ГЛАВА 1

Основы

Итак, мы начинаем! Я очень рад, что вы читаете эту книгу, и надеюсь, что она будет вам полезна. Такие ИИ-инструменты, как ChatGPT и ИИ-агенты, уже изменили и продолжают менять процесс нашего взаимодействия с приложениями. В свою очередь, GitHub Copilot и подобные ему инструменты уже изменили и продолжают менять процесс создания приложений программистами. GitHub Copilot умеет извлекать контекст из существующего кода или промптов на естественном языке, поэтому предоставляет более богатые и более мощные возможности генерации кода, чем другие приложения, созданные до него.

С этой книгой я помогу вам понять, как использовать функциональность GitHub Copilot для решения таких задач, как генерирование и автозавершение кода, автоматическая генерация тестов, преобразование и объяснение кода, работа с репозиториями, pull-запросами и задачами (issues), возникающими непосредственно в GitHub. Мы рассмотрим примеры использования Copilot в разных предметных областях и языках программирования. Вы научитесь эффективно применять его и писать промпты, позволяющие получать наилучший результат. Вы узнаете, как добавить в Copilot пользовательскую специализированную функциональность. А еще поймете, как он делает то, что делает, и почему вы иногда не получаете желаемый результат и как быть в таких ситуациях.

Copilot как общее название

В самом начале обсуждения необходимо сказать, что Copilot — популярное название для ИИ-приложений, которые собирают информацию, формулируют промпты и возвращают ответы и рекомендации. Например, у Microsoft есть система Office 365 Copilot (https://oreil.ly/DaSLT), обеспечивающая ту же функциональность для приложений Microsoft Office. Она анализирует контекст из Word, Teams, Outlook и так далее и предоставляет сводки, варианты ответов и предлагает другие полезные функции для взаимодействия.

В этой книге вместо полного названия GitHub Copilot я буду использовать просто Copilot, за исключением мест, где будут упоминаться другие системы, в названии которых тоже есть это слово. В таких случаях я буду указывать их полные официальные названия — например, Office 365 Copilot.

Однако прежде, чем начинать погружаться в тему, стоит ознакомиться с базовой информацией. Именно она и представлена в текущей главе. Мы начнем с общего описания GitHub Copilot. Затем рассмотрим базовую технологию, лежащую в его основе, общий принцип работы GitHub Copilot, особенности работы с ним и то, чем он отличается от таких инструментов, как ChatGPT. Кроме того, мы поговорим о том, где его взять и как установить.

Что такое GitHub Copilot

GitHub Copilot — облачная генеративная модель ИИ. Что же означают эти слова? В общем смысле ИИ можно определить как *компьютеры, решающие задачи, которые, как считалось ранее, могут решать только люди, поскольку имеют необходимые навыки и умеют рассуждать*. Кроме того, в последнее время у машин появилась возможность взаимодействовать с людьми так, как это делали бы сами люди, — благодаря использованию таких средств, как обработка естественного языка (Natural Language Processing, NLP), чат-интерфейсы и автоматизированная обработка и принятие решений с помощью ИИ-агентов.

СТОИТ ЛИ ОПАСАТЬСЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Когда речь заходит об ИИ, люди невольно мыслят образами поп-культуры: машины становятся разумными, обретают сознание и захватывают мир. Многие из нас, технических специалистов, понимают, что это нереально. Тем не менее программисты, пользующиеся такими инструментами, как GitHub Copilot, могут вообразить, что ИИ (в области программирования) может лишить их работы.

Безусловно, использование искусственного интеллекта уже привело к перестановкам на рабочих местах. И все же Copilot создавался не для этого. При всей кажущейся разумности, которую Copilot может демонстрировать в некоторых случаях, он может генерировать некорректный или неработоспособный код. И он по-прежнему нуждается в контроле со стороны человека. Я надеюсь, что благодаря этой книге Copilot займет почетное место в вашем инструментарии и поможет вам в работе над проектами и при выполнении ваших профессиональных обязанностей.

Слово *«облачный»* отсылает к способу, которым Copilot возвращает рекомендации и генерирует ответы. Речь идет об облачной среде под управлением GitHub, которая упрощает взаимодействие с ИИ-моделями. Определение *«генеративный»* отображает способность ИИ *генерировать* новые результаты на основе получен-

ного контекста. Copilot может выдавать ответы и предложения для разработки, опираясь на контекст и подсказки из среды пользователя. Насколько хорошо он это делает? Зависит от нескольких факторов, о которых мы поговорим далее. В качестве короткого примера на рис. 1.1 показано, как Copilot предлагает оптимизации для проекта в Visual Studio (VS) Code.

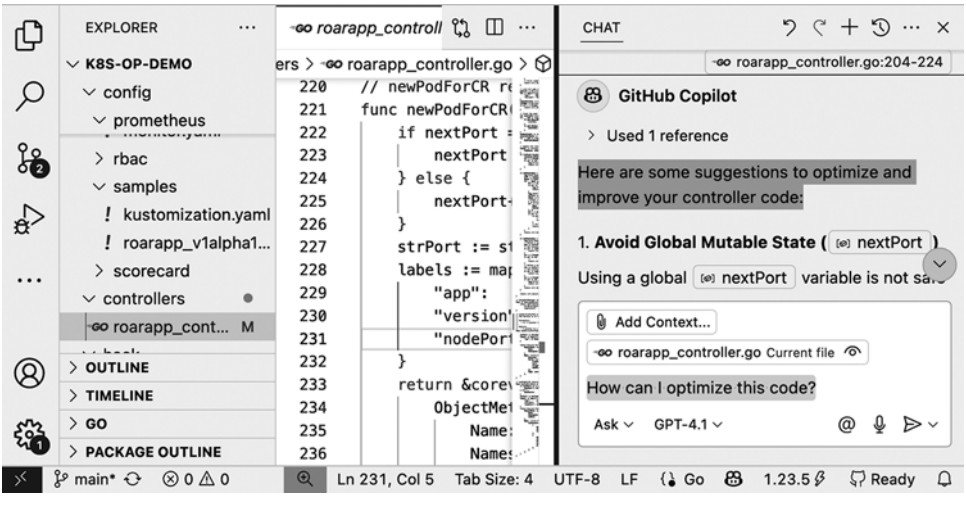

Рис. 1.1. Использование Copilot с проектом в VS Code

IDE по умолчанию

Описать все интегрированные среды разработки (IDE) в одной книге невозможно, поэтому в примерах, где фигурирует среда, будут использоваться VS Code или GitHub Codespace. Если вы работаете в другой IDE, то найдите в документации Copilot описание различий в использовании, элементах управления и т. д.

Как работает Copilot

Чтобы понять, как работает Copilot, необходимо иметь представление о некоторых компонентах, лежащих в основе функциональности его самого и других похожих ИИ-инструментов. В частности, эти компоненты определяют источник информации для формирования ответов Copilot и общий принцип его работы.

Copilot и многие ИИ-приложения получают свои данные от больших языковых моделей (large language models, LLM).

Большие языковые модели

LLM (без упоминания которых в наши дни не обходится ни одно обсуждение ИИ) — модели на базе искусственного интеллекта, обученные на огромных объемах существующих данных предсказывать следующие слова или другие виды контента (например, *токенов*), соответствующие заданным входным данным *(промпту)*.

От традиционных вычислительных моделей LLM отличаются способностью обрабатывать отформатированные данные или выдавать ответы на математические задачи, поскольку LLM учатся понимать контекст, синтаксис и структуру. Для этого они используют алгоритмы, которые анализируют огромное количество параметров, чтобы определить, какие слова или токены[1] лучше всего поставить следующими, исходя из их смысла. Таким образом, модели статистически формируют ответы, основываясь не только на входных данных, но и на форме их представления.

Прогностические способности модели изучаются и настраиваются на основании обширных наборов существующего контента. В процессе обучения модели ищут взаимосвязи между разными видами информации в любой заданной предметной области. С технической точки зрения, получив последовательность токенов в виде запроса, LLM способны анализировать синтаксис и структуру ввода, вычислять контекст на основе обучающих данных и прогнозировать последовательность токенов, которая с наибольшей вероятностью должна идти далее. Проще говоря, LLM, исходя из всех «переваренных» ими данных, умеют определять, что будет звучать естественно, и продолжить диалог.

> **НЕ ТОЛЬКО LLM**
>
> Помимо *традиционных* LLM, существуют и другие ИИ-модели — например, малые языковые модели (small language models, SLM) и модели логического вывода. Copilot постоянно обновляется, чтобы он мог использовать различные модели с разными возможностями и видами обучения, но для простоты я буду употреблять выражение «большие языковые модели».

Чтобы лучше понять, чем обучение LLM отличается от традиционной обработки, представьте, что изучаете новый язык, а потом общаетесь с человеком, для которого этот язык является родным. Вы можете заранее полистать словарь, запомнить отдельные слова и фразы. Но в общении с носителем языка необходимо также

[1] «Токен» — технический термин для элементарной единицы текста в модели (часто слово, подслово или символ).

понимать контекст того, что он говорит. Это поможет выбрать правильные слова или фразы для ответов. Кроме того, необходимо уметь сформулировать ответ в контексте, который будет осмысленным для собеседника. Язык имеет синтаксис и структуру, но они обретают смысл именно благодаря контексту общения. LLM могут определять контекст по входным данным и передавать соответствующий контекст в своих ответах.

Вернемся к Copilot. По умолчанию в нем используются LLM, разработанные и управляемые OpenAI (https://openai.com) — компанией, создавшей ChatGPT (https://chat.openai.com). В сотрудничестве с OpenAI компания GitHub разрабатывала Copilot в течение нескольких лет. Недавно были добавлены возможности использования других семейств моделей, включая модели Claude от Anthropic и Google Gemini. Во всех моделях общей характеристикой Copilot является ориентация на создание программного обеспечения, а языком общения — программный код.

Код и генеративный ИИ

Языки программирования по определению имеют конкретные обязательные синтаксические и семантические правила, в каждом языке разные. Когда вы пишете код на Python (а не на Go, к примеру), для создания программы используете разные токены и структуры. Тем не менее они подчиняются определенным правилам. Эти токены и структуры четко определены и образуют ограниченный набор. Copilot же предназначен для того, чтобы предоставлять рекомендации по программированию и сопутствующую информацию, которая соответствует синтаксису и структуре. Однако основное преимущество Copilot заключается в том, что он выдает ответы, имеющие отношение к тому, над чем работает программист или о чем спрашивает в промпте. Контекст, который передается в Copilot, может поступать из нескольких источников, включая следующие:

- код, написанный в редакторе в среде разработки;
- взаимодействие с моделью с помощью промптов на естественном языке (также известных как *чат-модели*);
- типичные операции разработки непосредственно в GitHub (например, pull-запросы).

Все эти варианты взаимодействия будут рассмотрены в книге. Независимо от интерфейса контекст (код, инструкции (directives), вводимые данные или элементы GitHub, с которыми вы работаете) преобразуются в *промпт* — вашу реплику в диалоге, на которую должен отреагировать ИИ. Обработка входных данных и формирование ответа на основании контекста, обучения и возможностей модели — *генеративная* часть работы Copilot как *генеративного ИИ*. Ответом могут быть предлагаемый код, ответ на вопрос или пошаговые инструкции. Это то, что, по мнению ИИ, завершает код, соответствует промпту или отвечает на запрос.

ЧАТ-МОДЕЛИ

Как мы чаще всего представляем себе программирование? Программист сидит перед терминалом и вводит код в IDE. Но прежде чем вводить код, он часто должен получить ответы на какие-то вопросы или что-то понять. Например, при изучении нового языка программист может сомневаться в том, как реализовать некую структуру данных или управляющую конструкцию. А может, он захочет узнать, как открывать файлы или выполнять операции чтения/записи в базе данных. А затем ему нужно будет разобраться в различиях версий сторонней платформы или выяснить, не устарела ли та или иная функциональность. Обычно такие сведения вы узнавали у коллег или искали на Stack Overflow. Но если вместо этого вы пользуетесь ИИ, то должны уметь задать ему вопросы или давать указания (то есть иметь возможность *вести диалог*).

Типичные *общедоступные* реализации генеративного ИИ (такие как ChatGPT) стали очень популярными потому, что могут общаться и отвечать на вопросы понятным для человека образом. В Copilot это возможно благодаря *Copilot Chat*. В чат-интерфейсе можно дать Copilot указание выполнить некоторые операции (например, «Сгенерируй юнит-тесты для кода») или задать интересующие вас вопросы (допустим, «Какая версия Kubernetes является текущей?» и даже «Что делает этот код?»). Формирование ответов на основе инструкций или вопросов в чат-интерфейсе — еще один пример *генеративной* составляющей *генеративного* ИИ.

В качестве аналогии представьте, что описываете в Zoom симптомы доктору, чтобы он попытался поставить диагноз, опираясь на свой многолетний опыт. Или что по телефону описываете автомеханику проблему с вашей машиной, чтобы он мог предложить решение. В таких случаях успех сильно зависит от ваших навыков коммуникации и контекста, который вы предоставляете, а также от квалификации профессионала, к которому вы обратились. В результате общения вы можете получить советы по решению проблемы (и некоторые из них могут оказаться неактуальными) или узнать, что профессионалу не хватает контекста или понимания, чтобы помочь вам. Генеративный ИИ работает по тому же принципу.

Когда Copilot дает вам рекомендации, пока вы пишете код, элементы среды разработки формируют контекст, в котором будет создаваться промпт для модели ИИ. Copilot может генерировать предложения, которые соответствуют стилю программирования, используемому в файлах вашего проекта. Это может быть как хорошо, так и плохо.

Предложения Copilot часто соответствуют стилю программирования реальных пользователей, и это может быть хорошо, если эти стили соответствуют лучшим практикам, и плохо, если Copilot слепо копирует неудачные практики из кодовой

базы. В последнем случае вы можете привыкнуть использовать ограниченные и неэффективные способы программирования либо же вам придется изучать более обширный набор рекомендаций, чтобы найти подходящий вариант. На ответы Copilot влияют количество и качество примеров, которые он сможет извлечь из вашей среды, а также степень его обученности.

Таким образом, генеративный ИИ, используемый в Copilot, может оказаться очень полезным. Но следует помнить, что на его рекомендации по написанию кода будет влиять контекст, с которым приходится работать Copilot, и данные, которые он использует для обучения. Вы можете обратить этот факт себе на пользу, предоставляя Copilot больше примеров стиля предложений, которые он должен выдавать. Но такая зависимость может стать недостатком, если контекст, доступный Copilot, ограничен. В дальнейших главах мы обсудим, как помочь Copilot выдавать наилучшие результаты.

Далее мы кратко рассмотрим общую схему работы Copilot, чтобы вы могли лучше понять, как он взаимодействует с контекстом при формировании ответа.

Общая схема работы Copilot

Чтобы понять принцип работы Copilot, можно рассмотреть основной процесс с точки зрения работы в одной из поддерживаемых IDE. В настоящее время в список поддерживаемых сред разработки для Copilot входят Visual Studio, VS Code, NeoVim, все представители семейства JetBrains и другие среды. Помимо этого, Copilot работает и в некоторых других средах, таких как в GitHub Codespaces (https://oreil.ly/QyEks).

Кроме того, GitHub выпустил пакет Copilot Language Server SDK (https://oreil.ly/6rDHq), с помощью которого GitHub Copilot можно интегрировать в любой редактор или IDE. А значит, в будущем можно ожидать появления новых приложений, интегрированных с Copilot.

Codespaces

GitHub Codespaces использует виртуальные машины (virtual machines, VM), работающие на облачной платформе, которые предоставляют полнофункциональную и настраиваемую среду разработки для пользователей GitHub. Интерфейс Codespaces напоминает интерфейс VS Code; эта среда может дополняться теми же расширениями, которые используются в VS Code, через браузер или путем подключения к разным IDE. GitHub предоставляет CodeSpaces как дополнительный сервис.

Когда вы пишете код и запрашиваете у Copilot варианты завершения, он автоматически сканирует различную информацию, чтобы получить контекст того, над чем вы работаете.

Текущий файл

Текущий файл, который редактируется пользователем, — ключевой источник контекста для Copilot.

Имя текущего активного файла

По описательному имени (например, `TestConfig.go`) Copilot может понять, что должен делать код.

Контент до и после текущей позиции курсора

Copilot может учитывать контекст из кода и комментариев, расположенных непосредственно до и после текущей позиции курсора в файле. Эта информация поможет Copilot решить, что именно нужно вставить, и/или понять, какие части кода отсутствуют.

Комментарии

Как и программисты-напарники или рецензенты кода, Copilot может по комментариям понять, что делает уже существующий код и что должен делать еще не написанный. Добавление комментариев — один из основных способов предоставления контекста для Copilot: чем точнее и подробнее комментарии, тем выше вероятность того, что предложенный им код будет актуальным.

Другие открытые в редакторе файлы

Copilot использует код, содержащийся в любых открытых файлах, как контекст. Это ключевой момент для сбора информации о текущей задаче и дополнения информации в модели. Так, одна из стратегий работы с устаревшей функциональностью LLM — открытие в редакторе файла с примером замены неактуальной функции. По этому примеру Copilot сможет понять, какие альтернативы будут предпочтительными, вместо того чтобы полагаться на устаревшие подходы, использовавшиеся при обучении LLM.

Локальный индекс

Copilot автоматически анализирует большинство файлов в проекте, открытом в таких IDE, как VS Code, и создает расширенный локальный индекс для проекта.

ИНДЕКСИРОВАНИЕ

В контексте Copilot *индексированием* называется процесс сканирования и организации кодовой базы для создания структуры данных, обеспечивающей быстрый и точный поиск. Наличие индекса позволяет Copilot предоставлять более точные и контекстно зависимые предложения и ответы по коду.

Локальный индекс, создаваемый и обновляемый автоматически при внесении изменений, хранится на машине пользователя. При использовании в GitHub (см. главу 9) Copilot создает удаленный индекс, связанный с репозиторием.

Если вы используете чат-интерфейс, то обычно он заранее заполняется контекстом из файла, выделенного фрагмента или команды терминала — в зависимости от того, над чем вы работали в последнее время. Впрочем, этот контекст можно изменить перед отправкой промпта (подробнее об этом поговорим в главе 3).

Таким образом, когда вы используете один из интерфейсов с установленным и активизированным Copilot, он собирает информацию из ряда источников по мере того, как вы вводите код (рис. 1.2).

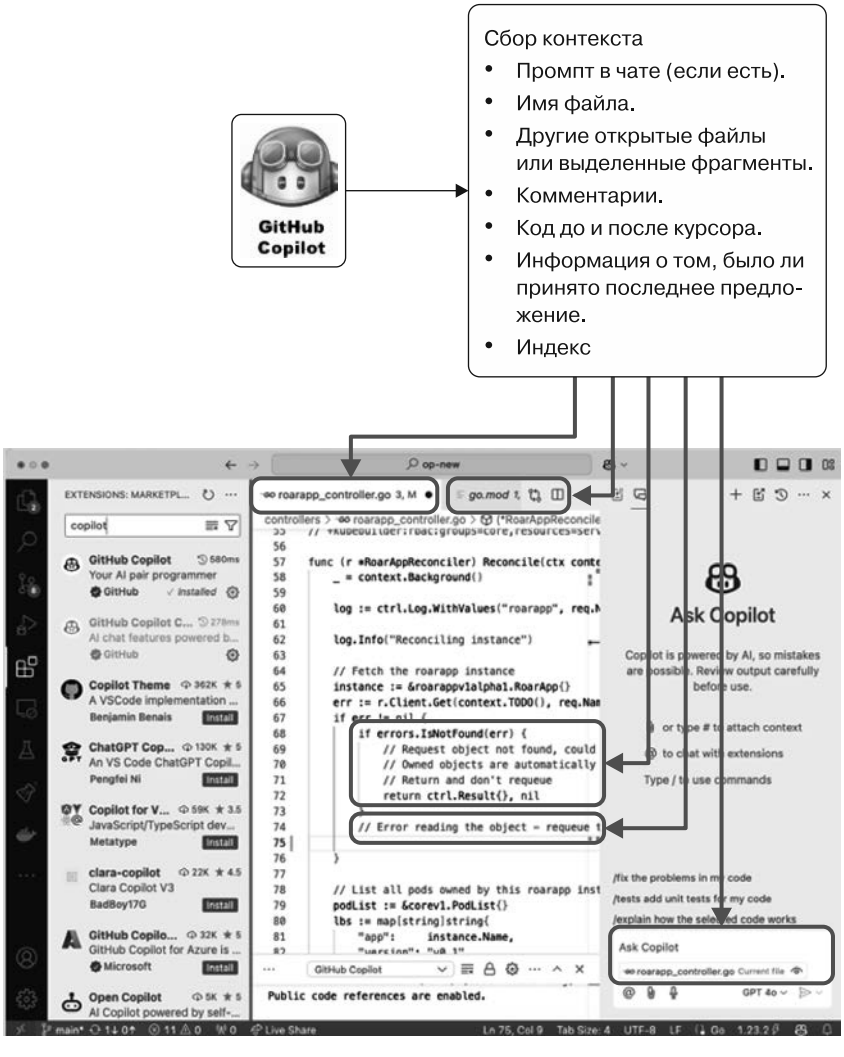

Рис. 1.2. Сбор контекста из среды IDE

Контекст обрабатывается и в конечном счете отправляется на GitHub, где синтезируется промпт (рис. 1.3).

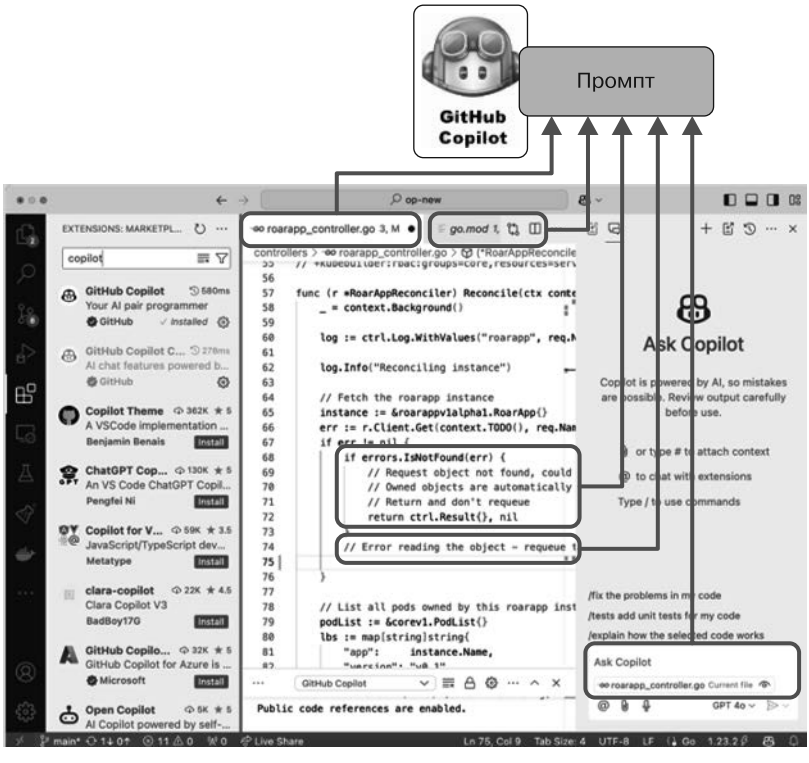

Рис. 1.3. Синтезирование промпта

Затем промпт передается через GitHub большой языковой модели, которая возвращает возможные завершения или ответы. После того как по промпту будут сгенерированы результаты, системы Copilot на GitHub выполняют дополнительную обработку результата (подробнее об этом поговорим позднее). Далее ответ возвращается в IDE, где вы можете оценить его и решить, как действовать дальше. Схема завершения кода показана на рис. 1.4.

Процесс продолжается и повторяется при дальнейшем взаимодействии пользователя с Copilot. Таким образом, Copilot действует как ассистент, помогающий вам разобраться с задачей, над которой вы работаете, будь то написание шаблонного кода, поиск сложного алгоритма, генерация данных или запросов, написание модульных тестов или изучение возможностей нового языка программирования.

Более подробно о том, как это происходит, мы поговорим в последующих главах, когда будем обсуждать работу с Copilot. А пока рассмотрим нюансы его использования, которые помогут вам в разработке программного обеспечения.

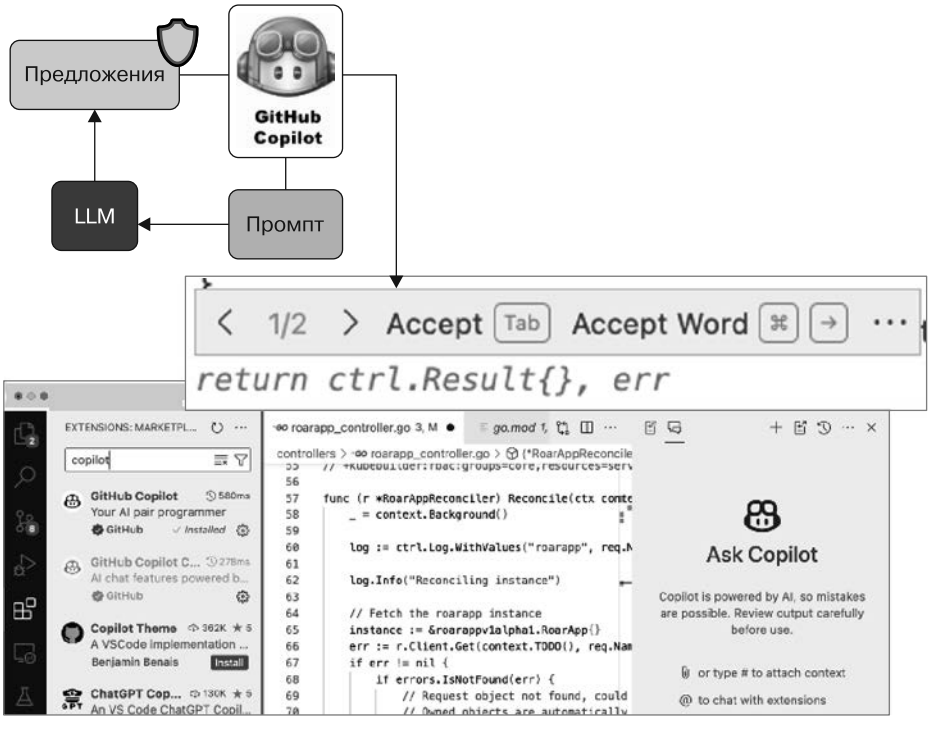

Рис. 1.4. Схема процесса от получения промпта до выдачи ответов

Особенности использования Copilot

Итак, теперь, когда вы знаете, что такое Copilot и как он работает, необходимо озвучить ключевой момент: Copilot, как и любой ИИ, может давать неверные или неполные ответы. Он может ошибаться или выдавать неожиданные результаты. Такое бывает нечасто, но при работе с ним нужно помнить о некоторых важных критериях. Это:

- актуальность;
- релевантность;
- полнота;
- точность;
- конфиденциальность;
- безопасность.

О них мы и поговорим в текущем разделе. Кстати говоря, эти критерии играют важную роль при взаимодействии не только с Copilot, но и с любым другим современным ИИ-инструментом, помогающим решать те или иные задачи.