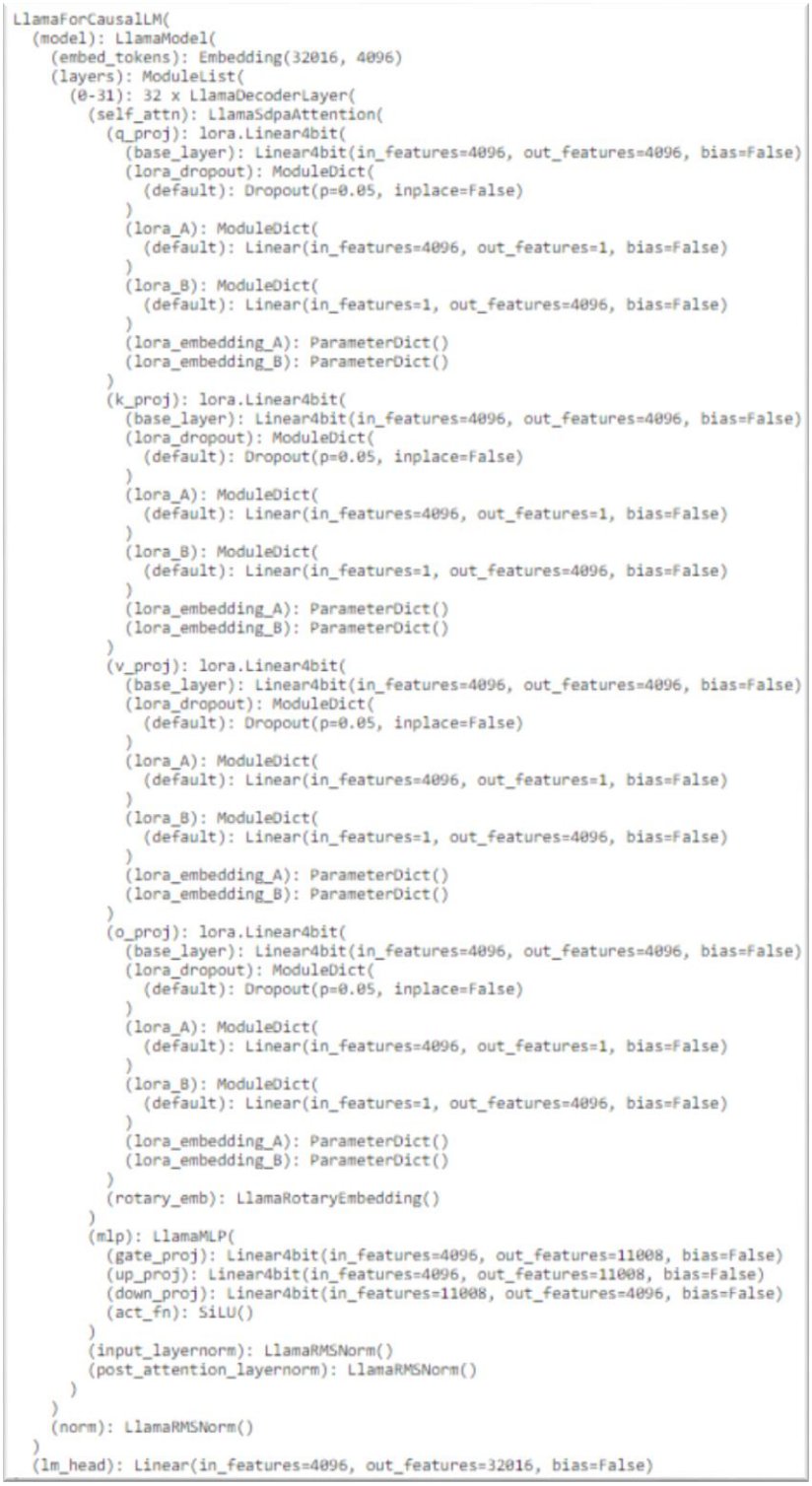

Рис. 2: Архитектура модели с адаптером.

Тестирование

Перед существующими, перечисленными в введении аналогами стояли частные задачи (чаще всего восстановление имен переменных и функций, исправление ошибок повторной компиляции), которые не включали в себя полную генерацию по результату работы декомпилятора кода, максимально близкого по структуре и функционалу к

исходному. Вследствие чего целесообразнее сравнивать эффективность разработанного решения (далее DecLlama) с наиболее близким аналогом – моделью CodeLlama без точной настройки. Для этого использовался описанный ранее датасет для тестирования. Рассматриваемые метрики: Sentence BLEU, Corpus BLEU, среднее расстояние редактирования (AED) и субъективная оценка. В ходе тестирования по классическим для seq2seq моделей метрикам были получены следующие результаты: 1) декомпилятор Hex-Rays (CodeLlama / DecLlama): 1.1) Sentence BLEU = 40.70% / 44.74%; 1.2) Corpus BLEU = 38.86% / 42.57%; 1.3) AED = 0.92 / 0.67; 2) декомпилятор RetDec (CodeLlama / DecLlama): 2.1) Sentense BLEU = 41.80% / 37.02%; 2.2) Corpus BLEU = 37.95% / 34.68%; 2.3) AED = 1.13 / 0.71. В случае с Hex-Rays DecLlama по всем метрикам лучше CodeLlama. Однако в случае с не использовавшимся для формирования обучающей выборки декомпилятором RetDec DecLlama почти по всем показателям, кроме AED, уступает модели без точной настройки.

Это можно было бы объяснить либо переобучением на примерах работы одного декомпилятора, либо недостатком метрик для адекватной оценки модели. Поэтому проводилась дополнительная субъективная оценка: каждый сгенерированный одной и другой моделью код просматривался и сравнивался с исходным. За один пример модель могла получить оценку по шкале от 0 до 2. 0 выставлялся, если сгенерированный код совершенно не отражал структуру и функционал исходного кода; 1 – если в нем не хватает некоторых важных функций и/или названия переменных и функций по-прежнему лишены смысла (например, v1 вместо node), но при этом по коду возможно сделать правильные выводы по работе программе; 2 – если сгенерированный код, несмотря на незначительные недочеты, довольно близок к исходному коду или реализует эквивалентный функционал, имеющий некритичные отклонения от оригинала. Результаты субъективной оценки: 1) декомпилятор Hex-Rays (CodeLlama / DecLlama): 0 – 46% / 5%, 1 – 16% / 8%, 2 – 38% / 87%, средняя оценка – 0.92 / 1.82; 2) декомпилятор RetDec (CodeLlama / DecLlama): 0 – 25% / 20%, 1 – 48% / 33%, 2 – 27% / 47%, средняя оценка – 1.02 / 1.27. Данные результаты говорят о том, что с задачей повышения читабельности декомпилированного кода для последующего анализа программы, эффективность выполнения которой объективнее, как ни странно, может оценить именно человек, лучше справляется модель с обученным адаптером, в том числе и при обработке вывода RetDec.

Чтобы провести оценку моделей по другим метрикам и сделать доступными результаты субъективной оценки каждого примера, было сформировано четыре CSV-файла, содержащих исходный код 100 программ на Си, декомпилированный (Hex-Rays / RetDec) код, сгенерированный моделью (CodeLlama / DecLlama) код и оценку каждого примера по шкале от 0 до 2.

Заключение

Проведенные тесты показывают, что модель с адаптером, несмотря на относительно небольшие для NLP объем датасета и количество эпох, достигает приличных результатов, в том числе при обработке кода, декомпилированного при помощи программ, примеров вывода которых не было в обучающей выборке (RetDec). Что подтверждает выдвинутую гипотезу. В дальнейшие планы работы над проектом входят: продолжение обучения модели на датасете большего объема и с примерами работы других декомпиляторов, проведение более масштабного тестирования как с классическими для задачи seq2seq метриками, так и с оценкой при помощи опроса специалистов и с проверкой возможности перекомпилирования результатов работы нейросети.

Контакты и полезные ссылки

- Почта автора: kostik_kislov@list.ru
- Репозиторий на GitHub: https://github.com/KislovKonstantin/Samsung-project-DecLlama
- Чекпоинт адаптера на Hugging Face: https://huggingface.co/KonstantinKislov/CodeLlama_adapter_for_solving_the_problem_of_increasing_the_readability_of_decompiled_C_code
- Видео демонстрация: https://youtu.be/dfIAFchQt0A
- Обучающий датасет: https://www.kaggle.com/datasets/kislovka/95131hr
- Тестовый датасет с оценкой: https://www.kaggle.com/datasets/kislovka/decllama-evaluation-dataset

Список литературы

1. Qibin Chen, Jeremy Lacomis, Edward J Schwartz, Claire Le Goues, Graham Neubig and Bogdan Vasilescu. Augmenting decompiler output with learned variable names and types. In 31st USENIX Security Symposium (USENIX Security 22), pages 4327–4343, 2022.
2. Xu Xiangzhe, Zhang Zhuo, Feng Shiwei, Ye Yapeng, Su Zian, Jiang Nan, Cheng Siyuan, Tan Lin and Zhang Xiangyu. LmPa: improving decompilation by synergy of large language model and program analysis. arXiv preprint arXiv:2306.02546v1 (2023).
3. Wai Kin Wong, Huaijin Wang, Zongjie Li, Zhibo Liu, Shuai Wang, Qiyi Tang, Sen Nie and Shi Wu. Refining decompiled C code with large language models. arXiv preprint arXiv:2310.06530v2 (2023).
4. Iman Hosseini and Brendan Dolan-Gavitt. Beyond the C: retargetable decompilation using neural machine translation. arXiv preprint arXiv:2212.08950v1 (2022).
5. Jeremy Lacomis, Pengcheng Yin, Edward J. Schwartz, Miltiadis Allamanis, Claire Le Goues, Graham Neubig and Bogdan Vasilescu. DIRE: a neural approach to decompiled identifier naming. International Conference on Automated Software Engineering (ASE 2019), pages 628-639, 2019.
6. Xin Jin, Kexin Pei, Jun Yeon Won and Zhiqiang Lin. SymLM: predicting function names in stripped binaries via context-sensitive execution-aware code embeddings. CCS '22: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pages 1631–1645, 2022.
7. Han Gao, Shaoyin Cheng, Yinxing Xue and Weiming Zhang. A lightweight framework for function name reassignment based on large-scale stripped binaries. The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021), pages 607–619, 2021.
8. CodeLlama-7b-hf. [Электронный ресурс] – URL: https://huggingface.co/codellama/CodeLlama-7b-hf.
9. Llama 2. [Электронный ресурс] - URL: https://huggingface.co/blog/llama2
10. FormAI-Dataset. [Электронный ресурс] – URL: https://github.com/FormAI-Dataset/FormAI-dataset/tree/main
11. Hex-Rays. [Электронный ресурс] - URL: https://hex-rays.com/decompiler
12. RetDec. [Электронный ресурс] - URL: https://github.com/avast/retdec