Санкт-Петербургский государственный университет
Кафедра системного программирования

Автоматизация и улучшение декомпиляции Kotlin

Евдокимов Данил

Научный руководитель: доцент кафедры Системного Программирования, к.ф.-м.н., Гориховский В. И.

Рецензент: главный инженер-программист, ООО «ШВАЧЕР», Васенина А. И.

Санкт-Петербург
2025

Введение

- Kotlin — основной язык Android с 2019 (Google)
- Нет декомпилятора, восстанавливающего именно Kotlin
- Существующий двухэтапный подход:
  - Байт-код $\longrightarrow$ Java $\longrightarrow$ Kotlin
- Теряются естественность и идиоматические конструкции языка:

Оригинал:

```
data class DataClass(
    val name: String,
    val age: Int
)
```

После декомпиляции:

```
class DataClass(name: String, age: Int) {
    val name: String
    val age: Int
    init {
        Intrinsics.checkNotNullParameter(name as Any, "name")
        ...
```

- Решением может быть применение специализированных больших языковых моделей, уже используемых для декомпиляции — например, **LLM4Decompile** для C.

Постановка задачи

Цель работы: Разработка подхода и инструмента для декомпиляции Kotlin.
Для достижения этой цели были сформулированы следующие задачи:

- Анализ существующих решений в области декомпиляции Java, конвертации Java в Kotlin и их комбинаций
- Разработка набора Kotlin-специфичных примеров и проведение декомпиляции с помощью существующих подходов
- Формирование набора метрик для количественной оценки качества декомпиляции
- Создание датасета для обучения моделей
- Экспериментальное сравнение генераций различных языковых моделей
- Частичное дообучение лучших моделей и оценка влияния дообучения на качество декомпиляции

Инструменты для декомпиляции и конвертации

Декомпиляторы Java			
Инструмент	**Разработчик**	**Особенности**	**Недостатки**
Fernflower	JetBrains	Встроен в IntelliJ IDEA	Менее стабильный результат
CFR	Lee Benfield	Частые обновления	—
JD-GUI	Emmanuel Dupuy	Графический интерфейс	GPL-3.0
Конвертеры Java в Kotlin			
Инструмент	**Способ использования**	**Особенности**	**Недостатки**
J2K	IntelliJ IDEA	Сохраняет структуру Java	Нет CLI, не идиоматично
CodeConvert	Онлайн	Идиоматичный Kotlin	Ограничения по объёму, нет API
Универсальные инструменты			
Инструмент	**Способ использования**	**Особенности**	**Недостатки**
ChatGPT (4o)	API, онлайн	Байт-код $\longrightarrow$ Kotlin	Возможны ошибки и аномалии

Качественное сравнение рассматриваемых подходов[1]

Комбинации инструментов:

- 9 вариантов: декомпилятор + ковертер
- Прямое использование ChatGPT: BytecodeChatGPT
- Например:
  - CFRJ2K
  - JDGUICodeConvert
  - FernflowerChatGPT

Синтетические тесты:

- 33 примера
- Ключевые конструкции:
  - `ScopeFunctions`
  - `Reified`
  - `ClassDelegation`

Пример SmartCast:

```
val obj: Any = "String"
if (obj is String) {
    println(obj.length)
}
```

[1]Подробное сравнение доступно на GitHub (дата обращения: 2025-05-26): https://github.com/akabynda/KotlinDecompilerSyntheticTests/tree/master/src/testResults

Оценка компилируемости декомпилированного кода

- Для каждого синтетического теста выполнена декомпиляция 10 подходами
- Для J2K протестированы два сценария:
  - **J2K (без правок)** — чистая конвертация
  - **J2K (с правками)** — автоматические исправления через IntelliJ IDEA
- Причины ограничений:
  - **J2K** — графический инструмент; генерирует ошибки, устраняемые средствами IDE
  - **ChatGPT** используется без правок; ошибки, как правило, не устраняются автоматически средствами IDE
  - **CodeConvert** не использовался; имеет малое количество бесплатных обращений

Комбинация	CFR	JD-GUI	Fernflower	Bytecode
ChatGPT	94%	97%	94%	100%
J2K (с правками)	58%	55%	58%	—
J2K (без правок)	58%	45%	27%	—

Доля успешно скомпилированных тестов после преобразования.

Пример использования рассматриваемых подходов

Оригинал:

```
fun main() {
    outerLoop@ for (i in 1..5) {
        for (j in 1..5) {
            if (i * j > 6) {
                println("Breaking out
                      of the outer
                      loop at i = $i,
                      j = $j")
                break@outerLoop
            }
            println("i = $i, j = $j")
        }
    }
}
```

CFRChatGPT:

```
fun main() {
    loop@ for (i in 1 until 6) {
        for (j in 1 until 6) {
            if (i * j > 6) {
                println("Breaking out
                      of the outer
                      loop at i = $i,
                      j = $j")
                break@loop
            }
            println("i = $i, j = $j")
        }
    }
}
```

CFRJ2K:

```
object LabeledBlocks {
    fun main() {
        block0@ for (i in 1..5) {
            for (j in 1..5) {
                if (i * j > 6) {
                    println((("Breaking
                          out of the outer
                           loop at i = \$i
                          , j = \$j")) as
                          Any)
                    break@block0
                }
                println(((("i = $i, j = $j
                     ")) as Any)
            }
        }
    }

    @JvmStatic
    fun main(args: Array<String>) {
        main()
    }
}
```

Метрики оценки декомпиляции

Структурные метрики:

- Program Size, Conditional Complexity
- Abrupt Control Flow, Labeled Blocks
- Local Variables, Cyclomatic Complexity
- Chapin Q, Pivovarsky Complexity

Halstead-метрики:
Объём, сложность, усилие, ошибки

Информационно-статистические метрики:

- Cross-Entropy, Conditional Entropy
- Perplexity, KL и JS-дивергенция

Статистические метрики вычисляются:

- **Попарно:** между оригиналом и декомпиляцией
- **Относительно статистической модели языка:** сравнение кода с распределением, полученным из корпуса реального кода
- Позволяет оценить как *сохранение стиля*, так и *Kotlin-специфичность*

Получение и сокращение признаков

Исходные данные:

- Посчитаны ранее указанные метрики для:
  - Синтетических тестов и их декомпиляций
  - `KStack-clean` — 25 000 Kotlin-фрагментов
  - `KExercises` — 15 000 задач в стиле HumanEval

Статистическая модель языка (LM):

- Распределение униграмм и биграмм для `KStack-clean` + `KExercises`

Методы сокращения:

- Признаки с низкой дисперсией (< 0.01)
- Сильно коррелирующие

Итог: 8 ключевых метрик

- Conditional Complexity
- Halstead Distinct Operators
- Conditional Entropy
- JSD, KL
- Conditional Entropy (от LM)
- Cross-Entropy (от LM), KL (от LM)

Сравнение по отобранным метрикам

Оригинал:

```
fun main() {
    for (i in 1..10) {
        if (i == 5) {
            println("Breaking the loop
                    at $i")
            break
        }
        if (i % 2 == 0) {
            println("Skipping even
                    number: $i")
            continue
        }
        println("Current value: $i")
    }
}
```

FernflowerJ2K:

```
object AbruptControlFlow {
    fun main() {
        for (i in 1..10) {
            var var1: String
            if (i == 5) {
                var1 = "Breaking..."
                println(var1)
                break
            }

            if (i % 2 == 0) {
                var1 = "Skipping..."
                println(var1)
            } else {
                var1 = "Current..."
                println(var1)
            }
        }
    }

    @JvmStatic
    fun main(args: Array<String>) {
        main()
    }
}
```

Ключевые метрики:

Метрика	Ориг	FJ2K
Cond Comp	4.0	4.0
Dist Ops	5	11
LM Cond En	-0.65	1.92
LM Cross En	7.74	9.18
LM KL	3.49	4.36
Cond En	7.06	
JS	0.42	
KL	9.24	

Вывод по результатам сравнения

- Комбинации с **ChatGPT**:
  - большее число успешных компиляций
  - метрики близкие к оригиналу
- Комбинации с **J2K**:
  - меньшее число успешных компиляций
  - использует конструкции Java
- **Прямая декомпиляция с ChatGPT:**
  - наибольшее число успешных компиляций
  - метрики близкие к оригиналу
  - не требует промежуточных преобразований

Ограничения:

- ChatGPT — универсальная модель (не для Kotlin)
- Возможны галлюцинации и нестабильность
- Нет контроля над архитектурой

Вывод:

- Требуется **специализированная модель** для декомпиляции Kotlin, обученная на парах байт-код — исходный код

Формирование датасета[2] для обучения

Исходные данные:

- `KStack-clean` (25 000 фрагментов)
- `KExercises` (15 000 задач)

Структура:

`{owner}__{repo}__{commit}/<path_to_file>`

Компиляция:

- `kotlinc -jvm-target 23`
- Зависимости: Gradle

Обработка:

- Автокоррекция импортов
- Удаление дубликатов
- Лог ошибок: `compile_errors.log`

Итог:

- 13 528 пар «байт-код — Kotlin»
- Разделение на train/test (90%/10%)

[2] `KExercises-KStack-clean-bytecode` (дата обращения: 2025-05-26): `https://huggingface.co/datasets/akabynda/KExercises-KStack-clean-bytecode`

Экспериментальное сравнение моделей

Цели эксперимента:

- Определить, какие модели лучше подходят для дообучения

Описание эксперимента:

- Использовано 100 тестов (97 репозиториев)
- 20 моделей: от 0.5B до 7B от JetBrains и лидеры Big Code Models Leaderboard[3]
- Для каждой модели:
  - Генерируется Kotlin из байт-кода
  - Вычисляются 8 ключевых метрик качества
  - Сравнение с эталоном по **расстоянию Чебышёва**, масштабированному по доле успешных компиляций

Выбраны компактные модели:

- `Qwen2.5-Coder-0.5B-Instruct`: $d_{\text{chebyshev}} = 10.99$, 84/97
- `deepseek-coder-1.3b-instruct`: $d_{\text{chebyshev}} = 8.81$, 92/97

[3] Big Code Models Leaderboard, URL: `https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard` (дата обращения: 2025-05-26)

Оценка эффективности дообучения моделей

Модель	$d_{\text{chebyshev}}$	Число компиляций
До дообучения		
`Qwen2.5-Coder-0.5B-Instruct`	10.99	84 / 97
`deepseek-coder-1.3b-instruct`	8.81	92 / 97
После дообучения		
`deepseek-coder-1.3b-instruct-Finetuned`	6.64	90 / 97
`Qwen2.5-Coder-0.5B-Instruct-Finetuned`	7.51	90 / 97
Для сравнения (не дообучена, большая модель)		
`Qwen2.5-Coder-7B-Instruct`	8.72	92 / 97

Вывод:

- Дообучение улучшает качество и снижает отклонения
- Компактные модели после дообучения сопоставимы с крупной моделью без него

Дообучение моделей: особенности

- В работе рассматривается только факт успешной компиляции, а не совпадение функциональности
  - Вариант решения — добавление к признакам оценку покрытия функциональными тестами
- У `deepseek-coder-1.3b-instruct` после дообучения число компиляций снизилось
  - Решение — изменение параметров генерации, например:
    - `doSample=True`
    - Увеличить число генерируемых токенов
  - Пример: до и после `doSample=True`

До настройки:

```
/**
 * This function calculates the salary of a
      person based on their annual income.
```

Код функции не сгенерирован;
комментарий не закрыт

После настройки:

```
fun calculateSalary(annualIncome: Double):
    Long {
    return Math.round(annualIncome)
}
```

Заключение

В ходе работы были достигнуты следующие результаты:

- Проведен анализ существующих подходов к декомпиляции Kotlin и выявлены их ограничения
- Сформирован и опубликован набор Kotlin-специфичных примеров
- Разработан набор метрик для оценки естественности декомпилированного кода
- Проведена декомпиляция с использованием 10 различных комбинаций инструментов и выполнена оценка по структурным и статистическим метрикам
- Создан и опубликован датасет из 13 528 пар «байт-код — исходный Kotlin»
- Проведено экспериментальное сравнение генераций различных языковых моделей
- Выбраны и дообучены компактные модели; достигнуто снижение отклонений и улучшение метрик качества
- Экспериментальное сравнение показало эффективность дообучения LLM для декомпиляции JVM байт-кода в Kotlin

Код и материалы: `https://github.com/akabynda/KotlinDecompiler`

Пример декомпиляции с JD-GUI+ChatGPT

Оригинал:

```
fun main() {
    val numbers = listOf(1, 2, 3, 4, 5)
    numbers.forEach { n -> println(n * n) }
}
```

Декомпиляция с ChatGPT:

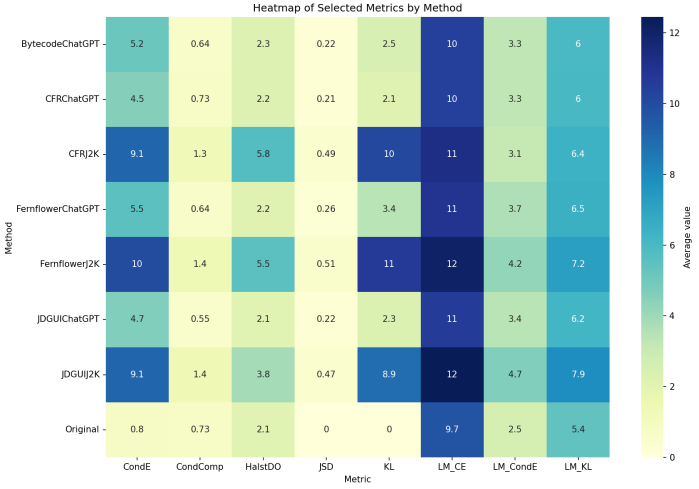
```
fun main() {
    val arrayOfInteger = arrayOf(1, 2, 3, 4, 5)
    val numbers = arrayOfInteger.toList()

    numbers.forEach {
        val n = it
        val i = n * n
        println(i)
    }
}
```

Пример декомпиляции с JD-GUI+J2K

```
@SourceDebugExtension(["SMAP\nLambda.kt\...])
object LambdaJ2K {
    fun main() {
        val arrayOfInteger = arrayOfNulls<Int>(5)
        arrayOfInteger[0] = 1
        ...
        arrayOfInteger[4] = 5
        val numbers: List<*> = listOf(*(arrayOfInteger as Array<Any>))
        val `$this$forEach$iv`: Iterable<*> = numbers
        val `$i$f$forEach` = 0
        val iterator = `$this$forEach$iv`.iterator()
        if (iterator.hasNext()) {
            val `element$iv` = iterator.next()!!
            val n = (`element$iv` as Number).toInt()
            val i = n * n
            println(i)
        }
    }
    @JvmStatic
    fun main(args: Array<String>) {
        main()
    }
```

Сравнение существующих подходов

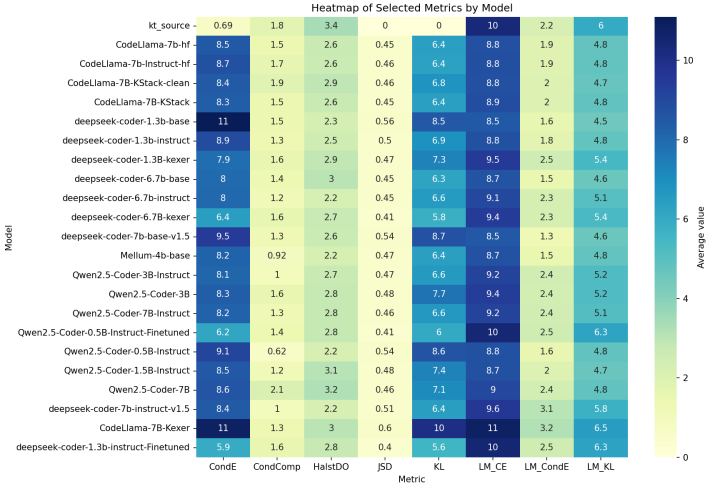


Средние значения ключевых метрик для различных подходов.

Параметры дообучения

- **Дообучались модели:**
  - `Qwen2.5-Coder-0.5B-Instruct`
  - `deepseek-coder-1.3b-instruct`
- **Данные:** тренировочный сплит `KExercises-KStack-clean-bytecode`
- **Параметры:**
  - `train_epochs=4`
  - `grad_accumulation=32`
  - `learning_rate=1e-4`
  - `weight_decay=0.05`
  - `max_grad_norm=0.7`
  - `warmup_ratio=0.1`
  - `bnb_4bit_quant_type="nf4"`
  - `LoRA: r=16, alpha=8, dropout=0.05`
- Параметры подобраны аналогично `deepseek-coder-1.3B-kexer`

Сравнение моделей после дообучения



Средние значения ключевых метрик для различных моделей.

Примеры генерации кода (1)

- **Оригинальный код:**

```
fun hasSignificantDecimalPart(number: Double, decimalThreshold: Double): Boolean {
    val decimalPart = Math.abs(number) - Math.abs(number.toInt())
    return decimalPart >= decimalThreshold
}
```

- **Qwen2.5-Coder-0.5B-Instruct (не компилируется):**

```
fun hasSignificantDecimalPart(a: Double, b: Double): Boolean {
    val absA = Math.abs(a)
    val absB = Math.abs(b)
    return absA - absB > 0.00000...
}
```

- **Qwen2.5-Coder-0.5B-Instruct-Finetuned:**

```
fun hasSignificantDecimalPart(num: Double, epsilon: Double): Boolean {
    val absNum = Math.abs(num)
    val absEpsilon = Math.abs(epsilon)

    return absNum > absEpsilon
}
```

Примеры генерации кода (2): оригинал

Оригинальный код (не входит обучающий датасет):

```
fun operationsQuiz(a: Boolean, b: Boolean, c: Boolean, d: Boolean): Boolean {
    var count = 0

    if (a) {
        count++
    }
    if (b) {
        count++
    }
    if (c) {
        count++
    }
    if (d) {
        count++
    }
    return count >= 2
}
```

Примеры генерации кода (2): deepseek-coder-1.3b-instruct

Генерация модели (не компилируется):

```
fun operationsQuiz(a: Boolean, b: Boolean, c: Boolean, d: Boolean): Boolean {
    var result = a
    if (b) {
        result = result || c
    }
    if (d) {
        result = result && (c xor b)
    }
    return result
}
```

Примеры генерации кода (2): deepseek-coder-1.3b-instruct-Finetuned

Генерация модели:

```
fun operationsQuiz(a: Boolean, b: Boolean, c: Boolean, d: Boolean): Boolean {
    var count = 0

    if (a) {
        count++
    }
    if (b) {
        count++
    }
    if (c) {
        count++
    }
    if (d) {
        count++
    }
    return count >= 2
}
```