

Large Language Models for Unit Testing: A Systematic Literature Review

面向单元测试的大型语言模型：一项系统性文献综述

Unit testing is a fundamental practice in modern software engineering, with the aim of ensuring the correctness, maintainability, and reliability of individual software components.

单元测试是现代软件工程中的一项基本实践，旨在确保单个软件组件的正确性、可维护性和可靠性。

Very recently, with the advances in Large Language Models (LLMs), a rapidly growing body of research has leveraged LLMs to automate various unit testing tasks, demonstrating remarkable performance and significantly reducing manual effort.

最近，随着大型语言模型（LLMs）的进步，大量迅速增长的研究开始利用 LLM 来自动化各种单元测试任务，展现出了卓越的性能并显著减少了人工投入。

However, due to ongoing explorations in the LLM-based unit testing field, it is challenging for researchers to understand existing achievements, open challenges, and future opportunities.

然而，由于基于 LLM 的单元测试领域仍处于不断的探索中，研究人员很难全面了解现有的成就、未解决的挑战以及未来的机遇。

This paper presents the first systematic literature review on the application of LLMs in unit testing until March 2025.

本文对截至 2025 年 3 月 LLM 在单元测试中的应用进行了首次系统性文献综述。

We analyze 105 relevant papers from the perspectives of both unit testing and LLMs.

我们从单元测试和 LLM 两个视角分析了 105 篇相关论文。

We first categorize existing unit testing tasks that benefit from LLMs, e.g., test generation and oracle generation.

我们首先对受益于 LLM 的现有单元测试任务进行了分类，例如测试生成和预言（Oracle）生成。

We then discuss several critical aspects of integrating LLMs into unit testing research, including model usage, adaptation strategies, and hybrid approaches.

接着，我们讨论了将 LLM 整合到单元测试研究中的几个关键方面，包括模型使用、适配策略和混合方法。

We further summarize key challenges that remain unresolved and outline promising directions to guide future research in this area.

我们进一步总结了尚未解决的关键挑战，并概述了有前景的方向，以指导该领域的未来研究。

Overall, our paper provides a systematic overview of the research landscape to the unit testing community, helping researchers gain a comprehensive understanding of achievements and promote future research.

总体而言，我们的论文为单元测试社区提供了研究概况的系统性综述，帮助研究人员全面了解现有成就并推动未来的研究。

Our artifacts are publicly available at the GitHub repository: <https://github.com/iSEngLab/AwesomeLLM4UT>.

我们的相关资料 (Artifacts) 已在 GitHub 仓库公开: <https://github.com/iSEngLab/AwesomeLLM4UT>。

1 INTRODUCTION

1 引言

Unit Testing (UT) aims to validate the correctness of individual components within a software system, and plays a crucial role in the software development and testing lifecycle [26, 27].

单元测试 (UT) 旨在验证软件系统中各个组件的正确性, 并在软件开发和测试生命周期中发挥着至关重要的作用 [26, 27]。

Nowadays, as modern software continues to evolve and support various critical industries, unit testing has become a standardized and even mandatory practice, forming the cornerstone of software quality and reliability [58].

如今, 随着现代软件不断演进并支持各种关键行业, 单元测试已成为一种标准化甚至强制性的实践, 构成了软件质量和可靠性的基石 [58]。

However, conducting unit testing manually is often time-consuming and labor-intensive for developers and testing professionals.

然而, 对于开发人员和测试专业人员来说, 手动进行单元测试往往既耗时又费力。

For example, prior work [16] has shown that developers typically spend more than 15% of their time writing unit tests.

例如, 先前的研究 [16] 表明, 开发人员通常花费超过 15% 的时间来编写单元测试。

To alleviate this burden, researchers have devoted considerable efforts to automating the unit testing process, such as test case and assertion generation [7, 25].

为了减轻这一负担, 研究人员投入了大量精力来自动化单元测试过程, 例如测试用例和断言生成 [7, 25]。

Thus, unit testing has long been an active research topic, extensively studied over the past decades and continuously attracting interest from both academia and industry [16, 82, 104, 123].

因此, 单元测试长期以来一直是一个活跃的研究课题, 在过去的几十年中得到了广泛的研究, 并持续吸引着学术界和工业界的兴趣 [16, 82, 104, 123]。

Recently, Large Language Models (LLMs) have been increasingly applied in Software Engineering (SE), fundamentally reshaping the research paradigm in the field [40, 104, 128].

最近, 大型语言模型 (LLMs) 越来越多地应用于软件工程 (SE) 中, 从根本上重塑了该领域的研究范式 [40, 104, 128]。

Built on the Transformer architecture, these models are typically pre-trained on large-scale unlabeled corpora to acquire generic language knowledge [101].

建立在 Transformer 架构之上, 这些模型通常在通过大规模无标签语料库上进行预训练, 以获取通用的语言知识 [101]。

Benefiting from their advanced model architecture, vast parameters and extensive training datasets, LLMs have demonstrated remarkable progress in various software development and testing tasks, including code generation [52, 105, 107] and program repair [127, 129, 130].

得益于其先进的模型架构、巨大的参数量和广泛的训练数据集，LLM 在各种软件开发和测试任务中取得了显著进展，包括代码生成 [52, 105, 107] 和程序修复 [127, 129, 130]。

In the domain of unit testing, the community has witnessed an explosion of studies equipped with LLMs [80, 86, 96, 109, 134], demonstrating notable advantages and indicating a promising future for further research.

在单元测试领域，社区见证了利用 LLM 进行研究的数量呈现爆炸式增长 [80, 86, 96, 109, 134]，展示了显著的优势，并预示了未来研究的光明前景。

However, due to the inherent complexity of unit testing and the rapid evolution of LLMs, integrating LLMs into unit testing workflows is a considerably complex undertaking, making it difficult for interested researchers to understand existing work.

然而，由于单元测试固有的复杂性和 LLM 的快速演进，将 LLM 整合到单元测试工作流程中是一项相当复杂的任务，这使得感兴趣的研究人员难以理解现有的工作。

For example, existing LLM-based unit testing studies span a wide range of research perspectives (e.g., empirical [96, 114], technical [30, 109] studies), testing scenarios (e.g., test generation [85, 135] and oracle generation [21, 134]), and model usage paradigms (e.g., fine-tuning [86, 131] and prompting engineering [64, 65]).

例如，现有的基于 LLM 的单元测试研究涵盖了广泛的研究视角（如实证研究 [96, 114]、技术研究 [30, 109]）、测试场景（如测试生成 [85, 135] 和预言生成 [21, 134]）以及模型使用范式（如微调 [86, 131] 和提示工程 [64, 65]）。

Despite the burgeoning interest and ongoing explorations in the field, the literature currently lacks a detailed and systematic review of the applications of LLMs in unit testing.

尽管该领域的兴趣日益浓厚且探索不断，但目前的文献缺乏对 LLM 在单元测试中应用的详细和系统的综述。

This gap makes it challenging for researchers to understand the relationship between LLMs and their use in unit testing, and conduct follow-up research.

这一空白使得研究人员难以理解 LLM 与其在单元测试中的使用之间的关系，也难以开展后续研究。

This Work. To bridge this gap, we present the first systematic literature review on the deployment of rapidly emerging LLM-based unit testing studies.

本项工作。为了弥补这一空白，我们就迅速涌现的基于 LLM 的单元测试研究的部署情况，提出了首个系统性文献综述。

Based on this, the community can gain a comprehensive understanding of existing LLM-based unit testing techniques, offering insights into their strengths, weaknesses, and research trends.

基于此，社区可以全面了解现有的基于 LLM 的单元测试技术，深入洞察其优势、劣势和研究趋势。

We collect 105 relevant papers and conduct a comprehensive analysis from both unit testing and LLMs perspectives.

我们收集了 105 篇相关论文，并从单元测试和 LLM 两个角度进行了全面分析。

From our analysis, we reveal crucial challenges and outline future research opportunities in the field.

通过分析，我们揭示了该领域的关键挑战，并概述了未来的研究机遇。

Overall, our work serves as a valuable resource for researchers and practitioners interested in navigating and advancing this rapidly developing area.

总的来说，对于有兴趣在这个快速发展的领域中探索和推进的研究人员和从业者来说，我们的工作是一份宝贵的资源。

Contributions. To sum up, this work makes the following contributions:

贡献。 综上所述，本文做出了以下贡献：

- **Survey Methodology. We conduct the first systematic literature review of 105 high-quality APR papers from 2020 to 2025 that utilize recent LLMs to tackle unit testing challenges.**

综述方法。我们对 2020 年至 2025 年间利用近期 LLM 解决单元测试挑战的 105 篇高质量论文进行了首次系统性文献综述。

(注：原文此处写的是 "APR papers" 即自动程序修复论文，但结合上下文这很可能是作者的笔误，实际指代的是 Unit Testing papers 单元测试论文)

- **Trend Analysis. We perform a detailed analysis of selected studies in terms of publication trends and distribution of publication venues.**

趋势分析。我们就出版趋势和出版场所的分布对选定的研究进行了详细分析。

- **UT Perspective. We conduct a comprehensive analysis from the perspective of unit testing to understand the distribution of unit testing tasks with LLMs and provide an in-depth discussion about how these tasks are solved with LLMs.**

单元测试视角。我们从单元测试的角度进行了全面分析，以了解涉及 LLM 的单元测试任务的分布，并深入讨论了如何利用 LLM 解决这些任务。

- **LLMs Perspective. We conduct a comprehensive analysis from the perspective of LLMs to uncover the commonly-used LLMs, the types of prompt engineering, the input of the LLMs, as well as the accompanying techniques with these LLMs.**

LLM 视角。我们从 LLM 的角度进行了全面分析，以揭示常用的 LLM、提示工程的类型、LLM 的输入以及与这些 LLM 配合使用的技术。

- **Challenges and Opportunities. We highlight some crucial challenges of applying LLMs in the unit testing field and pinpoint promising directions for future research.**

挑战与机遇。我们重点介绍了将 LLM 应用于单元测试领域的一些关键挑战，并指出了未来研究的前景的方向。

Paper Organization. Figure 1 summarizes the structure of this survey.

论文组织结构。图 1 总结了本综述的结构。

The remainder of this paper is organized as follows.

本文的其余部分组织如下。

Section 2 introduces some basic concepts about unit testing and LLMs.

第 2 节介绍了关于单元测试和 LLM 的一些基本概念。

Section 3 illustrates the survey methodology.

第 3 节阐述了综述方法论。

Section 4 and Section 5 conduct the analysis from the perspectives of unit testing and LLMs, respectively.

第 4 节和第 5 节分别从单元测试和 LLM 的角度进行了分析。

Section 6 discusses key challenges and research guidelines.

第 6 节讨论了关键挑战和研究指南。

Section 7 draws the conclusions.

第 7 节得出了结论。

2 BACKGROUND

2 背景

In this section, we introduce the core concepts relevant to this work, including LLMs, unit testing, and related surveys.

在本节中，我们将介绍与本研究相关的核心概念，包括 LLM、单元测试以及相关综述。

2.1 Large Language Models

2.1 大型语言模型

LLMs refer to a class of large-scale Transformer-based models that are pre-trained on massive textual corpora to understand and generate human-like text [12].

LLM 指的是一类基于 Transformer 的大规模模型，它们在海量文本语料库上进行预训练，以理解和生成类人文本 [12]。

To fully leverage the vast amount of unlabeled data, LLMs are typically trained using self-supervised learning objectives, such as masked language modeling [23], masked span prediction [108], and causal language modeling [67].

为了充分利用大量的无标签数据，LLM 通常使用自监督学习目标进行训练，例如掩码语言建模 [23]、掩码跨度预测 [108] 和因果语言建模 [67]。

LLMs are primarily built on the Transformer [101] architecture, which consists of an encoder for input representation and a decoder for output generation.

LLM 主要构建在 Transformer [101] 架构之上，该架构包含用于输入表示的编码器和用于输出生成的解码器。

Based on architecture design, LLMs can be categorized into three types: (1) encoder-only models (e.g., BERT [20] and CodeBERT [23]), designed for understanding tasks; (2) encoder-decoder models (e.g., T5 [79] and CodeT5 [108]), designed for translation tasks; and (3) decoder-only models (e.g., LLaMA [98] and CodeLLaMA[81]), designed for generation tasks.

根据架构设计，LLM 可分为三种类型：（1）仅编码器模型（如 BERT [20] 和 CodeBERT [23]），专为理解任务设计；（2）编码器-解码器模型（如 T5 [79] 和 CodeT5 [108]），专为翻译任务设计；以及（3）仅解码器模型（如 LLaMA [98] 和 CodeLLaMA [81]），专为生成任务设计。

In addition, based on their accessibility, LLMs can be categorized into black-box models (e.g., GPT-4 [68]), which are proprietary and closed-source, and open-source models (e.g., CodeLlama [81]), which provide public access to their architecture and weights.

此外，根据其可访问性，LLM 可分为黑盒模型（如 GPT-4 [68]），这类模型是专有且闭源的；以及开源模型（如 CodeLlama [81]），这类模型公开提供其架构和权重。

While commercial LLMs continue to dominate the top of the leaderboards, an increasing number of open-source models, such as CodeLlama [81] and DeepSeek-R1 [18], are emerging and demonstrating strong performance across a variety of tasks.

虽然商业 LLM 继续占据各大排行榜的榜首，但越来越多的开源模型，如 CodeLlama [81] 和 DeepSeek-R1 [18]，正在涌现并在各种任务中展现出强大的性能。

Owing to their advanced training mechanisms, model architectures and extensive training datasets, LLMs demonstrate impressive capabilities across a wide range of SE tasks [104, 128].

得益于其先进的训练机制、模型架构和广泛的训练数据集，LLM 在广泛的软件工程（SE）任务中展现了令人印象深刻的能力 [104, 128]。

In this work, we systematically investigate how LLMs have been applied to address the long-standing challenges of automating unit testing tasks.

在本研究中，我们系统地调查了 LLM 如何被应用于解决自动化单元测试任务中长期存在的挑战。

2.2 Unit Testing

2.2 单元测试

Software testing plays a crucial role in SE by evaluating and ensuring the correctness, reliability, and performance of software systems [104].

软件测试通过评估和确保软件系统的正确性、可靠性和性能，在软件工程中发挥着至关重要的作用 [104]。

Existing testing practices include unit testing, integration testing, system testing, and acceptance testing, each targeting different stages and levels in the validation process of software systems.

现有的测试实践包括单元测试、集成测试、系统测试和验收测试，每种测试针对软件系统验证过程中的不同阶段和层级。

Among them, unit testing is particularly important, as it serves as the foundation for detecting bugs at an early stage of development, thereby facilitating subsequent software development activities [58].

其中，单元测试尤为重要，因为它是开发早期发现缺陷的基础，从而促进了后续的软件开发活动 [58]。

With its long-established history, unit testing has garnered significant attention from both academia and industry, becoming an accepted and even mandatory practice in modern software engineering.

凭借其悠久的历史，单元测试赢得了学术界和工业界的极大关注，成为现代软件工程中公认甚至强制性的实践。

The primary objective of unit testing is to validate individual components or units of a program by isolating them and executing unit test cases.

单元测试的主要目标是通过隔离程序的各个组件或单元并执行单元测试用例来验证它们。

This practice helps ensure that each component functions as expected before being integrated with other components of the system.

这种做法有助于确保每个组件在与系统其他组件集成之前都能按预期运行。

As illustrated in Figure 2, given a focal method (i.e., the unit under test), its unit test typically consists of two components: (1) a test prefix, i.e., a sequence of statements that manipulate the unit under test to a specific state, and (2) a test oracle, i.e., an assertion that defines the expected behavior or condition to be satisfied in that state.

如图 2 所示，给定一个焦点方法（即被测单元），其单元测试通常由两部分组成：（1）测试前缀（Test Prefix），即一系列语句，将被测单元操作到特定状态；以及（2）测试预言（Test Oracle），即定义在该状态下应满足的预期行为或条件的断言。

In the literature, numerous approaches have been proposed to generate unit test cases automatically, including symbolic execution testing [11, 15], random testing [61, 73], and search-based testing [6, 25].

在文献中，已经提出了许多自动生成单元测试用例的方法，包括符号执行测试 [11, 15]、随机测试 [61, 73] 和基于搜索的测试 [6, 25]。

In addition, unit testing encompasses a variety of sub-tasks depending on the perspective, such as oracle generation [134] and test repair [116], each addressing specific challenges and scenarios.

此外，根据视角不同，单元测试包含多种子任务，如预言生成 [134] 和测试修复 [116]，每个子任务都解决特定的挑战和场景。

These tasks represent distinct aspects of unit testing across various levels of granularity and practical contexts, offering a comprehensive view of unit testing.

这些任务代表了不同粒度和实践背景下单元测试的不同方面，提供了单元测试的全面视图。

Given the complexity and significance of unit testing, this work focuses on how LLMs have been leveraged to support and automate various unit testing tasks.

鉴于单元测试的复杂性和重要性，本研究重点关注如何利用 LLM 来支持和自动化各种单元测试任务。

2.3 Related Surveys

2.3 相关综述

There exist several surveys or literature reviews on the general application of LLMs in the broader area of software engineering [22, 40, 104, 128, 136].

目前已有多篇关于 LLM 在更广泛的软件工程领域中一般应用的调查或文献综述 [22, 40, 104, 128, 136]。

Different from these studies targeting the whole software engineering/testing workflow, this work focuses specifically on the achievements of LLMs in the domain of unit testing, which remains relatively underexplored.

与这些针对整个软件工程/测试工作流程的研究不同，本研究专门关注 LLM 在单元测试领域的成就，该领域仍相对缺乏探索。

Moreover, unit testing and its related tasks have been widely surveyed in the past [16, 82, 123].

此外，单元测试及其相关任务在过去已被广泛调查过 [16, 82, 123]。

However, these studies primarily focus on traditional unit testing techniques and were conducted prior to 2014, before the emergence of modern LLMs.

然而，这些研究主要关注传统的单元测试技术，并且是在 2014 年之前进行的，早于现代 LLM 的出现。

Thus, they do not overlap with our research, as the first LLM-based unit testing work emerged in 2020.

因此，它们与我们的研究没有重叠，因为首个基于 LLM 的单元测试工作出现在 2020 年。

Table 1 presents a detailed comparison between our survey and existing literature, highlighting the novelty and scope of our work.

表 1 详细比较了我们的综述与现有文献，突出了我们工作的新颖性和范围。

In summary, to the best of our knowledge, this is the first systematic literature review specifically focusing on the applications of LLMs for unit testing.

总之，据我们所知，这是首个专门关注 LLM 在单元测试中应用的系统性文献综述。

3 SURVEY METHODOLOGY

3 综述方法论

In this section, we describe our methodology for conducting this systematic literature review, including the search strategy, paper collection process, and paper trend analysis.

在本节中，我们将描述进行这项系统性文献综述的方法论，包括搜索策略、论文收集过程和论文趋势分析。

3.1 Search Strategy

3.1 搜索策略

Following prior SE surveys [106, 126], we adopt a three-stage “Quasi-Gold Standard” (QGS) strategy [124] to collect relevant research papers systematically.

遵循先前的软件工程（SE）综述 [106, 126]，我们采用三阶段的“准金标准”（QGS）策略 [124] 来系统地收集相关研究论文。

This strategy combines manual and automated search processes to construct a set of known relevant studies, which is a common practice to refine search queries and improve retrieval accuracy.

该策略结合了手动和自动搜索过程来构建一组已知的相关研究，这是完善搜索查询和提高检索准确性的常见做法。

We first conduct a manual search to identify a seed set of relevant papers and derive a search string, as detailed in Section 3.2.

我们首先进行手动搜索以确定一组种子相关论文并推导出搜索字符串，详见 3.2 节。

We then utilize the search string to perform an automated search and employ a series of relatively strict filtering steps to extract the most relevant studies, as detailed in Section 3.3.

然后，我们利用该搜索字符串执行自动搜索，并采用一系列相对严格的筛选步骤来提取最相关的研究，详见 3.3 节。

We finally use a snowballing search to further complement the search results by manually inspecting references and citations, as detailed in Section 3.4.

最后，我们使用滚雪球搜索法，通过手动检查参考文献和引用来进一步补充搜索结果，详见 3.4 节。

Given the large volume of relevant papers, this strategy allows us to capture the most pertinent papers while maintaining higher efficiency and rigor than a purely manual process.

鉴于相关论文数量庞大，该策略使我们能够在保持比纯手动过程更高效率和严谨性的同时，捕获最相关的论文。

Particularly, we undertake the following three phases to search for and identify relevant studies.

具体而言，我们通过以下三个阶段来搜索和识别相关研究。

3.2 Manual Search and Search Items

3.2 手动搜索与搜索项

To construct the search items, we conduct a manual search from four top-tier SE conferences (ICSE, ESEC/FSE, ASE, ISSTA) and two journals (TOSEM and TSE), as listed in Table 2.

为了构建搜索项，我们对表 2 中列出的四个顶级 SE 会议（ICSE、ESEC/FSE、ASE、ISSTA）和两个期刊（TOSEM 和 TSE）进行了手动搜索。

The first two authors independently search these venues and propose an initial set of candidate papers involving both unit testing and LLMs.

前两位作者独立搜索这些渠道，并提出了一组涉及单元测试和 LLM 的候选论文初始集。

All authors then collaboratively review this collection to finalize the seed set of relevant publications.

随后，所有作者共同审查该集合，以最终确定相关出版物的种子集。

Then, we analyze the titles, abstracts, and keywords of these papers to identify search items, and conduct brainstorming to refine our search items, such as synonym substitution.

然后，我们分析这些论文的标题、摘要和关键词以识别搜索项，并进行头脑风暴来完善搜索项，例如同义词替换。

Finally, this iterative process formulates the set of search items, listed as follows.

最终，这一迭代过程制定了搜索项集，列出如下。

- **Search items related to unit testing:** “unit testing” OR “(unit) test cases” OR “(unit) test generation” OR “(unit) test repair” OR “(unit) test evolution” OR “regression testing” OR “test refactoring” OR “test oracle” OR “test reduction” OR “test selection” OR “test readability”

与单元测试相关的搜索项：“单元测试”或“（单元）测试用例”或“（单元）测试生成”或“（单元）测试修复”或“（单元）测试演进”或“回归测试”或“测试重构”或“测试预言”或“测试约减”或“测试选择”或“测试可读性”

- **Search items related to LLMs:** “Large Language Model(s)” OR “LLM(s)” OR “Pre-trained” OR “Pretraining” OR “PLM(s)” OR “(Code)BERT” OR “(Code)T5” OR “(Code)GPT” OR “Codex” OR “ChatGPT” OR “(Code)Llama” OR “GPT-” OR “DeepSeek()” OR “Mistral”

与 LLM 相关的搜索项：“大型语言模型”或“LLM”或“预训练”或“预训练过程”或“PLM”或“(Code)BERT”或“(Code)T5”或“(Code)GPT”或“Codex”或“ChatGPT”或“(Code)Llama”或“GPT-”或“DeepSeek()”或“Mistral”

3.3 Automatic Search and Study Selection

3.3 自动搜索与研究筛选

To perform the automated search, we utilize the above search items to collect relevant papers across four widely used databases, i.e., Google Scholar repository, ACM Digital Library, and IEEE Explorer Digital Library, at the end of January 2025.

为了执行自动搜索，我们在 2025 年 1 月底利用上述搜索项在四个广泛使用的数据库（即 Google Scholar 文库、ACM 数字图书馆和 IEEE Explorer 数字图书馆）中收集相关论文。

We restrict our search to articles published from 2017 onward, as the Transformer architecture [101], which serves as the foundation for LLMs, was introduced that year.

我们将搜索范围限制在 2017 年以后发表的文章，因为作为 LLM 基础的 Transformer 架构 [101] 是在那一年推出的。

3.3.1 Inclusion and Exclusion Criteria.

3.3.1 纳入与排除标准

We define a set of inclusion and exclusion criteria to filter out papers that do not align with the scope of this survey.

我们定义了一组纳入和排除标准，以过滤掉不符合本综述范围的论文。

Inclusion criteria. We define the following criteria for including papers:

纳入标准。我们定义了以下纳入论文的标准：

- **I1:** The paper proposes a technique, tool, or framework to address unit testing tasks using LLMs.
I1: 论文提出了一种利用 LLM 解决单元测试任务的技术、工具或框架。
- **I2:** The paper conducts an empirical study to evaluate LLMs in the context of unit testing.
I2: 论文进行了一项实证研究，以评估单元测试背景下的 LLM。
- **I3:** The paper focuses on specific unit testing tasks (e.g., assertion generation) where LLMs are employed.
I3: 论文关注采用了 LLM 的特定单元测试任务（例如断言生成）。

Exclusion criteria. We define the following criteria for excluding papers:

排除标准。我们定义了以下排除论文的标准：

- **C1:** The paper does not involve any unit testing tasks, e.g., unit test generation.
C1: 论文不涉及任何单元测试任务，例如单元测试生成。
- **C2:** The paper does not utilize any LLMs, e.g., only using traditional recurrent neural networks.
C2: 论文未使用任何 LLM，例如仅使用传统的循环神经网络。
- **C3:** The paper only mentions LLMs in future work rather than integrating LLMs in the approach.
C3: 论文仅在未来工作中提到 LLM，而非在方法中整合了 LLM。
- **C4:** The paper is a previously published conference paper extended to a journal by the same authors.
C4: 论文是同一作者将先前发表的会议论文扩展为期刊论文。
- **C5:** The paper is not an original research study, such as literature reviews, surveys, tool demonstrations, or editorials.
C5: 论文不是原创研究，例如文献综述、调查、工具演示或社论。
- **C6:** The paper is a duplicate publication where the preprint and the published version have different titles.
C6: 论文是重复发表，预印本和发表版本标题不同。

- **C7:** The paper is published in a workshop or a doctoral symposium.
C7: 论文发表在研讨会或博士生论坛上。
- **C8:** The paper is a grey publication, e.g., a technical report or thesis.
C8: 论文是灰色文献, 例如技术报告或学位论文。
- **C9:** The paper is inaccessible in full text or not written in English.
C9: 论文无法获取全文或非英文撰写。

During this process, the first two authors carefully review each paper to determine its eligibility based on the inclusion and exclusion criteria.

在此过程中, 前两位作者根据纳入和排除标准仔细审查每篇论文以确定其资格。

In cases where their decisions differ, the paper is referred to the third author for a final decision.

如果他们的决定不一致, 则将论文提交给第三位作者做最终决定。

For example, following exclusion criterion C4, there exists one study that extends a previously published conference paper [63] to a journal version [62] by the same authors.

例如, 根据排除标准 C4, 有一项研究将同一作者先前发表的会议论文 [63] 扩展为期刊版本 [62]。

In such cases, we retained only the extended journal version.

在这种情况下, 我们只保留了扩展后的期刊版本。

Besides, following exclusion criterion C6, we identify four papers whose published versions have different titles compared to their preprints.

此外, 根据排除标准 C6, 我们发现了四篇已发表版本的标题与预印本不同的论文。

In such cases, to avoid duplication, we retain only the published versions.

在这种情况下, 为避免重复, 我们只保留了已发表的版本。

After this step, we retained a total of 153 papers.

在此步骤之后, 我们要保留了总共 153 篇论文。

3.3.2 Quality Assessment.

3.3.2 质量评估

To further maximize the inclusion of high-quality papers, we design ten quality assessment questions to evaluate the relevance and rigor of included papers.

为了进一步最大化高质量论文的纳入, 我们设计了十个质量评估问题来评估所纳入论文的相关性和严谨性。

For each paper, its quality is evaluated by a three-tier scoring system: criteria are rated as “yes” (1 point), “partial” (0.5 points), or “no” (0 points).

对于每篇论文, 其质量通过三级评分系统进行评估: 标准评级为“是”(1分)、“部分”(0.5分)或“否”(0分)。

If a paper accumulates a total score below the threshold of 8 points, it will be excluded from further analysis, which reduces the number of papers to 99.

如果一篇论文的总分低于 8 分的阈值, 它将被排除在进一步分析之外, 这将论文数量减少到 99 篇。

The designed quality assessment criteria (QAC) are listed as follows.

设计的质量评估标准（QAC）列出如下：

- **QA1.** Is the paper primarily focused on LLMs, rather than using them only as baselines?
QA1. 论文是否主要关注 LLM，而不仅仅将其用作基线？
- **QA2.** Is the paper’s impact on the unit testing community explicitly stated?
QA2. 论文对单元测试社区的影响是否明确陈述？
- **QA3.** Are the research goals and key contributions explicitly defined?
QA3. 研究目标和关键贡献是否明确定义？
- **QA4.** Has the paper been published in a reputable venue?
QA4. 论文是否发表在知名场所？
- **QA5.** Does the paper provide open-source artifacts for reproducibility, such as datasets, code, or benchmarks?
QA5. 论文是否提供开源工件以供复现，如数据集、代码或基准？
- **QA6.** Is the implementation of the proposed technique described with sufficient clarity and detail?
QA6. 所提出技术的实现描述是否足够清晰详细？
- **QA7.** Are the experimental settings thoroughly explained, such as including hyperparameters and computing environments?
QA7. 实验设置是否解释透彻，例如包括超参数和计算环境？
- **QA8.** Are the utilized LLMs explicitly described, along with a clear explanation of how they are applied in the study?
QA8. 所使用的 LLM 是否明确描述，并清楚解释了它们如何在研究中应用？
- **QA9.** Are the evaluation metrics and results clearly aligned with research goals?
QA9. 评估指标和结果是否与研究目标清晰一致？
- **QA10.** Are both the contributions and limitations of the paper critically discussed?
QA10. 论文的贡献和局限性是否都经过了批判性讨论？

3.4 Snowballing Search

3.4 滚雪球搜索

To ensure the completeness of our study, we adopt a snowballing search approach [110] to manually incorporate papers that are previously overlooked yet remain pertinent to our study.

为了确保研究的完整性，我们采用滚雪球搜索方法 [110] 手动纳入先前被忽略但仍与我们的研究相关的论文。

Specifically, we examine all references (i.e., backward snowballing) or citations (i.e., forward snowballing) of collected papers to assess their quality and relevance to our survey.

具体而言，我们检查已收集论文的所有参考文献（即向后滚雪球）或引用（即向前滚雪球），以评估其质量以及与我们综述的相关性。

The manual inspection continues until no new relevant papers are identified, ultimately leading to the inclusion of an additional 6 papers in our survey.

手动检查持续进行，直到没有发现新的相关论文为止，最终在我们的综述中额外纳入了 6 篇论文。

This rigorous procedure helps ensure that our final corpus is comprehensive and provides a solid foundation for the subsequent analysis of LLM-based unit testing techniques.

这一严格的程序有助于确保我们的最终语料库是全面的，并为后续分析基于 LLM 的单元测试技术提供了坚实的基础。

Finally, we obtain 105 papers that are related to our work.

最终，我们获得了 105 篇与本职工作相关的论文。

3.5 Statistics of Collected Papers

3.5 已收集论文的统计数据

Figure 4 lists the number of relevant papers published across different publication venues.

图 4 列出了在不同出版场所发表的相关论文数量。

We find that 60% of the papers are published in peer-reviewed venues, with ICSE (11%) and TSE (8%) being the most popular conference and journal, respectively.

我们发现 60% 的论文发表在同行评审的场所，其中 ICSE (11%) 和 TSE (8%) 分别是最受欢迎的会议和期刊。

Following them are ASE (7%), FSE (7%), and ISSTA (5%), all of which are recognized as top-tier software engineering venues.

紧随其后的是 ASE (7%)、FSE (7%) 和 ISSTA (5%)，这些都被公认为顶级的软件工程场所。

This trend indicates researchers are increasingly prioritizing the dissemination of their work through high-quality, peer-reviewed venues, which in turn drives innovation and further advances in the field.

这一趋势表明，研究人员越来越重视通过高质量的同行评审场所传播他们的工作，这反过来推动了该领域的创新和进一步发展。

Meanwhile, around 40% of these papers, which are hosted on arXiv, have not undergone peer review.

同时，约 40% 的论文托管在 arXiv 上，尚未经过同行评审。

One possible reason for this phenomenon is the sudden surge of related studies emerging within a brief period.

这种现象的一个可能原因是相关研究在短时间内突然激增。

Given the inconsistent quality of these non-peer-reviewed papers, we carry out a rigorous assessment process to ensure only high-caliber works are selected for this study.

鉴于这些非同行评审论文的质量参差不齐，我们要执行严格的评估过程，以确保仅选择高质量的作品用于本研究。

Figure 3 further illustrates the publication trend of LLM-based unit testing.

图 3 进一步说明了基于 LLM 的单元测试的出版趋势。

We find that the number of relevant papers appears to be rising at an almost exponential pace.

我们发现相关论文的数量似乎正以近乎指数级的速度增长。

While only one paper was published in 2021 and two in 2022, the number increased to 14 in 2023 and surged to 73 in 2024, with 15 already appearing by March 2025.

虽然 2021 年仅发表了 1 篇论文，2022 年发表了 2 篇，但 2023 年增加到 14 篇，并在 2024 年激增至 73 篇，到 2025 年 3 月已出现 15 篇。

Since our data collection ends in March 2025, the figure may not reflect the full trend for the year, and the final count is expected to increase further.

由于我们的数据收集于 2025 年 3 月结束，该数字可能无法反映当年的完整趋势，最终计数预计将进一步增加。

These observations suggest that the application of LLMs to unit testing has gained substantial momentum since 2021 and will likely continue to be an active and growing area of research.

这些观察结果表明，自 2021 年以来，LLM 在单元测试中的应用获得了巨大的动力，并可能继续成为一个活跃且不断增长的研究领域。

4 ANALYSIS FROM UNIT TESTING PERSPECTIVE

4 从单元测试视角的分析

In this section, we present a comprehensive analysis from the perspective of unit testing and organize the collected studies according to different testing scenarios.

在本节中，我们从单元测试的角度进行了全面分析，并根据不同的测试场景对收集到的研究进行组织。

4.1 Taxonomy Analysis

4.1 分类体系分析

Figure 5 illustrates the distribution of unit testing tasks to which LLMs are applied.

图 5 展示了应用 LLM 的单元测试任务的分布。

It can be observed that test generation dominates the research landscape, constituting approximately 60% of the total research volume.

可以观察到，测试生成主导了研究格局，约占总研究量的 60%。

This phenomenon is reasonable, as test case generation has long been a central component in the unit testing pipeline and continues to attract substantial attention from both academia and industry.

这种现象是合理的，因为测试用例生成长期以来一直是单元测试流程中的核心组件，并持续吸引着学术界和工业界的广泛关注。

In addition, oracle generation represents the most popular task in LLM-based unit testing research, accounting for about 14% of the research proportion.

此外，预言 (Oracle) 生成是基于 LLM 的单元测试研究中最受欢迎的任务，约占研究比例的 14%。

This reflects the persistent challenges in automating this task and the growing interest in leveraging LLMs to address it.

这反映了自动化该任务面临的持续挑战，以及利用 LLM 解决该问题的兴趣日益增长。

Moreover, several other tasks have received moderate attention.

此外，其他几个任务也受到了一定的关注。

For example, bug reproduction is explored in 5 studies (4.3%), while test evolution and test smell detection each appear in 4 studies (3.4%).

例如，5 项研究（4.3%）探索了错误复现，而测试演进和测试异味（Test Smell）检测各出现在 4 项研究（3.4%）中。

Tasks such as test completion, test readability, and test minimization are addressed in 1–3 papers each, suggesting they are emerging but less mature application areas for LLMs.

诸如测试补全、测试可读性和测试最小化等任务各涉及 1-3 篇论文，表明它们是 LLM 新兴但不太成熟的应用领域。

Notably, we observe that some studies investigate underexplored aspects of unit testing, including test-to-code traceability, test refactoring, and test validation, each appearing in only a single study.

值得注意的是，我们观察到一些研究调查了单元测试中未被充分探索的方面，包括测试到代码的可追溯性、测试重构和测试验证，每一项仅出现在一项研究中。

This indicates a growing research interest in broadening the scope of unit testing tasks supported by LLMs thanks to LLMs' general knowledge gleaned from vast amounts of training data.

这表明，得益于 LLM 从大量训练数据中收集到的通用知识，研究界对拓宽 LLM 支持的单元测试任务范围的兴趣日益浓厚。

Overall, this trend highlights both the areas where LLMs have already shown strong potential and the opportunities for further exploration in more specialized or complex unit testing scenarios.

总的来说，这一趋势既突出了 LLM 已显示出强大潜力的领域，也突出了在更专业或复杂的单元测试场景中进一步探索的机会。

4.2 Unit Test Case Generation

4.2 单元测试用例生成

Unit test generation typically takes a focal method (i.e., the method under test) as input and produces a complete unit test comprising two key components: a test prefix and an assertion.

单元测试生成通常以焦点方法（即被测方法）作为输入，并生成包含两个关键组件的完整单元测试：测试前缀和断言。

The test prefix sets up the required execution context, while the assertion verifies whether the focal method behaves as expected.

测试前缀建立了所需的执行上下文，而断言则验证焦点方法的行为是否符合预期。

In the literature, to reduce manual efforts in writing unit tests, researchers have proposed various automated unit test generation approaches, including symbolic execution testing [11, 15], random testing [61, 73], and search-based testing [6, 25] strategies.

在文献中，为了减少编写单元测试的人工工作量，研究人员提出了各种自动单元测试生成方法，包括符号执行测试 [11, 15]、随机测试 [61, 73] 和基于搜索的测试 [6, 25] 策略。

Traditional approaches (e.g., EvoSuite [25]) are generally designed to maximize code coverage but often struggle to produce meaningful and human-readable test cases, primarily due to their limited understanding of the semantics of focal methods.

传统方法（例如 EvoSuite [25]）通常旨在最大化代码覆盖率，但往往难以生成有意义且人类可读的测试用例，这主要是因为它们对焦点方法的语义理解有限。

Therefore, recent research has turned to leveraging LLMs for test generation with the aim of producing more practical unit test cases from multiple dimensions, including correctness, coverage, and fault detection capability.

因此，最近的研究已转向利用 LLM 进行测试生成，旨在从正确性、覆盖率和故障检测能力等多个维度生成更实用的单元测试用例。

Existing LLM-based test generation techniques can be broadly categorized into three distinct groups, discussed as follows.

现有的基于 LLM 的测试生成技术大致可分为三类，讨论如下。

4.2.1 Training LLMs for Test Generation.

4.2.1 训练 LLM 进行测试生成

These techniques typically utilize supervised learning to train LLMs, thereby adapting them to the downstream task of unit test case generation.

这些技术通常利用监督学习来训练 LLM，从而使其适应单元测试用例生成的下游任务。

This is an intuitive yet effective strategy to enable LLMs to refine their pre-trained knowledge and weight parameters through limited-scale, domain-specific datasets.

这是一种直观而有效的策略，使 LLM 能够通过有限规模的特定领域数据集来优化其预训练知识和权重参数。

In the unit testing domain, the pre-training and fine-tuning paradigm has been extensively adopted during the early stages of LLM development, particularly for medium-scale models such as T5 and CodeT5, which contain hundreds of millions of parameters.

在单元测试领域，预训练和微调范式在 LLM 发展的早期阶段被广泛采用，特别是针对包含数亿参数的中等规模模型，如 T5 和 CodeT5。

The widespread adoption of this paradigm can be primarily attributed to the limited generalization capabilities of early-stage LLMs, which require task-specific training to achieve optimal performance in specialized domains like test case generation [2, 3, 99].

这种范式的广泛采用主要归因于早期 LLM 有限的泛化能力，这些模型需要特定任务的训练才能在测试用例生成等专业领域达到最佳性能 [2, 3, 99]。

As early as 2020, Tufano et al. [99] introduced AthenaTest, the first LLM-based unit test generation approach, which formulates the task as a sequence-to-sequence learning problem.

早在 2020 年，Tufano 等人 [99] 就推出了 AthenaTest，这是首个基于 LLM 的单元测试生成方法，它将任务表述为一个序列到序列的学习问题。

AthenaTest follows a two-stage training procedure: (1) denoising pre-training on a large, unsupervised Java corpus and (2) supervised fine-tuning for a downstream translation task of generating unit test cases.

AthenaTest 遵循两阶段训练过程：（1）在大型无监督 Java 语料库上进行去噪预训练，以及（2）针对生成单元测试用例的下游翻译任务进行监督微调。

To address the limitation of assertion knowledge in AthenaTest, A3Test [2] enhances BART with an assertion-aware self-supervision objective, i.e., predicting the masked tokens of a given focal method and its corresponding assert statements.

为了解决 AthenaTest 在断言知识方面的局限性，A3Test [2] 通过断言感知的自监督目标增强了 BART 模型，即预测给定焦点方法及其相应断言语句中的掩码标记。

Beyond the standard pre-training and fine-tuning pipeline, researchers have explored various strategies to improve the performance of training LLMs in test case generation, including reinforcement learning [91], domain adaptation [88] and data augmentation [36].

除了标准的预训练和微调流程外，研究人员还探索了各种策略来提高训练 LLM 生成测试用例的性能，包括强化学习 [91]、领域自适应 [88] 和数据增强 [36]。

For example, Shin et al. [88] investigate the advantages of domain adaptation for fine-tuning LLMs in the context of automated test case generation.

例如，Shin 等人 [88] 研究了在自动测试用例生成背景下，领域自适应对于微调 LLM 的优势。

However, most of these works rely on general-purpose, off-the-shelf LLMs that are typically trained on natural language and code corpora, without incorporating test-specific knowledge.

然而，这些工作大多依赖于通用的现成 LLM，这些模型通常在自然语言和代码语料库上进行训练，未包含测试特定的知识。

To address this gap, Rao et al. [80] propose CAT-LM, a GPT-style LLM with 2.7 billion parameters specifically trained to learn the mapping between methods under test and their corresponding test cases, making it more suitable for the test generation task.

为了填补这一空白，Rao 等人 [80] 提出了 CAT-LM，这是一个拥有 27 亿参数的 GPT 风格 LLM，专门用于学习被测方法与其相应测试用例之间的映射，使其更适合测试生成任务。

Similarly, He et al. [34] construct a large-scale dataset, UniTSyn, containing 2.7 million method-test function pairs across five programming languages, and propose UniTester, a specialized test generation LLM trained via continual fine-tuning of SantaCoder with an autoregressive objective.

类似地，He 等人 [34] 构建了一个大规模数据集 UniTSyn，包含五种编程语言的 270 万个方法-测试函数对，并提出了 UniTester，这是一个通过自回归目标对 SantaCoder 进行持续微调而训练出的专用测试生成 LLM。

4.2.2 Prompting LLMs for Test Generation.

4.2.2 提示 LLM 进行测试生成

These techniques typically construct prompts with diverse sources of project information to directly invoke LLMs for test case generation without requiring any additional training.

这些技术通常利用多种项目信息源构建提示（Prompt），直接调用 LLM 进行测试用例生成，而无需任何额外的训练。

In the unit testing community, prompt engineering has rapidly gained traction with the emergence of LLMs with billions of parameters, as their advanced capabilities make them well-suited for performing human-like interactions and generating high-quality test cases.

在单元测试社区，随着拥有数十亿参数的 LLM 的出现，提示工程迅速受到关注，因为它们的先进能力使其非常适合进行类人交互并生成高质量的测试用例。

Most prompt-based test generation techniques primarily follow a generation-and-refinement paradigm, where initial test cases are first generated based on prompts and then iteratively refined using dynamic execution feedback (e.g., code coverage and failure information) to enhance their quality [13, 30, 65, 77, 83, 85, 109, 121].

大多数基于提示的测试生成技术主要遵循“生成-优化”范式，即首先根据提示生成初始测试用例，然后利用动态执行反馈（如代码覆盖率和失败信息）进行迭代优化，以提高其质量 [13, 30, 65, 77, 83, 85, 109, 121]。

For example, TestART [30] queries LLMs to generate an initial set of test cases, then uses a compiler to collect runtime information.

例如，TestART [30] 查询 LLM 生成一组初始测试用例，然后使用编译器收集运行时信息。

This information is then fed into a coverage-guided testing framework and a template-based repair strategy to optimize the test cases iteratively.

随后，该信息被输入到一个覆盖率导向的测试框架和基于模板的修复策略中，以迭代地优化测试用例。

In addition, to design more effective prompts, researchers adopt a wide range of strategies to incorporate LLMs with valuable contextual information, including mutation testing [17], method slicing [109], demonstration retrieval [135], defect detection [119] and program analysis [113].

此外，为了设计更有效的提示，研究人员采用了广泛的策略将有价值的上下文信息结合给 LLM，包括变异测试 [17]、方法切片 [109]、范例检索 [135]、缺陷检测 [119] 和程序分析 [113]。

For example, Dakhel et al. [17] utilize mutation testing to augment the original prompt, enabling LLMs to generate tests capable of killing surviving mutants.

例如，Dakhel 等人 [17] 利用变异测试增强原始提示，使 LLM 能够生成可以杀死存活变异体的测试。

Given a focal method, Yang et al. [113] retrieve the most similar method within the same project and incorporate it, along with its corresponding test cases, into the prompt to guide LLMs in generating test cases with project-specific exemplars.

给定一个焦点方法，Yang 等人 [113] 检索同一项目中最相似的方法，并将其与相应的测试用例一起整合到提示中，以引导 LLM 利用项目特定的范例生成测试用例。

However, most of the aforementioned work has primarily focused on standard benchmarks such as Defects4J [45] and mainstream programming languages such as Java.

然而，上述大部分工作主要集中在标准基准测试（如 Defects4J [45]）和主流编程语言（如 Java）上。

Recently, the research community has shifted its focus toward emerging frontier domains that present unique and diverse technical challenges.

最近，研究社区已将重点转向新兴的前沿领域，这些领域提出了独特且多样化的技术挑战。

These studies include repository-level test case generation [59, 118], multi-language test case generation [76], high-performance computing software [47], industrial deployment [5, 84], Rust [14], programming problems [4], data-serialization libraries [137], game development [74], vulnerability exploitation [28], and bug reproduction [46].

这些研究包括仓库级测试用例生成 [59, 118]、多语言测试用例生成 [76]、高性能计算软件 [47]、工业部署 [5, 84]、Rust 语言 [14]、编程问题 [4]、数据序列化库 [137]、游戏开发 [74]、漏洞利用 [28] 以及错误复现 [46]。

Overall, these advancements demonstrate the growing versatility of prompt-based LLMs in adapting to diverse testing contexts, without the need for extensive retraining.

总的来说，这些进展证明了基于提示的 LLM 在适应不同测试环境方面日益增长的多功能性，且无需进行大量的重新训练。

4.2.3 Empirical Study.

4.2.3 实证研究

In addition to the aforementioned technical advancements, researchers have conducted extensive empirical studies to investigate the capabilities of LLMs in generating unit test cases.

除了上述技术进步外，研究人员还进行了广泛的实证研究，以调查 LLM 在生成单元测试用例方面的能力。

These studies systematically explore the actual performance of LLMs across various aspects, including fine-tuning [86, 92], prompt engineering [53, 72, 114], integration or comparison with traditional techniques [1, 9, 44, 96, 112], source code characteristics [42], context [90], quality assessment [94], retrieval-augmented generation [87], ChatGPT-specific evaluations [31, 117, 121] and benchmarking [8, 43, 133].

这些研究系统地探索了 LLM 在各个方面的实际性能，包括微调 [86, 92]、提示工程 [53, 72, 114]、与传统技术的整合或比较 [1, 9, 44, 96, 112]、源代码特征 [42]、上下文 [90]、质量评估 [94]、检索增强生成 [87]、ChatGPT 专项评估 [31, 117, 121] 以及基准测试 [8, 43, 133]。

For example, Shang et al. [86] conduct a large-scale empirical study to explore the potential of fine-tuning LLMs for unit testing, involving three tasks, five benchmarks, eight evaluation metrics, and 37 advanced LLMs across various architectures and sizes.

例如，Shang 等人 [86] 进行了一项大规模实证研究，以探索微调 LLM 用于单元测试的潜力，涉及三个任务、五个基准、八个评估指标以及 37 个不同架构和规模的先进 LLM。

Tang et al. [96] perform a systematic comparison of unit test cases generated by ChatGPT and EvoSuite based on several critical factors, including correctness, readability, code coverage, and bug detection capability.

Tang 等人 [96] 基于正确性、可读性、代码覆盖率和错误检测能力等几个关键因素，对 ChatGPT 和 EvoSuite 生成的单元测试用例进行了系统比较。

Yang et al. [114] empirically investigate the capabilities of five LLMs with various prompting settings.

Yang 等人 [114] 实证调查了五种 LLM 在不同提示设置下的能力。

These empirical studies provide critical insights into the strengths and limitations of LLM-based test generation and inform future technical design and evaluation.

这些实证研究为基于 LLM 的测试生成的优势和局限性提供了关键见解，并为未来的技术设计和评估提供了参考。

4.3 Unit Test Oracle Generation

4.3 单元测试预言生成

A test oracle formally defines the expected behavior of a software unit under test for a given test prefix, serving as a critical component in unit testing.

测试预言 (Test Oracle) 正式定义了被测软件单元在给定测试前缀下的预期行为，是单元测试中的关键组件。

Unlike test case generation, which focuses on producing inputs and exploring execution paths, generating reliable test oracles poses a non-trivial technical challenge, as it requires capturing the intended design specification rather than merely reflecting the implemented behavior.

与专注于产生输入和探索执行路径的测试用例生成不同，生成可靠的测试预言面临着不小的技术挑战，因为它需要捕获预期的设计规范，而不仅仅是反映已实现的行为。

This process demands a deep understanding of functional requirements, edge cases, and expected outcomes [39].

这个过程需要对功能需求、边界情况和预期结果有深刻的理解 [39]。

With the advent of LLMs, researchers have begun exploring their potential to automate oracle generation through both fine-tuning and prompting strategies.

随着 LLM 的出现，研究人员已开始探索通过微调和提示策略来自动化预言生成的潜力。

Table 3 summarizes existing LLM-based test oracle generation studies, which can be categorized into three key directions: whole oracle generation, assertion oracle generation, and exceptional oracle generation.

表 3 总结了现有的基于 LLM 的测试预言生成研究，这些研究可分为三个关键方向：整体预言生成、断言预言生成和异常预言生成。

We discuss these representative studies in detail below.

我们在下面详细讨论这些代表性研究。

4.3.1 Whole Oracle Generation.

4.3.1 整体预言生成

In 2022, Dinella et al. [21] introduce TOGA, a transformer-based approach to infer both exceptional and assertion test oracles based on the context of the focal method.

2022 年，Dinella 等人 [21] 推出了 TOGA，这是一种基于 Transformer 的方法，旨在根据焦点方法的上下文推断异常和断言测试预言。

TOGA is the first LLM-powered test oracle generation study by fine-tuning CodeBERT to (1) determine whether a test prefix raises an exception and (2) rank a set of candidate assertions.

TOGA 是首个基于 LLM 的测试预言生成研究，它通过微调 CodeBERT 来 (1) 确定测试前缀是否会引发异常，以及 (2) 对一组候选断言进行排序。

Furthermore, Liu et al. [58] identify three inappropriate settings in TOGA, i.e., generating test prefixes from correct program versions, evaluating with an unrealistic metric, and lacking a straightforward baseline.

此外，Liu 等人 [58] 发现了 TOGA 中的三个不当设置，即从正确的程序版本生成测试前缀、使用不切实际的指标进行评估以及缺乏直观的基线。

They then re-evaluate TOGA in a more realistic setting by reducing duplicates and noise during evaluation and introducing an additional ranking step to prioritize failed test cases.

随后，他们在一个更现实的设置中重新评估了 TOGA，通过减少评估过程中的重复和噪声，并引入额外的排序步骤来优先考虑失败的测试用例。

Similarly, Hossain et al. [38] conduct a series of replication studies to expand the understanding of TOGA's applicability, generalizability, precision, and fault detection capability across 25 real-world Java systems, 223.5K test cases, and 51K injected faults.

同样，Hossain 等人 [38] 进行了一系列复现研究，以扩展对 TOGA 在 25 个真实 Java 系统、22.35 万个测试用例和 5.1 万个注入故障中的适用性、泛化性、精确度和故障检测能力的理解。

Unlike early-stage TOGA, which fine-tunes CodeBERT as a component for oracle classifier and ranker, recent research explores fine-tuning or prompting LLMs as core backbones to generate oracles in an end-to-end manner.

与早期将 CodeBERT 微调为预言分类器和排序器组件的 TOGA 不同，最近的研究探索了微调或提示 LLM 作为核心骨干，以端到端的方式生成预言。

For example, Hayet et al. [33] introduce CHATASSERT, a feedback-driven oracle generation technique that utilizes prompt engineering to iteratively generate and refine oracles by incorporating dynamic and static information.

例如，Hayet 等人 [33] 推出了 CHATASSERT，这是一种反馈驱动预言生成技术，利用提示工程结合动态和静态信息来迭代生成和优化预言。

Khandaker et al. [48] propose AugmenTest to generate oracles for EvoSuite-generated test prefixes by prompting LLMs to infer the intended behavior of a focal method from documentation and developer comments.

Khandaker 等人 [48] 提出了 AugmenTest，通过提示 LLM 从文档和开发人员注释中推断焦点方法的预期行为，从而为 EvoSuite 生成的测试前缀生成预言。

In addition to the above novel techniques, researchers have conducted numerous empirical studies to explore the capabilities of LLMs in generating oracles.

除了上述新技术外，研究人员还进行了大量实证研究，以探索 LLM 在生成预言方面的能力。

For example, Hossain et al. [37] present a comprehensive study by fine-tuning seven code LLMs using six distinct prompts on a large dataset consisting of 110 Java projects.

例如，Hossain 等人 [37] 展示了一项全面的研究，在包含 110 个 Java 项目的大型数据集上，使用 6 种不同的提示微调了 7 个代码 LLM。

Hossain et al. [39] further conduct an empirical study to investigate the impact of Javadoc comments on test oracle generation by fine-tuning ten LLMs with three different prompts.

Hossain 等人 [39] 进一步进行了一项实证研究，通过使用三种不同的提示微调十个 LLM，调查 Javadoc 注释对测试预言生成的影响。

Konstantinou et al. [50] empirically investigate whether LLMs can identify the actual and expected program behavior.

Konstantinou 等人 [50] 实证调查了 LLM 是否能够识别实际和预期的程序行为。

Besides, Shin et al [89] undertake an empirical study to reassess the performance of prior oracle generation techniques (e.g., TOGA) and ChatGPT based on both static generation metrics (e.g., BLEU and CodeBLEU) and dynamic test adequacy metrics (e.g., line coverage and mutation score).

此外，Shin 等人 [89] 进行了一项实证研究，基于静态生成指标（如 BLEU 和 CodeBLEU）和动态测试充分性指标（如行覆盖率和变异分数），重新评估了先前的预言生成技术（如 TOGA）和 ChatGPT 的性能。

Their findings indicate a lack of statistically significant correlation between static and dynamic metrics, suggesting that existing static generation metrics do not reliably capture the quality of the generated oracles, and dynamic test adequacy metrics should serve as the principal evaluation criteria in this field.

他们的发现表明静态指标与动态指标之间缺乏统计学上的显著相关性，这表明现有的静态生成指标不能可靠地反映生成预言的质量，而动态测试充分性指标应作为该领域的主要评估标准。

4.3.2 Assertion Oracle Generation.

4.3.2 断言预言生成

In addition to the aforementioned whole test oracle generation work, researchers have conducted specialized explorations on assertion generation, including fine-tuning [62, 63, 100, 100, 122] and prompt engineering [64, 102].

除了上述整体测试预言生成工作外，研究人员还对断言生成进行了专门探索，包括微调 [62, 63, 100, 100, 122] 和提示工程 [64, 102]。

As a seminal work in this domain, Tufano et al. [100] frame assertion generation as a sequence-to-sequence task by fine-tuning LLMs on the ATLAS dataset, where each input pair consists of a test prefix and its focal method (i.e., the method under test), and the output is an assertion.

作为该领域的开创性工作，Tufano 等人 [100] 通过在 ATLAS 数据集上微调 LLM，将断言生成构建为序列到序列的任务，其中每个输入对由测试前缀及其焦点方法（即被测方法）组成，输出则是断言。

This work positions assertion generation as a downstream task for evaluating LLMs in the context of software engineering, laying the groundwork for its adoption in subsequent LLM research [62, 63, 122].

这项工作将断言生成定位为评估软件工程背景下 LLM 的下游任务，为其在随后的 LLM 研究中的采用奠定了基础 [62, 63, 122]。

He et al. [35] conduct an empirical study on the impact of the focal method identification strategy in ATLAS and reveal its limitations in assuming the last method call before assertions as the focal method.

He 等人 [35] 对 ATLAS 中焦点方法识别策略的影响进行了实证研究，并揭示了其局限性，即它假设断言之前的最后一个方法调用为焦点方法。

They then introduce ATLAS+, a revised dataset where focal methods are identified using various test-to-code traceability techniques, offering a more realistic and practical evaluation framework.

随后，他们推出了 ATLAS+，这是一个修订后的数据集，利用各种测试到代码的可追溯性技术识别焦点方法，提供了一个更现实和实用的评估框架。

Unlike the above work relying on training, Nashid et al. [64] utilize few-shot learning to query Codex to generate assertion statements given focal methods and test prefixes.

与上述依赖训练的工作不同，Nashid 等人 [64] 利用少样本学习来查询 Codex，根据给定的焦点方法和测试前缀生成断言语句。

Wang et al. [102] introduce CLAP, which prompts LLMs to generate assertions based on chain-of-thought reasoning and iteratively refines its predictions through interactions with both LLMs and a Python interpreter.

Wang 等人 [102] 推出了 CLAP，它提示 LLM 基于思维链（Chain-of-Thought）推理生成断言，并通过与 LLM 和 Python 解释器的交互迭代优化其预测。

Recently, Zhang et al. [134] conduct the first comprehensive study on fine-tuning various LLMs for automated assertion generation across two benchmarks, five LLMs, and two metrics.

最近，Zhang 等人 [134] 进行了首项全面研究，旨在跨越两个基准、五个 LLM 和两个指标，微调各种 LLM 以实现自动断言生成。

Their findings underscore the potential of LLM-based assertion generation to substantially alleviate the manual workload of unit testing experts in real-world software development, thereby inspiring further research in this domain.

他们的发现强调了基于 LLM 的断言生成在大幅减轻现实世界软件开发中单元测试专家的手动工作量方面的潜力，从而激发了该领域的进一步研究。

Building on this, Zhang et al. [132] introduce RetriGen, a retrieval-augmented automated assertion generation approach, which incorporates a novel hybrid assertion retriever to refine the assertion retrieval process by leveraging both lexical and semantic similarity to identify the most relevant assertions from external codebases.

在此基础上，Zhang 等人 [132] 推出了 RetriGen，这是一种检索增强的自动断言生成方法，它结合了一种新颖的混合断言检索器，利用词汇和语义相似性从外部代码库中识别最相关的断言，从而优化断言检索过程。

Moreover, Zhang et al. [131] propose AG-RAG, which optimizes both the retriever and generator within an end-to-end pipeline using a joint training strategy, enabling them to enhance their performance through collaborative learning mutually.

此外，Zhang 等人 [131] 提出了 AG-RAG，它利用联合训练策略在端到端管道中优化检索器和生成器，使它们能够通过相互协作学习来提高性能。

4.3.3 Exceptional Oracle Generation.

4.3.3 异常预言生成

Compared to the extensive research on assertion oracle generation, there has been only one study dedicated specifically to exception oracle generation.

与关于断言预言生成的广泛研究相比，只有一项研究专门致力于异常预言生成。

Zhang et al. [125] introduce exLong, an LLM-based framework for automatically generating exceptional behavior test cases, i.e., checking whether the method under test throws an exception or not.

Zhang 等人 [125] 推出了 exLong，这是一个基于 LLM 的框架，用于自动生成异常行为测试用例，即检查被测方法是否抛出异常。

Built on CodeLlama, exLong integrates program analysis to extract execution traces leading to throw statements, guard conditions, and relevant non-exceptional test cases.

exLong 建立在 CodeLlama 之上，整合了程序分析以提取导致抛出语句的执行轨迹、保护条件以及相关的非异常测试用例。

4.4 Unit Test Case Evolution

4.4 单元测试用例演进

During software evolution, source code needs to be continuously changed to satisfy new requirements or fix reported bugs.

在软件演进过程中，源代码需要不断更改以满足新需求或修复报告的错误。

To maintain software quality, it is essential to co-evolve the corresponding unit test cases alongside the changed source code.

为了保持软件质量，必须将相应的单元测试用例与更改后的源代码协同演进（即同步更新）。

However, this co-evolution process can pose significant challenges for developers, particularly given the constraints of limited regression testing resources and frequent releases of updated project versions.

然而，这种协同演进过程给开发人员带来了重大挑战，尤其是在回归测试资源有限和项目版本更新频繁的限制下。

To address this, Hu et al. [41] propose CEPROT, which fine-tunes CodeT5 to update outdated test cases based on source code modifications.

为了解决这个问题，Hu 等人 [41] 提出了 CEPROT，它微调 CodeT5 以根据源代码的修改来更新过时的测试用例。

Given a source code change and an associated test case, CEPROT first determines whether the test case is obsolete and requires updating; if deemed obsolete, it generates a new version of the test case accordingly.

给定源代码变更和关联的测试用例，CEPROT 首先确定测试用例是否已过时并需要更新；如果认为已过时，它将相应地生成测试用例的新版本。

Building on CEPROT, Liu et al. [56] develop SYNTER, a prompt-based approach that queries GPT-4 using contextual information to generate updated test cases.

在 CEPROT 的基础上，Liu 等人 [56] 开发了 SYNTER，这是一种基于提示的方法，利用上下文信息查询 GPT-4 以生成更新后的测试用例。

Given an obsolete test case to repair, SYNTER constructs three types of test-repair-oriented contexts via static analysis, and ranks them to select the most relevant one for guiding the repair process.

给定一个需要修复的过时测试用例，SYNTER 通过静态分析构建三种面向测试修复的上下文，并对其排序以选择最相关的上下文来指导修复过程。

In addition, Yaraghi et al. [116] introduce TARGET, a fine-tuning-based approach that adapts CodeT5+ to more realistic and executable test repair scenarios.

此外，Yaraghi 等人 [116] 推出了 TARGET，这是一种基于微调的方法，使 CodeT5+ 适应更现实和可执行的测试修复场景。

TARGET frames test repair as a language translation task via a two-step pipeline: collecting essential contextual information that characterizes the test breakage, and utilizing the information to construct input-output training pairs for finetuning LLMs.

TARGET 通过两步管道将测试修复构建为语言翻译任务：收集描述测试破坏特征的关键上下文信息，并利用该信息构建用于微调 LLM 的输入-输出训练对。

Overall, these studies highlight the emerging role of LLMs in automating test evolution, paving the way for more robust and adaptive regression testing in evolving software systems.

总的来说，这些研究突出了 LLM 在自动化测试演进中的新兴作用，为不断演进的软件系统中更稳健和自适应的回归测试铺平了道路。

4.5 Unit Test Completion

4.5 单元测试补全

Test completion attempts to automatically generate the next statement within an incomplete unit test case.

测试补全试图在一个不完整的单元测试用例中自动生成下一条语句。

This task takes as input the focal method under test, the test method signature, and the preceding statements within the incomplete test body, and produces the subsequent statement to be appended.

该任务将受测焦点方法、测试方法签名以及不完整测试体中的前序语句作为输入，并生成要追加的后续语句。

TECO [66] represents the first LLM-based test completion approach that leverages static analysis to extract code semantics and fine-tune CodeT5 to predict subsequent test statements.

TECO [66] 代表了首个基于 LLM 的测试补全方法，它利用静态分析提取代码语义并微调 CodeT5 以预测后续的测试语句。

Furthermore, CAT-LM [80] advances the field by pre-training on both source code and test cases, followed by supervised fine-tuning, demonstrating superior performance over TECO.

此外，CAT-LM [80] 通过在源代码和测试用例上进行预训练，随后进行监督微调，推动了该领域的发展，展现出优于 TECO 的性能。

This line of work suggests that LLMs can effectively assist in interactive test development workflows by incrementally completing unit test cases.

这一系列工作表明，LLM 可以通过增量补全单元测试用例，有效地辅助交互式测试开发 workflow。

4.6 Test Smell

4.6 测试异味 (Test Smell)

Test smells refer to potential issues in unit test cases that may degrade test quality, such as brittleness, slow execution, or lack of clarity.

测试异味指的是单元测试用例中可能降低测试质量的潜在问题，例如脆弱性、执行缓慢或缺乏清晰度。

Similar to code smells in production code, test smells suggest that a unit test may not be well-structured, robust, or efficient.

与生产代码中的代码异味类似，测试异味表明单元测试可能结构不佳、不稳健或效率低下。

Motivated by recent advances in LLMs, Lucas et al. [60] conduct an empirical study to assess the capability of LLMs in detecting test smells, demonstrating their potential to automate this process and improve testing efficiency.

受 LLM 最新进展的启发，Lucas 等人 [60] 进行了一项实证研究以评估 LLM 检测测试异味的能力，证明了其自动化该过程并提高测试效率的潜力。

Furthermore, Gao et al. [29] propose UTRefactor, an LLM-based test refactoring approach to eliminate test smells and improve the quality of unit test cases.

此外，Gao 等人 [29] 提出了 UTRefactor，这是一种基于 LLM 的测试重构方法，旨在消除测试异味并提高单元测试用例的质量。

UTRefactor extracts relevant contextual information from the test code, incorporates external knowledge, and guides LLMs through a chain-of-thought process to simulate manual refactoring with improved accuracy and consistency.

UTRefactor 从测试代码中提取相关上下文信息，结合外部知识，并通过思维链（Chain-of-Thought）过程引导 LLM，以更高的准确性和一致性模拟手动重构。

These studies suggest that LLMs can serve as valuable assistants in detecting and mitigating test smells, contributing to more maintainable and robust unit test cases.

这些研究表明，LLM 可以作为检测和缓解测试异味的有力助手，有助于构建更可维护和稳健的单元测试用例。

4.7 Test-to-Code Traceability Link

4.7 测试到代码的可追溯性链接

Test-to-code traceability involves establishing explicit links between unit tests and the corresponding software units they are intended to validate.

测试到代码的可追溯性涉及在单元测试与其旨在验证的相应软件单元之间建立明确的链接。

This practice is essential in unit testing, as it provides visibility into how tests align with production code, thereby facilitating test coverage analysis, debugging, and test maintenance.

这种做法在单元测试中至关重要，因为它提供了测试与生产代码对齐情况的可见性，从而促进了测试覆盖率分析、调试和测试维护。

To this end, Sun et al. [93] introduce TestLinker, a hybrid approach that combines heuristic rules with LLMs to establish test-to-code traceability links at the method level.

为此，Sun 等人 [93] 推出了 TestLinker，这是一种混合方法，结合启发式规则与 LLM 来建立方法级别的测试到代码可追溯性链接。

Specifically, TestLinker fine-tunes CodeT5 to learn the inherent semantic correlation between unit tests and focal methods, and then utilizes mapping rules to accurately align predicted function names with corresponding production method declarations.

具体而言，TestLinker 微调 CodeT5 以学习单元测试与焦点方法之间固有的语义相关性，然后利用映射规则将预测的函数名称与相应的生产方法声明准确对齐。

Despite its importance, this area remains under-explored, calling for further research on LLM-based techniques for test-to-code traceability to enable more maintainable and traceable unit test cases.

尽管其重要性不言而喻，但该领域仍未被充分探索，需要进一步研究基于 LLM 的测试到代码可追溯性技术，以实现更可维护和可追溯的单元测试用例。

4.8 Test Minimization

4.8 测试最小化

As software evolves, unit test cases tend to grow when software evolves, making it impractical to execute all test cases with the allocated testing budgets, especially for large software systems.

随着软件的演进，单元测试用例往往会不断增长，这使得在分配的测试预算内执行所有测试用例变得不切实际，尤其对于大型软件系统而言。

Test minimization attempts to improve the efficiency of unit testing by removing redundant test cases, thus reducing execution time and resource consumption while maintaining the adequacy criteria of the test suite, such as code coverage and fault detection capability.

测试最小化试图通过移除冗余的测试用例来提高单元测试的效率，从而在保持测试套件充分性标准（如代码覆盖率和故障检测能力）的同时，减少执行时间和资源消耗。

To this end, Pan et al. [75] propose LTM, a scalable and black-box test minimization approach based on LLMs and similarity analysis.

为此，Pan 等人 [75] 提出了 LTM，这是一种基于 LLM 和相似性分析的可扩展黑盒测试最小化方法。

LTM utilizes LLMs to extract test case embeddings and two measures to compute their similarity via two metrics, i.e., Cosine Similarity and Euclidean Distance.

LTM 利用 LLM 提取测试用例嵌入（Embeddings），并通过余弦相似度和欧几里得距离这两个指标来计算它们的相似性。

Based on the similarity, LTM applies a genetic algorithm to optimize the test minimization search space, which identifies the most effective subset of the original test cases within a given testing budget.

基于相似性，LTM 应用遗传算法来优化测试最小化搜索空间，从而在给定的测试预算内识别原始测试用例中最有效的子集。

However, research on LLM-based test minimization remains limited, suggesting the need for further exploration in broader regression testing scenarios such as test case prioritization and selection.

然而，基于 LLM 的测试最小化研究仍然有限，这表明需要在更广泛的回归测试场景（如测试用例优先级排序和选择）中进行进一步探索。

4.9 Test Readability

4.9 测试可读性

Traditional automated unit test generation techniques, particularly search-based tools (e.g., EvoSuite), are capable of producing test cases with high code coverage.

传统的自动单元测试生成技术，特别是基于搜索的工具（如 EvoSuite），能够生成具有高代码覆盖率的测试用例。

While these tools alleviate the burden of writing unit tests manually, the generated test cases often suffer from poor readability, making them difficult for developers to understand, interpret, or maintain.

虽然这些工具减轻了手动编写单元测试的负担，但生成的测试用例往往可读性较差，使得开发人员难以理解、解释或维护。

Thus, improving the readability of automatically generated test cases has therefore emerged as an important research direction.

因此，提高自动生成测试用例的可读性已成为一个重要的研究方向。

To address this, Delijouyi et al. [19] introduce UTGen, which integrates LLMs into the search-based test generation process, thus combining the strengths of both paradigms to generate effective and understandable test cases.

为了解决这个问题，Delijouyi 等人 [19] 推出了 UTGen，它将 LLM 整合到基于搜索的测试生成过程中，从而结合两种范式的优势来生成有效且易于理解的测试用例。

UTGen utilizes LLMs to provide contextually relevant test data, insert informative comments, and suggest suitable variable names via prompt engineering.

UTGen 利用 LLM 通过提示工程提供上下文相关的测试数据，插入信息丰富的注释，并建议合适的变量名称。

Similarly, Biagiola et al. [10] employ LLMs to improve the readability of test cases produced by EvoSuite, specifically focusing on renaming variables and test methods, while preserving functional correctness and coverage.

同样，Biagiola 等人 [10] 利用 LLM 来提高 EvoSuite 生成的测试用例的可读性，特别关注变量和测试方法的重命名，同时保持功能正确性和覆盖率。

In addition, Zhou et al. [138] introduce C3, a readability measurement tool that leverages LLMs to extract context-aware readability requirements from source code, aiming to assess and improve the readability of test inputs, especially for primitive and string types.

此外，Zhou 等人 [138] 推出了 C3，这是一种可读性测量工具，利用 LLM 从源代码中提取上下文感知的可读性需求，旨在评估和提高测试输入的可读性，特别是针对原始类型和字符串类型。

Specifically, C3 captures the expected input context from the tested code and checks the consistency of test inputs, and its integration into EvoSuite enables the generation of more readable test inputs by guiding the test generation with these extracted contexts.

具体而言，C3 捕获被测代码的预期输入上下文并检查测试输入的一致性，将其集成到 EvoSuite 中可以通过这些提取的上下文指导测试生成，从而生成更可读的测试输入。

These studies demonstrate the potential of LLMs in bridging the gap between traditional unit test generation techniques and human-oriented test comprehension.

这些研究展示了 LLM 在弥合传统单元测试生成技术与以人为本的测试理解之间差距的潜力。

Summary of Findings

发现总结

From the perspective of unit testing, our systematic analysis reveals the following key findings:

从单元测试的角度来看，我们的系统分析揭示了以下关键发现：

(1) LLMs have been applied across a wide range of unit testing tasks, demonstrating strong potential in both fundamental and emerging scenarios;

(1) LLM 已应用于广泛的单元测试任务，在基础场景和新兴场景中都展现出强大的潜力；

(2) despite this breadth, research efforts remain heavily concentrated on a few primary tasks, with test case generation and oracle generation being the most studied, accounting for 20% and 19% of the collected papers, respectively;

(2) 尽管广度很大，但研究工作仍主要集中在少数主要任务上，其中测试用例生成和预言生成是被研究最多的，分别占收集论文的 20% 和 19%；

(注：这里原文写的比例和前文4.1节提到的“60%”和“14%”有出入，可能是作者在不同统计口径下的笔误，或者此处20%仅指特定类型的生成。但在阅读时请留意这一数字差异)

(3) although emerging tasks such as test evolution, test smell detection, and test readability are gaining attention due to LLMs' general-purpose reasoning capabilities, they remain relatively underexplored.

(3) 尽管测试演进、测试异味检测和测试可读性等新兴任务因 LLM 的通用推理能力而受到关注，但它们仍相对缺乏探索。

5 ANALYSIS FROM LLM PERSPECTIVE

5 LLM 视角的分析

In this section, we perform an analysis of collected papers from the viewpoint of LLMs, specifically focusing on the distribution of LLMs used, their utilization strategies, and traditional techniques employed in conjunction with LLMs.

在本节中，我们从 LLM 的视角对收集到的论文进行分析，特别关注所使用的 LLM 分布、它们的使用策略以及与 LLM 结合使用的传统技术。

5.1 Adoption and Distribution of LLMs

5.1 LLM 的采用与分布

Figure 6 presents the distribution of LLMs adopted across the collected studies.

图 6 展示了收集到的研究中所采用的 LLM 的分布情况。

The results reveal a clear preference for a few dominant models, particularly those accessible via commercial platforms, while open-source alternatives are gaining traction for their flexibility and adaptability.

结果显示，研究人员明显偏好少数几种主导模型，特别是那些通过商业平台可访问的模型，而开源替代方案因其灵活性和适应性正逐渐受到关注。

Among all models, GPT-3.5 [69], released by OpenAI in November 2022, is the most commonly used LLM in the context of unit testing.

在所有模型中，OpenAI 于 2022 年 11 月发布的 GPT-3.5 [69] 是单元测试背景下使用最频繁的 LLM。

It is trained using a combination of supervised learning and reinforcement learning from human feedback, which enables it to generate fluent, human-like responses.

它结合了监督学习和基于人类反馈的强化学习进行训练，这使其能够生成流畅的、类人的回复。

Primarily featured in OpenAI's ChatGPT platform, its release has significantly advanced research on LLMs in unit testing, thus contributing to its top ranking in our collected studies.

作为 OpenAI ChatGPT 平台的主要特色，它的发布显著推进了单元测试中 LLM 的研究，从而使其在我们收集的研究中排名第一。

Following GPT-3.5, GPT-4 [68] is the second most commonly used LLM in our collected studies.

紧随 GPT-3.5 之后，GPT-4 [68] 是我们收集的研究中使用频率第二高的 LLM。

Released in March 2023, GPT-4 features a much larger parameter count and superior performance across a wide range of tasks than GPT-3.5.

GPT-4 于 2023 年 3 月发布，相比 GPT-3.5，它拥有更大的参数量，并在广泛的任务中表现出更优越的性能。

The third-ranked LLM is CodeLlama [81], which is an open-sourced model developed by Meta.

排名第三的 LLM 是 CodeLlama [81]，这是由 Meta 开发的开源模型。

Due to its open-source nature, researchers can not only perform prompt learning, similar to the GPT series models mentioned above, but also conduct additional training to meet domain-specific requirements.

由于其开源性质，研究人员不仅可以像上述 GPT 系列模型一样进行提示学习，还可以进行额外的训练以满足特定领域的需求。

For example, Shang et al. [86] fine-tune CodeLlama to support three key unit testing tasks, i.e., test generation, test evolution, and assertion generation.

例如，Shang 等人 [86] 微调 CodeLlama 以支持三个关键的单元测试任务，即测试生成、测试演进和断言生成。

In addition, GPT-4o [70], the successor to GPT-4, ranks fourth overall and has already been adopted by 12 studies since its release in May 2024.

此外，GPT-4 的继任者 GPT-4o [70] 总体排名第四，自 2024 年 5 月发布以来已被 12 项研究采用。

Compared with GPT-4, GPT-4o provides notable improvements in response speed, computational efficiency, and long-context reasoning, motivating some studies to explore the potential of CoT in unit testing.

与 GPT-4 相比，GPT-4o 在响应速度、计算效率和长上下文推理方面提供了显著改进，激励了一些研究探索思维链（CoT）在单元测试中的潜力。

Other less frequently used LLMs include a range of open-source models such as DeepSeek-Coder [32], CodeT5 [108], CodeBERT [23], and StarCoder [54].

其他使用较少的 LLM 包括一系列开源模型，如 DeepSeek-Coder [32]、CodeT5 [108]、CodeBERT [23] 和 StarCoder [54]。

These open-source models offer flexibility for customization and experimentation, particularly in scenarios that require fine-tuning or integration with task-specific workflows.

这些开源模型为定制和实验提供了灵活性，特别是在需要微调或与特定任务工作流程集成的场景中。

5.2 Utilization Strategies of LLMs

5.2 LLM 的使用策略

LLMs are typically pre-trained on large-scale corpora to acquire general-purpose knowledge.

LLM 通常在大规模语料库上进行预训练以获取通用知识。

Thus, a fundamental research challenge arises when integrating off-the-shelf LLMs with unit testing: how to effectively adapt general-purpose LLMs to the specialized tasks of unit testing.

因此，在将现成的 LLM 与单元测试集成时，出现了一个基础性研究挑战：如何有效地使通用 LLM 适应单元测试的专门任务。

In this section, we systematically categorize and analyze existing strategies for adapting LLMs to unit testing tasks.

在本节中，我们系统地分类并分析了现有的使 LLM 适应单元测试任务的策略。

5.2.1 Taxonomy Analysis.

5.2.1 分类体系分析

Figure 7 illustrates a hierarchical taxonomy of LLM utilization strategies in unit testing research, annotated with the number of studies adopting each approach.

图 7 展示了单元测试研究中 LLM 使用策略的层级分类体系，并标注了采用每种方法的研究数量。

These adaptation strategies can be broadly grouped into two main categories: model training and prompt engineering, each comprising several specific techniques that vary in complexity, resource demands, and levels of task alignment.

这些适配策略大致可分为两大类：模型训练和提示工程，每一类都包含若干具体技术，它们在复杂性、资源需求和任务对齐程度方面各不相同。

Overall, our analysis reveals that prompt engineering dominates the landscape with 96 studies, reflecting a growing trend toward leveraging LLMs without additional training.

总体而言，我们的分析显示提示工程占据了主导地位，共有 96 项研究，反映了无需额外训练即可利用 LLM 的增长趋势。

Zero-shot prompting (63 studies) and few-shot prompting (17 studies) are the most widely adopted strategies, while more advanced prompting techniques such as chain-of-thought (12), tree-of-thought (2), and guided tree-of-thought (2) highlight the growing sophistication of prompt engineering.

零样本提示（63 项研究）和少样本提示（17 项研究）是采用最广泛的策略，而更高级的提示技术，如思维链（12）、思维树（2）和引导式思维树（2），突显了提示工程日益增长的复杂性。

On the training side, full-parameter fine-tuning remains a significant strategy, with 35 studies adapting model weights for unit testing tasks, indicating continued interest in tailoring model weights for specific unit testing objectives.

在训练方面，全参数微调仍然是一项重要策略，有 35 项研究调整模型权重以用于单元测试任务，表明人们对针对特定单元测试目标定制模型权重仍保持着兴趣。

Although pre-training (2 studies) and reinforcement learning (2 studies) are less commonly used, they demonstrate potential for more targeted or optimization-aware adaptation.

虽然预训练（2 项研究）和强化学习（2 项研究）使用较少，但它们展示了更具针对性或优化感知适配的潜力。

Parameter-efficient fine-tuning (PEFT), while still emerging (4 studies), represents a practical direction toward efficient adaptation with minimal cost.

参数高效微调（PEFT）虽然仍处于起步阶段（4 项研究），但代表了以最小成本实现高效适配的实用方向。

In the following, we delve into each strategy in detail, discussing their technical characteristics, representative approaches, and observed trends in LLM-based unit testing research.

接下来，我们将详细深入探讨每种策略，讨论它们的技术特征、代表性方法以及在基于 LLM 的单元测试研究中观察到的趋势。

5.2.2 Model Training.

5.2.2 模型训练

To support task-specific adaptation of LLMs in unit testing, various training strategies have been proposed and evaluated across the literature.

为了支持 LLM 在单元测试中的特定任务适配，文献中提出并评估了各种训练策略。

These strategies differ in terms of training objectives, resource requirements, and expected performance gains.

这些策略在训练目标、资源需求和预期性能增益方面各不相同。

We categorize and summarize the main approaches into four types: (1) pre-training models on code-specific data, (2) full-parameter fine-tuning, (3) parameter-efficient fine-tuning (PEFT), and (4) reinforcement learning.

我们将主要方法分类并总结为四种类型：（1）在代码特定数据上预训练模型，（2）全参数微调，（3）参数高效微调（PEFT），以及（4）强化学习。

Each of these strategies represents a distinct point in the trade-off space between model generalization, adaptation cost, and task effectiveness.

这些策略中的每一种都代表了模型泛化能力、适配成本和任务有效性之间权衡空间中的一个独特位置。

Model Pre-training.

模型预训练

In the early stage, inspired by the success of LLMs in natural language, researchers attempt to adapt them to programming languages.

在早期阶段，受 LLM 在自然语言领域成功的启发，研究人员尝试将其适应于编程语言。

A straightforward approach is to train code-related LLMs with similar model architectures and training strategies but using code datasets rather than natural language datasets.

一种直接的方法是使用类似的模型架构和训练策略，但使用代码数据集而非自然语言数据集来训练代码相关的 LLM。

For example, to explore the potential of transfer learning in code-related tasks, Mastropaolo et al. [63] pre-train a T5 model with 1.5M Java methods and fine-tune it to generate assertion statements based on given focal methods and test prefixes.

例如，为了探索迁移学习在代码相关任务中的潜力，Mastropaolo 等人 [63] 使用 150 万个 Java 方法预训练了一个 T5 模型，并对其进行调整，以便根据给定的焦点方法和测试前缀生成断言语句。

Furthermore, Rao et al. [80] introduce CAT-LM, a 2.7B-parameter GPT-style LLM pre-trained on aligned focal methods and test cases, making it well-suited for accurate whole-test generation.

此外，Rao 等人 [80] 推出了 CAT-LM，这是一个 27 亿参数的 GPT 风格 LLM，在对齐的焦点方法和测试用例上进行了预训练，使其非常适合准确的全测试生成。

Full-Parameter Fine-tuning.

全参数微调

These techniques utilize supervised learning to train LLMs, thereby adapting them to the downstream test case generation task.

这些技术利用监督学习来训练 LLM，从而使其适应下游的测试用例生成任务。

This is an intuitive and effective way to allow LLMs to refine their pre-trained knowledge representations and weight parameters through limited-scale, domain-specific datasets.

这是一种直观且有效的方法，允许 LLM 通过有限规模的领域特定数据集来优化其预训练的知识表示和权重参数。

In the UT community, the training paradigm is extensively adopted during the early stages of LLMs with millions of parameters, such as T5 and CodeT5.

在单元测试社区，训练范式在拥有数百万参数的 LLM（如 T5 和 CodeT5）的早期阶段被广泛采用。

The prevalent implementation of this paradigm can be primarily attributed to the limited generalization capabilities of early-stage LLMs, which require task-specific training to achieve optimal performance in specialized domains like test case generation and assertion generation [2, 3, 99].

这种范式的普遍实施主要归因于早期 LLM 有限的泛化能力，这些模型需要特定任务的训练才能在测试用例生成和断言生成等专业领域达到最佳性能 [2, 3, 99]。

We observe that out of 30 studies, LLMs are fine-tuned using full parameter fine-tuning techniques to adapt to downstream unit testing tasks.

我们观察到，在 30 项研究中，LLM 使用全参数微调技术进行调整以适应下游单元测试任务。

(注：前文 5.2.1 中提到 35 项，此处为 30 项，可能是不同统计维度的细微差别)

For example, AthenaTest [99] is the first attempt at fine-tuning LLMs for test generation, inspiring a multitude of subsequent studies [2, 88].

例如，AthenaTest [99] 是微调 LLM 用于测试生成的首次尝试，启发了随后的众多研究 [2, 88]。

Parameter-Efficient Fine-tuning (PEFT).

参数高效微调 (PEFT)

However, it is quite computationally expensive and resource-intensive to fine-tune LLMs, particularly for models with billions of parameters, which usually demand substantial GPU resources.

然而，微调 LLM 的计算成本和资源消耗相当大，特别是对于拥有数十亿参数的模型，通常需要大量的 GPU 资源。

To this end, researchers introduce several parameter-efficient fine-tuning strategies in the domain of unit testing, including low-rank adaptation [8, 39, 125], (IA)3 [92] and prompt tuning [92].

为此，研究人员在单元测试领域引入了几种参数高效微调策略，包括低秩适应（LoRA）[8, 39, 125]、(IA)3 [92] 和提示微调（Prompt Tuning）[92]。

These studies aim to adapt LLMs with minimal weight updates, substantially reducing training costs.

这些研究旨在通过最小的权重更新使 LLM 适应任务，从而大幅降低训练成本。

For example, Storhaug et al. [92] conduct the first empirical study to extensively investigate the effectiveness of PEFT strategies, including LoRA, (IA)3, and prompt tuning, for LLM-based unit test generation.

例如，Storhaug 等人 [92] 进行了首项实证研究，广泛调查了 PEFT 策略（包括 LoRA、(IA)3 和提示微调）在基于 LLM 的单元测试生成中的有效性。

The results demonstrate that PEFT techniques, particularly LoRA, achieve comparable performance to full-parameter fine-tuning while significantly reducing computational costs, making task-specific adaptation of LLMs more feasible.

结果表明，PEFT 技术（尤其是 LoRA）取得了与全参数微调相当的性能，同时显著降低了计算成本，使得 LLM 的特定任务适配更加可行。

Such approaches make LLM-based unit testing more accessible and practical, particularly for academic or industrial settings with limited GPU resources.

这些方法使得基于 LLM 的单元测试更加易于获取和实用，特别是对于 GPU 资源有限的学术或工业环境。

Reinforcement Learning (RL).

强化学习 (RL)

In addition to improving fine-tuning efficiency through PEFT, some researchers also employ reinforcement learning to enhance the effectiveness of LLM-based unit testing research [91, 95].

除了通过 PEFT 提高微调效率外，一些研究人员还采用强化学习来增强基于 LLM 的单元测试研究的有效性 [91, 95]。

For example, Steenhoek et al. [91] design a lightweight static analysis-based reward mode to analyze the quality of LLM-generated test cases.

例如，Steenhoek 等人 [91] 设计了一种轻量级的基于静态分析的奖励模式来分析 LLM 生成的测试用例的质量。

They then utilize the reward mode to guide the reinforcement learning process, optimizing LLMs to generate unit tests that maximize expected reward across five code quality metrics.

然后，他们利用该奖励模式指导强化学习过程，优化 LLM 以生成在五个代码质量指标上最大化预期奖励的单元测试。

In addition, Takerngsaksiri et al. [95] propose PyTester, an RL-based test generation approach for Python in test-driven development settings.

此外，Takerngsaksiri 等人 [95] 提出了 PyTester，这是一种在测试驱动开发设置下用于 Python 的基于强化学习的测试生成方法。

5.2.3 Prompt Engineering.

5.2.3 提示工程

In addition to model training, prompt engineering has become a widely adopted strategy to guide LLM behavior for unit testing tasks.

除了模型训练外，提示工程已成为引导 LLM 执行单元测试任务的广泛采用策略。

These strategies range from simple zero-shot prompts to structured, multi-agent reasoning frameworks.

这些策略的范围从简单的零样本提示到结构化的多智能体推理框架。

We categorize existing prompting strategies in the context of unit testing as follows.

我们在单元测试的背景下将现有的提示策略分类如下。

Few-shot Prompting.

少样本提示

Few-shot prompting provides LLMs with several input-output pairs (i.e., demonstrations) before prompting them to generate a response for new, unseen queries.

少样本提示在提示 LLM 为新的、未见过的查询生成响应之前，先为其提供几个输入-输出对（即演示/范例）。

For example, in assertion generation, Nashid et al. [64] retrieve similar focal methods along with their corresponding assertions based on embedding-based or frequency-based similarity, using them as in-context examples for Codex.

例如，在断言生成中，Nashid 等人 [64] 基于基于嵌入或基于频率的相似性检索相似的焦点方法及其相应的断言，将它们用作 Codex 的上下文示例。

In addition, for test generation, to provide high-quality and diverse examples, Ni et al. [65] cluster all candidate examples based on semantic similarity and select one representative from each cluster.

此外，对于测试生成，为了提供高质量和多样化的示例，Ni 等人 [65] 基于语义相似性对所有候选示例进行聚类，并从每个聚类中选择一个代表。

The final prompt is then constructed with five examples, which are ordered by three different strategies: random, ascending, and descending order of cosine similarity with the target focal method.

最终的提示由五个示例构成，这些示例按照三种不同的策略排序：随机、按与目标焦点方法的余弦相似度升序或降序排列。

Zero-shot Prompting.

零样本提示

Zero-shot prompting directly queries LLMs to handle unseen tasks without providing any explicit examples.

零样本提示直接查询 LLM 来处理未见过的任务，而不提供任何明确的示例。

This prompt strategy entirely relies on LLMs' pre-existing knowledge and reasoning capabilities, thus requiring a powerful foundation model (such as ChatGPT) for effective interaction.

这种提示策略完全依赖于 LLM 现有的知识和推理能力，因此需要强大的基础模型（如 ChatGPT）来进行有效交互。

For example, Gu et al. [30] prompt ChatGPT to generate test cases by defining its role as a unit test case generator.

例如，Gu 等人 [30] 通过将 ChatGPT 的角色定义为单元测试用例生成器来提示其生成测试用例。

The prompt also specifies the testing requirements for JDK 1.8 and JUnit 4, as well as a quality requirement to ensure branch coverage in the given focal method.

该提示还指定了 JDK 1.8 和 JUnit 4 的测试要求，以及确保给定焦点方法中分支覆盖率的质量要求。

In oracle generation, Konstantinou et al. [49] design a prompt that begins with a role-playing introduction and a task explanation, concluding with a sentence that specifies the desired format of the LLM's response.

在预言生成中，Konstantinou 等人 [49] 设计了一个提示，以角色扮演介绍和任务解释开始，并以指定 LLM 响应所需格式的句子结束。

Chain-of-Thought (CoT).

思维链 (CoT)

CoT prompting attempts to improve LLMs' reasoning by decomposing complex problems into a sequence of intermediate reasoning steps.

思维链提示试图通过将复杂问题分解为一系列中间推理步骤来提高 LLM 的推理能力。

For example, Yang et al. [113] utilize CoT reasoning to structure the test generation process into two crucial stages: (1) providing LLMs with context information about the focal method to summarize its functionality, thus gaining a deep understanding of the method semantics; (2) guiding LLMs to iteratively generate divergent test cases that maximize branch coverage by incorporating counter-examples, thus allowing LLMs to reason step by step.

例如，Yang 等人 [113] 利用 CoT 推理将测试生成过程构建为两个关键阶段：（1）向 LLM 提供关于焦点方法的上下文信息以总结其功能，从而深入理解方法语义；（2）引导 LLM 通过结合反例迭代生成最大化分支覆盖率的发散测试用例，从而允许 LLM 逐步推理。

Similarly, Wang et al. [109] instruct LLMs via a step-by-step CoT reasoning process, where LLMs sequentially decompose focal methods, generate test cases, and iteratively refine errors, leading to more effective and higher-coverage unit test generation.

同样，Wang 等人 [109] 通过逐步的 CoT 推理过程指导 LLM，其中 LLM 依次分解焦点方法、生成测试用例并迭代修正错误，从而实现更有效和更高覆盖率的单元测试生成。

For example, during the method decomposition, LLMs first summarize the focal method's functionality, then recite the meaning and usage of all invoked non-local variables and methods, and finally break down the method into step-based meaningful slices.

例如，在方法分解过程中，LLM 首先总结焦点方法的功能，然后复述所有调用的非局部变量和方法的含义。

义及用法，最后将方法分解为基于步骤的有意义切片。

Tree-of-Thought (ToT).

思维树 (ToT)

This is an advanced reasoning technique that enhances the problem-solving capabilities of LLMs by structuring reasoning as a tree-based process.

这是一种先进的推理技术，通过将推理结构化为基础于树的过程来增强 LLM 的问题解决能力。

Unlike CoT, which follows a linear step-by-step reasoning chain, ToT enables LLMs to iteratively explore and evaluate multiple reasoning paths, discarding suboptimal ones and converging on a more robust solution.

与遵循线性逐步推理链的 CoT 不同，ToT 使 LLM 能够迭代地探索和评估多条推理路径，舍弃次优路径并收敛到更稳健的解决方案。

For example, in the context of unit test generation, Ouédraogo et al. [71] employ ToT by simulating a collaborative process among three virtual software testing experts, where each expert independently proposes test cases, exchanges them with the group for evaluation, and refines them iteratively based on collaborative insights.

例如，在单元测试生成的背景下，Ouédraogo 等人 [71] 通过模拟三位虚拟软件测试专家之间的协作过程来采用 ToT，其中每位专家独立提出测试用例，并在组内交换以进行评估，并根据协作见解迭代地完善它们。

This approach enhances test case quality by leveraging multiple reasoning pathways, ensuring coverage of typical use cases, edge cases, and exception-handling scenarios before merging the finalized test suite.

这种方法通过利用多种推理路径来提高测试用例质量，确保在合并最终测试套件之前覆盖典型的用例、边界情况和异常处理场景。

Guided Tree-of-Thoughts (GToT).

引导式思维树 (GToT)

GToT is an enhanced version of ToT that incorporates structured, step-by-step guidance mechanisms to steer the reasoning process toward more systematic and optimal solutions.

GToT 是 ToT 的增强版本，它结合了结构化的、循序渐进的指导机制，以引导推理过程走向更系统和最优的解决方案。

For example, Ouédraogo et al. [71] extend ToT by explicitly instructing LLMs to follow a structured framework, including method extraction, functional test generation, edge case identification, and iterative refinement.

例如，Ouédraogo 等人 [71] 扩展了 ToT，明确指示 LLM 遵循一个结构化框架，包括方法提取、功能测试生成、边界情况识别和迭代优化。

In this setting, three virtual experts not only propose and refine test cases but also systematically analyze method signatures and exception-handling scenarios before merging their insights into a complete JUnit 4 test suite.

在这种设置下，三位虚拟专家不仅提出和完善测试用例，还在将见解合并为完整的 JUnit 4 测试套件之前，系统地分析方法签名和异常处理场景。

Besides, by enforcing a well-defined reasoning structure, GToT improves test coverage, enhances code correctness, and minimizes test smells compared to the standard ToT approach.

此外，通过强制执行定义明确的推理结构，与标准 ToT 方法相比，GToT 提高了测试覆盖率，增强了代码正确性，并最大限度地减少了测试异味。

5.3 Enhancing LLMs via Traditional Techniques

5.3 通过传统技术增强 LLM

Although LLMs have demonstrated remarkable capabilities in various unit testing tasks, the complexity of unit testing poses unique challenges that often exceed the standalone capabilities of LLMs.

尽管 LLM 在各种单元测试任务中展现了卓越的能力，但单元测试的复杂性构成了独特的挑战，往往超出了 LLM 的独立能力。

To bridge this gap, recent studies have explored hybrid approaches that integrate LLMs with traditional software engineering techniques, aiming to harness the strengths of both paradigms.

为了弥补这一差距，最近的研究探索了将 LLM 与传统软件工程技术相结合的混合方法，旨在利用这两种范式的优势。

5.3.1 Taxonomy Analysis.

5.3.1 分类体系分析

Figure 8 presents a taxonomy of traditional techniques integrated with LLMs in the context of unit testing, annotated with the number of studies adopting each method.

图 8 展示了在单元测试背景下与 LLM 集成的传统技术的分类体系，并标注了采用每种方法的研究数量。

These studies fall into six major categories based on the supporting techniques employed: program repair (20 studies), program analysis (18 studies), SBST tools (7 studies), information retrieval (5 studies), mutation testing (3 studies), differential testing (3 studies).

这些研究根据所采用的支持技术分为六大类：程序修复（20 项研究）、程序分析（18 项研究）、SBST 工具（7 项研究）、信息检索（5 项研究）、变异测试（3 项研究）和差分测试（3 项研究）。

Among them, program repair and program analysis are the most frequently adopted techniques, indicating a strong emphasis on improving the correctness and contextual relevance of LLM-generated test cases.

其中，程序修复和程序分析是采用最频繁的技术，表明研究重点在于提高 LLM 生成的测试用例的正确性和上下文相关性。

Program repair techniques are primarily used to fix compilation and runtime errors in generated test code, while program analysis enhances prompt relevance by extracting meaningful structural or semantic information.

程序修复技术主要用于修复生成的测试代码中的编译和运行时错误，而程序分析则通过提取有意义的结构或语义信息来增强提示的相关性。

In the following, we examine each category in detail, discussing its technical motivation, representative approaches, and the observed impact on LLM-based unit testing research.

接下来，我们将详细检查每个类别，讨论其技术动机、代表性方法以及对基于 LLM 的单元测试研究产生的观察影响。

LLMs with Program Analysis.

LLM 结合程序分析

To facilitate LLMs' comprehension of the unit under test, an intuitive strategy is to provide them with all relevant information as prompts to the greatest extent possible.

为了促进 LLM 对被测单元的理解，一种直观的策略是尽可能在提示中提供所有相关信息。

However, this strategy inevitably introduces irrelevant noise in lengthy prompts and hampers LLMs' ability to extract essential semantic information due to the lost-in-the-middle phenomenon.

然而，这种策略不可避免地会在冗长的提示中引入不相关的噪声，并由于“迷失在中间（lost-in-the-middle）”现象而阻碍 LLM 提取关键语义信息的能力。

To this end, researchers utilize program analysis techniques to represent the unit under test and its context information more effectively, thus increasing LLMs' ability to comprehend source code accurately.

为此，研究人员利用程序分析技术更有效地表示被测单元及其上下文信息，从而提高 LLM 准确理解源代码的能力。

For example, in the context of test generation, TELPA [113] performs backward and forward program analysis to feed LLMs with a limited number of relevant invocation methods rather than the whole focal class.

例如，在测试生成的背景下，TELPA [113] 执行向后和向前的程序分析，向 LLM 提供有限数量的相关调用方法，而不是整个焦点类。

Chen et al. [13] conduct program slicing to extract representative API usage and perform dependency analysis to build object construction graphs, which can provide guidance for LLM to generate meaningful test cases.

Chen 等人 [13] 进行程序切片以提取代表性的 API 用法，并执行依赖分析以构建对象构造图，这可以为 LLM 生成有意义的测试用例提供指导。

Wang et al. [109] parse the focal method via static analysis to extract context information for LLMs to understand the usage and structure of the focal method.

Wang 等人 [109] 通过静态分析解析焦点方法，为 LLM 提取上下文信息，以理解焦点方法的用法和结构。

In test evolution, Yaraghi et al. [116] utilize static analysis via the Spoon library to construct method-level and class-level call graphs of the focal class, enabling the identification and prioritization of relevant repair contexts for LLMs.

在测试演进中，Yaraghi 等人 [116] 利用 Spoon 库通过静态分析构建焦点类的方法级和类级调用图，从而能够为 LLM 识别和优先考虑相关的修复上下文。

LLMs with Information Retrieval.

LLM 结合信息检索

LLMs are typically trained on vast datasets, retaining learned knowledge in the form of parameters up to a fixed cutoff point.

LLM 通常在海量数据集上进行训练，将学到的知识以参数形式保留到固定的截止时间点。

While fine-tuning provides a viable mechanism for incremental knowledge integration, frequently updating LLMs with the latest data is often impractical due to the vast number of model parameters.

虽然微调为增量知识集成提供了一种可行的机制，但由于模型参数数量庞大，频繁用最新数据更新 LLM 往往是不切实际的。

As a result, when generating test cases, LLMs may struggle with outdated knowledge and a lack of project-specific context.

结果，在生成测试用例时，LLM 可能会受困于过时的知识和项目特定上下文的缺失。

In particular, LLMs may lack awareness of new knowledge (such as updated libraries or frameworks) after their last training date and fail to incorporate critical project-level information (such as method invocation of the unit under test), which reduces the quality of generated test cases and leads to hallucinations.

特别是，LLM 可能缺乏对其最后训练日期之后的新知识（如更新的库或框架）的认知，并且无法整合关键的项目级信息（如被测单元的方法调用），这会降低生成的测试用例的质量并导致幻觉。

To address this challenge, researchers leverage information retrieval techniques to dynamically provide LLMs with relevant, up-to-date context information through prompts.

为了解决这一挑战，研究人员利用信息检索技术通过提示动态地为 LLM 提供相关的、最新的上下文信息。

For example, Nashid et al. [64] construct prompts for assertion generation by retrieving demonstrations similar to the test prefix based on embedding or frequency analysis.

例如，Nashid 等人 [64] 通过基于嵌入或频率分析检索与测试前缀相似的演示（范例），来构建断言生成的提示。

LLMs with Program Repair.

LLM 结合程序修复

Although LLMs have achieved impressive performance on unit testing, generating syntactically correct test code remains challenging, often resulting in compilation and runtime errors.

尽管 LLM 在单元测试方面取得了令人印象深刻的性能，但生成语法正确的测试代码仍然具有挑战性，通常会导致编译和运行时错误。

To address this, researchers have developed various strategies to identify and repair errors in generated test cases automatically.

为了解决这个问题，研究人员开发了各种策略来自动识别和修复生成的测试用例中的错误。

Broadly, these program repair techniques fall into two categories: dynamic feedback-based repair [66] and static pattern-based repair [2].

从广义上讲，这些程序修复技术分为两类：基于动态反馈的修复 [66] 和基于静态模式的修复 [2]。

For example, Yuan et al. [121] adopt a validate-and-repair paradigm, iteratively refining buggy test cases by re-prompting ChatGPT with compilation error messages.

例如，Yuan 等人 [121] 采用验证并修复的范式，通过用编译错误消息重新提示 ChatGPT 来迭代优化有缺陷的测试用例。

In contrast, inspired by the advancements of traditional templates program repair [130], Gu et al. [30] design five expert-informed repair templates to fix common compilation errors (such as syntax, import, and scope errors) as well as runtime errors in generated test cases.

相比之下，受传统模板程序修复进展 [130] 的启发，Gu 等人 [30] 设计了五个专家告知的修复模板，以修复生成的测试用例中常见的编译错误（如语法、导入和作用域错误）以及运行时错误。

LLMs with Mutation Testing.

LLM 结合变异测试

Most LLM-based unit testing approaches have demonstrated promising performance in maximizing code coverage.

大多数基于 LLM 的单元测试方法在最大化代码覆盖率方面表现出有前景的性能。

However, while code coverage is widely regarded as a useful metric, its correlation with actual bug detection capability remains weak.

然而，虽然代码覆盖率被广泛认为是一个有用的指标，但它与实际缺陷检测能力的相关性仍然很弱。

Thus, researchers employ mutation testing to simulate real bugs to optimize the bug detection ability of test cases generated by LLMs.

因此，研究人员采用变异测试来模拟真实缺陷，以优化 LLM 生成的测试用例的缺陷检测能力。

For example, MuTAP [17] prompts LLMs to generate initial test cases via zero-shot and few-shot learning.

例如，MuTAP [17] 通过零样本和少样本学习提示 LLM 生成初始测试用例。

MuTAP then applies mutation testing to assess how well the generated test cases detect faults (i.e., kill mutants).

然后，MuTAP 应用变异测试来评估生成的测试用例检测故障（即杀死变异体）的能力。

If any mutants survive, MuTAP iteratively augments the initial prompt with surviving mutants to re-prompt LLMs to generate improved test cases until all mutants are detected or no further improvements can be made.

如果有任何变异体存活，MuTAP 会迭代地用存活的变异体增强初始提示，重新提示 LLM 生成改进的测试用例，直到检测到所有变异体或无法进一步改进为止。

Similarly, ACH [24] attempts to construct mutants that represent faults that are both relevant to the issue at hand and undetected by existing test cases.

类似地，ACH [24] 试图构建变异体，这些变异体代表了既与当前问题相关又未被现有测试用例检测到的故障。

These mutants are then used as prompts for LLMs to generate new test cases capable of detecting them.

然后，这些变异体被用作提示，让 LLM 生成能够检测它们的新测试用例。

LLMs with Differential Testing.

LLM 结合差分测试

Differential testing attempts to detect inconsistencies or bugs by running multiple implementations of a program with the same inputs and comparing their outputs.

差分测试试图通过使用相同的输入运行程序的多个实现并比较它们的输出来检测不一致或错误。

Given a program under test, Li et al. [55] prompt ChatGPT to infer the program's intention and ask ChatGPT to generate multiple compilable programs that share the same intention as the original.

给定一个被测程序，Li 等人 [55] 提示 ChatGPT 推断程序的意图，并要求 ChatGPT 生成多个与原始程序具有相同意图的可编译程序。

They then apply differential testing between the program under test and the ChatGPT-generated programs, using the inferred intention to identify failure-inducing test cases.

然后，他们在被测程序和 ChatGPT 生成的程序之间应用差分测试，利用推断出的意图来识别导致失败的测试用例。

Similarly, Liu et al. [57] first utilize LLMs to generate multiple program variants and test inputs.

同样，Liu 等人 [57] 首先利用 LLM 生成多个程序变体和测试输入。

They then execute test inputs on both the program under test and its variants to construct the test oracle.

然后，他们在被测程序及其变体上执行测试输入以构建测试预言。

Zhong et al. [137] prompt LLMs to generate high-quality unit test cases for JSON library and adopt differential testing to detect potential bugs by comparing the results from fastjson and fastjson2.

Zhong 等人 [137] 提示 LLM 为 JSON 库生成高质量的单元测试用例，并采用差分测试通过比较 fastjson 和 fastjson2 的结果来检测潜在错误。

LLMs with SBST Tools.

LLM 结合 SBST（基于搜索的软件测试）工具

Another line of work combines LLMs with traditional search-based software testing (SBST) tools to leverage their respective strengths.

另一类工作将 LLM 与传统的基于搜索的软件测试（SBST）工具相结合，以利用它们各自的优势。

For example, TELPA [113] utilizes the traditional search-based tool Pynguin to generate initial test cases for easily reachable branches, which are then utilized to construct prompts for LLMs, allowing them to address harder-to-cover branches further.

例如，TELPA [113] 利用传统的基于搜索的工具 Pynguin 为容易到达的分支生成初始测试用例，然后利用这些用例构建 LLM 的提示，使其能够进一步处理更难覆盖的分支。

Similarly, when Pynguin fails to increase test coverage within a given testing budget, CODAMOSA [51] invokes LLMs to generate new seed tests, which are used to resume the generation process of Pynguin.

同样，当 Pynguin 在给定的测试预算内无法增加测试覆盖率时，CODAMOSA [51] 调用 LLM 生成新的种子测试，用于恢复 Pynguin 的生成过程。

CoverUp [77] further extends CodaMosa by incorporating feedback-driven refinements based on coverage information to improve test case quality iteratively.

CoverUp [77] 通过结合基于覆盖率信息的反馈驱动改进，进一步扩展了 CodaMosa，以迭代地提高测试用例质量。

6 CHALLENGES AND OPPORTUNITIES

6 挑战与机遇

In this section, we discuss key challenges and highlight potential research directions across both technical and practical dimensions.

在本节中，我们讨论了关键挑战，并从技术和实践两个维度重点介绍了潜在的研究方向。

6.1 Challenges

6.1 挑战

Despite the promising progress in applying LLMs to unit testing, our survey reveals several persistent challenges that hinder their broader adoption and effectiveness.

尽管将 LLM 应用于单元测试取得了可喜的进展，但我们的调查揭示了几个持续存在的挑战，阻碍了其更广泛的采用和有效性。

These challenges span from technical limitations in modeling complex software units to broader concerns around dataset quality, bug detection reliability, and model deployability.

这些挑战涵盖了从对复杂软件单元建模的技术限制，到围绕数据集质量、Bug 检测可靠性和模型可部署性等更广泛的担忧。

We summarize the key challenges observed in the literature as follows.

我们将文献中观察到的关键挑战总结如下。

6.1.1 Challenges in Testing Complex Units Under Test.

6.1.1 测试复杂被测单元的挑战

Unit testing aims to validate the functionality of individual software units in isolation.

单元测试旨在隔离验证单个软件单元的功能。

However, in real-world software systems, these units rarely operate independently, and they often rely on complex interactions with other functions, classes, or modules.

然而，在现实世界的软件系统中，这些单元很少独立运行，它们通常依赖于与其他函数、类或模块的复杂交互。

This interconnected nature of modern codebases introduces significant challenges for LLM-based unit testing, particularly in understanding and reasoning over the broader execution context.

现代代码库的这种互连性质给基于 LLM 的单元测试带来了重大挑战，特别是在理解和推理更广泛的执行上下文方面。

Our analysis reveals three development stages in how contextual information is incorporated into LLMs for unit testing.

我们的分析揭示了将上下文信息纳入单元测试 LLM 的三个发展阶段。

From our collected papers, we observe that early studies typically feed the focal method into LLMs, prompting them to generate corresponding test cases in the form of machine translation.

从我们收集的论文中观察到，早期的研究通常将焦点方法输入 LLM，提示它们以机器翻译的形式生成相应的测试用例。

However, this strategy lacks critical contextual details, such as variable declarations and invoked methods, which are often necessary to produce valid and meaningful test cases.

然而，这种策略缺乏关键的上下文细节，例如变量声明和被调用的方法，而这些通常是生成有效且有意义的测试用例所必需的。

Later, with the advent of larger models featuring expanded input windows, researchers incorporate the entire focal class as additional context to capture inter-class dependencies.

随后，随着具有扩展输入窗口的更大模型的出现，研究人员将整个焦点类作为附加上下文纳入，以捕获类内依赖关系。

Although this strategy enriches the available information, it also introduces excessive input size and noise, making it difficult for LLMs to identify and focus on relevant elements.

尽管这种策略丰富了可用信息，但它也引入了过大的输入尺寸和噪声，使得 LLM 难以识别和聚焦于相关元素。

Moreover, even this expanded input fails to capture cross-file or cross-module dependencies that are common in large-scale projects.

此外，即使是这种扩展后的输入也无法捕获大型项目中常见的跨文件或跨模块依赖关系。

Recently, some efforts have sought to mitigate these issues by leveraging program analysis techniques to extract semantically related methods, function call relationships, and usage contexts of the unit under test.

最近，一些工作试图通过利用程序分析技术来提取语义相关的方法、函数调用关系以及被测单元的使用上下文，从而缓解这些问题。

While this strategy improves the relevance of provided information, striking a balance between sufficient context and manageable input size remains an open challenge, especially in repository-level unit testing scenarios.

虽然这种策略提高了所提供信息的相关性，但在仓库级单元测试场景中，如何在充足的上下文和可管理的输入大小之间取得平衡仍然是一个开放性的挑战。

Future research may explore more advanced static and dynamic analysis techniques (e.g., dataflow analysis) to better identify and structure relevant context.

未来的研究可以探索更先进的静态和动态分析技术（例如数据流分析），以更好地识别和构建相关上下文。

Besides, program reduction techniques, such as dead code elimination, can be employed to remove source code that is irrelevant to the behavior of the unit under test, thereby reducing complexity while preserving the essential functionality.

此外，可以采用程序约减技术（如死代码消除）来移除与被测单元行为无关的源代码，从而在保留基本功能的同时降低复杂性。

6.1.2 Challenges in Detecting Real-world Software Bugs.

6.1.2 检测现实世界软件 Bug 的挑战

A fundamental purpose of unit testing is to detect individual component bugs at the early stage of software development, thereby improving software reliability.

单元测试的一个根本目的是在软件开发早期检测单个组件的 Bug，从而提高软件可靠性。

However, current evaluation metrics for unit testing primarily emphasize generation accuracy, code coverage, and defect detection capabilities, often overlooking its impact on bug detection.

然而，当前的单元测试评估指标主要强调生成准确性、代码覆盖率和缺陷检测能力，往往忽视了其对 Bug 检测的影响。

As illustrated in Figure 9, we summarize the distribution of evaluation metrics adopted in test generation studies based on our collected papers.

如图 9 所示，我们根据收集到的论文总结了测试生成研究中采用的评估指标分布。

It can be observed that the vast majority of studies evaluate the quality of generated test cases using code coverage (38 papers), pass rate (27 papers), and compilation rates (21 papers).

可以观察到，绝大多数研究使用代码覆盖率（38 篇）、通过率（27 篇）和编译率（21 篇）来评估生成的测试用例的质量。

There are only eight papers to identify whether the generated test cases can uncover real-world bugs.

只有 8 篇论文识别了生成的测试用例是否能发现现实世界的 Bug。

After a careful analysis, we conclude that the challenges of existing unit testing tools in detecting real-world bugs are threefold.

经过仔细分析，我们得出结论，现有单元测试工具在检测现实世界 Bug 方面面临三重挑战。

First, the primary reason why current studies lack sufficient focus on bug detection is the limitation of the dataset.

首先，当前研究缺乏对 Bug 检测足够关注的主要原因是数据集的局限性。

For example, existing studies typically utilize projects from GitHub as evaluation datasets, which usually fail to include real-world bugs.

例如，现有研究通常利用 GitHub 上的项目作为评估数据集，这些项目通常不包含现实世界的 Bug。

(注：这里指的是直接从 GitHub 拉取的最新代码通常被认为是“正确”的版本，不包含已知未修复的 Bug)

These datasets guide researchers to evaluate the correctness of test cases (such as syntactic correctness) rather than bug detection capabilities.

这些数据集引导研究人员评估测试用例的正确性（如语法正确性），而不是 Bug 检测能力。

To address this, six papers employ mutation testing to simulate bugs for evaluation.

为了解决这个问题，有 6 篇论文采用变异测试来模拟 Bug 进行评估。

To address this gap, we call for the development of more comprehensive and realistic datasets tailored for evaluating LLM-driven unit testing approaches.

为了填补这一空白，我们呼吁开发更全面、更现实的数据集，专门用于评估 LLM 驱动的单元测试方法。

Second, oracle generation has long remained a persistent challenge in various software testing activities, including unit testing.

其次，预言生成长期以来一直是各种软件测试活动（包括单元测试）中的一个持续挑战。

For example, Yang et al. [114] report that GPT-4 generates valid unit test cases for only 15.74% of bugs (65/413), and among them, only 40% (39/65) are correctly detected, resulting in a final detection rate of 9.44% (39/413).

例如，Yang et al. [114] 报告称，GPT-4 仅为 15.74% 的 Bug (65/413) 生成了有效的单元测试用例，而在

这些用例中，只有 40% (39/65) 被正确检测到，导致最终检测率仅为 9.44% (39/413)。

Similarly, Zhang et al. [134] demonstrate that assertions generated by CodeT5 are able to detect only 15 bugs for 835 bugs from Defects4J-v2.0.

同样，Zhang 等人 [134] 证明，CodeT5 生成的断言在 Defects4J-v2.0 的 835 个 Bug 中只能检测到 15 个。

The third challenge arises from the complexities inherent in real-world bug detection scenarios.

第三个挑战源于现实世界 Bug 检测场景中固有的复杂性。

Although some studies (e.g., Yang et al. [114]) attempt to evaluate bug detection, their assessment scenarios lack realism.

尽管一些研究（如 Yang 等人 [114]）试图评估 Bug 检测，但他们的评估场景缺乏现实性。

For example, they typically generate test cases on the fixed program and run them on the buggy program.

例如，他们通常在修复后的程序上生成测试用例，然后在有 Bug 的程序上运行它们。

However, in practice, developers expect unit testing tools to uncover bugs in the buggy program and its fixed version is not available at that time.

然而，在实践中，开发人员期望单元测试工具能在有 Bug 的程序中发现 Bug，而当时其修复版本是不可用的。

Besides, test cases generated by LLMs on the fixed program version may implicitly contain some information related to the patch of the bug, which should be unknown during bug finding and results in information leakage issues.

此外，LLM 在修复后的程序版本上生成的测试用例可能隐式包含了一些与 Bug 补丁相关的信息，而这些信息在 Bug 查找过程中应该是未知的，这会导致信息泄露问题。

Therefore, to be consistent with the real-world usage scenario, test cases should be generated on the buggy program version instead of the bug-fixed program version.

因此，为了与现实世界的使用场景保持一致，应该在有 Bug 的程序版本上生成测试用例，而不是在 Bug 修复后的程序版本上生成。

A crucial challenge lies in the issue of false positives, i.e., determining whether a failing test case is caused by its assertion error or by an actual bug in the program.

一个关键挑战在于假阳性（误报）问题，即确定一个失败的测试用例是由其断言错误引起的，还是由程序中的实际 Bug 引起的。

These false positives hinder the practical application of unit testing techniques in real-world scenarios.

这些假阳性阻碍了单元测试技术在现实场景中的实际应用。

6.1.3 Challenges in Developing Unit Testing-driven LLMs.

6.1.3 开发单元测试驱动型 LLM 的挑战

As illustrated in Figure 6, GPT-series models are the most widely adopted in the domain of unit testing.

如图 6 所示，GPT 系列模型是单元测试领域采用最广泛的模型。

However, the proprietary and black-box nature of these commercial models poses several challenges for real-world deployment, including concerns related to deployment environments and potential privacy breaches.

然而，这些商业模型的专有和黑盒性质给现实世界的部署带来了一些挑战，包括与部署环境和潜在隐私泄露相关的担忧。

For example, due to concerns regarding data privacy, when designing LLM-based unit testing tools, most organizations tend to avoid using commercial LLMs in production workflows.

例如，出于对数据隐私的担忧，在设计基于 LLM 的单元测试工具时，大多数组织倾向于避免在生产工作流程中使用商业 LLM。

Instead, they often prefer open-source alternatives that can be fine-tuned with domain-specific data in controlled environments.

相反，他们通常更喜欢开源替代方案，这些方案可以在受控环境中使用特定领域的数据进行微调。

However, the computational cost of fine-tuning large-scale models remains prohibitive for most companies, leading them to rely on medium-sized models.

然而，微调大规模模型的计算成本对于大多数公司来说仍然过高，导致他们依赖于中等规模的模型。

However, these models often fail to match the performance of their commercial counterparts, as observed in our collected studies.

然而，正如我们在收集的研究中观察到的那样，这些模型的性能往往无法与商业模型相媲美。

Among existing LLMs, CAT-LM [80] stands out as the only model specifically designed for test case generation in relation to the method under test.

在现有的 LLM 中，CAT-LM [80] 脱颖而出，它是唯一专门为基于被测方法生成测试用例而设计的模型。

However, it still relies on traditional NLP pre-training objectives and has a relatively smaller parameter scale compared to other models.

然而，它仍然依赖于传统的 NLP 预训练目标，并且与其他模型相比，其参数规模相对较小。

These observations point to an urgent need for the development of unit testing-oriented LLMs.

这些观察结果表明，迫切需要开发面向单元测试的 LLM。

Therefore, we advocate for the development of more unit test-oriented LLMs.

因此，我们倡导开发更多面向单元测试的 LLM。

However, developing such LLMs presents inherent challenges due to the unique characteristics of unit testing.

然而，由于单元测试的独特性，开发此类 LLM 面临着固有的挑战。

First, while software repositories host abundant production code, the number of high-quality, labeled unit test cases remains limited, posing a significant barrier to large-scale pre-training.

首先，虽然软件仓库拥有丰富的生产代码，但高质量、有标签的单元测试用例数量仍然有限，这对大规模预训练构成了重大障碍。

Second, designing pre-training objectives tailored specifically to unit testing, such as coverage maximization and assertion inference, requires further research and innovation to ensure models can generalize effectively in real-world testing scenarios.

其次，设计专门针对单元测试的预训练目标（如覆盖率最大化和断言推断），需要进一步的研究和创新，以确保模型能够在现实世界的测试场景中有效泛化。

6.2 Opportunities

6.2 机遇

6.2.1 Opportunities in Developing End-to-End Testing-and-Debugging Framework.

6.2.1 开发端到端测试与调试框架的机遇

While unit testing primarily aims to detect software bugs at the early stage of development, and program debugging focuses on automatically analyzing and fixing such bugs, the two tasks are inherently connected and mutually reinforcing.

虽然单元测试主要旨在开发早期检测软件 Bug，而程序调试侧重于自动分析和修复这些 Bug，但这两项任务在本质上是相互关联且相辅相成的。

In particular, high-quality unit test cases not only expose defects, but also support key debugging activities such as root cause analysis, fault localization, patch generation, and patch validation.

特别是，高质量的单元测试用例不仅能暴露缺陷，还能支持关键的调试活动，如根本原因分析、故障定位、补丁生成和补丁验证。

Despite this close relationship, prior work has largely treated unit testing and program debugging as separate research areas, limiting opportunities for integrated solutions and cross-domain enhancement.

尽管关系密切，但先前的工作在很大程度上将单元测试和程序调试视为独立的研究领域，限制了集成解决方案和跨领域增强的机会。

In the future, with LLMs serving as the backbone, their powerful understanding and reasoning capabilities present an opportunity to bridge this gap, enabling the integration of a fully automated framework for unit testing and program debugging.

在未来，随着 LLM 作为骨干，其强大的理解和推理能力为弥合这一差距提供了机会，从而实现单元测试和程序调试的全自动框架集成。

In such a framework, LLMs could first be used to automatically generate test cases and meaningful oracles to uncover potential defects in real-world programs.

在这样的框架中，首先可以使用 LLM 自动生成测试用例和有意义的预言，以发现现实世界程序中的潜在缺陷。

Upon identifying faults, program debugging components could be triggered to perform tasks such as root cause analysis, fault localization, and patch generation.

一旦识别出故障，就可以触发程序调试组件来执行诸如根本原因分析、故障定位和补丁生成等任务。

These patches can then be validated against test cases and reintegrated into the unit testing process, forming a self-reinforcing feedback loop that continuously enhances both testing and debugging effectiveness.

然后，可以根据测试用例验证这些补丁，并将其重新集成到单元测试过程中，形成一个自我强化的反馈循环，不断增强测试和调试的有效性。

This direction attempts to unify two well-known research domains that are often developed in isolation into an interactive pipeline thereby benefiting the whole software development lifecycle.

这一方向试图将通常孤立发展的两个知名研究领域统一为一个交互式管道，从而使整个软件开发生命周期受益。

More importantly, such integration not only broadens the application scope of these two research areas but also boosts the capabilities of recent LLMs in advancing software quality assurance.

更重要的是，这种集成不仅拓宽了这两个研究领域的应用范围，还提升了近期 LLM 在推进软件质量保证方面的能力。

6.2.2 Opportunities in Exploring More Unit Testing Tasks.

6.2.2 探索更多单元测试任务的机遇

As discussed in Section 2.2, a majority of collected papers are concentrated on a limited number of unit testing scenarios, particularly test case generation and oracle generation.

正如 2.2 节所讨论的，大多数收集到的论文集中在有限数量的单元测试场景上，特别是测试用例生成和预言生成。

However, several important unit testing scenarios remain largely underexplored in the context of LLMs.

然而，在 LLM 的背景下，几个重要的单元测试场景在很大程度上仍未得到充分探索。

For example, there is only one paper on the use of LLMs in test minimization, and still no research on test prioritization and test selection.

例如，关于在测试最小化中使用 LLM 的论文只有一篇，而关于测试优先级排序和测试选择的研究仍然空白。

These tasks typically differ from more widely studied ones, posing unique challenges when integrated with LLMs.

这些任务通常不同于研究更广泛的任务，在与 LLM 集成时提出了独特的挑战。

For example, test case prioritization requires analyzing and ranking tens of thousands of test cases based on various features to determine an optimal execution order.

例如，测试用例优先级排序需要根据各种特征对成千上万个测试用例进行分析和排序，以确定最佳执行顺序。

Unlike the well-explored test generation task, where LLMs address it as a machine translation task by mapping a focal method to its corresponding test code, test prioritization feeds an entire test suite into LLMs, which far exceeds the context window limitations of LLMs.

与探索已久的测试生成任务不同（LLM 将其作为机器翻译任务，通过将焦点方法映射到其相应的测试代码来解决），测试优先级排序将整个测试套件输入 LLM，这远远超出了 LLM 的上下文窗口限制。

To address this issue, a promising direction is to combine LLMs with traditional unit testing techniques.

为了解决这个问题，一个有前途的方向是将 LLM 与传统的单元测试技术相结合。

Specifically, it is promising to leverage LLMs' code semantic understanding capabilities by encoding test cases into vector representations, which are then combined with similarity-based test prioritization algorithms.

具体而言，利用 LLM 的代码语义理解能力，将测试用例编码为向量表示，然后将其与基于相似性的测试

优先级排序算法相结合，是有前景的。

6.2.3 Opportunities in Integrating Multi-modal Context Information.

6.2.3 整合多模态上下文信息的机遇

From our collected papers, in the early stage of LLM-based unit testing research [2, 99], LLMs are typically provided with the focal method under test and tasked with directly generating the corresponding test case.

根据我们收集的论文，在基于 LLM 的单元测试研究的早期阶段 [2, 99]，通常向 LLM 提供受测焦点方法，并任务其直接生成相应的测试用例。

Later, due to the dependencies among various units, studies attempt to adopt traditional techniques, such as program analysis and information retrieval, to enrich prompts with more relevant contextual information, such as invoked functions and variable definitions.

后来，由于各个单元之间的依赖关系，研究试图采用传统技术（如程序分析和信息检索）来用更相关的上下文信息（如被调用的函数和变量定义）丰富提示。

However, most existing studies remain largely focused on providing accurate code-level context, often overlooking other valuable input types, such as documentation, API references, and bug reports.

然而，大多数现有研究仍主要集中在提供准确的代码级上下文上，往往忽略了其他有价值的输入类型，如文档、API 参考和 Bug 报告。

Given the advanced natural language understanding capabilities of LLMs, there is great potential to extract semantic intent and behavioral expectations from these textual sources, which may not be explicitly reflected in the code.

鉴于 LLM 先进的自然语言理解能力，从这些文本源中提取语义意图和行为预期具有巨大的潜力，而这些在代码中可能没有明确反映。

Moreover, for certain types of software, such as Android applications, the multimodal capabilities of LLMs can be further explored to process diverse inputs, such as graphical user interfaces (GUIs), to support richer and more effective unit testing.

此外，对于某些类型的软件（如 Android 应用程序），可以进一步探索 LLM 的多模态能力来处理多样化的输入（如图形用户界面 GUI），以支持更丰富、更有效的单元测试。

We believe that the integration of source code and other multi-modal contexts (such as textual and visual information) represents a promising direction for improving the completeness and accuracy of LLM-driven unit testing research.

我们相信，源代码与其他多模态上下文（如文本和视觉信息）的整合代表了提高 LLM 驱动的单元测试研究的完整性和准确性的一个有前景的方向。

7 CONCLUSION

7 结论

Unit testing is a crucial and even mandatory practice in modern software engineering, facilitating software development and maintenance.

单元测试是现代软件工程中至关重要甚至强制性的实践，它促进了软件开发和维护。

Recently, Large Language Models (LLMs) have brought transformative capabilities to the unit testing domain, yielding impressive progress and indicating substantial potential in follow-up research.

最近，大型语言模型（LLMs）为单元测试领域带来了变革性的能力，取得了令人印象深刻的进展，并预示着后续研究的巨大潜力。

In this paper, we conduct the first systematic literature review of LLM-based unit testing, covering publications from 2020 to March 2025.

在本文中，我们对基于 LLM 的单元测试进行了首次系统性文献综述，涵盖了从 2020 年到 2025 年 3 月的出版物。

We analyze 105 relevant studies from two complementary perspectives: the unit testing dimension, which includes tasks such as test generation, oracle generation, and test evolution; and the LLM dimension, which examines model usage, adaptation strategies, and integration with traditional techniques.

我们从两个互补的视角分析了 105 项相关研究：单元测试维度，包括测试生成、预言生成和测试演进等任务；以及 LLM 维度，考察模型使用、适配策略以及与传统技术的集成。

We also reveal that despite promising progress, challenges remain in areas such as testing complex units, detecting real-world bugs, and developing test-oriented LLMs.

我们还揭示了尽管取得了可喜的进展，但在测试复杂单元、检测现实世界 Bug 以及开发面向测试的 LLM 等领域仍存在挑战。

To guide future research, we highlight several research opportunities, including building end-to-end testing-and-repair pipelines, expanding support for underexplored tasks, and leveraging multimodal context information.

为了指导未来的研究，我们要重点介绍几个研究机遇，包括构建端到端的测试与修复管道、扩展对未充分探索任务的支持，以及利用多模态上下文信息。

Overall, this survey will serve as a comprehensive reference for researchers and practitioners, and help advance the development of LLM-based unit testing research.

总的来说，本综述将作为研究人员和从业者的综合参考，并有助于推动基于 LLM 的单元测试研究的发展。
