Автоматизация метаморфизма кода Go с использованием больших языковых моделей

Д. К. Мурадян, О. В. Цветков

Аннотация—**Метаморфизм кода является мощной техникой защиты программного обеспечения от реверс-инжиниринга, однако его ручная реализация крайне трудоемка. В данной работе представлен инструментарий для автоматизации процесса метаморфизма кода на языке Go с использованием больших языковых моделей. Инструментарий реализует модульную архитектуру и применяет две ключевые техники: вставку «мертвого» кода и модификацию потока управления, делегируя генерацию преобразований внешним LLM. Проведена экспериментальная оценка с использованием трех современных больших языковых моделей (gemini-2.5-flash-preview-04-17, deepseek-chat-v3-0324, gemini-2.0-flash). Эффективность оценивалась по метрикам функциональной эквивалентности, изменения количества строк кода, цикломатической и когнитивной сложностей. Результаты демонстрируют возможность автоматизации метаморфизма с помощью LLM, значительное усложнение кода по метрикам CC и CogC, но выявляют проблемы со стабильностью сохранения функциональности при модификации потока управления у некоторых моделей. Работа показывает перспективность использования LLM для задач обфускации кода, но подчеркивает необходимость дальнейших исследований для повышения надежности и оценки реальной стойкости кода.**

Ключевые слова—**метаморфизм кода, обфускация кода, большие языковые модели, LLM, Go, автоматизация, защита программного обеспечения, сложность кода, цикломатическая сложность, когнитивная сложность**

I. Введение

Защита программного кода от несанкционированного анализа и модификации стала критической задачей современной кибербезопасности. В эпоху усложнения программных систем и роста целенаправленных атак на интеллектуальную собственность традиционные методы защиты утрачивают эффективность. Реверс-инжиниринг представляет особую угрозу, позволяя злоумышленникам восстанавливать алгоритмы и внутреннюю логику программ [1].

Одним из продвинутых методов противодействия реверс-инжинирингу является метаморфизм кода – техника, при которой программа способна изменять собственную структуру при сохранении исходной функциональности [2]. В отличие от полиморфизма, метаморфизм не полагается на шифрование, а использует набор семантически эквивалентных преобразований кода. Это приводит к генерации большого числа функционально идентичных, но структурно различных вариантов программы, что делает неэффективными сигнатурные методы обнаружения и значительно усложняет ручной анализ.

Несмотря на потенциальную эффективность метаморфизма, его практическое применение затруднено высокой сложностью и трудоемкостью ручной реализации техник обфускации. Существующие автоматизированные инструменты часто ограничены в своих возможностях или ориентированы на специфические платформы [3]. В последние годы значительный прогресс в области искусственного интеллекта, в частности появление больших языковых моделей (LLM), обученных на огромных массивах кода [4], открывает новые перспективы для автоматизации сложных задач программной инженерии, включая метаморфизм кода.

В данной работе представлен инструментарий, разработанный для автоматизации процесса метаморфизма кода на языке Go с использованием LLM. Мы исследуем возможность применения современных нейронных сетей для генерации метаморфических преобразований, реализующих техники вставки «мертвого» кода и модификации потока управления. Целью работы является не только создание прототипа такого инструментария, но и количественная оценка его эффективности, включая анализ сохранения функциональной эквивалентности и степени достигаемой обфускации кода с использованием стандартных метрик сложности.

Статья организована следующим образом: Раздел 2 рассматривает связанные работы и существующие подходы к метаморфизму кода. Раздел 3 детально описывает методологию и архитектуру разработанного инструментария. Раздел 4 представляет систему метрик, используемых для объективной оценки эффективности метаморфических преобразований. В Разделе 5 приводятся результаты экспериментальной оценки и их всесторонний анализ. Завершает статью Раздел 6, содержащий заключение и обсуждение перспективных направлений будущих исследований.

II. Связанные работы

Методы обфускации и метаморфизма кода являются предметом исследований в области компьютерной безопасности и программной инженерии на протяжении нескольких десятилетий. Фундаментальные техники обфускации, такие как вставка неиспользуемого кода, модификация потока управления, переименование идентификаторов и преобразование данных, были систематизированы и подробно описаны в основополагающих работах, например, Collberg и др. [1]. Метаморфизм, как развитие идей обфускации, фокусируется на самомодификации

Статья получена 1 мая 2025

Давид Каренович Мурадян, МГУ им. М.В. Ломоносова, (email: itzkhaotic1@gmail.com).

Олег Вячеславович Цветков, РТУ МИРЭА, (email: oleg@tsv.one).

кода с целью уклонения от обнаружения, что детально рассмотрено в работе Szor [2]. Различные аспекты эволюции и реализации метаморфических техник также анализируются в обзорах Sharma и Sahay [5] и Brezinski и Ferens [6].

Несмотря на глубокую теоретическую проработку, автоматизация метаморфизма остается сложной задачей. Существующие инструментарии часто либо являются коммерческими продуктами с закрытым исходным кодом, либо представляют собой академические прототипы или генераторы вредоносного ПО, ориентированные на низкоуровневые представления кода (байт-код, ассемблер) и специфические платформы [6]. Обзоры средств защиты программного обеспечения, такие как работа Schrittwieser и др. [3], часто отмечают ограничения существующих общедоступных инструментов обфускации, которые редко реализуют сложные метаморфические преобразования и не обладают адаптивностью. Большинство подходов либо применяют предопределенный набор правил, либо требуют значительного ручного вмешательства для настройки.

Исследования в области применения машинного обучения и формальных методов чаще фокусировались на обнаружении и анализе метаморфического кода [7, 8], а не на его автоматизированной генерации с целью защиты легитимного ПО. Подходы к изучению правил трансформации из образцов [8] интересны, но не решают задачу создания метаморфического кода по запросу.

Данная работа направлена на восполнение этого пробела. В отличие от классических инструментов, мы предлагаем подход, использующий возможности современных LLM для автоматизации метаморфических преобразований непосредственно на уровне исходного кода языка Go. Мы не фокусируемся на низкоуровневых представлениях и не привязываемся к конкретным архитектурам, делегируя интеллектуальную часть генерации преобразований внешним моделям ИИ, что отличает наш подход от большинства существующих решений.

III. Методология и Инструментарий

Для решения задачи автоматизации метаморфизма кода Go с использованием больших языковых моделей была разработана и реализована специализированная методология, воплощённая в программном инструментарии. Основой инструментария является модульная архитектура, реализованная на языке Go, что обеспечивает гибкость, производительность и удобство разработки. Проект структурирован с разделением на пользовательский интерфейс командной строки и внутреннюю логику преобразования кода.

Центральным компонентом системы является пакет `internal/rewriter`, инкапсулирующий всю логику, связанную с метаморфическими преобразованиями. Ключевая особенность данного модуля заключается в том, что он выступает в роли абстрактного интерфейса к внешним LLM, не реализуя самостоятельно алгоритмы обфускации. Его задача — получить фрагмент исходного Go-кода, сформулировать соответствующий запрос к выбранной модели нейронной сети и обработать ее ответ. Пакет поддерживает работу с различными API нейронных сетей, выбор конкретного API осуществляется через перечисление `APIType` (например, поддерживаются варианты для Gemini и OpenRouter). Фабричная функция `NewLLMRewriterWithAPI` отвечает за создание экземпляра переписчика, сконфигурированного для работы с указанным API. Основные методы переписчика включают `RewriteFile`, который читает содержимое входного файла и инициирует процесс его переписывания через LLM, и `SaveRewrittenFile`, сохраняющий полученный результат.

Взаимодействие с LLM является ядром процесса метаморфизма в данной архитектуре. Модуль переписывания формирует запрос, включающий исходный фрагмент кода и текстовый промпт. Именно промпт инструктирует нейронную сеть выполнить преобразование, соответствующее одной из целевых техник метаморфизма: вставки «мертвого» кода или модификации потока управления. Качество генерации метаморфического варианта напрямую зависит от точности и ясности этого промпта, а также от способности конкретной LLM интерпретировать инструкции и генерировать семантически эквивалентный, но структурно измененный Go-код. Предполагается, что модели, обученные на больших кодовых корпусах [4], обладают необходимыми возможностями для выполнения таких преобразований.

На Рисунке 1 представлена общая схема работы разработанного инструментария, от инициализации системы до оценки результатов метаморфических преобразований.

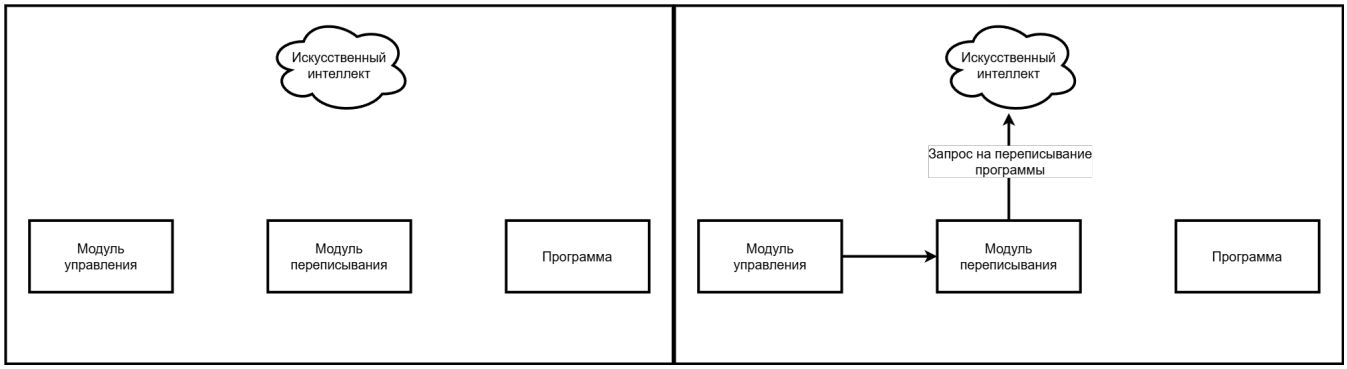
Пользовательский интерфейс реализован в виде CLI-приложения. Этот компонент отвечает за парсинг аргументов командной строки, таких как путь к входному файлу, путь для сохранения результата и выбор используемого API нейронной сети. После валидации входных данных и выбора API, CLI создает экземпляр переписчика из пакета `internal/rewriter` и вызывает его методы для выполнения основной задачи.

Типичный сценарий работы инструментария выглядит следующим образом: пользователь запускает CLI, указывая исходный файл, желаемый выходной файл и используемый API LLM. Приложение парсит аргументы, создает соответствующий объект переписчика. Переписчик читает исходный файл, формирует и отправляет запрос, включая код и промп, к API выбранной нейронной сети. Получив ответ с модифицированным кодом, переписчик сохраняет результат в указанный выходной файл.

После записи модифицированного кода в файл, проводится замер метрик с выводом результатов в командную строку.

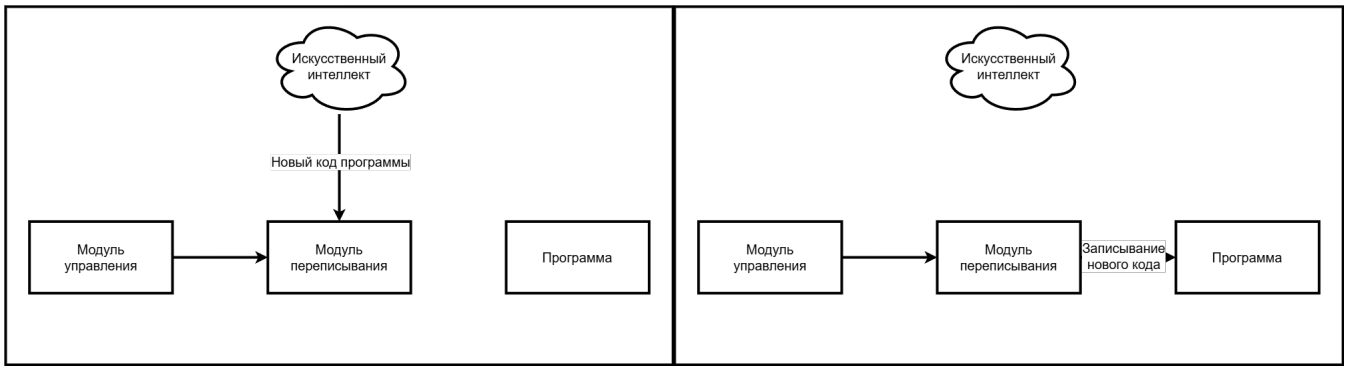
Такая архитектура обладает рядом преимуществ. Четкое разделение логики CLI и ядра переписывания упрощает поддержку и тестирование. Добавление поддержки новых LLM API или даже новых техник метаморфизма путем изменения логики формирования промптов требует модификации только внутреннего пакета `internal/rewriter`, не затрагивая пользовательский интерфейс. Это обеспечивает хорошую расширяемость инструментария.

Оценка качества сгенерированного метаморфического кода проводится на отдельном этапе с использованием метрик функциональной эквивалентности (FE), изменения количества строк кода (LOC_{fin}), цикломатической сложности (CC_{fin}) [9] и когнитивной сложности



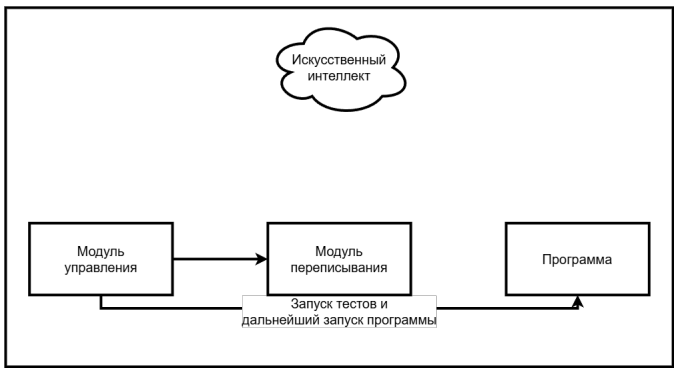
(a) Инициализация системы и загрузка исходного кода

(b) Формирование запроса к языковой модели



(c) Генерация метаморфического кода

(d) Валидация и сохранение результата



(e) Анализ и оценка метрик обфускации

Рис. 1: Процесс метаморфизма кода: от исходного файла до оценки результатов преобразования.

($CogC_{fin}$) [10], что позволяет количественно измерить как корректность преобразований, так и достигнутый уровень обфускации.

IV. Метрики оценки

Для объективной оценки результатов работы разработанного инструментария и сравнения эффективности различных подходов к автоматическому метаморфизму кода недостаточно просто констатировать факт его изменения. Необходимо ввести количественные показатели, которые позволили бы измерить как сохранение исходной функциональности, так и достигнутый уровень усложнения и запутывания кода. Поэтому в данном исследовании был использован набор конкретных метрик, описанных ниже.

Ключевой метрикой является функциональная эквивалентность (FE), определяющая степень сохранения семантики программы после метаморфических преобразований. Поскольку основной принцип метаморфизма – изменение формы без изменения содержания, проверка FE является обязательным шагом валидации. В данной работе FE определяется как процент успешно пройденных тестов из заранее подготовленного набора, охватывающего основные сценарии использования модифицируемого кода:

$$FE = \frac{\text{Количество пройденных тестов}}{\text{Общее количество тестов}} \times 100\% \qquad (1)$$

Идеальным результатом является $FE = 100\%$, любое существенное отклонение свидетельствует о внесении семантических ошибок нейронной сетью в процессе генерации кода.

Для оценки изменения объема кода используется метрика количества строк кода (LOC). Хотя LOC является относительно поверхностным показателем, он приобретает значение в контексте техники вставки «мертвого» кода, которая напрямую подразумевает добавление но-

вых строк. Измеряется абсолютное изменение ΔLOC и относительное изменение LOC_{fin}:

$$\Delta LOC = LOC_{\text{metamorphic}} - LOC_{\text{original}} \quad (2)$$

$$LOC_{fin} = \frac{\Delta LOC}{LOC_{\text{original}}} \times 100\% \quad (3)$$

Значительный рост LOC_{fin} ожидаем при вставке кода, однако интерпретировать его следует в комплексе с другими метриками сложности.

Для количественной оценки структурной сложности программы применяется цикломатическая сложность (CC), предложенная Маккейбом [9]. Данная метрика основана на анализе графа потока управления и определяет количество линейно независимых путей в коде, что коррелирует с трудоемкостью тестирования. Формула расчета:

$$CC = E - N + 2P \quad (4)$$

где E — количество рёбер в графе, N — количество узлов, P — количество связных компонентов. В контексте метаморфизма, особенно при модификации потока управления, ожидается рост CC из-за усложнения управляющих конструкций. Рассчитывается относительное изменение:

$$\Delta CC = CC_{\text{metamorphic}} - CC_{\text{original}} \quad (5)$$

$$CC_{fin} = \frac{\Delta CC}{CC_{\text{original}}} \times 100\% \quad (6)$$

Положительное значение CC_{fin} свидетельствует об усложнении структуры программы.

Для оценки сложности кода с точки зрения его понимания человеком-разработчиком используется когнитивная сложность ($CogC$). В отличие от CC, метрика $CogC$ штрафует за конструкции, прерывающие линейный поток чтения кода (условия, циклы, ‘goto‘, рекурсия) и за их вложенность [10]. Обе реализованные техники метаморфизма потенциально увеличивают $CogC$. Измеряется относительное изменение:

$$\Delta CogC = CogC_{\text{metamorphic}} - CogC_{\text{original}} \quad (7)$$

$$CogC_{fin} = \frac{\Delta CogC}{CogC_{\text{original}}} \times 100\% \quad (8)$$

Рост $CogC_{fin}$ интерпретируется как повышение сложности восприятия и анализа кода человеком.

Совокупность этих четырех метрик (FE, LOC_{fin}, CC_{fin}, $CogC_{fin}$) позволяет получить многогранную картину результатов метаморфизма, сравнивая как корректность преобразований, так и эффективность различных нейронных сетей и техник в задаче обфускации кода.

V. Экспериментальная оценка и анализ результатов

После разработки и реализации инструментария был проведен цикл вычислительных экспериментов для количественной оценки эффективности предложенного подхода и сравнения результатов, достигаемых различными моделями искусственного интеллекта. Экспериментальная установка включала описанный инструментарий, тестовую программу на Go и три нейронные сети на основе архитектуры Трансформер: `gemini-2.5-flash-preview-04-17`, `deepseek-chat-v3-0324` и `gemini-2.0-flash`, доступ к которым осуществлялся через API. Для каждой модели и целевой техники метаморфизма (вставка «мертвого» кода или модификация потока управления) использовался специализированный промпт [11—13], содержащий подробное объяснение техники метаморфизма, пример использования, основные замечания по генерации кода на Go и исходный код, который подлежит модификации. Проверка функциональной эквивалентности FE проводилась отдельно для каждого сгенерированного варианта. В данном разделе анализируются показатели FE и метрики обфускации (LOC_{fin}, CC_{fin}, $CogC_{fin}$) для успешно скомпилированных модификаций.

Сводные усредненные данные экспериментов представлены в Таблице I.

Анализ результатов позволяет сравнить влияние техник метаморфизма и эффективность различных моделей LLM. Техника вставки «мертвого» кода стабильно обеспечивает высокую функциональную эквивалентность (92-100%) при значительном увеличении объема кода и существенном росте метрик сложности. Это указывает на способность LLM генерировать семантически нейтральный, но объемный и структурно сложный дополнительный код.

Модификация потока управления демонстрирует иные тенденции. Прирост объема кода значительно ниже, что ожидаемо. Ключевым эффектом является резкое увеличение когнитивной сложности ($CogC_{fin}$ до 1121%), подтверждая гипотезу о том, что данная техника наиболее эффективно затрудняет понимание кода человеком [10]. Однако сохранение функциональной эквивалентности (FE) при этой технике оказалось серьезной проблемой для моделей `deepseek-chat-v3-0324` (43%) и `gemini-2.0-flash` (53%). Лишь модель `gemini-2.5-flash-preview-04-17` смогла обеспечить 100% FE при модификации потока, показав при этом максимальный прирост $CogC_{fin}$. Низкий прирост цикломатической сложности у некоторых моделей при модификации потока может свидетельствовать о генерации конструкций, запутанных для человека, но не обязательно увеличивающих число путей выполнения так же сильно, как вставка сложного «мертвого» кода.

Сравнение моделей LLM выявило различные профили эффективности. Модель `gemini-2.5-flash-preview-04-17` проявила себя как наиболее эффективная, обеспечивая максимальные показатели усложнения, особенно при вставке кода, и являясь единственной моделью, сохранившей 100% FE при модификации потока. Модель `deepseek-chat-v3-0324` показала наименее выраженные результаты, с наименьшим усложнением и крайне низкой FE при модификации потока. Модель `gemini-2.0-flash` продемонстрировала компромиссный результат: идеальная FE при вставке кода с хорошим усложнением, но низкая FE при модификации потока, несмотря на высокий прирост $CogC_{fin}$. Выбор оптимальной модели зависит от приоритетов: максимальная обфускация или гарантированная функциональность.

В целом, эксперименты подтвердили эффективность автоматизированного метаморфизма с использованием LLM для кода Go. Достигнут значительный прирост

Таблица I: Медианные значения метрик для различных моделей нейронных сетей и техник метаморфизма

Модель	**Техника**	FE **(%)**	LOC_{fin} **(%)**	CC_{fin} **(%)**	$CogC_{fin}$ **(%)**
gemini-2.5-flash-preview-04-17					
	Вставка «мертвого» кода	93	**612**	**640**	**905**
	Модификация потока	**100**	**297**	**167**	**1121**
deepseek-chat-v3-0324					
	Вставка «мертвого» кода	92	161	284	417
	Модификация потока	43	89	90	321
gemini-2.0-flash					
	Вставка «мертвого» кода	**100**	245	443	619
	Модификация потока	53	211	72	702

метрик сложности, что демонстрирует потенциал подхода для усложнения анализа. Однако выявлены ключевые проблемы: нестабильность сохранения функциональной эквивалентности, особенно при модификации потока управления, что подчеркивает необходимость тщательного тестирования. Текущие метрики дают лишь косвенную оценку стойкости к реверс-инжинирингу, и требуется разработка более точных методов оценки. Зависимость от качества промптов и вариативность ответов LLM также требуют дальнейшего изучения.

Анализ метрик обфускации (LOC_{fin}, CC_{fin}, $CogC_{fin}$) количественно подтвердил, что применение разработанного подхода приводит к значительному усложнению программного кода. Особенно впечатляющие результаты показала модель `gemini-2.5-flash-preview-04-17`, продемонстрировав как наилучшую функциональную эквивалентность при модификации потока управления (100%), так и максимальные показатели усложнения по метрикам когнитивной сложности. Относительный прирост цикломатической и когнитивной сложности свидетельствует о потенциале метода для затруднения анализа и понимания кода как автоматизированными средствами, так и человеком.

VI. Заключение и Будущие направления

В ходе выполнения данной работы был успешно разработан и апробирован инструментарий для автоматизации метаморфизма кода на языке Go, использующий возможности современных больших языковых моделей. Основной целью являлось исследование применимости нейронных сетей для генерации семантически эквивалентных, но структурно измененных вариантов кода, реализующих техники вставки «мертвого» кода и модификации потока управления.

Ключевым результатом работы является демонстрация принципиальной возможности автоматизации метаморфизма с помощью LLM. Созданный прототип инструментария с модульной архитектурой способен итеративно модифицировать Go-код, делегируя генерацию преобразований внешним API нейронных сетей. Проведенные эксперименты с тремя различными моделями LLM показали, что они способны генерировать код, который в большинстве случаев успешно компилируется и сохраняет исходную функциональность, особенно при использовании техники вставки «мертвого» кода.

Анализ метрик обфускации (LOC_{fin}, CC_{fin}, $CogC_{fin}$) количественно подтвердил, что применение разработанного подхода приводит к значительному усложнению программного кода. Особенно впечатляющие результаты показала модель gemini-2.5-flash-preview-04-17, продемонстрировав как наилучшую функциональную эквивалентность при модификации потока управления (100%), так и максимальные показатели усложнения по метрикам когнитивной сложности. Относительный прирост цикломатической и когнитивной сложности свидетельствует о потенциале метода для затруднения анализа и понимания кода как автоматизированными средствами, так и человеком.

Вклад данной работы заключается в предложении нового подхода к автоматизации метаморфизма, основанного на использовании LLM и применимого к исходному коду Go. Проведенное сравнение различных LLM в контексте этой специфической задачи также представляет практический и научный интерес.

Несмотря на достигнутые результаты, выявленные ограничения определяют направления для дальнейших исследований. Основными вызовами остаются обеспечение стопроцентной гарантии сохранения функциональной эквивалентности при сложных преобразованиях потока управления, а также разработка более точных и комплексных метрик для оценки реальной стойкости обфусцированного кода к инструментам реверс-инжиниринга.

Перспективные направления дальнейших исследований включают:

- Расширение набора реализуемых техник метаморфизма.
- Исследование и оптимизация методов взаимодействия с LLM, в том числе fine-tuning специализированных моделей для задач преобразования кода.
- Интеграция механизмов обратной связи для адаптивного управления процессом метаморфизма.
- Разработка более совершенных метрик оценки стойкости обфускации к современным инструментам анализа.
- Тестирование инструментария на более крупных и сложных реальных проектах на Go для оценки практической применимости и масштабируемости.

Развитие предложенного подхода имеет потенциал для создания нового поколения интеллектуальных инструментов защиты программного обеспечения.

Библиография

[1] C. Collberg, C. Thomborson и D. Low, «A Taxonomy of Obfuscating Transformations,» Department of

Computer Science, University of Auckland, Technical Report 148, 1997.

[2] P. Szor, «Art of Computer Virus Research and Defense,» в *Proceedings of the 11th European Institute for Computer Antivirus Research Conference (EICAR)*, 2005.

[3] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik и E. Weippl, «Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?» *ACM Computing Surveys (CSUR)*, 2016.

[4] M. Chen и др., «Evaluating Large Language Models Trained on Code,» *arXiv preprint arXiv:2107.03374*, 2021.

[5] A. Sharma и S. K. Sahay, «Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey,» *International Journal of Computer Applications (IJCA)*, т. 90, № 2, с. 7—11, март 2014.

[6] K. Brezinski и K. Ferens. *Metamorphic Malware and Obfuscation – A Survey of Techniques, Variants and Generation Kits*. Preprint. (окт. 2021).

[7] W. Wong и M. Stamp, «Hunting for metamorphic engines,» *Journal in Computer Virology*, т. 2, № 3, с. 211—229, 2006.

[8] M. Campion, M. Dalla Preda и R. Giacobazzi, «Learning metamorphic malware signatures from samples,» *Journal of Computer Virology and Hacking Techniques*, т. 17, с. 167—183, 2021.

[9] T. J. McCabe, «A Complexity Measure,» *IEEE Transactions on Software Engineering*, т. SE-2, № 4, с. 308—320, 1976.

[10] SonarSource, *Cognitive Complexity: A new way of measuring understandability*, SonarSource Blog, URL: https : / / www . sonarsource . com / docs / CognitiveComplexity.pdf, 2023.

[11] *MetamorphLLM Prompt Dead Code Insertion technique*, GitHub, URL: https : / / gist . github . com / Hekzory / 12e4357763be058a6e17a496c8330e9d, 2025.

[12] *MetamorphLLM Prompt Control Flow Modification technique*, GitHub, URL: https : / / gist . github . com / Hekzory/4264e30175938c2b4c3fc4cdf24f4274, 2025.

[13] *MetamorphLLM Prompt Repository combination of Dead Code Insertion AND Control Flow Modification*, GitHub, URL: https : / / gist . github . com / Hekzory / d97f7249658784af3782ac28cfaa1b29, 2025.

[14] *MetamorphLLM Project Repository*, GitHub, URL: https : / / github . com / Hekzory / MetamorphLLM / blob / master/internal/rewriter/rewriter.go, 2025.

Automating Go Code Metamorphism Using Large Language Models

David Muradyan, Oleg Tsvetkov

***Abstract*—Code metamorphism is a powerful technique for protecting software from reverse engineering, but its manual implementation is extremely labor-intensive. This paper presents a toolkit for automating the code metamorphism process for the Go language using large language models. The toolkit implements a modular architecture and applies two key techniques: dead code insertion and control flow modification, delegating the generation of transformations to external LLMs. An experimental evaluation was conducted using three modern LLMs (gemini-2.5-flash-preview-04-17, deepseek-chat-v3-0324, gemini-2.0-flash). Effectiveness was assessed using metrics of functional equivalence, change in lines of code, cyclomatic complexity, and cognitive complexity. The results demonstrate the feasibility of automating metamorphism with LLMs, a significant increase in code complexity according to CC and CogC metrics, but reveal issues with the stability of maintaining functional equivalence during control flow modification for some models. The work shows the promise of using LLMs for code obfuscation tasks but highlights the need for further research to improve reliability and assess the real-world resilience of the code.**

***Keywords*—code metamorphism, code obfuscation, large language models, LLM, Go, automation, software protection, code complexity, cyclomatic complexity, cognitive complexity**

References

[1] C. Collberg, C. Thomborson, and D. Low, «A taxonomy of obfuscating transformations», Department of Computer Science, University of Auckland, Technical Report 148, 1997.

[2] P. Szor, «Art of computer virus research and defense», in *Proceedings of the 11th European Institute for Computer Antivirus Research Conference (EICAR)*, 2005.

[3] A. Sharma and S. K. Sahay, «Evolution and detection of polymorphic and metamorphic malwares: A survey», *International Journal of Computer Applications (IJCA)*, vol. 90, no. 2, pp. 7–11, Mar. 2014.

[4] K. Brezinski and K. Ferens. *Metamorphic malware and obfuscation – a survey of techniques, variants and generation kits*. Preprint. (Oct. 2021).

[5] W. Wong and M. Stamp, «Hunting for metamorphic engines», *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006.

[6] M. Campion, M. Dalla Preda, and R. Giacobazzi, «Learning metamorphic malware signatures from samples», *Journal of Computer Virology and Hacking Techniques*, vol. 17, pp. 167–183, 2021.

[7] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl, «Protecting software through obfuscation: Can it keep pace with progress in code analysis?», *ACM Computing Surveys (CSUR)*, 2016.

[8] M. Chen *et al.*, «Evaluating large language models trained on code», *arXiv preprint arXiv:2107.03374*, 2021.

[9] T. J. McCabe, «A complexity measure», *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[10] SonarSource, *Cognitive complexity: A new way of measuring understandability*, SonarSource Blog, URL: https://www.sonarsource.com/docs/CognitiveComplexity.pdf, 2023.

[11] *Metamorphllm project repository*, GitHub, URL: https://github.com/Hekzory/MetamorphLLM/blob/master/internal/rewriter/rewriter.go, 2025.

[12] *Metamorphllm prompt dead code insertion technique*, GitHub, URL: https://gist.github.com/Hekzory/12e4357763be058a6e17a496c8330e9d, 2025.

[13] *Metamorphllm prompt control flow modification technique*, GitHub, URL: https://gist.github.com/Hekzory/4264e30175938c2b4c3fc4cdf24f4274, 2025.

[14] *Metamorphllm prompt repository combination of dead code insertion and control flow modification*, GitHub, URL: https://gist.github.com/Hekzory/d97f7249658784af3782ac28cfaa1b29, 2025.