Отчет по индивидуальному проекту Оптимизация параметров компиляции в clang

Шаго Павел Евгеньевич
Группа: 22.Б07-пу

Сентябрь 2025

1 Постановка задачи

Целью данного проекта является разработка автоматизированного метода подбора оптимального набора флагов компиляции clang, который обеспечивает улучшение ключевых характеристик сгенерированных бинарных файлов (время исполнения и размер). В качестве критерия качества принимается многомерная метрика, включающая время выполнения целевого приложения (минимизация) и размер итогового исполняемого файла (минимизация).

Для проверки качества модели формируется два независимых множества примеров:

- **Обучающее множество:** набор из 2000 примеров, каждая запись содержит исходный код тестового бенчмарка, конфигурацию флагов компиляции и замеры времени выполнения и размера бинарника.
- **Тестовое множество:** набор из 200 примеров, не использованных при обучении.

2 Формулировка задачи обучения

2.1 Обучающее множество

Каждый пример задается вектором бинарных признаков фиксированной длины, соответствующих параметру или отсутствию каждого из $d = 57$ выбранных флагов clang, а также в будущем дополнительными статическими фичами исходного кода (количество функций, глубина вложенности циклов, число операций ввода-вывода), всего (для первой версии) $d_{\text{total}} = 57$ признаков. Обучающее множество состоит из 2000 таких векторов и соответственных целевых значений метрик.

2.2 Результаты разведочного анализа данных

Предварительный анализ показал, что распределение времени выполнения имеет значимую дисперсию (от нескольких миллисекунд до нескольких секунд), что указывает на высокую чувствительность разных флагов. Размер бинарников варьируется от 30 до 200 КБ. Корреляционный анализ выявил сильное влияние флагов `-O3` и `-flto` на уменьшение времени и роста размера соответственно.

2.3 Описание тестового множества

Тестовое множество сформировано аналогично обучающему: 200 примеров, с тем же распределением признаков, но взятое из независимого пула бенчмарков.

2.4 Предобработка данных

На этапе предобработки выполнены следующие шаги:

1. Проверка на дублирование и удаление повторяющихся записей.
2. Нормализация числовых признаков (статических фич) методом z-оценки.
3. Кодирование флагов компиляции:
    - бинарные флаги (например, `-fomit-frame-pointer`) кодируются как 0/1;
    - многозначные флаги (например, `-flto=thin`, `-flto=auto`) представляются с помощью one-hot кодирования.

При генерации данных был использован алгоритм случайной выборки: комбинации флагов отбирались равновероятно из множества из 57 флагов, описанного в разделе 3.

3 Описание модели

В работе используется полносвязная нейронная сеть (MLP) с архитектурой:

- Входной слой: размер $d_{\text{total}} = 57$
- Два скрытых слоя по 128 нейронов с функцией активации ReLU
- Выходной слой: два нейрона, отвечающих за предсказание времени выполнения и размера

Метод обучения: стохастический градиентный спуск с адаптивным шагом (Adam). Целевая функция: среднеквадратичная ошибка (MSE) по обеим метрикам. Гиперпараметры обучения:

- Размер батча: 32
- Количество эпох: 100
- Начальный шаг обучения (learning rate): 10^{-3}

4 Описание программы

Реализация выполнена на Python 3.13. Использованные библиотеки:

- `PyTorch` для построения и обучения нейросети
- `subprocess` для автоматизации компиляции бинарников через clang
- `pandas` и `numpy` для работы с данными

Скрипт сбора данных и обучения объединен в файле `tune.py`, для удобства запуска предусмотрен Makefile.

5 Код

Ссылка на репозиторий:
`https://github.com/shgpavel/edu_ml/clang-ml`

6 Вычислительный эксперимент

Обучение проводилось на машине с GPU Intel Arc A770. Среднее время одной эпохи составило около 15 секунд, общее время обучения 100 эпох – порядка 25 минут. При обучении наблюдалось переобучение после 70-й эпохи, для борьбы с этим был введен Dropout (0.2) в скрытых слоях.

7 Выводы

В ходе эксперимента было показано, что предлагаемая MLP-модель способна предсказывать метрики качества бинарников с приемлемой точностью (MAE менее 5% от среднего значения) и эффективно использовать статические признаки исходного кода вместе с конфигурацией флагов.