中图分类号：TP391　　单位代号：10280
密　　级：公开　　学　　号：23721378

上海大学

专业硕士学位论文

SHANGHAI UNIVERSITY PROFESSIONAL MASTER'S DISSERTATION

题目	面向生成式代码的 自适应水印技术研究

作　　者　施颖
学科专业　电子信息
导　　师　张新鹏
完成日期　二〇二六年五月

姓　　名：施颖　　学号：23721378

论文题目：面向生成式代码的自适应水印技术研究

上海大学

本论文经答辩委员会全体委员审查，确认符合上海大学专业硕士学位论文质量要求。

答辩委员会签名:

主　　席:

委　　员:

导　　师:

答辩日期:　　年　　月　　日

姓　　名：施颖　　学号：23721378

论文题目：面向生成式代码的自适应水印技术研究

上海大学学位论文原创性声明

本人郑重声明: 所呈交的学位论文是本人在导师指导下, 独立进行研究工作所取得的成果。除了文中特别加以标注和致谢的内容外，论文中不包含其他人已发表或撰写过的研究成果。参与同一工作的其他研究者对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名:

日期:　　年　　月　　日

上海大学学位论文使用授权说明

本人完全了解上海大学有关保留、使用学位论文的规定, 即: 学校有权保留论文及送交论文复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容。

(保密的论文在解密后应遵守此规定)

学位论文作者签名:　　导师签名:

日期:　　年　　月　　日　　日期:　　年　　月　　日

上海大学工学专业硕士学位论文

面向生成式代码的自适应水印技术研究

作　　者: 施颖

导　　师: 张新鹏

学科专业: 电子信息

通信与信息工程学院

上海大学

2026 年 5 月

A Dissertation Submitted to Shanghai University for the
Degree of Professional Master in Engineering

Research on Adaptive Watermarking for Generative Code

Candidate: Ying Shi
Supervisor: Xinpeng Zhang
Major: Electronic Information

School of Communication and Information Engineering
Shanghai University
May, 2026

摘　要

随着大语言模型在代码生成任务中的广泛应用，生成代码的来源鉴别与溯源问题逐渐受到关注。数字水印方法已在自然语言生成任务中得到应用，但代码不同于普通文本，其语法结构约束严格、表达空间受限，对候选词的微小扰动可能导致编译错误或逻辑异常。因此，直接将文本水印方法应用于代码生成任务中，容易破坏代码的正确性与可执行性。同时，代码中可安全嵌入水印的位置有限，水印嵌入强度的提升往往伴随着对程序语义的干扰，如何在不影响代码功能的前提下嵌入可检测水印并实现稳定识别，成为一项技术挑战。在此背景下，本文围绕生成代码水印嵌入强度与代码正确性之间的权衡问题开展研究，主要工作如下：

1）针对现有方法难以兼顾水印嵌入强度与代码正确性的问题，提出一种基于熵自适应的代码水印生成方法。具体而言，代码由生成模型基于词表概率分布逐步采样得到，即在每一步根据当前上下文对候选词进行概率选择。本文在该过程中计算候选词分布的熵值以刻画不确定性，并据此动态调节嵌入策略，仅在高熵位置施加偏置。由于高熵位置选择空间较大，对其进行调制对语法和语义影响较小，因此能够在保证代码可编译性和功能正确性的前提下实现水印的嵌入。与固定阈值方法相比，该方法能够根据上下文自适应调整嵌入强度，减少对关键语法位置的干扰。实验结果表明，该方法在保持代码正确性的同时，提高了水印检测成功率，并在多种任务场景下表现出良好的适用性与鲁棒性。

2）在此基础上，针对语法关键位置易受水印偏置扰动而导致代码生成错误的问题，提出一种基于语法结构感知的词表调制方法。该方法结合语法关键词集合与预测概率分布对生成位置进行动态判定，并据此设计分层词表裁剪策略，在语法位置施加强约束，在语义位置采用较为宽松的自适应裁剪，从而形成一种位置感知的解码机制。该方法可与现有基于红绿色表的水印机制协同使用，在不削弱水印可检测性的前提下，提高嵌入过程稳定性。实验结果表明，引入该策略后生成代码正确性显著提升，同时水印检测结果保持稳定，整体性能优于未采用该策略的相关方法。

关键词：生成代码水印；文本水印；自适应水印；语法结构感知；版权保护

ABSTRACT

With the widespread adoption of large language models in code generation tasks, the problem of source attribution and traceability of generated code has attracted increasing attention. Digital watermarking techniques have been applied in natural language generation; however, code differs fundamentally from plain text due to its strict syntactic constraints and limited expressive space. Even minor perturbations to candidate tokens may lead to compilation errors or logical faults. Therefore, directly applying text watermarking methods to code generation often compromises code correctness and executability. Moreover, the number of positions where watermarks can be safely embedded in code is limited, and increasing embedding strength typically introduces interference with program semantics. Consequently, how to embed detectable watermarks without affecting code functionality while ensuring reliable detection remains a significant challenge.To address this issue, we study the trade-off between watermark embedding strength and code correctness. The main contributions are as follows:

1) To overcome the difficulty of balancing embedding strength and correctness, an entropy-adaptive watermarking method for code generation is proposed. Specifically, code is generated through stepwise sampling from the token probability distribution conditioned on the current context. In this process, the entropy of the candidate distribution is computed to characterize uncertainty, and the embedding strategy is dynamically adjusted accordingly, applying bias only at high-entropy positions. Since high-entropy positions offer greater selection flexibility, such modulation has minimal impact on syntax and semantics, enabling watermark embedding while preserving compilability and functional correctness. Compared with fixed-threshold methods, the proposed approach adaptively adjusts embedding strength based on context and reduces interference with critical syntactic positions. Experimental results demonstrate that the method improves watermark detection success rates while maintaining code correctness, and exhibits strong applicability and robustness across diverse tasks.

2) To address the issue that syntactically critical positions are susceptible to watermark-induced perturbations, a syntax-aware vocabulary modulation method is proposed. This method dynamically classifies generation positions by combining a predefined set of syntactic keywords with the model’ s predictive probability distribution. Based on this, a hierarchical vocabulary pruning strategy is designed, imposing stronger constraints on syntactic positions while introducing relatively looser adaptive pruning for semantic positions, thereby forming a position-aware decoding mechanism. The proposed method can be integrated with existing red-green token partitioning watermark schemes, improving embedding stability without sacrificing detectability. Experimental results show that incorporating this strategy significantly enhances code correctness while maintaining stable watermark detection performance, achieving overall superior performance compared to baseline methods.

Keywords: Code Generation Watermarking; Text Watermarking; Adaptive Watermarking; Syntactic Structure Awareness; Copyright Protection

目 录

第一章　绪论

1.1　研究背景与意义

近年来，基于深度学习[1]的大规模预训练模型已成为人工智能发展的核心驱动力，推动人工智能技术迈入新的发展阶段。以 OpenAI、Google DeepMind 等为代表的研究团队相继推出 GPT-5、Gemini 3 等具有代表性的高性能大语言模型（Large Language Models, LLMs），在自然语言理解、文本生成、多轮对话以及复杂逻辑推理等任务中均表现出优异的性能。随着模型参数规模持续增长、训练数据覆盖范围扩展、推理策略不断优化，大语言模型的泛化能力与跨任务迁移能力显著增强。其应用场景也由最初的文本补全与对话任务，逐步拓展至代码生成（Code Generation）[2]、数学推理[3]与数据分析[4]等多个专业领域，并呈现出明显的跨领域融合发展趋势[5-7]。

其中，代码生成能力的突破尤为引人关注。依托大规模代码语料预训练与指令微调技术，模型能够理解自然语言的需求描述，并在包括 Python、Java、C++ 等多种主流编程语言[8]以及常见开发框架下，自动生成语法结构正确、语义完整且逻辑连贯的程序代码。相较于传统基于模板或规则的代码生成方法，基于 LLM 的生成方式在处理开放性问题与复杂逻辑方面具有更强的适应能力[9-10]，在函数实现、算法设计与接口封装等任务上均表现出良好的泛化性能。随着大模型代码生成能力的不断增强，软件开发流程正发生显著变化，以自然语言为核心的人机交互式编程模式逐渐得到发展与广泛应用[11-13]。

在软件工程实践层面，基于 LLM 的编程助手正在重塑传统开发模式。开发者可以通过自然语言提示快速生成函数实现、构建单元测试、编写接口示例或进行代码重构，从而减少重复性工作。模型还可用于注释生成、文档撰写以及错误定位与修复，提升开发效率与代码可维护性。对于初学者或非计算机专业人员而言，自然语言驱动编程降低了技术门槛[14-15]，使更多用户能够参与软件构建与原型设计，推动“低代码”“无代码”及智能化开发工具的发展。代码生成模型正逐步融入软件生命周期管理流程，成为新一代软件生产体系中的重要基础设施。

然而，技术能力的快速提升也带来了新的安全与治理挑战。一方面，代码生成模型在特定条件下可能被用于生成恶意代码，如勒索程序、后门程序及漏洞利用脚本

等[16-17]，并能够快速生成多种变体，从而降低攻击门槛并放大规模化传播能力。另一方面，由于模型训练过程中广泛使用开源代码数据，其输出可能无意复现受版权或许可证约束的内容，引发知识产权与合规风险[18-19]。在企业环境中，若缺乏有效管理，可能导致责任界定不清并增加合规成本；在学术与教育场景中，未披露的模型生成代码也可能影响成果归属与评价公平性[20-21]。

在上述背景下，随着生成代码质量不断提升，机器生成内容与人工编写代码在结构与风格上逐渐趋同[22]，传统基于代码特征或相似度分析的方法已难以有效区分其来源。当生成内容涉及安全漏洞、版权纠纷或不当使用时，若无法有效判定其是否由特定模型生成，将严重影响责任追溯、风险评估与法律界定。因此，构建对生成内容的可识别与可追溯机制，已成为当前人工智能安全治理体系中的关键议题。

为应对上述挑战，文本水印（Text Watermarking）技术被广泛认为是实现生成内容来源识别的重要技术路径之一。不同于在文本或代码中嵌入可见标识的做法，水印机制通常以隐性方式融入生成过程，使输出内容在整体层面呈现特定特征，而在正常阅读中难以被察觉。通过预设规则或密钥，可以在后续检测阶段对生成结果进行分析，从而判断其是否来自特定模型。该类方法旨在保持内容表达质量与使用体验基本不受影响的前提下，实现对生成内容来源的有效区分。由于兼具隐蔽性与可识别性，文本水印被视为支持生成式模型责任追踪与内容溯源的重要技术基础，在构建可审计的人工智能应用体系中具有重要价值。

然而，现有水印方法主要面向自然语言文本设计，直接应用于代码生成场景时会面临新的挑战。代码具有严格的语法约束与执行语义要求，其表达空间受限且对生成扰动高度敏感；同时，代码可通过重构、变量替换或结构调整等方式进行等价变换，从而削弱甚至破坏水印信息。因此，在保证代码正确性与功能一致性的前提下，实现兼具隐蔽性与鲁棒性的水印机制仍具有较大难度。

从研究现状来看，面向代码生成内容的可溯源技术仍处于起步阶段，现有研究多集中于自然语言场景，对代码这一同时具备工程属性与法律属性的生成对象关注不足。随着生成式人工智能在软件开发、教育教学与开源生态中的广泛应用[23-24]，构建可靠的代码来源识别机制具有重要的现实意义，有助于明确责任归属、降低合规风险，并推动生成式人工智能的规范化应用。

1.2 国内外研究概况

随着生成式人工智能的快速发展，水印技术已成为保障生成内容可追溯和验证内容来源的重要技术手段。在国内外学术界和工业界，水印技术的发展经历了从图像版权保护到文本生成追溯[25-26]，并逐步扩展至代码生成等各领域的演进过程。

1.2.1 文本水印研究

文本水印技术最早主要应用于数字文本版权保护领域，其基本思想是在不显著影响文本可读性的前提下，在文本中嵌入隐蔽的标识信息，以实现内容来源验证与版权追踪。早期研究多依赖于文本格式层面的修改，例如通过调整字符间距、行间距[27-28]、字体样式[29]或插入零宽度字符[30]等方式嵌入水印信息。这类方法实现简单，对文本内容本身的语义影响较小，但其鲁棒性较弱，在文本复制、格式转换或重新排版等操作下容易被破坏[31]，因此在实际应用中具有局限性。

随着自然语言处理技术的发展，研究者开始探索基于语言内容层面的文本水印方法。例如，通过同义词替换[32-33]、句式改写[34]或语序调整[35]等方式，在保持语义基本一致的前提下嵌入水印信息。这类方法利用自然语言表达的多样性来实现隐蔽的信息编码，相比纯格式类水印具有更好的隐蔽性。然而，由于需要依赖人工设计的替换规则或语义资源库，其嵌入容量和自动化程度仍然受到一定限制。

近年来，随着生成式模型能力的持续提升，文本水印研究逐步由基于规则的后处理方法转向面向生成过程的嵌入机制。相较于依赖格式调整或语义替换的传统方法，生成式水印能够在生成阶段直接融合水印信息，在保持文本自然性的同时提升其隐蔽性。现有方法大致可分为两类。一类通过构建端到端模型，将文本与水印信息作为联合输入，实现隐式嵌入与提取。例如，Abdelnabi 等提出的 AWT 方法[36]基于 Transformer 编码—解码结构完成信息融合，后续研究进一步结合预训练模型以提升水印容量与表达能力。另一类方法则利用生成模型的改写能力，在语义一致的前提下嵌入水印信息。Lau 等提出的 WATERFALL 方法[37]通过释义生成来实现水印注入，在保证文本流畅性的同时增强隐蔽性与抗分析能力。此外，Kirchenbauer 等提出的基于概率调制的水印方法 WLLM[38]已成为主流方向之一。该方法通过划分词表并对特定集合施加概率偏置，使生成文本在统计层面具备可检测特征。由于无需重新训练模型，仅在生成阶段调制概率分布，该方法具有实现简单、通用性强等优势，并为后续研究奠定了基础。

1.2.2 代码水印研究

随着代码生成大模型的发展，代码生成技术在软件开发中的应用逐渐增多，例如代码补全、自动编程以及程序修复等。然而，与自然语言文本相比，面向代码生成内容的水印研究仍处于起步阶段，相关研究相对较少。由于程序代码具有严格的语法结构与可执行性要求，现有文本水印方法难以直接迁移至代码生成场景，因此亟需针对代码特性进行专门设计。

目前，仅有少数研究尝试将水印技术应用于生成代码的来源识别问题。Lee 等人基于 WLLM 提出了 SWEET[39] 方法。该方法通过将词汇表随机划分为“绿色”和“红色”标记，在生成过程中优先选择高熵的绿色词元。由于这种划分对攻击者不可见，SWEET 能够以较高置信度判断代码是否由模型生成，并具有较好的隐蔽性和检测精度，为后续代码水印研究奠定了基础。

在此基础上，部分研究从不同角度对代码水印机制进行了探索。Li 等人提出 ToSyn 方法[40]，通过词元级同义替换隐式嵌入水印，从而实现对代码生成接口知识产权的保护。该方法结合统计检验进行水印检测，在隐蔽性与检测可靠性方面取得了较好效果。随后，Li 等人进一步提出 ACW 方法[41]，通过选择性语义保持的幂等代码转换实现水印嵌入，无需对模型进行额外训练或微调。实验结果表明，该方法在效率与抗篡改能力方面优于部分已有方法。Ning 等人提出 MCGMARK 方法[42]，面向生成恶意代码场景，通过控制词元选择过程隐式嵌入水印，并在水印中编码用户身份信息；同时，该方法引入跳过策略以避免关键位置被修改，从而在一定程度上提升了水印的鲁棒性。Yang 等人提出 SrcMarker 方法[43]，通过双通道无损嵌入标识比特串，确保水印嵌入不改变代码语义；该方法结合学习型嵌入与基于规则的转换策略，提高了水印表达的多样性，并通过特征近似技术实现端到端训练。

现有代码水印方法主要从概率调制、语义替换以及代码转换等不同角度展开研究，但整体仍处于发展阶段。由于程序代码具有更强的结构约束，关键字、运算符以及分隔符等结构性词元在程序构建中起着重要作用，一旦这些位置被错误修改，可能导致代码无法编译或执行。因此，相比自然语言文本，代码水印在嵌入策略设计上需更加谨慎。如何在保证代码语法正确性与程序可执行性的前提下，实现稳定、隐蔽且可检测的水印嵌入，仍然是当前代码水印研究需要重点解决的问题。

1.2.3 面临的主要挑战

尽管近年来文本水印与代码水印技术取得了一定进展[44]，但现有方法在实际应用中仍面临多方面挑战[45]，主要包括生成质量与嵌入强度之间的权衡、对编辑与转换操作的鲁棒性不足、水印容量受限以及跨场景适配能力有限等问题。

在生成质量方面，现有水印方法多通过调制候选词元的概率分布以实现水印嵌入。当偏置策略设计不当或强度过大时，可能扰动模型原有的生成分布，从而降低了生成内容的自然性与连贯性。在自然语言场景中，这通常表现为表达不够流畅或语义衔接受损；而在代码生成任务中，该问题更为突出。程序代码不仅要求语义正确，还需严格满足语法规则，一旦关键位置的词元被不当修改，可能直接导致代码无法编译或执行。

在鲁棒性方面，生成内容在传播与使用过程中通常会经历多种编辑与转换，例如文本中的改写、摘要压缩与翻译，以及代码中的格式化、变量重命名与自动重构等。这类操作可能改变词元分布特征，从而削弱甚至破坏水印信号，影响检测的稳定性。因此，如何在保证生成质量的前提下提升水印在多种编辑与转换条件下的鲁棒性，是当前研究的重要问题。

此外，在实际生成过程中，并非所有位置都适合嵌入水印信号。在代码生成任务中，大量词元对应的概率分布高度集中，属于低不确定性位置，其生成结果较为稳定，难以通过概率调制嵌入有效水印信号。若在此类位置强行施加偏置，不仅难以提升嵌入效果，反而可能影响生成质量。因此，如何识别适合嵌入水印的关键位置并提高嵌入效率，成为重要研究方向。同时，不同生成场景之间的结构特性差异也对水印方法提出了更高要求。与自然语言相比，程序代码具有更强的结构约束，关键字、运算符及分隔符等结构性词元对程序结构具有决定性作用；若在这些位置进行不合理调制，可能破坏程序的语法正确性。另一方面，代码中变量名、函数名等语义性词元的分布特征与结构性词元存在显著差异，统一的水印策略往往难以兼顾不同类型词元的特点。

综上所述，在保证生成质量与语法正确性的前提下，提高水印嵌入的稳定性与检测可靠性，并针对代码生成场景设计结构感知的水印机制，仍是当前生成模型水印研究亟待解决的关键问题。本文正是在上述背景下，对代码生成场景中的水印嵌入方法进行研究与改进。

1.3 本文主要研究内容

针对当前大语言模型在代码生成任务中的广泛应用，以及由此带来的内容归属与版权保护问题，本文围绕代码生成场景下的数字水印技术展开研究。相较于自然语言文本，程序代码具有更强的结构约束与执行语义要求，其生成过程往往表现出低不确定性位置占比较高、语法依赖关系紧密等特点。这使得现有面向文本的水印方法在直接应用于代码场景时，容易对关键位置产生干扰，从而影响代码的语法正确性与执行正确性。因此，在保证代码质量不受影响的前提下，实现隐蔽且可检测的水印嵌入，成为亟需解决的关键问题。围绕上述问题，本文从“嵌入位置选择”与“嵌入过程约束”两方面展开研究，提出两类针对代码生成特点的水印优化方法。具体内容如下：

针对代码生成过程中不同位置不确定性差异显著的问题，提出一种基于熵自适应的代码水印生产方法。该方法利用生成过程中候选词元概率分布所反映的不确定性信息，对嵌入策略进行动态调整：在不确定性较高的位置适当引入更强的水印偏置，以提升嵌入有效性；而在不确定性较低的位置则降低偏置大小，甚至保持原有生成分布，以避免对关键结构产生干扰。通过这种方式，可以在保证代码语法与语义一致性的前提下，实现水印嵌入强度与生成质量之间的有效平衡。

针对语法关键位置对代码正确性敏感的问题，提出一种基于语法结构感知的词表调制方法。该方法通过构建语法词元集合，对生成过程中的关键结构位置进行识别，并在语法位置对候选词进行筛选，去除概率较低且可能影响语法正确性的候选项，同时在语义位置采用较为宽松的自适应裁剪策略以保留原有生成分布特性。在此基础上，对保留的候选词施加水印调制，使嵌入过程更加符合代码结构特性。该方法可与现有基于概率调制的水印机制协同使用，在保证可检测性的同时，提高生成过程的稳定性与代码正确性。

1.4 本文的结构安排

本文共分为五章，内容安排如下：

第一章介绍了生成式代码水印的研究背景与现实意义，分析代码生成模型快速发展带来的版权保护与内容溯源问题，以及现有水印方法在代码场景下面临的主要挑战。在此基础上，综述了文本水印与代码水印的国内外研究现状，并说明本文的

主要研究内容、创新点及整体结构安排。

第二章从理论基础与关键技术两个方面展开介绍。首先概述生成式模型与代码生成模型的基本原理，其次介绍数字水印与文本水印的相关理论及评价指标。随后分析代码与自然语言在结构和语义上的差异，讨论抽象语法树、编译约束以及功能正确性对代码水印设计的影响，最后总结生成式代码水印的相关技术与关键挑战。

第三章提出了基于熵自适应的代码生成水印方法。详细描述水印嵌入与检测流程，利用生成过程中概率分布的熵信息动态调节嵌入强度，在保证代码语法与功能正确性的前提下提升水印鲁棒性。同时给出方法的实现细节，包括算法流程、关键参数选择及策略优化，并通过实验对比分析其在检测性能、代码质量影响及抗攻击能力方面相较固定阈值方法的优势。

第四章提出了基于语法结构感知的词表调制水印方法。在现有概率调制方法基础上，引入代码语法结构与层级信息，对水印嵌入空间进行进一步优化。通过语法分析与调制策略设计，在降低对生成代码质量影响的同时提升水印的可检测性与稳定性。本章详细介绍方法设计、实现过程与实验评估，并分析其在不同编程语言场景下的适用性与扩展能力。

第五章对全文研究工作进行总结，概括所提出方法的主要创新点与实验结果，分析其在实际生成式代码场景中的应用价值与局限性，并对未来研究方向进行展望，包括提升水印鲁棒性及改进检测与防攻击策略等。

1.5 本章小结

本章首先介绍了代码生成水印技术的研究背景与研究意义。随着大语言模型在代码生成领域的广泛应用，生成代码的来源鉴别、版权保护以及内容安全等问题逐渐受到关注，代码水印技术因此成为当前人工智能安全研究中的重要方向。随后，本文分别综述了文本水印与代码水印的国内外研究现状，对现有方法的基本思想、实现机制以及存在的问题进行了分析。现有研究虽然能够在一定程度上实现水印嵌入与检测，但在代码场景下仍然面临语法约束强、语义敏感度高以及生成质量易受影响等挑战。最后，本章介绍了本文的主要研究内容与整体结构安排，为后续章节中面向代码生成任务的水印方法研究与改进奠定了基础。

第二章　代码水印技术基础

2.1　生成式模型概述

生成式模型（Generative Models）通过学习数据分布以实现样本生成，其核心目标是刻画数据的统计特性，并生成与训练数据在分布上相一致的内容。与判别式模型（Discriminative Models）侧重输入与标签之间的映射关系不同[46]，生成式模型关注对数据本身分布的建模，因此能够在无监督或弱监督条件下完成生成任务。

针对序列数据（如文本与代码），主流生成方法通常采用基于上下文的逐步生成机制，即在已有历史信息的基础上依次生成后续元素，从而逐步构建完整序列。这种生成方式使模型能够在局部决策的基础上形成全局一致的结构与语义表达。在具体生成过程中，模型在每一步根据当前分布对候选进行选择，不同选择会影响后续生成路径，因此整体过程具有明显的路径依赖特性。同时，生成分布通常呈现不均匀性，使得模型在生成时需要在高概率选择与一定程度的随机性之间进行权衡，以兼顾生成质量与多样性。

根据生成对象不同，生成式模型可应用于多种任务场景[47]。其中，自然语言文本生成主要关注语义表达的流畅性与连贯性，而代码生成任务则进一步受到语法结构与执行语义的严格约束。下面分别对文本生成模型与代码生成模型进行介绍。

2.1.1　文本生成模型

文本生成模型的目标是基于给定的上下文生成符合语言规律的文本序列，同时保持语义逻辑一致性、语法合理性和信息完整性。自回归语言建模是文本生成的核心概率框架，通过逐步预测下一个词元或词语的条件概率来实现序列生成。对于长度为 T 的文本序列 $x = (x_1, x_2, \ldots, x_T)$，自回归建模可以表示为：

$$P(x) = \prod_{t=1}^{T} P(x_t \mid x_{<t}; \theta) \tag{2.1}$$

其中，θ 表示模型参数，$x_{<t} = (x_1, x_2, \ldots, x_{t-1})$ 表示历史上下文。通过最大化训练语料的对数似然，模型能够学习到序列中词语之间的依赖模式和语义结构。

早期语言建模方法多依赖循环神经网络（Recurrent Neural Network，RNN）及其

变体，在序列建模任务中取得了一定进展。然而，这类模型在处理长序列时往往面临梯度传播困难和远距离依赖建模能力不足的问题，同时其递归计算方式也限制了并行效率。为克服上述局限，基于自注意力机制的模型逐渐成为主流，其中 Transformer 架构[48]具有代表性。

自注意力机制的核心在于对序列中各位置之间的相关性进行全局建模，使得每个位置在生成表示时都能够直接利用序列中其他位置的信息，从而在单次前向计算中捕获完整的上下文依赖关系。其计算过程可表示为：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.2}$$

其中，查询矩阵 Q、键矩阵 K 和值矩阵 V 由输入表示 X 经过不同线性映射得到，即 $Q = XW_Q$、$K = XW_K$、$V = XW_V$，d_k 表示键向量的维度。该过程通过计算查询与键之间的相似度来获得注意力权重，并对值向量加权求和，从而生成新的表示。

如图 2.1 所示，Transformer 模型整体采用编码器–解码器（Encoder–Decoder）结构[49]。输入序列首先经过嵌入层映射到连续向量空间，并叠加位置编码以引入序列顺序信息。编码器由多层相同结构堆叠而成，每一层包含多头自注意力模块和前馈神经网络，并结合残差连接与层归一化以提升训练稳定性；解码器在此基础上引入掩码自注意力机制，以防止当前位置在计算注意力时访问其后续位置信息，从而保证自回归生成过程的因果性，同时通过编码器–解码器注意力模块融合源序列的语义表示，逐步生成目标序列。

得益于自注意力机制的全局建模能力以及非递归的结构设计，Transformer 可以在各位置上并行计算，大幅提升训练效率。同时，多头注意力机制能够从不同子空间刻画序列特征，使模型具备更强的表达能力。基于上述优势，Transformer 已成为现代自然语言处理与代码生成任务中的核心基础架构。

在 Transformer 架构的基础上，现代文本生成模型采用大规模预训练策略，即先在海量语料上进行泛语言建模，再在具体生成任务上进行微调[50-52]。预训练目标通常采用最大对数似然估计形式：

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t}) \tag{2.3}$$

这种训练方式使模型能够获得跨领域、跨任务的泛化能力[53-54]，而微调过程则使其在特定生成任务上具备更好的适应性。生成模型在大型语料上的训练不仅学习到了

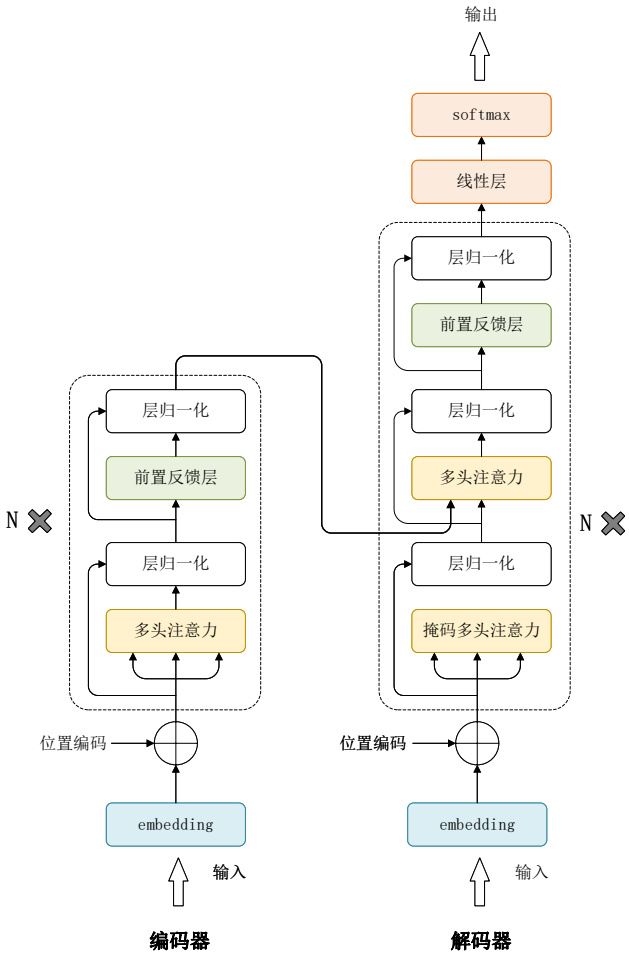


图 2.1 Transformer 模型结构示意图

语言的统计规律，还能够在多个语义层面捕捉深层次的文本结构。

在生成阶段，模型需要根据条件概率分布选择具体的词元序列，而不仅仅是简单地选取概率最大的结果。为此，研究者提出了多种解码策略以平衡生成质量与多样性。例如，束搜索（Beam Search）[55]通过保留多个高概率候选路径来提升整体序列的最优性，但可能导致生成结果趋于保守；随机采样（Sampling）则根据概率分布进行随机选择，使生成结果更加多样，但可能降低稳定性。在此基础上，温度调节（Temperature Scaling）通过调整分布的平滑程度控制生成的随机性，高温度对应更均匀的分布从而提升多样性，而低温度则使模型更倾向于选择高概率词。此外，Top-k 采样与 Top-p（nucleus sampling）方法[56]通过截断低概率候选集合，在保证生成质量的同时进一步提升文本的自然性与连贯性。

随着参数规模不断增长和训练数据规模的扩大，大规模语言模型（Large Language Models, LLMs）如 GPT 系列在自然语言生成任务中取得了显著突破。GPT-3[57]通过

数百亿参数规模和海量语料训练，实现了在多个语言生成任务上无需任务特定微调即可展现通用生成能力；GPT-4[58]更进一步，通过多模态输入和更强的语义理解能力，使模型能够处理更复杂的语言生成任务。GPT-5 在此基础上，通过优化训练策略和扩展模型架构，进一步获得了更强的多步推理、逻辑链条保持能力。

文本生成模型的技术进步不仅推动了自然语言系统的发展，同时也为其他领域生成任务提供了架构基础。例如在对话生成、文本摘要、内容创作辅助等应用中，现代语言模型通过对上下文和语义信息的深度理解，使生成结果在信息完整性和语言流畅性上达到了前所未有的水平，为自然语言理解和交互技术奠定了坚实基础。

2.1.2 代码生成模型

与自然语言生成相比，代码生成不仅需要生成文本形式的符号序列，还必须满足更严格的结构约束和语义规则。在编程语言中，语法规则明确规定了语句结构、变量作用域、类型一致性等多个层面的约束，因此代码生成模型需要在保持概率生成能力的同时，具备对语言结构的深入理解。

代码生成的概率建模同样可以采用自回归形式：

$$P(c) = \prod_{t=1}^{T} P(c_t \mid c_{<t}; \theta) \tag{2.4}$$

其中，$c = (c_1, c_2, \ldots, c_T)$ 表示代码 token 序列，模型参数 θ 表示其生成策略在语法和语义约束下的学习参数。不同于自然语言，代码序列中的每个 token 不仅有词汇意义，还具有语法和执行语义关联，这对模型提出了更高的表示学习要求。

在代码生成任务中，模型不仅需要理解代码的统计规律，还要学习抽象语法树（Abstract Syntax Tree, AST）等语法结构信息，这些结构信息在描述代码逻辑和执行过程方面具有核心作用。为了有效结合结构信息与概率生成机制，许多研究提出将 AST 信息编码进模型输入，使模型在生成时对语法约束有明确学习依据。这种语法增强方式使得生成结果更符合语言规范，并减少语法错误。

Transformer 架构在代码生成任务中同样被广泛应用，但需要对原始输入进行结构性增强。例如，在输入层融入缩进信息、代码块边界和变量作用域标记等，使注意力机制能够捕捉程序内部的层级结构。此外，也有研究将图神经网络（Graph Neural Networks, GNNs）[59]与 Transformer 融合，使模型能够学习到程序控制流和数据流中的复杂依赖关系[60-61]。

在训练策略上，代码生成模型通常在大规模开源代码库上进行预训练[62-63]，这些代码库包括多种编程语言、风格和功能实现。例如在函数补全场景下，模型需要理解函数定义、输入输出约定以及语义约束，使生成结果在语义和功能上达到可运行标准。预训练目标仍然以最大似然估计为基础，但训练数据需要经过清洗和结构化处理，以减少噪声代码对模型学习的负面影响。

近年来，多种专用的代码生成模型相继涌现，推动了该领域的技术演进。OpenAI 开发的 Codex 模型[22]基于 GPT 架构，通过在公开代码库上的大规模预训练，模型具备了从自然语言描述生成可执行代码的能力，并构成 GitHub Copilot 的技术基础。同期 Google DeepMind 推出的 AlphaCode[24]结合大规模采样与过滤机制，在编程竞赛任务上展现了解决复杂问题的潜力；Salesforce 发布的 CodeT5[64]采用编码器-解码器架构，通过标识符感知预训练任务增强了对于代码语义的理解。研究社区陆续开源的 StarCoder[65]、DeepSeekCoder[66]等模型，则通过多语言支持、检索增强学习等技术，进一步提升了代码生成的准确性与泛化能力。Meta 开发的 Code Llama[67]是一款针对代码生成优化的开源模型，在长上下文处理和跨语言迁移方面表现优异，推动了该领域研究成果的广泛可用性。

总体而言，代码生成模型在概率建模、结构理解和语义推理等方面已展现出较强能力，从最初的简单代码补全发展到能够处理复杂编程任务的智能代理系统。新一代模型的涌现，反映出该领域正从单纯的代码生成器向能够解决复杂工程问题的智能编程系统演进。

2.2 文本水印理论基础

数字水印是一种通过在原始数据中嵌入隐蔽信息以实现内容标识与来源追溯的技术，其核心目标是在不显著影响数据质量的前提下，使嵌入信息在后续传播过程中能够被可靠识别。传统数字水印主要应用于图像、音频和视频等连续信号，通过利用信号在时域或频域中的冗余特性实现信息嵌入。

与连续信号不同，文本数据由离散符号构成，表达空间受限且语义约束严格，这使得文本水印在设计上面临更大的挑战。一方面，对文本的微小修改可能引入语义偏差甚至语法错误；另一方面，文本缺乏显式冗余结构，限制了水印信息的嵌入容量。因此，文本水印需要在信息嵌入能力与文本自然性之间进行精细权衡。

从系统角度来看，文本水印通常包含水印嵌入与水印检测两个基本阶段。在嵌入阶段，通过特定策略将隐式信息编码到文本中；在检测阶段，则通过分析文本统计特征判断其是否包含水印信号。该过程通常不依赖原始文本，属于盲检测范式，从而在开放环境中具有较强的实用性。其通用流程如图 2.2 所示。

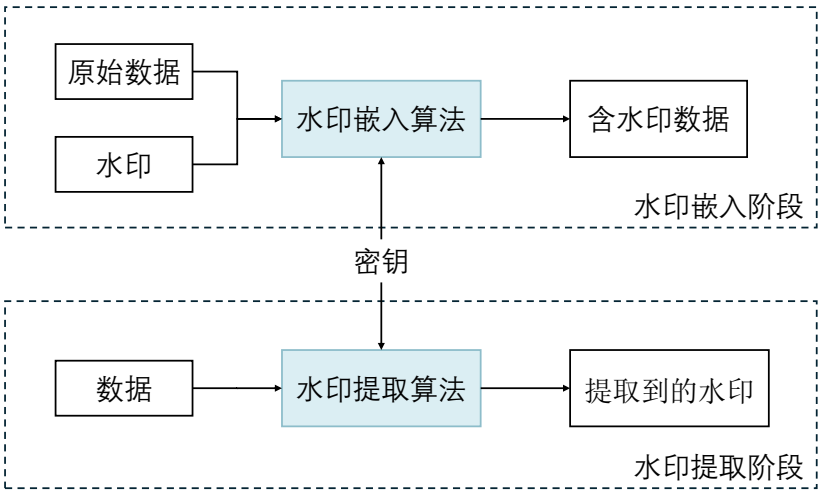


图 2.2　水印通用框架

从形式上看，文本水印可以理解为在文本生成或编辑过程中引入可控扰动，使生成文本在统计意义上呈现出特定模式。这种模式在局部难以察觉，但在整体层面可通过统计方法进行识别。从信息论角度来看，该过程本质上是在有限表达空间中实现隐蔽信息嵌入与语义保持之间的平衡。

基于水印嵌入方式的不同，现有文本水印方法大致可分为两类：一类是对已有文本进行修改的自然文本水印方法，另一类是在文本生成过程中引入可控机制的生成式文本水印方法。前者主要通过词汇替换、句法变换或语义改写等方式嵌入水印信号，后者则依托生成模型，在文本生成过程中通过对解码策略或概率分布进行干预以实现水印注入。更进一步地，这两类方法在实现机制上又可以细分为多个子类别，其整体分类结构如图 2.3 所示。

总体而言，文本水印为生成内容的来源识别与版权保护提供了重要技术手段。然而，由于文本数据的离散性与语义敏感性，其在嵌入策略、检测方法及鲁棒性等方面仍面临诸多挑战，这也为后续研究提供了重要方向。基于上述分类，下一节将首先简要介绍自然文本水印方法，随后进一步分析生成式文本水印技术。

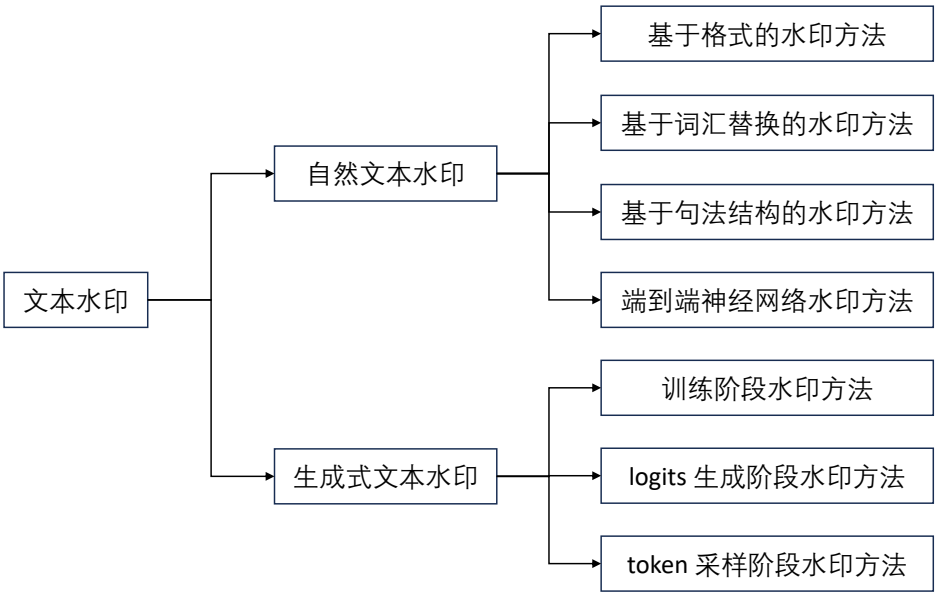


图 2.3　文本水印方法分类框架

2.2.1　自然文本水印技术

自然文本水印方法主要通过对已有文本进行修改以嵌入隐式信息，其核心思想是在尽量保持语义与可读性的前提下，对文本不同层次特征进行可控变换。根据嵌入层级的不同，现有方法通常可分为基于格式、基于词汇替换、基于句法结构以及端到端神经网络生成四类。

基于格式的水印方法通过调整文本排版或字符编码等非语义特征嵌入信息，例如利用间距变化或编码差异进行隐式表示，其优点是不影响语义内容，但对格式变化较为敏感。基于词汇替换的方法通过对词语进行同义替换实现信息嵌入，在保持句法结构不变的前提下改变词汇选择，该类方法实现简单，但易受后续编辑或重写影响。基于句法结构的方法则通过语序调整或句式转换等结构变换进行编码，在保持语义等价的同时利用结构差异承载水印，其隐蔽性较强，但依赖较强的语言结构分析能力。端到端神经网络水印方法基于生成模型直接学习水印嵌入过程，通过联合建模文本内容与水印信息实现隐式编码，在文本自然性与隐蔽性方面表现较优，但通常训练成本较高且可解释性较弱。

总体而言，自然文本水印方法通过不同层级的文本修改实现信息嵌入，在隐蔽性与鲁棒性之间进行权衡。但由于依赖对文本内容的显式或隐式修改，其在面对重写或编辑操作时仍可能受到影响。为进一步提升鲁棒性，研究逐渐转向基于生成过

程的水印方法。

2.2.2 生成式文本水印技术

随着大语言模型在文本生成任务中的广泛应用，研究者逐渐将水印嵌入过程从“文本修改”转向“生成过程内嵌”。生成式文本水印方法通过在模型生成过程中引入可控机制，使模型输出的文本天然具备水印特性，从而避免对已有文本进行显式修改。从生成流程的角度来看，大语言模型的文本生成主要经历模型训练、logits 生成以及 token 采样三个阶段。相对应地，生成式文本水印方法也可划分为训练阶段水印、logits 生成阶段水印以及 token 采样阶段水印三类。

（1）训练阶段水印方法

训练阶段水印通过在模型训练过程中引入特定机制，使水印信息隐式编码到模型参数中。其基本思想是在训练数据中构造包含水印信息的样本，使模型在学习过程中形成对特定模式的响应，从而在推理阶段表现出可检测的行为特征。一种常见实现方式是对训练数据进行有控制的修改，如在部分样本中引入特定模式或结构，使模型在遇到相关输入时产生预期输出。这种机制本质上类似于在模型中嵌入隐式触发信号，使水印信息与模型行为建立关联。训练阶段方法的优点在于水印嵌入过程对生成阶段透明，不会影响正常推理流程，同时具有较强的隐蔽性。然而，其局限性也较为明显：水印通常仅在特定输入条件下才能被触发，信息容量有限，且一旦需要更新水印，往往需要重新训练模型，计算成本较高。

（2）logits 生成阶段水印方法

logits 生成阶段水印是在模型输出概率分布后、采样之前对分布进行调制，从而在不改变模型参数的前提下实现水印嵌入。具体而言，该方法通过对原始 logits 向量施加与密钥相关的偏置，使某一部分候选词在采样过程中更容易被选中。其核心思想是对词汇空间进行划分，并在生成过程中动态调整不同子集的概率权重。例如，可以将候选词划分为若干子集，并通过提升某一子集的概率，使生成序列在统计意义上呈现出特定分布特征。检测阶段则通过分析文本中各类 token 的分布情况，判断其是否符合预期水印模式。该类方法的优势在于实现灵活、无需修改模型结构，且能够在生成过程中持续嵌入水印信号。同时，由于水印嵌入以概率调制形式进行，对文本语义影响较小。然而，其性能依赖于分布调制策略的设计，不合理的偏置可能影响文本质量。此外，在低不确定性（低熵）场景下，水印嵌入空间受限，也对方法

设计提出更高要求。

（3）token 采样阶段水印方法

token 采样阶段水印在保持原始概率分布不变的前提下，通过控制采样过程实现水印嵌入。该方法通常利用采样过程中的随机性，将水印信息编码到 token 选择路径中。一种典型思路是引入伪随机序列，通过固定随机种子生成确定性随机数，并利用该序列指导 token 的选择过程。例如，在每一步生成中将随机数与候选词的概率进行比较，从而决定具体采样结果。由于该过程在统计上保持对原始分布的无偏性，因此能够最大限度地保留生成文本的自然性。采样阶段方法的优势在于对模型输出分布无直接干扰，能够较好地保持文本质量，同时具备一定的抗伪造能力。然而，该类方法通常依赖随机序列与文本之间的对齐关系来进行检测，对文本编辑操作较为敏感。此外，过强的确定性控制可能降低生成结果的多样性。

总体而言，生成式文本水印方法通过在不同生成阶段引入控制机制，实现了水印与文本生成过程的深度融合。其中，训练阶段方法侧重于模型层面的嵌入，logits 阶段方法关注概率分布调制，而采样阶段方法则利用生成过程的随机性进行编码。这三类方法在嵌入能力、文本质量及鲁棒性之间形成不同权衡，也为后续水印方法的设计提供了多种实现路径。

2.2.3 水印方法的评价指标

为了系统评估水印方法的性能，通常从不可感知性、鲁棒性以及可检测性三个维度进行量化分析。尽管相关指标最初主要应用于文本水印研究，但其核心思想同样适用于代码生成场景，并构成后续方法设计与实验评估的基础。

不可感知性用于衡量水印嵌入对生成质量的影响。在生成任务中，通常采用困惑度（Perplexity, PPL）作为评价指标，其定义为：

$$PPL = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P(x_t \mid x_{<t})\right) \tag{2.5}$$

其中，T 表示序列长度，x_t 为第 t 个生成 token，$P(x_t \mid x_{<t})$ 表示在给定历史上下文条件下生成该 token 的概率。PPL 越低，说明生成结果越符合模型原始分布，即水印对生成质量的影响越小。

鲁棒性用于衡量水印在经过扰动或攻击后仍能被正确检测的能力。通常通过攻

击后水印检测准确率进行度量，其定义为：

$$Acc_{robust} = \frac{N_{\text{correct}}^{attack}}{N_{\text{total}}} \tag{2.6}$$

其中，$N_{\text{correct}}^{attack}$ 表示在攻击后仍能正确检测出水印的样本数量，N_{total} 则表示总样本数量。该指标反映了水印在扰动条件下的稳定性。在代码场景中，此类扰动可表现为结构保持的变换，如格式调整或标识符替换等。

可检测性用于评估水印信号的显著性与可靠性。通常基于统计假设检验构建检测统计量，例如基于绿色 token 比例的 z-score：

$$z = \frac{S - \mathbb{E}[S]}{\sqrt{\text{Var}(S)}} \tag{2.7}$$

其中，S 表示序列中绿色 token 的数量，$\mathbb{E}[S]$ 和 $\text{Var}(S)$ 分别为其在无水印假设下的期望与方差。z 值越大，说明水印信号越显著。在此基础上，可进一步定义检测准确率：

$$Accuracy = \frac{N_{\text{correct}}}{N_{\text{total}}} \tag{2.8}$$

综上，上述指标分别从生成质量保持、抗扰动能力以及检测可靠性三个方面刻画水印方法的整体性能。在实际应用中，需要在不可感知性与可检测性之间进行权衡，从而在保证生成质量的前提下实现稳定的水印嵌入效果。

2.3 编程语言结构与代码生成特性

与自然语言相比，编程语言在形式结构、语义约束与执行机制上具有更强的规则性与确定性。这种差异决定了代码生成任务在建模目标、结构表达以及评价标准方面均呈现出不同于文本生成的技术特征。自然语言生成强调语义连贯与表达合理性，而代码生成不仅要求语法合法，还必须满足类型约束、作用域规则以及运行时语义一致性。因此，理解编程语言的结构特性，是分析代码生成模型行为与能力边界的基础。

在结构层面，编程语言与自然语言存在显著差异。自然语言允许一定程度的模糊性、歧义与冗余表达，其语义合理性依赖语境判断；而编程语言属于形式语言，其语法由严格的产生式规则定义，任何不符合规则的符号序列都会导致编译失败。代

码中的关键字、括号匹配与语句嵌套均受到严格约束，使得单个符号错误即可破坏整体结构。

此外，代码具有显式的类型系统与作用域机制。在类型系统约束下，变量与表达式必须满足一致性要求；在作用域规则下，标识符的定义与引用必须严格对应，否则将产生语义错误。这些机制使代码生成不仅依赖局部 token 预测，还需要对全局结构关系进行建模。同时，代码中存在控制流与数据流依赖关系，例如条件分支与循环结构决定执行路径，变量赋值关系影响数据传递，这些依赖通常跨越多个语句，使生成过程具有明显的长距离结构约束。

为刻画程序结构，编程语言通常通过抽象语法树（Abstract Syntax Tree, AST）表示层级结构。如图2.4所示，AST 是基于语法规则构建的树状表示，每个节点对应一个语法单元。相比线性 token 序列，AST 能够显式表达嵌套关系与层级结构，是理解程序结构的重要中间表示。

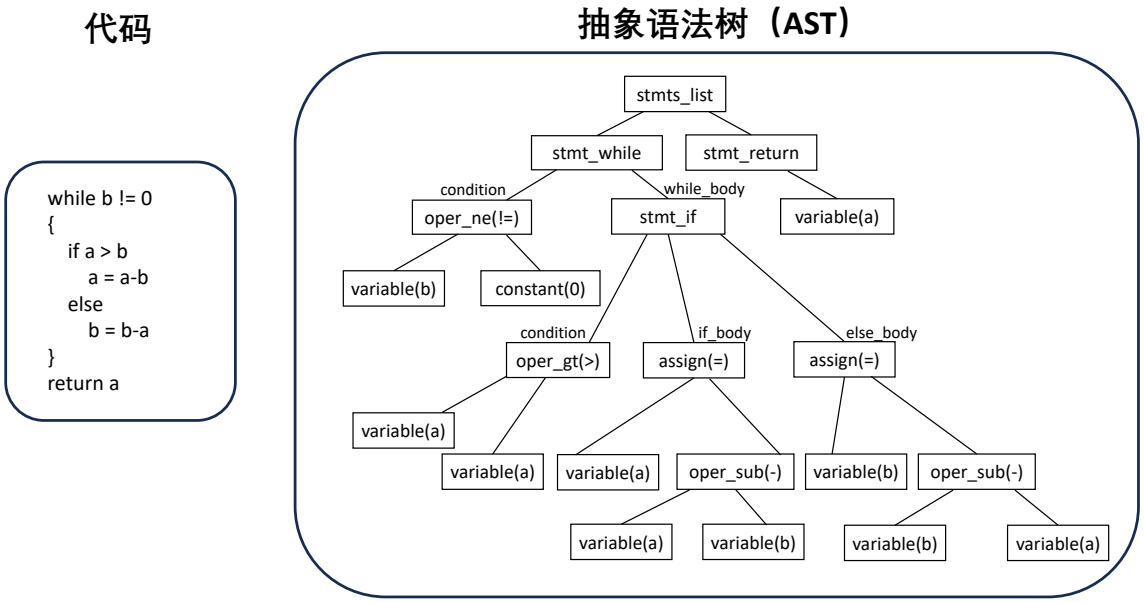


图 2.4　代码的抽象语法树示例

语法产生式定义了 AST 的构建规则，使代码在解析过程中被映射为具有层级关系的树状结构。该表示能显式刻画程序的嵌套关系，例如函数内部包含条件语句，条件语句中再嵌套循环结构。结构层级信息使模型能够区分不同语法单元之间的父子关系，从而提升对复杂程序结构的建模能力。在生成任务中引入结构信息，从而有助于减少语法错误并增强结构一致性。部分代码生成模型进一步结合 AST 结构进行

建模，通过图结构或树结构对程序进行表示学习，使模型在预测 token 时不仅依赖线性上下文，还结合其在语法树中的结构位置。然而，完全基于 AST 的方法存在计算复杂度较高以及部分语义信息难以表达的问题，因此当前主流方法通常采用“序列建模为主、结构信息增强”的混合方式，以平衡生成效率与结构一致性。

在语义约束方面，代码生成结果必须同时满足可编译性与执行正确性。可编译性要求代码通过语法与类型检查，而执行正确性则要求程序在运行时产生符合预期的输出。即使代码在语法上合法，逻辑错误仍可能导致执行失败，因此代码生成任务本质上是结构合法性与语义正确性的联合建模问题。单元测试反馈机制为代码生成提供了重要外部监督信号。通过执行测试用例，可以判断生成代码是否满足预期功能，并将结果用于优化生成策略。这使得代码生成从单纯的概率建模问题扩展为结合执行验证的优化过程。然而，该过程也面临计算开销较大以及测试覆盖不完全等问题，限制了其在实际场景中的应用。

总体而言，编程语言在语法约束、类型系统、作用域机制以及控制与数据流依赖方面均表现出高度结构化特征。这些特性使代码生成任务在建模复杂度上显著高于自然语言生成任务，并要求模型具备更强的结构建模与语义推理能力。

2.4 生成式代码水印相关技术与基础

2.4.1 生成式代码水印基本方法

随着大语言模型在代码生成、程序修复与自动测试等任务中的广泛应用，自动生成代码逐渐融入软件开发流程。然而，这也带来了代码来源难以追溯、知识产权归属不清以及潜在恶意代码难以识别等问题。为实现生成代码的可追踪性与版权保护，生成式水印技术被引入到代码生成场景中。

与自然语言文本相比，代码具有更严格的语法结构与执行语义约束。自然语言中常见的同义表达在代码中较为有限，任意的词元替换可能导致语法错误或语义偏移。因此，生成式代码水印需要在不破坏程序语法正确性与功能一致性的前提下，在生成过程或生成结果中嵌入可检测的统计信号或结构特征。

围绕水印嵌入的实现方式，现有技术大致可以归纳为三类：基于结构等价变换的方法、基于后处理的方法以及基于概率分布调制的方法。这三类方法在嵌入时机与实现机制上各有特点，共同构成生成式代码水印的技术基础。

基于结构等价变换的方法利用代码中语义保持的等价转换来承载水印信息。具体而言，该类方法通过在不改变程序行为的前提下，对代码进行语法层面的替换或改写，例如变量重命名、表达式等价改写或控制结构转换等。不同的等价变换选择可以映射为不同的水印比特，从而在生成代码的结构中隐式嵌入信息。这类方法通常不依赖模型内部状态，适用于模型无关的场景，但其嵌入能力受限于可用的等价变换空间。

基于后处理的方法则在代码生成完成后，通过语义保持的转换对已有代码进行水印嵌入。该类方法不干预模型生成过程，而是在生成结果上施加变换，例如代码结构重排、冗余表达插入或格式调整等。根据实现方式的不同，后处理方法既可以基于规则设计，也可以结合学习模型对嵌入位置与方式进行优化。该类方法具有较好的通用性，可直接作用于任意来源的代码，但其嵌入效果依赖于后处理阶段的可操作空间。

与上述两类方法不同，基于概率分布调制的方法直接作用于代码生成模型的解码过程，通过对每一步的输出概率分布施加可控偏置，使生成序列在统计上呈现特定特征。该类方法通常基于自回归生成框架，在不修改模型参数的前提下，通过调整候选词元的选择概率实现水印嵌入。

具体而言，在第 i 个生成位置，将词汇表 V 划分为两个互补子集：绿色列表 G_i 与红色列表 R_i，其中 $R_i = V \setminus G_i$，绿色列表的比例记为 γ。在生成过程中，对绿色列表中的词元施加固定偏置 δ，以提高其被选中的概率。设原始输出 logits 为 $l_i[v]$，则偏置后的 logits 表示为：

$$\tilde{l}_i[v] = \begin{cases} l_i[v] + \delta, & v \in G_i \\ l_i[v], & v \in R_i \end{cases} \tag{2.9}$$

经过 softmax 归一化后，得到新的概率分布：

$$\tilde{p}_i[v] = \frac{\exp(\tilde{l}_i[v])}{\sum_{u \in V} \exp(\tilde{l}_i[u])}. \tag{2.10}$$

由于绿色列表中的词元在概率上被系统性增强，生成序列中绿色词元的出现频率将高于随机情况，从而形成可检测的水印信号。在检测阶段，可以基于统计假设检验判断水印是否存在。设生成序列长度为 n，其中绿色词元数量为 k，则在零假设

（无水印）下，绿色词元服从参数为 γ 的二项分布。进一步可构造如下 z 统计量：

$$z = \frac{k - \gamma n}{\sqrt{n\gamma(1-\gamma)}}. \tag{2.11}$$

当 z 值超过预设阈值时，可认为该序列中存在水印信号。为提高安全性与抗攻击能力，绿色列表通常由密钥驱动的随机过程生成，使不同位置的划分具有不可预测性。

总体而言，结构等价变换方法与后处理方法主要作用于生成结果层面，而概率分布调制方法直接作用于生成过程本身。三类方法在嵌入机制与实现路径上形成互补，为后续面向代码语法约束与语义一致性的水印优化方法提供了基础。

2.4.2 基于熵选择的代码水印方法

在生成式代码水印研究中，一个核心挑战是如何在保证生成代码功能正确性的同时嵌入可检测的水印信号。与自然语言文本不同，代码具有严格的语法和语义约束，任何微小的结构性改变都可能导致编译错误或逻辑错误。因此，单纯依赖概率调制可能会破坏代码的可执行性。为此，一些研究提出了基于熵选择（entropy-based selection）的水印嵌入策略，通过量化每个 token 的生成不确定性来决定水印施加的位置，从而降低对代码功能的干扰。

基于熵选择的方法利用 token 熵作为判断标准，仅在高熵位置施加水印偏置。具体而言，设第 i 个 token 的生成概率分布为 $\tilde{p}_i[v]$，其熵可以表示为：

$$H_i = -\sum v \in V \tilde{p}_i[v] \log \tilde{p}_i[v]. \tag{2.12}$$

在生成第 i 个 token 时，该方法首先计算该位置的具体熵值 H_i。若 H_i 超过预设阈值 τ，则认为该位置具有较高的生成自由度，可安全嵌入水印；否则不进行水印调制。该策略可以形式化为：

$$\text{ApplyBias}(i) = \begin{cases} 1, & H_i > \tau \\ 0, & H_i \leq \tau \end{cases} \tag{2.13}$$

当满足条件时，模型会在绿色 token 上施加概率偏置，从而实现水印的嵌入。通过这种方式，水印主要集中在语义灵活的区域，例如变量名、常量值或可替换的表

达式，而避免对语法关键 token（如关键字、操作符、控制结构）产生干扰，从而降低对代码功能的影响。

基于熵的选择性水印方法具有明显优势：

(1) 可控性：通过调节熵阈值 τ 和偏置幅度 δ，可以在水印可检测性与代码正确性之间实现平衡。

(2) 安全性：水印主要嵌入高熵、语义灵活的 token 位置，不直接干扰关键语法结构，从而保持生成代码的可编译性和逻辑正确性。

(3) 灵活性：适用于不同类型的代码生成任务，无需依赖特定语言或模型结构。

然而，该方法仍然存在一定的局限性。基于熵选择的方法通常采用固定阈值 τ 和固定偏置 δ。如图2.5所示，在实际代码生成过程中，不同上下文环境下 token 分布差异较大，固定参数难以适应所有场景。当熵阈值过低时，该方法会对大量低熵 token 施加偏置，从而破坏代码逻辑，导致生成结果错误；当熵阈值过高时，则仅在少数高熵 token 上施加偏置，生成代码能够保持正确性，但水印嵌入不足，检测成功率较低。

提示Prompt

```
Write a python function takes in an integer and check whether the frequency of each digit in the integer is less than or equal to the digit itself.
assert validate(1233) == True
assert validate(112) == False
```

(a) 未嵌入水印代码

```
def validate(n):
    for i in str(n):
        if str(n).count(i) > int(i):
            return False
    return True
```

(b) SWEET(低阈值) 水印代码

```
def validate(num):
    num_str = str(num)
    freq_dict = {}
    for digit in num_str:
        if digit in freq_dict:
            freq_dict[digit] += 1
        else:
            freqDict[digit] = 1    # ✗ 未定义变量 'freqDict'
    for digit, freq in freq_dict.items():
        if int(digit) > freq:    # ✗ 逻辑错误
            return False
    return True
```

正确性: ×
可检测性: √

(c) SWEET(高阈值) 水印代码

```
def validate(n):
    for char in str(n):
        if str(n).count(char) > int(char):
            return False
    return True
```

正确性: √
可检测性: ×

图 2.5 基于熵选择方法在不同熵阈值下的局限性示例

单纯依赖熵信息并不能完全反映代码结构的重要性。一些语法关键 token 虽然可能具有中等熵值，但其语义仍然十分关键，对这些位置进行概率扰动仍然可能影响代码功能。因此，仅依赖静态阈值进行控制的方法在复杂代码生成场景中仍然存在

一定局限性。

综上所述，现有生成式代码水印方法通常通过概率调制和熵选择策略实现水印嵌入，但在适应不同代码上下文和保证语法安全性方面仍存在改进空间。这也为后续提出更加自适应的代码水印方法提供了研究基础。

2.5 本章小结

本章围绕代码水印技术的基础理论与相关问题展开介绍。首先，从生成式模型出发，说明了自回归建模与概率采样机制，揭示生成过程本质上是受控的概率决策过程。随后，介绍了文本水印的基本概念、主要方法及评价指标，并分析了水印方法在生成质量、鲁棒性与可检测性之间的基本权衡关系。在此基础上，进一步讨论了代码生成的结构特性，指出其相较自然语言生成具有更严格的语法与语义约束。最后，对生成式代码水印的主要方法进行了概括，并从整体层面说明其在实际应用中仍存在一定局限性。总体而言，代码水印是在严格结构约束下的概率调制问题，如何在保证生成正确性的前提下实现有效嵌入，是后续方法设计的核心问题。

第三章　基于熵自适应的代码生成水印方法

本章围绕生成式代码水印的功能保真与鲁棒性优化问题展开研究，关注两个核心矛盾：一是如何在不破坏代码语法与执行正确性的前提下实现有效水印嵌入；二是如何提升水印在不同代码上下文及潜在结构化攻击下的可检测性和稳定性。针对现有方法在固定熵阈值下易破坏关键词元或嵌入强度不足的问题，本章从代码生成过程中词元熵的分布特性与上下文敏感性出发，提出一种基于熵自适应的水印嵌入框架，以降低水印嵌入对语法结构和逻辑功能的干扰，从而提升整体水印系统的可靠性与实用性。

3.1　引言

随着大语言模型在代码生成中的广泛应用，生成代码的来源可追踪性与版权保护问题日益突出。生成式代码不仅承载着实际功能，还具有严格的语法和执行语义约束，任何对关键词的干扰都可能导致编译错误或逻辑异常。因此，研究能够在保证代码功能正确性的前提下嵌入可检测信号的水印方法显得尤为重要。传统生成式代码水印方法多依赖固定概率偏置或静态熵阈值选择策略，例如通过在高熵位置施加固定偏置实现水印嵌入。然而，在实际生成过程中，不同上下文环境下 token 分布差异较大，固定参数往往难以同时兼顾水印可检测性和代码功能完整性。熵阈值过低时，水印容易破坏语法关键词，导致生成代码不可执行；熵阈值过高时，仅在少数高熵位置嵌入水印，导致检测成功率下降。这种两难局面限制了现有方法在复杂代码生成场景下的实用性。

为了解决这一问题，本章提出基于熵自适应的代码水印方法。该方法结合 token 的局部生成不确定性和上下文敏感性，自适应地调整水印嵌入策略。在生成每个 token 时，模型会计算其熵值，并结合上下文信息动态决定是否施加概率偏置以及偏置大小。高熵且语义灵活的 token 被优先选中进行水印嵌入，而低熵或语法关键词则避免干预，从而在保证代码可编译性和正确性的同时，实现稳定可靠的水印嵌入。通过实验验证，基于熵自适应的水印方法在不同代码生成任务中能够保持较高的嵌入成功率和检测精度，同时显著降低对代码功能的破坏风险。

3.2 总体框架

为在保证代码功能正确性的前提下，实现具有高可检测性的水印嵌入，本文提出一种熵感知水印框架。该框架的核心思想是在自回归解码的每一个时间步，根据当前候选词的预测熵动态调整水印嵌入强度，从而使水印信号的分布与代码局部的不确定性相匹配。该方法不改变底层语言模型的参数，仅通过调整 logits 实现水印嵌入，因此可无缝应用于任何能够输出 logits 的生成模型。

本章方法的整体架构如图 3.1 所示，主要包含三个功能模块：熵计算模块、自适应偏置生成模块以及水印嵌入与采样模块。框架的输入是用户提供的提示词序列 $x = \{x_1, x_2, \ldots, x_{M-1}\}$，输出是带有水印的代码序列 $y = \{y_1, y_2, \ldots, y_N\}$。整个生成过程遵循标准的自回归方式，但在每一步 t 引入额外的熵感知处理。

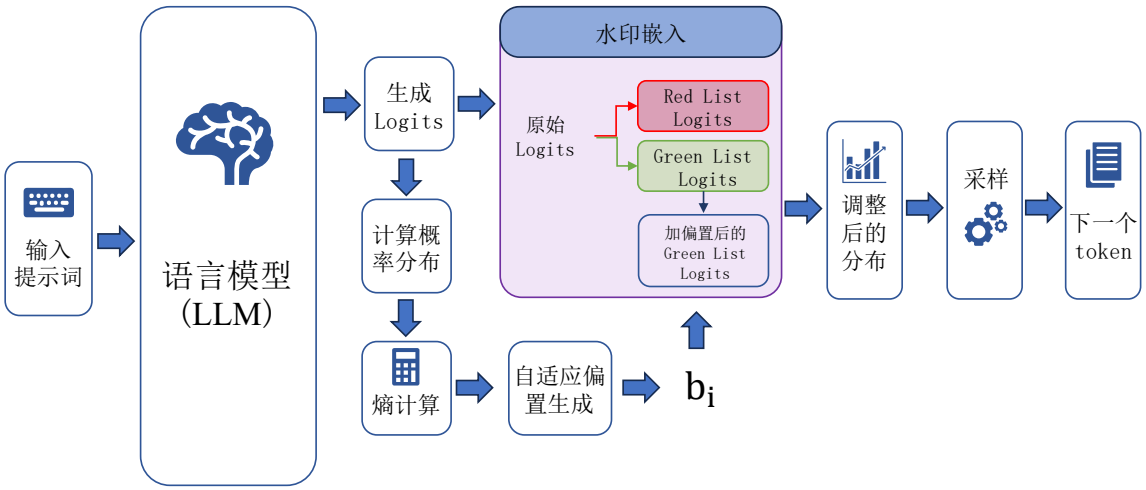


图 3.1 熵自适应的代码生成水印总体框架图

从整体上看，熵自适应的代码生成水印框架具备以下几个关键特性：

(1) 熵感知（Entropy-Awareness）：框架实时计算每个 token 的熵，使水印强度的调节严格依赖于局部上下文的不确定性，避免了盲目施加均匀水印可能导致的语法破坏。

(2) 自适应（Adaptivity）：偏置函数具有连续、有界的特性，能够平滑响应熵值的变化，既避免了固定阈值带来的硬切换，也保证了水印强度始终处于可控范围内。

(3) 无模型修改（Model-agnostic）：所有水印操作均在解码阶段完成，不涉及对语言模型权重的任何改动，因此该水印方法可以即插即用地应用于各种预训练代码生成模型。

(4) 端到端可检测（End-to-end Detectability）：水印信号被系统地嵌入到高熵 token 中，检测阶段只需重现绿名单划分并统计高熵区域的绿名单比例，即可通过 z 检验判断代码是否包含水印。

熵自适应的代码生成水印框架通过将水印强度与 token 级熵动态耦合，在保证代码功能正确的前提下显著提升了水印的可检测性。其模块化设计使得嵌入与检测两个阶段既独立又协同，为代码生成模型的可信溯源提供了一种灵活且鲁棒的解决方案。3.3 节和 3.4 节将分别对水印嵌入算法和检测算法进行详细阐述。

3.3 水印嵌入

水印嵌入的目标是在代码生成过程中，向每个 token 的采样概率中注入可控的统计偏差，同时最大限度地保持代码的功能正确性和自然度。本章方法的水印嵌入机制由四个关键步骤构成：熵计算、自适应偏置生成、绿名单划分与偏置应用、以及采样。本节将依次阐述这些步骤，并给出完整的嵌入算法。

3.3.1 熵计算

在自回归生成的每一步 t，语言模型基于已生成的上下文 $y_{<t}$ 输出 logits 向量 $s_t \in \mathbb{R}^{|V|}$（V 为词表），然后通过 softmax 函数得到下一个 token 的概率分布：

$$p_t(v) = \frac{\exp(s_t[v])}{\sum_{v' \in V} \exp(s_t[v'])}, \quad \forall v \in V. \tag{3.1}$$

该分布的熵定义为：

$$H_t = -\sum_{v \in V} p_t(v) \log p_t(v). \tag{3.2}$$

熵 H_t 反映了模型在当前步预测的不确定性：值越大，表示模型对下一个 token 的预测越分散（即有多种可能的选择）；值越小，表示模型越确信（如语法关键字几乎唯一）。在代码生成中，低熵区域通常对应于语法强制性的 token（如 ‘if‘、‘for‘、括号等），而高熵区域则包括变量名、字符串字面量、注释等自由度较高的部分。本方法利用这一特性，在低熵区域施加微弱水印以避免破坏语法结构，在高熵区域则加强水印以提升可检测性。

3.3.2 自适应偏置机制

在生成式代码水印嵌入过程中，偏置强度的设计直接决定了水印信号的可检测性与代码生成质量之间的平衡关系。传统方法通常采用固定偏置策略，即在满足条件的生成位置上施加统一强度的概率调制。然而，这种方式隐含地假设所有可嵌入位置具有相同的扰动容忍度，而这一假设在代码生成场景中并不成立。实际生成过程中，不同位置的概率分布具有显著差异，其不确定性呈现连续变化特征：部分位置（如关键字或固定结构）分布高度集中，而部分位置（如变量命名或表达式细节）则具有较高自由度。因此，固定偏置难以同时兼顾不同上下文下的嵌入需求。

针对上述问题，本文引入熵自适应偏置机制，使水印嵌入强度能够根据当前位置的不确定性动态调整。其基本思想是将偏置设计为词元级熵的单调递增函数：当熵值较低时，模型对该位置预测具有较强确定性，此时应避免施加扰动；当熵值较高时，候选空间更为丰富，可以在不显著影响生成质量的前提下引入更强的水印信号。通过这种方式，可以在不同生成区域实现差异化调制，从而提高整体嵌入的精细程度。

具体而言，设第 t 个生成位置的概率分布熵为 H_t，本文定义偏置函数 b_t 如下：

$$b_t = \min\left(g \cdot \left(1 + \beta \cdot \left(1 - e^{-\alpha \cdot \max(0, H_t - H_0)}\right)\right),\ m\right), \tag{3.3}$$

其中，函数通过指数形式实现平滑增长，并结合上限约束控制偏置范围。各参数的具体含义如下：

- g：基础偏置，对应于刚超过阈值时的初始调制强度；
- α：增长率因子，控制偏置随熵增加的敏感程度；
- β：放大因子，决定偏置的最大增长幅度；
- H_0：熵阈值，用于区分可嵌入区域与不可嵌入区域；
- m：偏置上限，用于限制最大扰动幅度。

从函数形式上看，该机制可以分为三个典型区域进行理解：

（1）低熵区域（$H_t \leq H_0$）。此时 $\max(0, H_t - H_0) = 0$，函数输出为 $b_t = 0$，即不施加任何偏置。该区域通常对应语法关键位置或高确定性位置，如关键字、运算符或固定结构。通过完全关闭水印嵌入，可以有效避免对代码语法与执行逻辑的干扰。

（2）过渡区域（$H_t > H_0$ 且未达到饱和）。当熵值刚超过阈值时，指数项逐渐增

大，偏置从基础值 g 开始连续增长。该区域对应具有一定灵活性的生成位置，水印嵌入强度随不确定性逐步提升，实现“软激活”过程。相比于硬阈值策略，该连续变化能够避免生成分布的突变，从而减少不自然现象。

（3）高熵饱和区域。随着 H_t 持续增大，指数项趋于稳定，偏置逐渐逼近上限 m。该区域通常对应高自由度位置，例如变量名或局部表达式结构，在这些位置可以施加较强水印而不显著影响程序语义。通过设置上限，可以防止偏置过大导致生成分布严重偏移。

该函数设计具有以下几个关键性质：

（1）连续性。指数函数保证了偏置随熵值平滑变化，避免传统阈值方法中“开/关”切换带来的不连续问题，从而提升生成结果的自然性。

（2）有界性。通过 min 操作对偏置进行截断，使其始终处于可控范围内，防止过强扰动破坏生成质量。

（3）熵感知性。偏置强度由熵值驱动，使水印嵌入自动适应不同上下文环境，实现从“统一调制”向“精细调制”的转变。

（4）参数可调性。参数 α 决定曲线的上升速度：较大的 α 会使偏置在接近阈值时迅速增长，适用于需要较强嵌入的场景；较小的 α 则使增长更加平缓，有助于保持生成稳定性。参数 β 控制最大放大倍数，从而影响水印信号的整体强度；而参数 g 则决定偏置的起始水平。通过对这些参数的组合调节，可以针对不同任务需求灵活配置水印嵌入策略。

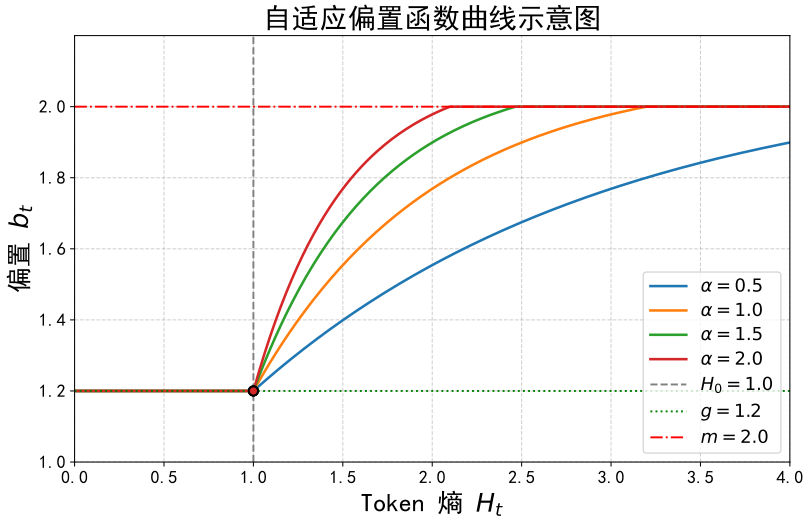


图 3.2 自适应偏置函数曲线示意图。横轴为 token 熵 H_t，纵轴为偏置 b_t。图中标注了阈值 H_0、基础偏置 g、上限 m，并展示了不同 α 取值下的曲线形状，说明偏置如何随熵值变化。

图 3.2 展示了不同参数下偏置随熵值变化的曲线。当 $H_t \leq H_0$ 时，偏置为 0，水印完全不激活；当 H_t 超过 H_0 后，偏置开始指数增长，逐渐逼近上限 m。参数 α 控制上升的陡峭程度，β 决定最终能达到的最大倍数。通过调节这些参数，可以针对不同数据集和任务灵活调整水印的强度分布。

3.3.3 色表划分与偏置应用

在得到当前位置的自适应偏置 b_t 后，需要将其具体作用到候选 token 集合上，从而在生成过程中引入可检测的统计偏置。该过程主要包括两个关键步骤：基于哈希函数的色表划分，以及对绿色 token 的概率调制。

首先，为了在生成阶段与检测阶段之间建立一致的映射关系，需要构造一种可重现的随机划分机制。为此，本文采用基于哈希函数的伪随机划分策略：利用前一个生成 token y_{t-1} 作为随机种子，对词表 V 进行确定性随机划分，从而得到当前步的绿名单 G_t 和红名单 R_t。具体地，划分过程可表示为：

$$(G_t, R_t) = \text{Split}(V, \gamma, \text{hash}(y_{t-1})), \tag{3.4}$$

其中 $\gamma \in (0, 1)$ 表示绿名单的比例，hash$(\cdot)$ 为一个伪随机哈希函数，Split 表示基于种子的随机划分过程。

从实现角度来看，Split 通常通过如下方式构造：首先对词表中每个 token $v \in V$ 计算其对应的哈希值 hash(y_{t-1}, v)，然后根据哈希值对所有 token 进行排序或映射到 $[0, 1]$ 区间。随后选择哈希值最小的前 $\gamma|V|$ 个 token 作为绿名单，其余 token 划入红名单。该过程本质上等价于在给定随机种子的条件下对词表进行一次随机置换，并截取前缀作为绿色集合。

这种基于哈希的划分方式具有以下重要性质：（1）确定性：在给定相同上下文（即相同的 y_{t-1}）时，生成阶段与检测阶段能够得到完全一致的划分结果，从而保证水印可检测性；（2）随机性：由于哈希函数具有良好的均匀性，不同位置的划分结果在统计上近似独立，使水印信号在序列中均匀分布；（3）不可见性：划分过程依赖于内部种子，对外不可观测，增加了攻击者恢复或规避水印的难度。

在完成色表划分后，下一步是将自适应偏置 b_t 应用于绿名单 token。设当前时刻

模型输出的原始 logits 为 $s_t[v]$，则经过调制后的 logits 为：

$$s'_t[v] = \begin{cases} s_t[v] + b_t, & v \in G_t, \\ s_t[v], & v \in R_t. \end{cases} \tag{3.5}$$

该操作本质上是在对数空间中对绿色 token 进行平移，从而在 softmax 归一化后改变其相对概率。具体而言，softmax 函数对 logits 的指数形式具有放大效应，因此即使 b_t 取值较小，也能够在概率空间中产生可观的偏移。对于任意 token v，其调整后的概率可写为：

$$p'_t[v] = \frac{\exp(s'_t[v])}{\sum_{u \in V} \exp(s'_t[u])}. \tag{3.6}$$

对于绿名单中的 token，由于 $s'_t[v] = s_t[v] + b_t$，其概率将按比例被放大为原来的 $\exp(b_t)$ 倍（在归一化前），而红名单 token 的相对概率则相应下降。因此，从整体上看，绿色 token 在采样过程中的被选中概率将系统性提升。

在完成 logits 调制后，从新的概率分布中进行采样，得到当前生成 token：

$$y_t \sim \text{softmax}(s'_t). \tag{3.7}$$

通过在每一步重复上述过程，生成序列中绿色 token 的出现频率将显著高于其理论比例 γ，从而在统计意义上形成稳定的偏置模式。这种偏置在单个位置上难以察觉，但在长序列上会逐渐累积，使得检测器能够通过统计检验区分带水印与无水印文本。

值得注意的是，偏置 b_t 与绿名单比例 γ 在该过程中共同作用：γ 决定了潜在可嵌入 token 的覆盖范围，而 b_t 决定了实际嵌入强度。较大的 γ 会增加可调制 token 数量，但同时也会稀释单个 token 的统计贡献；较大的 b_t 则会增强单步偏移，但可能对生成分布产生更明显影响。因此，两者需要协同设计，以在水印可检测性与生成质量之间取得平衡。

综上，基于哈希函数的色表划分与自适应偏置应用构成了生成式水印嵌入的核心执行机制：前者提供了稳定且不可见的随机结构，后者则通过概率调制在该结构上注入统计信号，从而实现对生成序列的隐式标记。

3.3.4 嵌入算法流程

算法 3.1 总结了熵自适应代码水印方法的完整嵌入过程。算法输入为提示词序列 x 和超参数集合 $\{g, \alpha, \beta, H_0, m, \gamma\}$，输出为带有水印的代码序列 y。在每一步 t，首先计算 logits 和熵；若熵超过阈值，则根据式 (3.3) 计算自适应偏置，并根据前一个 token 的哈希值生成绿/红名单，然后按照式 (3.5) 调整 logits；最后从调整后的分布中采样得到当前 token。该过程循环直至生成结束标记或达到最大长度。

算法 3.1: 熵自适应的代码水印嵌入算法

输入: 提示词序列 $x = \{x_1, \ldots, x_{M-1}\}$；参数 g, α, β, H_0, m

输出: 水印代码序列 $y = \{y_1, \ldots, y_N\}$

初始化 y_0 为提示词的最后一个 token
for $t = 0, 1, 2, \ldots$ **do**
    计算 logits s_t 和概率分布 $p_t = \text{softmax}(s_t)$
    计算 token 级熵 $H_t = -\sum_v p_t[v] \log p_t[v]$
    根据前一个 token y_{t-1} 的哈希值生成绿/红名单 (G_t, R_t)
    if $H_t > H_0$ **then**
        根据式 (3.3) 计算自适应偏置 b_t
        按照式 (3.5) 调整 logits（将 b_t 加到绿名单 token 上）
    end
    从调整后的分布 $\text{softmax}(s'_t)$ 中采样得到 y_t
    if y_t 为结束标记 **then**
        break
    end
end
return $y = \{y_1, \ldots, y_t\}$

3.4 水印检测

水印检测的目标是判断一段给定的代码是否由本章所提出的框架生成，而无需访问水印嵌入时的随机状态。检测器利用水印嵌入阶段留下的统计痕迹，即在高熵区域绿名单 token 的比例系统性偏高，通过假设检验的方法做出判定。本节将详细阐述熵自适应代码水印方法的检测算法、统计原理以及与嵌入阶段的协同机制。

3.4.1 检测算法流程

算法 3.2: 熵自适应的代码水印检测算法

输入: 提示词 x；待检测序列 y；参数 $\gamma, H_0, z_{\text{th}}$

输出: True（水印代码）或 False（人类编写）

初始化计数器 $N^H = 0$，$N_G^H = 0$
for $t = 0, 1, \ldots, N-1$ **do**
    计算 $p_t = \text{softmax}(s_t)$ 与熵 H_t
    if $H_t > H_0$ **then**
        根据前一个 token 的哈希值重建绿/红名单 (G_t, R_t)
        $N^H \leftarrow N^H + 1$
        if $y_t \in G_t$ **then**
            $N_G^H \leftarrow N_G^H + 1$
        end
    end
end
计算 z 分数 $z = \dfrac{N_G^H - \gamma N^H}{\sqrt{\gamma(1-\gamma)N^H}}$
if $z > z_{\text{th}}$ **then**
    return True
end
else
    return False
end

算法 3.2 给出了完整的水印检测流程。与嵌入阶段类似，检测过程同样依赖于模型在每个生成位置的概率分布信息，其核心思想是在不访问嵌入参数（如偏置 b_t）的情况下，仅通过统计分析判断生成序列是否存在系统性偏置。

具体而言，检测流程首先基于提示词 x 和已生成序列逐步重建生成过程。在每一个位置 t，利用同一语言模型重新计算该位置的概率分布 p_t，并据此计算熵值 H_t。该步骤的作用在于识别“可嵌入区域”，即筛选出在嵌入阶段可能施加水印的位置。只有当 $H_t > H_0$ 时，该位置才被认为是高不确定性区域，才纳入统计范围。

在筛选出高熵位置后，检测算法利用与嵌入阶段一致的哈希函数与划分规则，基于前一个 token y_{t-1} 重建当前步的绿/红名单划分 (G_t, R_t)。由于该划分是确定性的，检测端无需访问任何额外信息即可精确复现嵌入时的随机结构。随后，通过判断实际生成 token y_t 是否落入绿名单，对绿 token 数量进行累积统计。

通过遍历整个序列，最终得到两个统计量：高熵位置总数 N^H 以及其中属于绿名单的 token 数量 N_G^H。这两个量构成后续统计检验的基础。直观上，如果序列不包含水印，则绿 token 出现的频率应接近其理论比例 γ；而若存在水印，则由于嵌入阶段对绿 token 进行了概率提升，实际观测到的比例将系统性偏大。

基于上述统计量，可以构造标准化的 z 分数，对观测结果进行显著性检验。最终，通过将 z 值与预设阈值 z_{th} 进行比较，判断序列中是否存在水印信号。

3.4.2 检测原理与统计检验

在原假设成立时，由于绿/红名单的划分仅由哈希函数随机生成，与生成内容无关，因此每个高熵位置的 token 落入绿名单的概率应为 γ。此外，由于不同位置的划分由不同的上下文种子驱动，其结果在统计上近似独立。因此，可以将高熵位置上的观测过程近似建模为一组独立伯努利试验，从而有：

$$N_G^H \sim \text{Binomial}(N^H, \gamma). \tag{3.8}$$

在实际应用中，当 N^H 较大时（通常大于 30），根据中心极限定理，该二项分布可以近似为正态分布：

$$N_G^H \approx \mathcal{N}(\gamma N^H,\ \gamma(1-\gamma)N^H). \tag{3.9}$$

基于此，可以构造标准化统计量：

$$z = \frac{N_G^H - \gamma N^H}{\sqrt{\gamma(1-\gamma)N^H}}. \tag{3.10}$$

在原假设成立时，z 近似服从标准正态分布 $\mathcal{N}(0,1)$。而在备择假设下，由于嵌入阶段对绿 token 进行了概率提升，N_G^H 的期望将大于 γN^H，从而使 z 呈现显著正偏。因此，可以通过单侧检验判断是否拒绝原假设。

需要指出的是，上述独立性假设在严格意义上并不完全成立，因为生成过程本身具有序列依赖性。然而，由于绿/红名单划分依赖于哈希函数，其随机性在一定程度上削弱了这种依赖关系，使得该近似在实践中具有较好效果。

3.4.3 检测阈值的选取与误报控制

检测阈值 z_{th} 的选择直接影响检测系统的可靠性，其本质是在假阳性率（False Positive Rate, FPR）与检测率（True Positive Rate, TPR）之间进行权衡。在原假设下，z 服从标准正态分布，因此可以通过分位数函数确定阈值：

$$z_{\text{th}} = \Phi^{-1}(1 - \text{FPR}),$$

其中 $\Phi(\cdot)$ 为标准正态分布的累积分布函数。

在实际应用中，不同场景对误报的容忍度不同。例如，在版权保护或法律取证场景中，通常要求极低的误报率（如 10^{-3} 或更低），此时需要设置较高的阈值（如 $z_{\text{th}} \geq 3$）；而在内容筛查等场景中，可以适当放宽误报约束，以提高检测灵敏度。

此外，检测性能还受到样本规模的影响。当高熵 token 数量 N^H 较小时，统计波动较大，z 分数的稳定性降低，此时正态近似可能不够精确。为提高检测可靠性，可以采用以下策略：一是增加检测序列长度，以提升统计显著性；二是在小样本情况下采用精确二项检验替代正态近似。

3.4.4 与嵌入阶段的协同机制

检测算法与嵌入阶段的协同依赖于三个关键参数的一致设置。首先，熵阈值 H_0 必须在两个阶段完全相同，以确保 “高熵 token” 的定义一致，若该阈值存在偏差，检测器关注的位置集合将与水印实际嵌入的位置错位，从而导致检测性能下降。其次，绿名单比例 γ 也需要保持一致，这样才能在统计检验中保证期望概率的匹配。最后，

检测阶段必须使用与嵌入阶段完全相同的伪随机函数和种子生成规则（即基于前一个 token 的哈希值），以正确重建每个位置的绿/红名单划分，从而确保统计推断的一致性。

3.5 实验结果与分析

3.5.1 实验设置

所有实验均以 Qwen2.5-Coder-3B[68] 作为基础语言模型开展。在数据集的选择上，我们采用了三个在代码生成领域广泛使用的基准来全面评估所提方法的性能。HumanEval+ 数据集源自 HumanEval 基准[22]，并在此基础上通过引入更丰富的测试用例进行增强[69]。该数据集包含 164 个手工编写的 Python 编程问题，每个问题均配有更为详尽的测试用例，能够更精确地衡量生成代码的功能正确性，是代码生成任务中最具代表性的评估基准之一。MBPP+ 则是在 MBPP 数据集[70] 基础上的扩展版本，同样通过补充测试用例提升评测严格性[69]，其包含约 500 个基础编程任务，涵盖循环、条件、字符串操作等常见编程概念，用于检验模型在多样化简单任务上的泛化能力。为了验证本方法在不同编程语言上的适用性，我们还引入了 HumanEvalPack[71]，该数据集将 HumanEval 扩展至多语言环境，包括 Python、C++ 和 Java 三个版本，从而能够评估水印方法在跨语言代码生成任务中的迁移能力。所有数据集的评估均采用官方提供的测试用例，并通过 pass@1 指标进行度量。

在对比方法的选择上，我们将本章方法与三种代表性的无训练水印方法进行对比，这些方法均无需修改模型参数，仅通过调整生成过程或检测策略嵌入水印信号。KGW[38]是最基础的硬水印方法，它在每一步生成时对绿名单中的 token 施加固定偏置 δ，从而在整体上提高绿名单 token 的出现频率，该方法直接体现了水印嵌入的核心思想，但未考虑代码的结构特性。EWD[72]在检测阶段引入了熵加权的思想，对高熵 token 赋予更高的检测权重以提升检测鲁棒性，然而其嵌入阶段仍然采用固定偏置，未能从根本上解决低熵区域的干扰问题。SWEET[73]是当前代码水印任务中的代表性方法，它通过设定固定的熵阈值，仅对超过阈值的 token 施加固定偏置，从而避免对语法关键区域的干扰，但固定偏置的强度选择仍需在正确性和可检测性之间进行权衡。上述方法为本章提出的方法提供了全面的对比基准，有助于验证熵自适应机制的有效性。

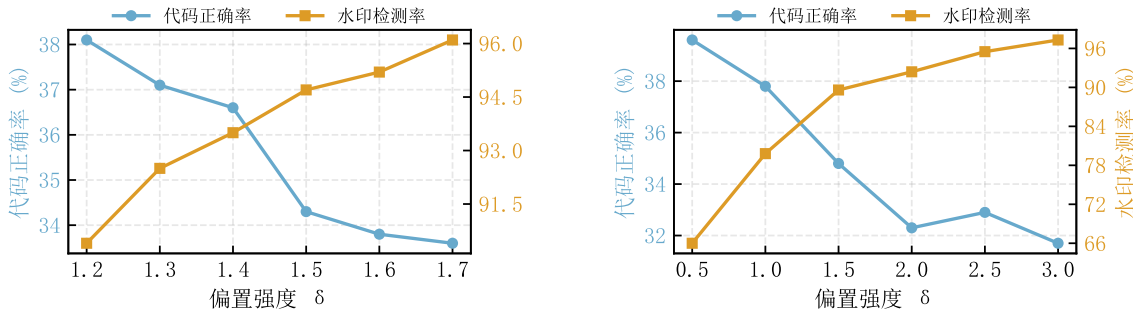
在评价指标方面，我们从代码正确性和水印可检测性两个维度对方法进行综合评估。对于代码正确性，我们采用 pass@1 指标，即生成的代码能够一次通过所有测试用例的概率。为了降低单次采样带来的随机性，我们对每个编程任务进行 20 次独立采样，并取平均值作为最终结果。对于水印可检测性，我们采用了三个互补的指标。AUROC（Area Under the Receiver Operating Characteristic curve）[74]衡量检测器在不同阈值下区分水印代码与人类编写代码的整体能力，其值越接近 100% 表示检测性能越好。此外，我们还报告了在固定假阳性率下的真阳性率，即 TPR@0.05 和 TPR@0.01，分别对应假阳性率为 5% 和 1% 时的检测成功率。这两个指标能够更直观地反映检测器在实际应用中低误报要求下的实用性，例如在需要严格控制误判的场景下，TPR@1% FPR 的值尤为重要。通过上述多维度的评估体系，我们能够全面衡量该方法在保持代码质量的同时提升水印可检测性的能力。

3.5.2 参数选择

本章方法涉及多个关键超参数，包括基础偏置 g、动态增长率 α、强度调节因子 β、熵阈值 H_0 以及偏置上限 m。这些参数共同决定了水印嵌入强度及其随上下文不确定性的动态调节方式，因此合理的参数配置对于在“代码正确性—水印可检测性”之间取得平衡至关重要。为此，本文采用分阶段、数据驱动的参数选择策略。

在粗粒度搜索阶段，我们基于验证集对关键参数进行网格搜索。所有实验均在 Qwen2.5-Coder-3B 模型上开展，并分别在 HumanEval+ 和 MBPP+ 数据集中随机抽取 20% 样本构建验证集。参考 SWEET 框架中的偏置机制，我们以 δ 表征基础偏置强度，并固定绿色词表比例 $\gamma = 0.5$。具体地，在 $\delta \in [1.0, 2.0]$ 区间内以 0.1 为步长进行搜索，同时对 $\alpha, \beta \in [0.5, 1.5]$ 进行联合搜索。参数范围的设定基于预实验观察：较小取值会导致水印信号不足，从而降低检测性能；而过大的取值则会对候选词分布施加过强扰动，显著影响代码生成质量。评估过程中同时考虑 pass@1 与 AUROC，以刻画生成正确性与水印可检测性的综合表现。实验结果表明，随着 δ 增大，AUROC 整体呈上升趋势，但当偏置超过一定范围时，pass@1 会明显下降。这一现象在 Fig. 3.3(a) 中得到体现：当 $\delta \in [1.2, 1.4]$ 时，pass@1 基本保持稳定，而在 $\delta = 1.5$ 时出现显著下降，说明过强的偏置会破坏代码语义正确性。因此，在粗选阶段将候选区域缩小至 $\delta \in [1.2, 1.4]$，并筛选出若干在两项指标之间具有良好权衡的 (α, β) 组合。

在此基础上，我们进一步在候选区域内开展细粒度敏感性分析，以确定最具鲁



(a) MBPP+ (b) HumanEval+

图 3.3 SWEET 方法在 MBPP+ 与 HumanEval+ 数据集上 pass@1 与 AUROC 随偏置参数 δ 的变化关系

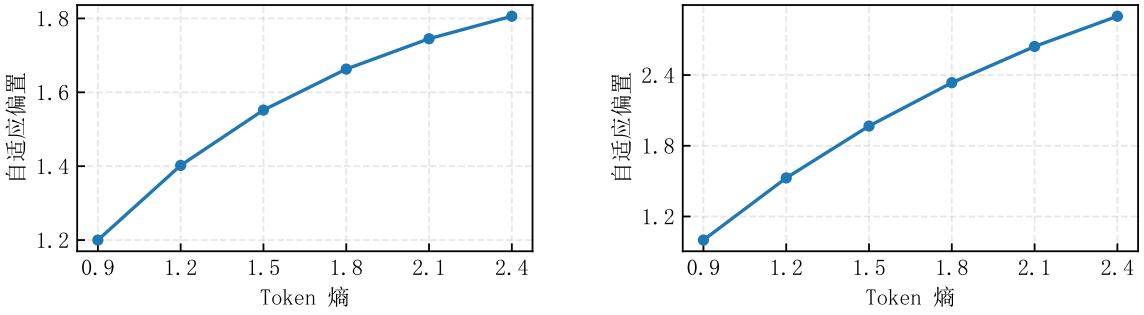
棒性的参数配置。具体而言，通过逐步调整 δ、α 和 β，观察其对 pass@1 与 AUROC 的局部影响。结果表明，在 MBPP+ 数据集中，较小的基础偏置更有利于维持语法正确性，因此选择 $\delta = 1.2$ 作为基准偏置；同时设置 $\alpha = 1.0$，使模型在高于熵阈值时能够适度增强偏置，从而提升对不确定位置的调节能力；设置 $\beta = 0.65$，以在高熵区域增强水印嵌入，同时通过上限约束避免整体偏置过大。偏置上限设置为 $m = 2.0$，以保证嵌入强度始终处于安全范围内。

对于 HumanEval+ 数据集，由于其包含更多自由格式内容（如注释与字符串）以及更高比例的高熵位置，我们基于其熵分布统计独立进行参数选择。通过相同的两阶段流程，最终采用相对较小的基础偏置 g 与较大的 β，以强化高不确定区域的水印嵌入能力，同时通过较低的 α 控制偏置增长速度，从而减少对低熵及关键语法位置的干扰。此外，适当提高偏置上限 m，以适应其更宽松的表达空间。实验结果（如 Fig. 3.3 所示）表明，该配置能够在不降低代码正确性的前提下，显著提升水印检测性能。

总体而言，参数选择过程揭示了不同任务场景下的差异性：MBPP+ 更强调语法严谨性与函数正确性，因此需要较低的偏置强度与上限约束；而 HumanEval+ 中高熵位置占比较高，允许在局部区域施加更强偏置以提升水印信号强度。在多语言场景中，由于语法约束更强，参数设置需在稳定性与检测性能之间进行更为保守的权衡。

3.5.3 实验结果与分析

图 3.4 展示了本方法中 token 级熵与自适应偏置之间的关系。从图中可以看出，不同熵区间对应不同的偏置调节策略：对于低熵 token（通常为关键语法元素，如关



(a) MBPP+ (b) HumanEval+

图 3.4 自适应偏置与 token 熵之间的关系

键字、控制流结构等），模型仅施加较小的偏置，从而避免对程序语法结构造成破坏；而对于高熵 token（如变量名、常量或自由文本区域），偏置随熵值增加而逐渐增强，并最终受限于上界 m。这种连续且上下文感知的调节机制，使得水印主要嵌入在对语义影响较小的区域，从而在保证代码正确性的同时提升水印信号强度。

具体而言，图 3.4(a) 给出了 MBPP+ 数据集上的变化趋势，可以观察到偏置在中高熵区域平滑增长且整体幅度受控，有利于维持代码的语法严谨性；而图 3.4(b) 则展示了 HumanEval+ 数据集上的对应结果，由于该数据集中高熵位置占比较高，偏置增长更为显著，从而能够在更大范围内增强水印嵌入能力。两者共同表明，本方法能够根据不同任务的数据分布特性，自适应调整水印嵌入策略。

表 3.1 MBPP+ 数据集实验结果

Method	pass@1	AUROC	TPR@0.05	TPR@0.01
KGW	36.6	77.7	42.1	16.8
EWD	36.6	91.3	70.4	35.8
SWEET($\delta = 1.3$)	37.1	92.5	66.6	42.1
SWEET($\delta = 1.7$)	33.6	96.1	79.9	62.8
Ours	37.8	96.2	84.7	67.7

表 3.1 和表 3.2 分别给出了在 MBPP+ 与 HumanEval+ 数据集上的实验结果。从整体上看，现有方法在代码正确性与水印可检测性之间普遍存在明显的权衡关系。

具体而言，KGW 由于在所有 token 上施加均匀偏置，缺乏对上下文不确定性的区分能力，导致水印信号较弱，其 AUROC 和 TPR 指标均处于较低水平，难以实现有效检测。EWD 在检测阶段引入熵权重，对高熵 token 赋予更高重要性，从而在一定程度上提升了检测性能，但由于其嵌入阶段缺乏精细控制，整体效果仍受到限制。

表 3.2　HumanEval+ 数据集实验结果

Method	pass@1	AUROC	TPR@0.05	TPR@0.01
KGW	35.4	78.9	40.2	11.0
EWD	35.4	81.7	46.3	22.0
SWEET($\delta = 1.1$)	36.2	81.4	47.5	26.8
SWEET($\delta = 2.0$)	33.5	92.4	76.2	61.6
Ours	37.2	92.6	77.4	67.1

SWEET 方法通过在高熵位置选择性嵌入水印，进一步改善了检测能力，但其性能对偏置强度参数 δ 高度敏感。当 δ 取较小值时（如 MBPP+ 上的 $\delta = 1.3$），其 pass@1 可保持在较高水平，但检测性能仍低于本方法；而当 δ 增大（如 $\delta = 1.7$ 或 2.0）时，虽然 AUROC 和 TPR 显著提升，但代码正确性出现明显下降，反映出强偏置对程序语义的破坏作用。

相比之下，本方法在两个数据集上均表现出更加均衡且稳定的性能。在 MBPP+ 数据集上，本方法在保持最高 pass@1（37.8%）的同时，实现了最优的检测性能（AUROC = 96.2，TPR@0.05 = 84.7%，TPR@0.01 = 67.7%）；在 HumanEval+ 数据集上，本方法同样在正确性与检测性两方面均优于基线方法，尤其是在低误报率（FPR = 0.01）条件下，TPR 提升更为显著。

这一结果表明，本方法能够有效缓解传统方法中存在的“强嵌入—低正确性”与“弱嵌入—低可检测性”之间的矛盾。其核心原因在于：通过基于熵的连续自适应调节机制，本方法能够在高不确定性区域增强水印嵌入强度，同时在低熵关键位置抑制偏置，从而实现对不同类型 token 的差异化处理。该机制不仅提高了水印信号的整体强度，还有效避免了对代码语法结构与语义功能的破坏，因此在多个评估指标上均取得了更优的综合表现。

3.5.4　跨语言泛化结果

为了验证本章方法在不同编程语言上的适用性和鲁棒性，我们进一步在 HumanEvalPack 数据集上进行了实验。该数据集将 HumanEval 扩展至多语言环境，包含 Python、C++ 和 Java 三个版本，每个版本均保持相同的编程问题语义，但语法和语言特性各不相同。这一设置能够有效评估水印方法在跨语言代码生成任务中的迁移能力，检验其是否过度依赖特定语言的语法结构。

表 3.3 展示了在 Java 和 C++ 子集上的实验结果，Python 结果已在前面两表中呈现。从表中可以清晰地看到，本章方法在所有语言上均保持了对基线方法的优势。在 Java 任务上，本章方法的 pass@1 达到 24.4%，比最优基线 SWEET 高出 3.1 个百分点，同时 AUROC 达到 88.9%，显著优于其他方法。在 C++ 任务上，本方法的 pass@1 为 39.0%，AUROC 为 85.0%，同样全面领先。值得注意的是，EWD 在 Java 上取得了 86.2% 的 AUROC，但其 pass@1 与 KGW 相同（18.9%），说明其检测性能的提升是以牺牲代码正确性为代价的。SWEET 虽然通过熵筛选减轻了对语法关键区域的干扰，但固定偏置的选择难以同时兼顾两种语言的不同特性，导致在 C++ 上的 AUROC 仅为 75.7%，甚至低于 EWD。

本方法之所以能够实现跨语言的稳定性能，根本原因在于其熵自适应机制不依赖于特定语言的语法先验，而是动态地根据每个 token 的预测不确定性调整水印强度。无论是 Java 的强类型语法还是 C++ 的复杂指针操作，模型在这些语言的关键字和控制流位置仍会表现出低熵，而在标识符、注释和字符串字面量等位置则呈现高熵。本方法通过统一的熵阈值和自适应偏置函数，能够在不同语言中自动识别出适合嵌入水印的高熵区域，从而在不破坏语法结构的前提下注入水印信号。这一特性使得本方法无需针对每种语言重新调参即可取得良好效果，体现了其良好的泛化能力和实用性。

表 3.3　HumanEvalPack 多语言实验结果

方法	Java		C++	
	pass@1	AUROC	pass@1	AUROC
KGW	18.9	76.0	36.5	74.1
EWD	18.9	86.2	36.5	78.1
SWEET	21.3	83.3	38.4	75.7
Ours	24.4	88.9	39.0	85.0

3.6　本章小结

本章围绕代码生成任务中的水印嵌入问题，提出了一种熵自适应水印框架。针对传统方法在固定熵阈值和恒定偏置条件下难以兼顾代码正确性与水印可检测性的局限，本章通过引入 token 级预测熵，对水印强度进行动态调节。在不修改底层语言

模型参数的前提下，框架通过实时计算生成过程中的熵值，并利用连续、有界的偏置函数，在高熵区域平滑注入水印信号，同时尽量避免对低熵语法关键位置产生干扰，从而降低对代码语法结构和功能正确性的影响。

在整体结构上，该方法由熵计算模块、自适应偏置生成模块以及水印嵌入与采样模块构成。三者协同工作形成完整的嵌入与检测流程：嵌入阶段通过自适应偏置实现差异化水印注入，检测阶段则结合高熵 token 筛选与统计检验实现水印识别，并在熵阈值、绿名单比例和哈希函数等关键参数上保持一致，以保证检测结果的稳定性和可复现性。本章提出的熵自适应机制通过将水印强度与 token 级不确定性动态耦合，使水印能够更合理地分布在适合嵌入的位置，从而在保持代码生成质量的同时提升水印可检测性。实验结果表明，该方法能够有效缓解代码水印中正确性与可检测性之间的矛盾，并在不同编程语言场景下表现出良好的稳定性与通用性，为生成代码的版权保护与可信溯源提供了一种可行的技术方案。

第四章　基于语法结构感知的词表调制水印方法

本章围绕生成式代码水印中语法结构约束与嵌入稳定性之间的协调问题展开研究，重点关注在代码生成过程中引入语法感知机制，对候选词表进行结构约束与裁剪，以降低水印调制对关键语法位置的干扰。不同于基于整体概率分布的统一调节策略，本文从代码语法结构出发，识别对语法正确性具有关键作用的位置，并据此约束候选词集合，避免低概率或不适配候选对程序结构造成破坏。同时，在语法约束较弱或语义表达空间较大的位置，适当保留调制空间，以兼顾生成灵活性与水印嵌入需求。因此，如何结合语法结构感知与词表裁剪机制，在保证代码语法正确性的前提下提升水印嵌入的稳定性与可控性，构成本章的核心研究问题。

4.1　引言

随着大规模语言模型在代码生成任务中的持续发展，模型在自动编程、代码补全及软件开发辅助等场景中展现出良好的应用潜力。与自然语言不同，代码具有严格的语法规则和层次化结构，其生成过程不仅需要满足语义合理性，还必须保证语法完整性与可执行性。在自回归生成框架下，每一步预测结果都会直接影响后续结构展开，使得局部生成决策对整体程序正确性具有放大效应，尤其是在关键语法位置，任何不符合语法约束的候选选择都可能导致结构破坏或执行失败。因此，在对生成过程进行干预时，有必要从代码的语法结构出发，对解码过程中的候选空间进行结构化控制，而非仅依赖概率层面的整体调节。

现有水印方法通常通过对候选词分布施加统一偏置实现信息嵌入，该类方法在自然语言生成中具有一定适用性，但在代码场景下存在明显局限。一方面，代码中不同位置在语法结构中的作用差异显著，关键字、运算符及结构符号等位置具有强约束性，其候选集合本身高度受限；另一方面，变量名或常量表达等位置则具有较强的表达冗余，对候选变化具有更高容忍度。统一的分布调制策略未对这种结构差异进行区分，容易在关键语法位置引入不合理候选，从而破坏程序结构，降低生成代码的可执行性与稳定性。此外，传统基于固定策略的解码方法难以根据当前位置的结构属性动态调整候选空间，限制了生成过程的精细化控制能力。

针对上述问题，本文提出一种基于语法结构感知的解码策略，从候选空间控制的角度对生成过程进行建模。该方法通过构建语法关键词集合，并结合预测概率分布对生成位置进行动态判定，在此基础上对候选词表实施分层裁剪。在语法关键位置施加强约束，剔除低概率且不符合语法预期的候选项；在语义位置则采用较为宽松的自适应裁剪策略，根据结构概率动态调整裁剪强度，以在保证语法约束的同时维持必要的表达灵活性。该策略可与基于红绿色表的水印机制协同使用，在不削弱水印可检测性的前提下，实现对解码过程的结构化约束与精细化调控。

4.2 总体框架

为在保证代码生成正确性的前提下实现稳定有效的水印嵌入，本文构建了一种基于语法结构感知的代码水印生成总体框架。该框架以生成模型的逐步预测过程为基础，在不改变模型结构的前提下，通过对每一步候选词分布进行动态调制，实现对水印信号的嵌入与控制。整体流程如图4.1所示可概括为：在生成过程中获取当前位置的预测分布，结合语法结构信息对候选空间进行约束处理，并在此基础上对特定词集合施加偏置，从而引导生成结果在保持语法正确性的同时体现水印特征。

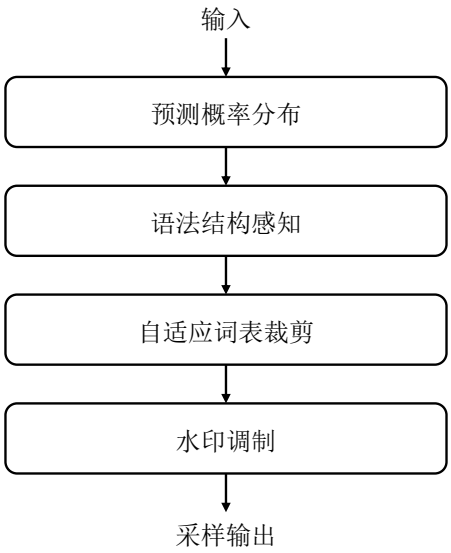


图 4.1　语法结构感知的词表调制水印总体框架图

从模块划分上看，所提出的框架主要由语法结构感知模块、位置自适应裁剪模块以及水印调制模块三部分组成，各模块之间协同作用，共同完成水印嵌入过程。首

先，语法结构感知模块通过对当前生成状态及预测概率分布进行分析，结合预先构建的语法关键词集合，对当前位置进行语法属性判定，将生成过程划分为语法位置与语义位置。该模块为后续差异化解码策略提供结构先验。

在获得位置类型信息后，位置自适应裁剪模块根据不同位置的结构特性，对候选词空间进行动态约束。在语法位置，由于候选空间本身受限且对生成结果影响显著，模块采用较强裁剪策略，对低概率候选词进行严格过滤，以避免其在后续调制过程中被放大，从而降低语法错误或结构破坏的风险；而在语义位置，考虑到表达具有一定冗余性，该模块采用较为宽松的自适应裁剪方式，根据当前位置的结构概率动态调整裁剪强度，在保证基本合理性的同时尽可能保留候选空间，为后续调制提供灵活性。通过这种基于位置类型的连续化控制方式，能够在生成稳定性与表达能力之间取得平衡。

在完成候选空间约束后，水印调制模块对特定词集合施加偏置以实现水印嵌入。具体而言，依据当前上下文构建对应的绿色词集合，并对该集合中的候选词施加加性调制，使其在后续采样过程中具有更高的被选中概率，从而在生成序列中形成稳定且可检测的统计特征。该调制过程与第三章所述方法保持一致，但其作用范围被限制在裁剪后的候选空间内，使水印嵌入过程能够更好地与结构约束相协调，从而有效减少对关键语法位置的干扰。

综合上述三个模块，本文方法在每一步生成过程中形成如下处理流程：首先根据当前预测分布估计下一位置的语法属性；随后依据位置类型对候选词分布进行分层裁剪与自适应调节；最后对绿色词集合施加偏置并完成采样生成。该过程无需引入额外模型或复杂结构，仅在原有解码流程中进行轻量级调整，具有良好的可扩展性与工程实现可行性。与传统统一调制方法相比，所提出的框架能够充分利用代码生成过程中不同位置的结构差异，在语法位置强化约束以保障生成稳定性，在语义位置保持适度灵活性以支持水印嵌入，从而在生成质量与水印可检测性之间实现更加合理的平衡。后续各小节将对语法结构感知机制、位置自适应裁剪策略以及水印调制与检测方法进行详细介绍。

4.3 基于语法结构感知的词表调制水印方法

在前述总体框架的基础上，本文进一步对水印嵌入过程中的关键机制进行具体设计。本节围绕语法结构感知与色表调制策略展开，从语法位置判定、候选词裁剪以及水印嵌入与检测三个方面，对所提出的方法进行系统阐述。通过引入位置感知信息，实现对不同生成位置的差异化调制，从而在保证代码正确性的前提下提升水印嵌入的稳定性与可检测性。

4.3.1 语法结构感知机制

为实现对代码生成过程中不同位置的精细化调制，首先需要对当前位置的语法属性进行有效判定。与自然语言相比，代码生成过程受到显式语法规则约束，不同类型 token 在结构中的作用差异显著。因此，仅依赖整体概率分布难以准确刻画位置敏感性，有必要引入语法层面的先验信息。

本文采用语法元素集合与概率分布相结合的方式构建语法结构感知机制。首先，针对 Python、C++ 和 Java 三类主流编程语言，预先构建语法元素集合 $\mathcal{K}$。如表 4.1 所示，该集合将语法元素划分为关键字、运算符、分隔符、空白符和类型五类。这些元素覆盖了代码结构中最核心的语法组成部分，能够反映程序控制结构、表达式构造以及类型约束等关键信息。

在生成第 t 个 token 时，模型输出条件概率分布 $P(v \mid x_{<t})$。鉴于完整词表规模较大，本文采用 top-k 截断对分布进行近似，得到高置信候选集合 $\mathcal{V}_t^{top}$。其中，k 用于限定参与语法强度估计的候选数量，从而聚焦于预测分布的主要概率质量，降低低概率噪声的干扰，其具体取值将在实验部分进行分析。在此基础上，区别于基于候选占比的粗粒度统计方式，本文引入概率加权机制，对候选中语法元素的概率质量进行建模，以实现对语法信息的更精细刻画。

具体而言，对 top-k 候选中属于语法集合 $\mathcal{K}$ 的 token，其概率进行累加，定义当前位置的语法强度为：

$$\alpha_t = \sum_{v \in \mathcal{V}t^{top} \cap \mathcal{K}} P(v \mid x< t) \tag{4.1}$$

该指标不仅考虑语法 token 是否出现，还进一步引入其预测概率大小，从而能够反映模型在当前位置对语法结构的“置信程度”。例如，当 top-3 候选中语法 token 占

表 4.1　Python、C++ 和 Java 的五类语法元素对比

类别	Python	C++	Java
关键字	True, False, None, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield	auto, break, case, catch, class, const, constexpr, continue, else, enum, for, if, namespace, new, nullptr, private, protected, public, return, static, struct, switch, template, this, throw, try, typedef, typename, using, virtual, while, override, …	abstract, assert, break, case, catch, class, const, continue, default, else, enum, extends, final, for, if, implements, import, new, null, package, private, protected, public, return, static, super, switch, this, throw, try, void, while, …
空白符	space, \n, \t	space, \n, \t	space, \n, \t
类型	int, float, complex, str, bytes, bool, list, tuple, set, dict, NoneType	int, float, double, bool, char, short, long, void, unsigned, size_t, ptrdiff_t, wchar_t, char8_t, …	byte, short, int, long, float, double, boolean, char, String, Object
分隔符	( ), [ ], { }, “: ; ”, @, ->, …	( ), [ ], { }, °, “: ; ”, ->, …	( ), [ ], { }, °, “: ; ”, ->; …
运算符	+, -, *, /, %, **, //, ==, !=, >, <, >=, <=, +=, -=, /=, %=, //=, **=, &, |	+, -, *, /, %, ++, –, ==, !=, >, <, >=, <=, &&, ||, !, &, ^, ~, <<, >>, >>>, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=	+, -, *, /, %, ++, –, ==, !=, >, <, >=, <=, &&, ||, !, &, ^, ~, <<, >>, >>>, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=

据较高概率质量时，说明当前位置更可能处于结构敏感区域，如关键字、括号或运算符位置；反之则更偏向语义表达区域。

基于语法强度 α_t，定义位置判定函数如下：

$$\text{Type}(t) = \begin{cases} \text{Syntax}, & \alpha_t > \eta \\ \text{Semantic}, & \text{otherwise} \end{cases} \tag{4.2}$$

其中 η 为语法强度阈值，用于区分语法敏感位置与语义表达位置，其取值影响最终判定结果，本章将在实验部分对其进行调优与分析。

该机制的核心思想在于：利用模型自身的预测分布作为结构信号来源，通过语法词表约束对 top-k 候选进行加权统计，从而实现对代码生成过程中语法关键位置的动态识别。与传统依赖显式语法解析的方法相比，该机制无需构建抽象语法树或引

入额外解析器，而是直接基于生成概率分布进行轻量级结构感知。因此，该方法在保持计算效率的同时，能够较好地捕捉代码生成过程中的结构敏感区域，并具备较强的跨语言适用性。

总体而言，该语法结构感知机制为后续水印调制提供了位置级别的结构先验，使生成过程能够在语法关键位置与语义表达位置之间进行差异化处理，从而为提升代码水印的稳定性与可用性奠定基础。

4.3.2 基于位置类型的词表调制策略

在前文中，本文已对代码生成过程中的语法位置进行了判定，并分析了不同位置在生成约束上的差异。在此基础上，本节进一步关注解码阶段的候选空间控制问题，提出一种基于位置类型的词表调制策略。该策略的核心思想是在每一步生成时，根据当前位置的语法属性，对模型输出的候选词表进行差异化裁剪，从而在保证语法正确性的同时抑制低概率噪声 token，为后续水印嵌入提供更加稳定的分布基础。

在自回归生成过程中，第 t 步模型输出词表 $\mathcal{V}$ 上的 logits 向量 z_t，对应的概率分布为：

$$p_t(v) = \frac{\exp(z_t(v))}{\sum_{v' \in \mathcal{V}} \exp(z_t(v'))}, \quad v \in \mathcal{V}. \tag{4.3}$$

该分布通常具有明显的长尾特性，即少数高概率 token 集中在分布头部，而大量低概率 token 分布在尾部区域。在代码生成任务中，这些尾部 token 往往对应语法不匹配或语义不相关的候选，若直接参与采样，容易引入噪声并降低生成稳定性。因此，有必要在采样前对候选词表进行有效裁剪。

与传统基于统一阈值或累计概率的全局裁剪方法不同，本文引入位置类型信息，对不同语法位置采用差异化的裁剪策略。具体而言，对于每一个时间步 t，根据前文的判定方法，将当前位置标记为语法位置或语义位置，即 $\text{type}_t \in \{\text{syntax}, \text{semantic}\}$。该划分反映了生成过程中不同位置在结构约束上的差异，其中语法位置对应关键语法单元，而语义位置则主要承担表达性内容生成。

为实现对不同位置的精细控制，本文在裁剪过程中引入分层阈值机制。设语法位置对应的裁剪阈值为 β_s，语义位置对应的基础裁剪阈值为 β_m，且满足 $\beta_s > \beta_m$。

据此，对候选词表进行如下筛选：

$$\mathcal{F}_t^{(\text{syntax})} = \{v \in \mathcal{V} \mid p_t(v) \geq \beta_s\}, \tag{4.4}$$

$$\mathcal{F}_t^{(\text{semantic})} = \{v \in \mathcal{V} \mid p_t(v) \geq \beta_m(t)\}. \tag{4.5}$$

在语法位置，由于生成结果需要严格满足程序语法结构约束，模型预测分布通常呈现出较强的集中性，因此采用较高的固定裁剪阈值 β_s，以更严格地过滤低概率 token，从而减少不符合语法模式的候选项干扰。该策略有助于提升关键语法结构生成的稳定性与确定性。

相比之下，在语义位置，生成过程更侧重于表达能力与多样性，若采用固定裁剪强度，难以同时兼顾不同生成状态下的分布差异。为此，本文在语义位置中进一步引入基于语法强度 α_t 的自适应裁剪机制。需要指出的是，α_t 在前文中已用于位置类型判定，在此基础上，本文将其进一步用于刻画语义位置内部的结构接近程度。当 α_t 接近判定阈值 η 时，说明当前位置虽被判定为语义位置，但仍具有一定结构约束，应适当提高裁剪强度；而当 α_t 较低时，说明当前位置主要承担语义表达功能，应降低裁剪强度以保留更多候选。

据此，将语义位置的裁剪阈值定义为：

$$\beta_m(t) = \beta_{\text{min}} + \frac{\alpha_t}{\eta}(\beta_{\text{max}} - \beta_{\text{min}}), \tag{4.6}$$

其中 β_{min} 与 β_{max} 分别表示裁剪阈值的下界与上界，且在语义位置中满足 $\alpha_t \leq \eta$。该设计使得语义位置的裁剪强度能够随语法强度连续变化，在接近语法边界时增强约束，在纯语义区域保持较高自由度，从而在生成稳定性与表达多样性之间取得更细粒度的平衡。

最终，在每一步生成过程中，根据当前位置类型选择对应的候选词表参与后续采样过程：

$$\tilde{\mathcal{V}}_t = \begin{cases} \mathcal{F}_t^{(\text{syntax})}, & \text{if type}_t = \text{syntax}, \\ \mathcal{F}_t^{(\text{semantic})}, & \text{if type}_t = \text{semantic}. \end{cases} \tag{4.7}$$

通过上述方式，词表裁剪策略在不同语法功能位置上实现了分层且连续的调制：在语法位置通过固定阈值强化结构约束，在语义位置通过自适应阈值维持表达灵活性。该机制使模型能够根据当前位置的结构属性与不确定性动态调整候选空间，从而在生成稳定性与表达多样性之间取得更优平衡。

总体而言，该策略在不改变模型参数的前提下，通过引入位置类型信息及语法强度信号对生成分布进行细粒度控制。一方面有效抑制低概率噪声 token 对关键语法结构的干扰，另一方面在非关键位置保留必要的表达自由度，并通过自适应机制进一步提升分布调制的精细程度，从而为后续水印嵌入提供更加稳定且可控的候选分布基础。

4.3.3 水印嵌入与检测

在完成基于位置类型的词表裁剪后，本节在裁剪后的候选空间 $\tilde{\mathcal{V}}_t$ 上引入红绿词表机制实现水印嵌入。具体而言，在每一步生成过程中，将候选词表划分为绿色词表 $\mathcal{V}^{(g)}$ 与红色词表 $\mathcal{V}^{(r)}$，满足：

$$\mathcal{V}^{(g)} \cup \mathcal{V}^{(r)} = \tilde{\mathcal{V}}_t, \quad \mathcal{V}^{(g)} \cap \mathcal{V}^{(r)} = \emptyset. \tag{4.8}$$

在解码阶段，对绿色词表中的 token 施加统一的 logits 偏置 δ。该过程与第三章所述红绿色表水印机制一致，相关概率建模与检测方法在第三章中已详细给出，因此本章不再赘述。不同之处在于，本章方法将水印嵌入限制在经过词表裁剪后的有效候选空间中，使水印调制过程与位置感知的生成约束相匹配。

在该设置下，水印信号仍表现为绿色 token 的统计性偏移，其检测方式与第三章保持一致，即基于计数统计量构造二项分布检验量进行判定。由于词表裁剪已提前过滤部分低概率 token，水印嵌入主要作用于高置信候选集合，从而在不改变检测框架的前提下，提高嵌入过程的稳定性与一致性。

4.4 实验结果与分析

4.4.1 参数选择

为确定语法位置判定阈值，本文基于 Qwen2.5-Coder-3B 模型在 MBPP+ 数据集上的生成过程，对所提出的语法概率（syntax probability）的经验分布进行了统计分

析。如图 4.2 所示，该分布呈现出明显的双峰特性：一类样本集中在接近 0 的区域，对应于非结构性 token（如变量名、常量及语义内容相关词）；另一类样本集中在接近 1 的区域，对应于结构性 token（如关键字、运算符及语法分隔符等）。两者之间在中间区间形成显著的低密度“谷底”，表明该指标在不同类型 token 之间具有较强的可分性。

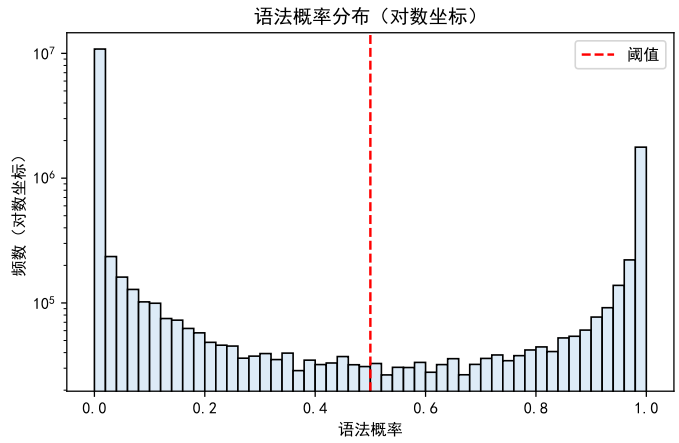


图 4.2 Qwen2.5-Coder-3B 在 MBPP+ 数据集上统计得到的语法概率分布

上述现象表明，语法概率在统计意义上可视为由结构 token 分布与非结构 token 分布构成的混合分布。在该模型下，判定阈值的选择可转化为在两类分布的交界区域寻找最优分割点。理论上，该分割点应位于概率密度函数的局部最小值处，从而最大化两类 token 的区分度并降低误分类风险。结合经验分布可以观察到，该低密度区域稳定出现在约 0.5 附近，因此本文初步选取 $\tau = 0.5$ 作为语法判定阈值。

在此基础上，本文进一步在 MBPP+ 数据集上基于 KGW 水印方法，对不同阈值设置进行了验证。需要说明的是，语法概率分布基于 MBPP+ 数据集统计得到，而参数敏感性实验在 MBPP 数据集上进行，两者在任务分布上具有一致性。具体而言，在 $\tau \in \{0.3, 0.4, 0.5, 0.6\}$ 的取值范围内，分别评估对应设置下的水印检测性能（AUROC）与代码生成质量（pass@1），实验结果如表 4.2 所示。

可以观察到，随着阈值 τ 的增大，模型对语法位置的判定逐渐趋于保守。在较小阈值（如 $\tau = 0.3$）下，部分语义位置被误判为语法位置，从而施加了过强的裁剪约束，限制了生成分布的表达能力，导致生成质量与检测性能均受到影响。随着 τ 增大至 0.5，该问题得到有效缓解，语法位置与语义位置的划分更加合理，从而在保证结构约束的同时保留必要的生成灵活性，使得 AUROC 与 pass@1 均达到最优。当阈

表 4.2 MBPP+ 数据集上不同语法判定阈值 τ 的性能对比（KGW 方法）

τ	AUROC (%)	pass@1 (%)
0.3	75.72	35.6
0.4	76.71	36.8
0.5	78.43	37.8
0.6	77.95	37.2

值进一步增大至 $\tau = 0.6$ 时，部分语法关键位置被划分为语义位置，从而削弱了应有的结构约束，使低概率候选更易参与生成过程，进而对整体性能产生一定影响。因此，过大的阈值同样会降低方法的有效性。

综合理论分析与实验结果，本文最终选取 $\tau = 0.5$ 作为语法位置判定阈值。该取值能够在语法约束与生成灵活性之间取得较优平衡，为后续词表裁剪与水印嵌入提供稳定可靠的结构依据。

在确定语法判定阈值后，本文进一步对词表裁剪阶段的概率截断阈值进行设计。该阈值用于控制低概率 token 的过滤强度，从而直接影响生成过程中对原始分布的干预程度。考虑到语法位置与语义位置在概率分布形态上的显著差异，本文采用分层策略分别进行设置。在语法位置处，模型输出分布通常呈现出较强的尖峰特性，即少数 token 占据主要概率质量，而其余 token 多为语法不兼容或低贡献噪声。因此，本文采用相对较高的裁剪阈值（如 0.1），以有效去除尾部低概率 token，同时增强候选空间的结构一致性，从而提高语法生成的稳定性。

相比之下，在语义位置中，概率分布通常较为平缓，多个 token 之间的概率差距较小，且这些候选往往承载不同语义表达方式。若采用过强的裁剪策略，将导致表达空间显著收缩，从而降低生成多样性甚至影响语义合理性。因此，在语义位置中不再采用固定阈值，而是设定一个较低的裁剪区间（如 $[10^{-5}, 10^{-3}]$），并在该区间内引入基于语法强度 α_t 的自适应调节机制。

具体而言，当 α_t 接近语法判定阈值时，说明当前位置虽属于语义位置，但仍具有一定结构约束，应适当提高裁剪阈值以增强稳定性；而当 α_t 较低时，说明当前位置主要承担语义表达功能，应降低裁剪阈值以保留更多候选。该机制使裁剪强度能够根据当前位置的不确定性连续变化，从而在不同语义场景下实现更为精细的控制。

通过上述分层与自适应相结合的裁剪策略，本文实现了对不同语法功能位置的差异化调制：在语法位置强化结构约束，在语义位置通过动态调整保留表达自由度。该设计不仅能够有效抑制低概率噪声 token 的干扰，同时避免过度裁剪对生成多样

性的影响，从而在水印嵌入强度与生成质量之间取得更优平衡。

4.4.2 实验结果与分析

为验证本文所提出位置感知词表裁剪策略的有效性，本文在 MBPP+ 与 HumanEval+ 数据集上，将该策略作为解码阶段的通用模块，引入至多种典型水印方法中进行对比实验。具体而言，实验分别在 KGW、EWD、SWEET 以及第三章提出的方法基础上，引入本章算法，并对生成代码的功能正确性（pass@1）与水印检测性能（AUROC）进行评估。

需要说明的是，表中“Ours”表示第三章提出的基于熵的水印方法，“+ 本章算法”表示在对应方法的解码阶段引入本章提出的位置感知词表裁剪策略。

表 4.3 MBPP+ 数据集上不同方法在引入本章算法前后的性能对比

Method	pass@1	AUROC
KGW	36.6	77.7
KGW + 本章算法	37.8	78.4
EWD	36.6	91.3
EWD + 本章算法	37.2	92.1
SWEET	37.1	92.5
SWEET + 本章算法	37.6	92.8
Ours	37.8	96.2
Ours + 本章算法	38.2	96.2

表 4.4 HumanEval+ 数据集上不同方法在引入本章算法前后的性能对比

Method	pass@1	AUROC
KGW	35.4	78.9
KGW + 本章算法	36.4	80.3
EWD	35.4	81.7
EWD + 本章算法	36.1	82.6
SWEET	36.2	81.4
SWEET + 本章算法	36.7	82.5
Ours	37.2	92.6
Ours + 本章算法	37.6	92.8

从实验结果可以观察到，引入本章算法后，各类水印方法在两个数据集上整体

呈现出较为稳定的性能提升趋势。在生成质量方面（pass@1），无论是 KGW、EWD 还是 SWEET 方法，在引入本章算法后均取得了不同程度提升。其中，在 MBPP+ 数据集上，KGW 的 pass@1 从 36.6 提升至 37.8，提升幅度达到 1.2；EWD 从 36.6 提升至 37.2；SWEET 从 37.1 提升至 37.6。在 HumanEval+ 数据集上，各方法同样表现出一致提升趋势。该结果表明，通过对生成过程中的候选空间进行结构约束，可以有效减少低概率异常 token 对生成过程的干扰，从而提升生成代码的稳定性与功能正确性。

在水印检测性能方面（AUROC），本章算法同样能够在多数情况下带来稳定增益。例如，在 MBPP+ 数据集上，KGW 的 AUROC 从 77.7 提升至 78.4，EWD 从 91.3 提升至 92.1，SWEET 从 92.5 提升至 92.8；在 HumanEval+ 数据集上，KGW 从 78.9 提升至 80.3，EWD 从 81.7 提升至 82.6，SWEET 从 81.4 提升至 82.5。说明该策略在不改变原有水印机制的前提下，能够通过优化有效候选空间，增强水印信号在生成过程中的稳定累积，从而提升检测性能。

对于第三章提出的方法（Ours），本章算法同样能够进一步提升生成质量。在 MBPP+ 数据集上，pass@1 从 37.8 提升至 38.2；在 HumanEval+ 数据集上，则从 37.2 提升至 37.6。同时，其 AUROC 基本保持稳定，分别维持在 96.2 与 92.8 附近。这说明本章算法在提升代码正确性的同时，并未破坏原有较强的水印检测能力，体现出较好的兼容性与稳定性。

从机制上看，这种性能提升主要来源于本章算法对生成过程的结构化约束。在语法关键位置，通过采用较强裁剪策略，能够有效过滤低概率且不符合语法模式的候选项，从而降低错误生成风险；而在语义位置，则采用相对宽松的裁剪方式，并结合结构概率动态调整裁剪强度，在保证表达能力的同时减少对原始分布的过度干扰。这种基于位置类型的差异化控制，使生成分布更加符合代码结构特性，从而在生成质量与水印稳定性之间取得更合理的平衡。

总体而言，实验结果表明，本文提出的位置感知词表裁剪策略具有良好的通用性与可插拔性，能够在不同水印方法中稳定提升代码生成质量，并在多数情况下进一步增强水印检测性能。该方法无需修改模型参数，仅通过对解码阶段进行轻量级调整，即可实现对生成分布的有效控制，从而为代码水印的稳定嵌入提供支持。

4.4.3 消融实验分析

为进一步验证本章算法中不同裁剪策略的有效性，本文在 KGW 方法基础上进行了消融实验。实验主要围绕语法位置与语义位置的差异化裁剪机制展开，对不同裁剪方式下的生成质量与水印检测性能进行分析。实验结果如表 4.5 所示。

表 4.5 KGW 方法在 MBPP+ 数据集上的消融实验结果

Method	pass@1	AUROC
KGW	36.6	77.7
KGW + 全局弱裁剪（自适应）	37.1	78.0
KGW + 语法强裁剪 + 语义弱裁剪 (10^{-5})	37.8	76.6
KGW + 语法强裁剪 + 语义弱裁剪 (10^{-3})	38.0	77.8
KGW + 本章算法	37.8	78.4

其中，“全局弱裁剪（自适应）”表示所有生成位置均采用基于结构概率动态调整的弱裁剪策略；“语法强裁剪 + 语义弱裁剪”表示区分语法位置与语义位置，但语义位置采用固定阈值而非动态调整；“本章算法”则表示在语义位置进一步结合结构概率进行连续化自适应裁剪。

从实验结果可以看出，仅采用全局弱裁剪虽然能够在一定程度上提升 AUROC，但由于缺乏对语法关键位置的强约束，其 pass@1 提升较为有限。这说明仅依赖宽松裁剪难以有效减少关键结构位置中的错误生成风险。

进一步地，在区分语法位置与语义位置后，模型整体性能得到明显改善。当语义位置采用固定弱裁剪阈值 10^{-5} 时，pass@1 提升至 37.8，但 AUROC 下降至 76.6。这表明过小的裁剪阈值会保留大量低概率尾部 token，使得生成分布中噪声候选增多，从而削弱水印信号在有效生成路径中的稳定累积，最终导致检测性能下降。

相比之下，当语义位置采用固定阈值 10^{-3} 时，pass@1 提升至 38.0，同时 AUROC 回升至 77.8。说明适度增强语义位置的裁剪强度能够有效减少低概率异常 token 对生成过程的干扰，从而提升代码生成稳定性与水印信号一致性。但由于固定阈值策略无法适应不同生成状态下的结构差异，其整体检测性能仍略低于本章算法。

本文提出的方法则在语义位置进一步结合结构概率动态调整裁剪强度，使裁剪策略能够根据当前位置的结构属性自适应变化。在结构概率较低时，采用较弱裁剪以保留表达空间；而在结构概率接近语法边界时，则适当增强约束以减少异常候选

干扰。因此，该方法在保持较高生成质量的同时，取得了最优 AUROC（78.4），说明其能够在生成稳定性与水印检测性能之间实现更加合理的平衡。

总体而言，消融实验结果表明，本章算法中的“语法位置强裁剪 + 语义位置自适应弱裁剪”设计是提升整体性能的重要因素。通过结合位置类型与结构概率对候选空间进行动态控制，能够有效协调代码生成质量与水印信号稳定性，从而实现更优的综合性能。

4.4.4 跨语言泛化分析

为进一步验证本文所提出解码策略在不同编程语言中的适用性，本文在 HumanEvalPack 多语言数据集上进行了扩展实验。该数据集在 HumanEval 基础上构建，包含 Python、Java 与 C++ 三种语言版本，各语言在问题语义上保持一致，但在语法结构与表达方式上存在显著差异。因此，该数据集能够有效评估方法在跨语言代码生成场景中的鲁棒性与泛化能力。

在本节实验中，本文以经典 KGW 方法为基础，并在其解码阶段引入本章算法，对比分析不同语言下的性能变化。实验结果如表 4.6 所示。

表 4.6　HumanEvalPack 多语言实验结果（KGW 引入本章算法前后对比）

方法	Java		C++	
	pass@1	AUROC	pass@1	AUROC
KGW	18.9	76.0	36.5	74.1
KGW + 本章算法	20.3	77.8	37.8	76.0

从实验结果可以看出，在引入本章算法后，KGW 方法在不同语言上的性能均取得稳定提升。在 Java 任务上，pass@1 从 18.9% 提升至 20.3%，AUROC 从 76.0% 提升至 77.8%；在 C++ 任务上，pass@1 从 36.5% 提升至 37.8%，AUROC 从 74.1% 提升至 76.0%。该结果表明，本文提出的位置感知词表裁剪策略不仅适用于单一语言环境，在不同编程语言中同样能够有效改善生成质量并增强水印检测性能。

该结果表明，本章算法在不同编程语言中均能够保持稳定效果，其原因在于该方法并非直接依赖固定语法模板进行约束，而是结合不同语言预先构建的语法关键词集合，并利用模型预测分布对当前位置的结构属性进行动态判定。虽然 Java 与 C++ 在关键字体系、语法形式及表达习惯上存在明显差异，但在代码生成过程中，关键

语法位置通常仍会表现出较高的结构概率，而变量名、函数名及常量等语义位置则具有更强的生成自由度。

基于这一特性，本章方法能够在不同语言中对语法位置与语义位置实施差异化裁剪：在结构敏感位置采用更强的约束，以减少低概率异常 token 对程序结构的干扰；而在语义相关位置保留相对宽松的候选空间，从而维持生成过程的表达能力与水印嵌入空间。由于该策略主要基于生成分布的结构特征进行调制，因此即使面对不同语言，也能够保持较好的稳定性与一致性。

总体而言，实验结果表明，本文提出的位置感知词表裁剪策略具有良好的跨语言适用能力，能够作为一种通用解码模块应用于不同代码水印方法之中，从而提升生成过程的稳定性与可靠性。

4.5 本章小结

本章针对传统代码水印方法在统一调制过程中容易破坏代码结构稳定性的问题，提出了一种基于语法结构感知的词表调制水印方法。该方法结合语法关键词集合与模型预测分布，对生成位置进行动态语法属性判定，并基于位置类型对候选词空间实施差异化裁剪：在语法关键位置采用较强约束以减少低概率异常 token 对程序结构的干扰，在语义位置则通过自适应弱裁剪保留必要的表达空间，从而在保证代码正确性的同时提升水印嵌入稳定性。实验结果表明，本文方法能够在多个代码生成数据集及不同编程语言场景下稳定提升生成质量与水印检测性能，并对多种典型水印方法具有良好的兼容性与通用性。

第五章　总结与展望

5.1　总结

随着大语言模型在软件工程领域的广泛应用，生成代码的来源鉴别与版权保护已成为人工智能安全治理中亟待解决的关键问题。针对程序代码语法结构严苛、对扰动敏感以及现有文本水印方法易破坏代码正确性等挑战，本文深入研究了面向生成式代码的数字水印技术，旨在实现水印嵌入强度与代码质量之间的有效平衡。

首先，本文提出了一种基于熵自适应的水印生成策略，用于降低固定嵌入方式对代码关键位置生成质量的影响。该方法利用生成过程中候选词分布的熵值刻画当前预测的不确定性，并据此动态调整水印嵌入强度。在高熵位置，候选词分布较为分散，适度施加偏置对代码语义与结构的影响较小，因此更适合进行水印嵌入；而在低熵、语法结构较确定的位置，则减少水印干预以避免破坏代码正确性。通过这种基于生成状态的自适应分配机制，水印信息能够更加合理地分布于生成序列中，从而提升整体嵌入稳定性。实验结果表明，该方法在保证代码正确性与生成质量的同时，提高了水印检测性能，并具有较好的鲁棒性。

其次，针对语法关键位置对扰动更为敏感的问题，本文进一步提出一种语法结构感知的词表调制方法。该方法引入语法信息对生成位置进行动态判定，从而区分不同语法约束强度下的代码区域。在语法关键位置采用更严格的候选约束，以降低错误生成风险；在语义相对自由的位置则采用更宽松的裁剪策略，以保留水印嵌入空间。该分层调制机制使解码过程更具结构感知能力，从而减少水印偏置对语法正确性的影响，同时维持水印信号的有效性。实验结果表明，该方法在提升代码正确性的同时未降低检测性能，整体优于未引入结构感知约束的基线方法。

总体而言，本文围绕代码生成场景中水印嵌入强度与代码正确性之间的核心矛盾，分别从生成不确定性建模与语法结构约束两个角度提出了改进方案。所提出的方法在不显著影响代码语义一致性与可执行性的前提下，实现了更稳定的水印嵌入与更可靠的检测性能。实验结果表明，两种方法在不同数据集与任务场景下均表现出良好的有效性与鲁棒性，为生成代码的可信标识与来源溯源提供了一种兼顾正确性与可检测性的技术路径。

5.2 展望

本文围绕代码生成过程中的水印嵌入问题，重点研究了在保证代码正确性的前提下提升水印可检测性与稳定性的方法。尽管当前方法在实验中取得了一定效果，但在更复杂的真实应用场景中，仍存在一些值得进一步探索的方向。

首先，从方法建模角度来看，当前工作主要基于生成过程中的概率分布特征以及局部语法约束来指导水印嵌入，但对更高层次的程序语义结构利用仍然相对有限。例如，目前方法更多聚焦于 token 级别的决策，而对函数级或模块级的整体语义一致性考虑不足。未来可以进一步结合抽象语法树、控制流结构等程序分析信息，从更结构化的角度对生成过程进行约束，使水印嵌入不仅在局部合理，也在全局语义层面更加稳定。

其次，从生成过程的稳定性来看，本文方法主要针对单步解码阶段进行优化，而在长代码生成任务中，早期生成决策可能对后续输出产生累积影响，从而导致水印分布不稳定或误差传播问题。特别是在多函数或复杂逻辑生成场景下，这种局部策略的局限性会更加明显。因此，未来可以探索跨步骤的水印一致性约束机制，例如引入序列级优化目标或全局水印保持策略，以增强长序列生成过程中的鲁棒性。

最后，当前方法的评估主要集中在标准代码生成任务与静态指标上，而在真实软件工程场景中的验证仍相对不足。未来可以进一步结合实际开发环境或复杂工程代码库，评估方法在真实使用条件下的适用性与可靠性。

总体而言，代码水印仍然是一个需要在生成质量、结构约束与可追溯性之间进行多目标权衡的问题。本文工作在这一方向上进行了初步探索，但在语义建模深度、长序列稳定性以及真实场景适配性方面仍有进一步提升空间。

参考文献

[1] LECUN Y, BENGIO Y, HINTON G. Deep learning[J]. nature, 2015, 521(7553): 436-444.

[2] CHEN L, GUO Q, JIA H, et al. A survey on evaluating large language models in code generation tasks[J]. arXiv preprint arXiv:2408.16498, 2024.

[3] 赵葛剑, 张新鹏. DeepSeek: 从“概率生成”到“因果推理”[J]. 自然杂志, 2025, 47(2): 79-84.

[4] 梁忻健, 姚迪, 文永明, 等. 基于大模型的时态数据分析算法综述[J]. 集成技术, 2025, 14(5): 1-19.

[5] 陈露, 张思拓, 俞凯. 跨模态语言大模型: 进展及展望[J]. 中国科学基金, 2023, 37(5): 776-785.

[6] ZHOU K, HASSAN F H, HOON G K. The state of the art for cross-modal retrieval: A survey[J]. IEEE Access, 2023, 11: 138568-138589.

[7] ZHANG X, LI S, SHI N, et al. Cross-modal consistency in multimodal large language models[J]. arXiv preprint arXiv:2411.09273, 2024.

[8] XIONG Z, WANG D, LI Y, et al. CrossPL: Evaluating Large Language Models on Cross Programming Language Code Generation[J]. arXiv preprint arXiv:2507.19904, 2025.

[9] DONG Y, JIANG X, LIU H, et al. Generalization or memorization: Data contamination and trustworthy evaluation for large language models[C]//Findings of the Association for Computational Linguistics: ACL 2024. 2024: 12039-12050.

[10] FRIED D, AGHAJANYAN A, LIN J, et al. Incoder: A generative model for code infilling and synthesis[J]. arXiv preprint arXiv:2204.05999, 2022.

[11] JIANG J, WANG F, SHEN J, et al. A survey on large language models for code generation[J]. arXiv preprint arXiv:2406.00515, 2024.

[12] DHRUV A, DUBEY A. Leveraging large language models for code translation and software development in scientific computing[C]//Proceedings of the Platform for Advanced Scientific Computing Conference. 2025: 1-9.

[13] HUSEIN R A, ABURAJOUH H, CATAL C. Large language models for code completion: A systematic literature review[J]. Computer Standards & Interfaces, 2025, 92: 103917.

[14] HOU W, JI Z. Comparing large language models and human programmers for generating programming code[J]. Advanced Science, 2025, 12(8): 2412279.

[15] 陈铭松, 焦跃军. 嵌入式软件语言及开发新范式[J]. Embedded Technology and Intelligent Systems, 2025, 2: 223.

[16] MIRSKY Y, DEMONTIS A, KOTAK J, et al. The threat of offensive ai to organizations[J]. Computers & Security, 2023, 124: 103006.

[17] LIN Z, CUI J, LIAO X, et al. Malla: Demystifying real-world large language model integrated malicious services[C]//33rd USENIX Security Symposium (USENIX Security 24). 2024: 4693-4710.

[18] PUDASAINI S, MIRALLES-PECHUÁN L, LILLIS D, et al. Survey on AI-generated plagiarism detection: The impact of large language models on academic integrity[J]. Journal of Academic Ethics, 2024: 1-34.

[19] SIMMONS A, HOLANDA M, CHAMON C, et al. Ai generated code plagiarism detection in computer science courses: A literature mapping[C]//2024 IEEE Frontiers in Education Conference (FIE). 2024: 1-7.

[20] 杜静, 冯凯瑞, 王东. 生成式人工智能背景下的高校学术道德失范及其治理策略.[J]. Journal of Soochow University Educational Science Edition, 2025, 13(3).

[21] 吴兰岸, 闫寒冰, 黄发良, 等. 大型语言模型在高等教育中的应用分析与现实挑战.[J]. Modern Educational Technology, 2023, 33(8).

[22] CHEN M, TWOREK J, JUN H, et al. Evaluating large language models trained on code[J]. arXiv preprint arXiv:2107.03374, 2021.

[23] DONG Y, LI G, JIN Z. CODEP: grammatical seq2seq model for general-purpose code generation[C]//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. 2023: 188-198.

[24] LI Y, CHOI D, CHUNG J, et al. Competition-level code generation with alphacode [J]. Science, 2022, 378(6624): 1092-1097.

[25] WU H, LIU G, YAO Y, et al. Watermarking neural networks with watermarked images[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2021, 31(7): 2591-2601.

[26] YANG Z, ZHAO G, WU H. Watermarking for large language models: A survey[J]. Mathematics, 2025, 13(9): 1420.

[27] BRASSIL J T, LOW S, MAXEMCHUK N F, et al. Electronic marking and identification techniques to discourage document copying[J]. IEEE Journal on Selected Areas in Communications, 1995, 13(8): 1495-1504.

[28] LOW S H, MAXEMCHUK N F, BRASSIL J T, et al. Document marking and identification using both line and word shifting[C]//Proceedings of INFOCOM'95: vol. 2. 1995: 853-860.

[29] KIM Y W, MOON K A, OH I S. A text watermarking algorithm based on word classification and inter-word space statistics.[C]//ICDAR. 2003: 775-779.

[30] RIZZO S G, BERTINI F, MONTESI D. Content-preserving text watermarking through unicode homoglyph substitution[C]//Proceedings of the 20th International Database Engineering & Applications Symposium. 2016: 97-104.

[31] JOHNSON N F, DURIC Z, JAJODIA S. Information hiding: steganography and watermarking-attacks and countermeasures: steganography and watermarking: attacks and countermeasures: vol. 1[M]. Springer Science & Business Media, 2001.

[32] TOPKARA U, TOPKARA M, ATALLAH M J. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions[C]//Proceedings of the 8th workshop on Multimedia and security. 2006: 164-174.

[33] HAO W, XIANG L, LI Y, et al. Reversible natural language watermarking using synonym substitution and arithmetic coding[J]. 2018.

[34] BARMAWI A. Linguistic based steganography using lexical substitution and syntactical transformation[C]//2016 6th International Conference on IT Convergence and Security (ICITCS). 2016: 1-6.

[35] CHANG C Y, CLARK S. The secret’s in the word order: Text-to-text generation for linguistic steganography[C]//Proceedings of COLING 2012. 2012: 511-528.

[36] ABDELNABI S, FRITZ M. Adversarial watermarking transformer: Towards tracing text provenance with data hiding[C]//2021 IEEE symposium on security and privacy (SP). 2021: 121-140.

[37] LAI Z, ZHANG X, CHEN S. Adaptive ensembles of fine-tuned transformers for llm-generated text detection[C]//2024 International Joint Conference on Neural Networks (IJCNN). 2024: 1-7.

[38] KIRCHENBAUER J, GEIPING J, WEN Y, et al. A watermark for large language models[C]//International Conference on Machine Learning. 2023: 17061-17084.

[39] LEE T, HONG S, AHN J, et al. Who wrote this code? watermarking for code generation[J]. arXiv preprint arXiv:2305.15060, 2023.

[40] LI Z, WANG C, WANG S, et al. Protecting intellectual property of large language model-based code generation apis via watermarks[C]//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023: 2336-2350.

[41] LI B, FU Z, ZHANG M, et al. Efficient and Universal Watermarking for LLM-Generated Code Detection[J]. arXiv preprint arXiv:2402.07518, 2024.

[42] NING K, CHEN J, ZHONG Q, et al. MCGMark: An Encodable and Robust Online Watermark for Tracing LLM-Generated Malicious Code[J]. arXiv preprint arXiv:2408.01354, 2024.

[43] YANG B, LI W, XIANG L, et al. Towards code watermarking with dual-channel transformations[J]. arXiv preprint arXiv:2309.00860, 2023.

[44] LIU A, PAN L, LU Y, et al. A survey of text watermarking in the era of large language models[J]. ACM Computing Surveys, 2024, 57(2): 1-36.

[45] FAHAD I A, FUAD M M N. Can We Trust the Source? A Systematic Review of Watermarking and Attribution for AI Generated Code[J]. A Systematic Review of Watermarking and Attribution for AI Generated Code, 2026.

[46] BERNARDO J, BAYARRI M, BERGER J, et al. Generative or discriminative? getting the best of both worlds[J]. Bayesian statistics, 2007, 8(3): 3-24.

[47] KUMAR S, MUSHARAF D, MUSHARAF S, et al. A comprehensive review of the latest advancements in large generative AI models[C]//International conference on advanced communication and intelligent systems. 2023: 90-103.

[48] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[49] SUTSKEVER I, VINYALS O, LE Q V. Sequence to sequence learning with neural networks[J]. Advances in neural information processing systems, 2014, 27.

[50] HAN Z, GAO C, LIU J, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey[J]. arXiv preprint arXiv:2403.14608, 2024.

[51] DING N, QIN Y, YANG G, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models[J]. Nature machine intelligence, 2023, 5(3): 220-235.

[52] WANG L, CHEN S, JIANG L, et al. Parameter-efficient fine-tuning in large models: A survey of methodologies[J]. arXiv preprint arXiv:2410.19878, 2024.

[53] GROSSE R, BAE J, ANIL C, et al. Studying large language model generalization with influence functions[J]. arXiv preprint arXiv:2308.03296, 2023.

[54] BUDNIKOV M, BYKOVA A, YAMSHCHIKOV I P. Generalization potential of large language models[J]. Neural Computing and Applications, 2025, 37(4): 1973-1997.

[55] VIJAYAKUMAR A, COGSWELL M, SELVARAJU R, et al. Diverse beam search for improved description of complex scenes[C]//Proceedings of the AAAI Conference on Artificial Intelligence: vol. 32: 1. 2018.

[56] HOLTZMAN A, BUYS J, DU L, et al. The curious case of neural text degeneration [J]. arXiv preprint arXiv:1904.09751, 2019.

[57] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.

[58] ACHIAM J, ADLER S, AGARWAL S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[59] SCARSELLI F, GORI M, TSOI A C, et al. The graph neural network model[J]. IEEE transactions on neural networks, 2008, 20(1): 61-80.

[60] KUANG L, ZHOU C, YANG X. Code comment generation based on graph neural network enhanced transformer model for code understanding in open-source software ecosystems[J]. Automated Software Engineering, 2022, 29(2): 43.

[61] CHENG J, FOSTIROPOULOS I, BOEHM B. Gn-transformer: Fusing sequence and graph representation for improved code summarization[J]. arXiv preprint arXiv:2111.08874, 2021.

[62] LE H, WANG Y, GOTMARE A D, et al. Coderl: Mastering code generation through pretrained models and deep reinforcement learning[J]. Advances in Neural Information Processing Systems, 2022, 35: 21314-21328.

[63] POESIA G, POLOZOV O, LE V, et al. Synchromesh: Reliable code generation from pre-trained language models[J]. arXiv preprint arXiv:2201.11227, 2022.

[64] WANG Y, LE H, GOTMARE A, et al. Codet5+: Open code large language models for code understanding and generation[C]//Proceedings of the 2023 conference on empirical methods in natural language processing. 2023: 1069-1088.

[65] LI R, ALLAL L B, ZI Y, et al. Starcoder: may the source be with you![J]. arXiv preprint arXiv:2305.06161, 2023.

[66] ZHU Q, GUO D, SHAO Z, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence[J]. arXiv preprint arXiv:2406.11931, 2024.

[67] ROZIERE B, GEHRING J, GLOECKLE F, et al. Code llama: Open foundation models for code[J]. arXiv preprint arXiv:2308.12950, 2023.

[68] HUI B, YANG J, CUI Z, et al. Qwen2. 5-coder technical report[J]. arXiv preprint arXiv:2409.12186, 2024.

[69] LIU J, XIA C S, WANG Y, et al. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation[J]. Advances in neural information processing systems, 2023, 36: 21558-21572.

[70] AUSTIN J, ODENA A, NYE M, et al. Program synthesis with large language models [J]. arXiv preprint arXiv:2108.07732, 2021.

[71] MUENNIGHOFF N, LIU Q, ZEBAZE A, et al. Octopack: Instruction tuning code large language models[C]//NeurIPS 2023 workshop on instruction tuning and instruction following. 2023.

[72] LU Y, LIU A, YU D, et al. An entropy-based text watermarking detection method [C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024: 11724-11735.

[73] LEE T, HONG S, AHN J, et al. Who wrote this code? watermarking for code generation[C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024: 4890-4911.

[74] FAWCETT T. An introduction to ROC analysis[J]. Pattern recognition letters, 2006, 27(8): 861-874.

攻读专业硕士学位期间取得的研究成果

[1] SHI Ying, BAO Siyuan, WU Hanzhou, et al. Entropy-Aware Watermarking for Code Generation Models[C]//Machine Learning for Cyber Security. Singapore: Springer Nature Singapore, 2026: 193-206.

[2] 张新鹏, 施颖, 吴汉舟. 一种基于语法结构感知的代码水印生成方法、设备及介质: 202610446667.7 [P]. 2026-04-07.

致　谢

论文至此收笔，致谢落下最后一段文字。像是一段被时间缓慢折叠过的旅程，终于在某个安静的节点展开全貌。回望时才发现，许多看似寻常的日子，其实都在不动声色地参与塑形，塑造思维的轮廓，也塑造对“成长”这个词更复杂的理解。

这一段时光并不总是线性前进的，它更像是一条时而清晰、时而隐入雾中的路径。推进与停顿交替出现，思考与空白彼此穿插，而正是在这种交错之中，时间被一点点填满。那些不被刻意强调的瞬间，恰恰构成了最真实的底色：停顿、犹豫、重新开始，以及在这些往复之间逐渐形成的某种持续的回声。

首先感谢我的导师张新鹏教授，工科的严谨与文人的风骨在您身上相融，诗词引路，思想照人，学生铭记于心。感谢我的指导老师吴汉舟教授。从选题到方法，从一字一句到谋篇布局，您始终以无比的耐心给予细致指导。

感谢课题组的同学们。一起改代码、讨论问题、互相打气的日子，我会一直记得。谢谢你们让这段枯燥的科研时光变得温暖又热闹。所谓“同行”，或许正是在这样的过程中被具体化的概念。

感谢家人始终如一的支持与理解。不以结果衡量，不给过度期待，只是安静地托举。这种信任的重量，我记在心里。

也感谢朋友与生活中那些看似轻描淡写的时刻。它们不断提醒我，人生并不只由单一轨道构成，“自我”也不由单一身份覆盖。那些对话、笑意与思考，使时间在学业之外仍然保持流动。

学生时代的夏天我们总在告别，可夏天永远炙热明媚。这是最后一个被称作“学生时代”的夏天，却不是关于青春的终章。从此山长水远，我仍会以少年般滚烫的心跳，去爱每一个六月的光斑与蝉鸣。青春未竟，热烈如初。

施颖

上海大学

2026 年 5 月 7 日