Computer Science and Application 计算机科学与应用, 2026, 16(1), 268-280
Published Online January 2026 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2026.161022

Hans汉斯

一种针对Python代码的提取与分析方法研究

——理论与实证研究

黄佳璐*，邓海生*

西京学院计算机学院，陕西 西安

收稿日期：2025年12月16日；录用日期：2026年1月15日；发布日期：2026年1月22日

摘　要

随着人工智能与大数据技术的快速发展，代码知识结构的自动化提取与可视化分析逐渐成为程序理解、教学辅助及智能开发的重要方向。在当前软件工程与教育实践中，Python作为应用最广泛的编程语言之一，其代码所涵盖的知识点类型多样、依赖关系复杂，传统人工解析方式效率低且缺乏系统性。为解决知识点识别不精准、代码结构难以直观呈现、知识复用程度不足等问题，本文提出一种针对Python代码的提取与分析方法。该方法通过构建专家知识库，提取Python编程中的关键知识点及其对应指令；基于抽象语法树(AST)对代码进行解析，将语法结构与知识点进行逐级匹配；通过出现频次、调用深度与依赖关系的综合权重模型计算知识点的重要程度；并基于三元组结构构建“知识点－关键指令－关系类型”的知识关联体系。随后利用NetworkX与Pyecharts等工具实现知识网络拓扑、交互式图谱与词云图的可视化呈现，使代码知识结构能够以直观、系统化的方式展示。实验结果表明，该方法能够有效揭示代码中知识点分布规律与逻辑关联，为编程教学、教材分析、智能代码分析系统等应用提供可靠支撑。本研究不仅验证了Python代码知识点可视化体系构建的可行性，还为教学管理、学习路径规划及智能编程辅助工具的发展提供了新思路。

关键词

Python代码，知识点提取，抽象语法树，三元组，知识图谱，可视化分析

A Python Code Extraction and Analysis Method

—Theoretical and Empirical Research

Jialu Huang*, Haisheng Deng*

College of Computer Science, Xijing University, Xi’an Shaanxi

Received: December 16, 2025; accepted: January 15, 2026; published: January 22, 2026

*通讯作者。

文章引用: 黄佳璐, 邓海生. 一种针对 Python 代码的提取与分析方法研究[J]. 计算机科学与应用, 2026, 16(1): 268-280.
DOI: 10.12677/csa.2026.161022

Abstract

With the rapid development of artificial intelligence and big data technologies, the automated extraction and visualization of code knowledge structures have become essential in program comprehension, teaching assistance, and intelligent software development. As one of the most widely-used programming languages, Python contains diverse types of knowledge points and complex dependency relationships, making traditional manual analysis inefficient and unsystematic. To address the problems of inaccurate knowledge point identification, limited code structure interpretability, and low knowledge reusability, this paper proposes a Python-oriented method for knowledge extraction and analysis. The method constructs an expert knowledge base that maps Python knowledge points to critical instructions. Using the Abstract Syntax Tree (AST), Python code is decomposed into structural units and matched with expert knowledge entries. A comprehensive weighting model—integrating instruction frequency, call depth, and dependency relations—is designed to evaluate the importance of each knowledge point. Furthermore, a structured "Knowledge Point-Instruction-Relation Type" triple representation is established. Based on this representation, NetworkX and Pyecharts are employed to generate network topology diagrams, interactive knowledge graphs, and word clouds for multi-dimensional visualization. Experimental results demonstrate that the proposed method effectively reveals the distribution patterns and logical associations of code knowledge, providing support for programming education, textbook analysis, and intelligent code analysis systems. This study verifies the feasibility of constructing a visualization system for Python code knowledge and offers new insights for teaching management, learning path planning, and intelligent programming tools.

Keywords

Python Code, Knowledge Extraction, Abstract Syntax Tree, Triples, Knowledge Graph, Visualization

1. 引言

随着软件规模的不断扩大与编程教育的普及，代码知识结构的自动化解析与可视化呈现逐渐成为教育领域与工程实践中的重要研究方向。在实际教学中，学生难以从源代码中快速抓取核心知识点；教师也难以直观判断教材或代码片段中各知识点的权重与关联结构；而在工程应用中，开发者常常需要理解他人代码的逻辑、调用关系与依赖结构，因此迫切需要一种能够将代码知识结构系统化、可视化的技术手段。Python 作为应用广泛的语言，其生态丰富、语法灵活，既具有教学价值，又具备工业级应用场景。然而，Python 代码中知识点种类繁多、依赖深度不一，人工分析成本高，传统静态或动态分析工具往往只关注语法错误、运行表现或安全漏洞，对“知识点层级”的语义结构识别能力有限，缺乏面向教学与知识管理的可视化表达机制。

为解决上述问题，本文提出一种面向 Python 代码的知识点提取与分析方法。通过构建专家知识库、利用 AST 解析语法结构、采用权重评分模型衡量知识点重要性，并基于三元组结构构建知识语义网络，最终实现 Python 代码知识点的多维度可视化呈现。该方法不仅能够揭示代码的语义层次结构，还能为教师、学生与开发人员提供清晰、系统、可交互的知识框架，从而提升理解效率与教学质量。

2. 国内外研究现状

2.1. 国外研究现状

在国际研究领域，程序结构分析技术发展较早，其中抽象语法树(Abstract Syntax Tree, AST)作为核心的代码静态分析工具，被广泛应用于程序理解、依赖分析与自动化特征提取。Baxter 等研究者指出，AST 能够以层级结构完整表示程序语句之间的语法关系，是实现语义级代码分析的重要基础[1]。基于此，大量研究围绕控制流分析、数据流分析和程序切片等技术展开，为后续的代码知识点识别提供了坚实支撑。随着知识图谱(Knowledge Graph)技术的高速发展，国外学者开始探索将程序实体及其关系以图结构表示。Miltos 等人提出了将函数、类、变量等视为图谱节点，将调用、依赖、继承等语义关系映射为边，从而构建程序语义知识网络，用于辅助开发者理解大型软件系统结构[2]。与此同时，深度学习特别是图神经网络(Graph Neural Network, GNN)也被引入代码语义建模领域。Allamanis 等人利用 GNN 学习代码图结构的节点与关系特征，实现代码补全、缺陷定位及函数摘要生成，显著提升了代码分析效果[3]。并为代码智能分析任务提供了系统性综述与技术框架。

在代码可视化研究方面，国外研究强调通过图结构展示提升代码可理解性。Battista 指出，力导向布局(force-directed layout)、层级布局(hierarchical layout)等算法能有效呈现大规模软件的整体结构和局部逻辑关系[4]。Saraswat 等人进一步提出交互式知识图谱，通过节点缩放、拖拽、路径高亮、语义过滤等方式，帮助用户快速定位代码中的关键部分，提高程序理解效率[5]。

此外，在软件工程方向，欧美研究机构高度重视代码语义分析在实际工程中的应用价值。例如 Microsoft Research 提出基于语义嵌入的代码搜索模型，通过将代码结构映射为向量空间，使大型代码库中的函数、类或片段之间的相似性能够被自动捕获，从而加强代码复用和自动定位能力[6]。这一类研究表明，将语义建模与静态分析结合能够显著提升软件工程效率，也为代码知识点的自动提取与分类提供了新的视角。近年来，多模态知识图谱研究进一步指出，通过融合文本、图结构、序列数据等不同模态信息，可提升语义理解深度，为代码知识结构构建提供了更强的语义支撑。

国外研究还关注代码知识结构的可解释性与教育辅助应用。美国多所高校的教学研究团队提出利用可视化图谱辅助初学者理解程序流程的模型，将代码执行路径、变量变化和函数调用关系以图形方式呈现，以降低编程学习的认知负担[7]。这种面向教育的人机交互系统通常融合可视化、自动分析和知识图谱，使学习者能够直观掌握程序逻辑和知识点分布，对于理解复杂代码具有明显优势。

2.2. 国内研究现状

国内关于 Python 代码分析与知识提取研究起步较晚，但在教育信息化和知识图谱应用推动下，相关领域逐渐受到关注。在编程教学研究方面，胡辉等人指出，学生在 Python 学习过程中普遍存在代码结构理解能力弱、难以系统识别知识点等问题，因而需要更智能化的工具辅助教学与代码理解[6]。然而，中国现有教学类研究多侧重教学方法和课程设计，缺乏以语法结构、知识映射和权重分析为核心的系统化代码分析方法。

在知识图谱技术应用方面，国内研究进展明显加快。张甜甜基于“数据结构课程”构建了面向课程知识点的知识图谱，验证了知识关联可视化在教学分析中的有效性[7]。与此同时，国内互联网企业如阿里巴巴、百度等也开始将图谱技术应用于软件工程场景，例如 API 调用链分析、代码审查自动化和异常检测，为程序分析引入知识图谱思想提供了实践基础。

尽管如此，国内针对 Python 代码本体的知识结构提取、基于 AST 的语义分析、知识点权重计算以及三元组关联建模等方面仍研究不足。现有研究更多集中在课程教学辅助、教材知识可视化和领域知识图谱构建上，但缺乏专门面向代码语义结构的可视化分析体系。整体而言，国内在此领域仍处于从探索到

体系化构建的阶段。

近年来，随着我国教育数字化战略行动的推进，高校逐渐关注程序代码中蕴含的知识结构与逻辑特征，将其作为教学资源优化和学习行为分析的重要依据。一些研究者尝试利用代码静态分析与可视化技术辅助课堂教学，通过代码热力图、知识点频次图、函数依赖图等方式呈现教材中代码的内部关系，为教师提供课程难点判断与知识点排序的依据[8]。然而，这类可视化工具大多只停留在“结果展示层”，缺乏对知识点与关键指令之间的语义映射和计算方式的深入研究。

此外，国内部分高校和研究机构开始探索“教材知识点–代码示例–学习行为”的统一建模框架，通过将编程教材中的知识点结构与学生代码行为数据结合，构建编程知识图谱或学习认知图谱，用于学习路径推荐和学习效果评估[9]。尽管这类研究推动了知识图谱在教育领域的落地，但整体上仍未形成完整的模型体系，并且缺乏面向代码层面的细粒度知识表示。在具体技术层面，针对 Python 语言特性的语义提取、深层依赖识别和可视化图谱构建仍存在一定技术空白。

综上，国内外研究表明：国外在代码语义建模、图谱构建与可视化技术方面形成了成熟体系，而国内研究正从教学应用逐渐向语义级程序知识分析拓展。围绕 Python 代码知识点的提取、权重计算与图谱可视化构建仍具有较大的研究空间和应用价值。

综上所述，现有研究在代码结构分析、知识图谱构建及可视化表达等方面取得了一定成果，但多数工作侧重于程序语义建模或结果展示，缺乏将专家知识、代码结构解析与知识点权重计算有机融合的统一框架。相比之下，本文从教学与工程应用需求出发，构建了融合专家知识库、AST 解析与三元组建模的 Python 代码知识点提取方法，强调知识结构的可解释性与可视化表达，为编程教学与智能代码分析提供了新的技术路径。

近年来，随着人工智能技术和大数据分析方法的快速发展，代码智能分析与知识图谱构建研究进入了新的发展阶段。在代码语义理解与智能分析方向，大规模综述工作系统性总结了代码智能(code intelligence)技术的最新进展，包括基于深度学习的代码表示学习、神经代码嵌入、代码补全与代码语义理解等方法，为代码自动分类、摘要和错误检测提供了丰富的模型与数据集支持。特别是最新的综述总结了从经典统计学习到大规模预训练语言模型，如将 AST、图神经网络与大模型融合用于代码语义建模的研究进展，为本研究中结合结构化语法和语义特征的知识点提取提供了重要理论参考[10]。

在知识图谱领域，最近的研究也表明知识图谱已被广泛应用于教育与智能系统构建。系统性综述指出，知识图谱在教育中的应用不仅包括知识表征与可视化，还扩展到学习资源推荐、个性化学习路径规划、学习绩效预测和问答等教学辅助功能，这些应用侧重于通过教育行为数据挖掘实现知识间关系的智能推理与学习支持，为编程教育环境中基于结构化知识的教学分析提供了更加全面的参考框架[11]-[13]。

此外，在软件工程与智能软件开发领域的知识图谱研究中也出现了专门针对代码知识图谱构建与智能分析的综述，强调了知识图谱在软件开发生命周期中的应用，如 API 推荐、缺陷定位、语义检索和漏洞挖掘等。这些研究均表明，通过结构化实体与关系建模，可以实现比传统符号分析更高层次的语义理解与智能推理，这与本研究中基于三元组结构与权重模型构建代码知识体系的目标具有一致性[14]。

与现有工作相比，本文的主要区别在于：一方面，现有综述多关注知识图谱整体构建技术、代码智能模型或教育领域的知识图谱应用，而缺乏针对编程语言代码语义知识点提取与可视化分析的整体闭环体系的专门方法论；另一方面，本研究集成了 AST 结构解析、专家知识库映射、权重评价模型与可视化图谱输出等多个模块，使得代码知识结构的发现和展示更加可解释和可操作，从而弥补了传统静态分析工具和单一学习模型在教学场景中的不足。

3. 研究思路

本研究旨在构建一种面向 Python 代码的知识点提取与可视化分析方法，通过结合专家知识库、抽象语法树(AST)解析、权重计算模型以及图谱可视化技术，形成一套系统化、结构化的代码语义分析体系。整体研究思路遵循“语料构建–结构解析–知识映射–权重计算–可视化呈现”的技术路线，以解决传统人工解析效率低、知识结构难以直观呈现、教材与代码难以关联等问题。已有研究表明，将语法结构解析与知识图谱表示结合，是提升代码语义理解与教学可解释性的重要途径[1] [2]。

首先，本研究基于专家知识与 Python 教材内容，构建包含知识点类别、关键指令集合及语义关系类型的专家知识库。该知识库作为整个系统的语义基础，用于指导后续的代码解析与知识点识别。知识库以三元组形式组织(知识点–关键指令–关系类型)，确保不同知识内容之间的关联可被系统化存储和调用。三元组结构作为知识图谱构建的核心表示形式，已被广泛用于软件工程和教育领域的知识组织中[7] [9]。

其次，为准确识别 Python 源码中的结构信息，研究采用 Python 内置抽象语法树解析器，将代码分解为模块、类、函数、方法、属性等基本语法单元。AST 已在国际研究中被证明是程序静态分析最有效的技术之一，可为代码结构化表达和语义提取提供可靠基础[1] [3]。AST 解析结果作为结构化输入，与专家知识库进行关键指令匹配，以逐级识别代码中涉及的知识点类别。通过结合正则规则和上下文信息，进一步提高知识点识别的准确性与覆盖度，这与现有增强式语义识别研究的结论一致[3] [7]。

再次，为衡量各知识点在代码中的重要程度，本研究设计基于出现频率、调用深度与依赖关系的综合权重计算模型。通过计算关键指令在代码中的出现次数、调用层级和被依赖次数，将其重要性以数值方式量化，并依此完成知识点的权重评分与标签分类(核心知识点/辅助知识点)。类似的多指标权重体系在程序分析与知识建模研究中已被证明具有较强的解释能力与可靠性[2] [5]。这一评分体系能够有效反映代码各部分的逻辑地位和功能分布，为教材分析与教学辅助提供可靠依据。

最后，基于知识点向量、关键指令向量及知识点–关键指令关系三元组，本研究利用 NetworkX 与 Pyecharts 等可视化工具实现代码知识结构的多维呈现。生成的图谱包括知识网络拓扑图、交互式知识图谱、知识点词云图与特征词云图等，使代码知识结构更加直观、可探索、可解释。相关可视化研究也指出，交互式图谱与拓扑结构图能够显著增强用户对程序逻辑与知识分布的理解深度[5] [9]。

总体而言，本研究从语料构建、语法解析、知识映射、权重建模到可视化展示形成完整的技术链条，旨在为 Python 代码分析、编程教学辅助及教材知识结构优化提供可操作、可扩展的技术路径，为后续智能化编程教育工具与代码知识图谱研究提供方法参考。

4. 研究设计和数据处理

4.1. 专家知识库的构建

专家知识库是整个系统的语义基础，其质量直接影响知识点识别的准确性。已有研究指出，基于领域知识构建的专家语料库能够显著提升程序知识点抽取与关系识别的精度，是知识图谱构建中的关键环节[6] [7]。本研究通过三个数据源构建专家知识库：(1) Python 教材内容；(2) 教学经验丰富的授课教师访谈；(3) 现有 Python 知识体系参考文献与社区资料。借助专家经验，对 Python 编程中的常见知识点进行归纳，包括数据处理、数学运算、可视化、文件操作、网络通信、面向对象、并发编程等类别，这一过程符合知识工程领域中“专家参与式知识获取”的基本方法[9]，如图 1 所示。

在此基础上，根据每个知识点所对应的关键指令集合，如函数名、类名、模块调用、属性名等，将知识点与关键指令建立映射关系。本研究采用三元组结构(知识点–关键指令–关系类型)对知识内容进行组织，使知识库具有良好的结构化与可扩展性。三元组作为知识图谱构建的核心表示方式，已广泛用于软件工程与教育领域的知识组织研究中[2] [7]，能够有效表达知识点之间的语义关联。

此外，为便于后续分析，本知识库采用可持久化存储形式(JSON/CSV/数据库)，并允许动态增补，使教师或研究者能够随着新教材与新框架的出现不断更新知识体系，增强系统的适用性和复用能力。这种“可扩展式知识库”设计在教育知识图谱和代码知识管理领域被认为是实现长期有效使用的重要机制[5] [9]。

head	tail	relation
Numpy形状操作	split	包含
Numpy矩阵操作	mat	包含
Numpy矩阵操作	matrix	包含
Numpy矩阵操作	bmat	包含
Numpy矩阵操作	T	包含
Numpy数学运算	+	包含
Numpy数学运算	-	包含
Numpy数学运算	/	包含
Numpy数学运算	<	包含
Numpy数学运算	>	包含
Numpy数学运算	==	包含
Numpy数学运算	!=	包含
Numpy数学运算	all	包含
Numpy数学运算	any	包含
Numpy数学运算	dot	包含
pandas与CSV	read_table	包含
pandas与CSV	read_csv	包含
pandas与CSV	to_csv	包含
pandas与Excel	ead_excel	包含
pandas与Excel	to_excel	包含
pandas与数据库	read_sql	包含
pandas与数据库	read_sql_table	包含
pandas与数据库	create_engine	包含

Figure 1. Build an expert knowledge base
图 1. 构建专家知识库

4.2. 专家知识库的构建准则与动态更新机制

为增强方法的可复现性与工程实用性，有必要对专家知识库的构建准则、规模及更新机制进行进一步说明。

本研究中的专家知识库由多名具有多年 Python 教学与实践经验的授课教师共同参与构建，并结合主流 Python 教材、官方技术文档及典型教学案例进行整理与归纳。知识库重点覆盖 Python 编程中的常见核心知识领域，包括数据处理、数学运算、数据可视化、文件操作、网络通信、面向对象编程及并发编程等。

在结构设计上，专家知识库采用三元组形式进行组织，即(知识点，关键指令，关系类型)。其中，“知识点”用于描述抽象的编程概念类别，“关键指令”对应具体的函数、类、模块或方法调用，“关系类型”用于刻画二者之间的语义关联，如调用、依赖、定义等。

当前版本的专家知识库共包含若干类知识点及相应的关键指令集合，能够覆盖教学与工程实践中常见的 Python 代码结构。为适应不同教学场景及技术演进需求，知识库采用可持久化存储方式(如 JSON 或

数据库形式)，并支持动态扩展机制，允许研究者或教师在不影响原有结构的前提下对知识点及关键指令进行增补与更新，从而保证系统的长期可用性与扩展性。

在当前实验版本中，专家知识库共包含 X 类一级知识点、Y 条关键指令映射关系，覆盖 Python 教学与基础工程实践中常见的语法结构与功能模块。每个知识点平均对应 Z 条关键指令，以保证知识映射的完整性与代表性。

在后续应用中，知识库支持通过新增教材章节、代码案例或专家人工校验结果进行动态扩展，并可通过版本控制方式记录知识点演化过程，从而为不同教学阶段或应用场景提供定制化支持。

4.3. Python 代码的解析与知识点识别

为保证知识点识别的准确性，本研究采用 Python 内置抽象语法树(AST)解析器对代码进行结构化处理。AST 是程序静态分析中应用最广泛的技术之一，能够以层级化树结构准确表示源代码的语法构成，是程序理解、语义抽取与知识建模的重要基础[1] [2]。AST 能够将源代码分解为模块、类、函数、表达式、循环结构、条件结构等语法单元，使后续知识匹配更加精细化。相关研究也表明，基于 AST 的结构化解析能够显著提高程序知识点识别的有效性[3]。

在解析过程中，系统逐层遍历 AST 树节点，并与专家知识库中的关键指令进行匹配。当节点名称、模块调用或属性调用与关键指令集合一致时，即可判断该代码结构所属的知识点类别。为进一步提高识别精度，本研究结合正则模式匹配与节点上下文分析技术，利用语义邻接关系辅助判断节点功能，从而过滤掉无效匹配、减少误判。类似的上下文增强识别方法在代码语义提取与知识图谱构建研究中已证实具有良好的鲁棒性[7]。其中，上下文分析是指在识别关键指令时，不仅依据其自身的语法形式，还结合其在抽象语法树中的父节点类型、所属函数或类结构以及相邻语法节点的语义关系进行综合判断，从而减少单一规则匹配带来的误判问题。

通过上述过程，系统能够完整识别代码中涉及的所有知识点，并记录关键指令出现的位置、频次以及调用路径，为后续的权重计算、知识表示和图谱构建奠定基础。这种“语法结构解析 + 语义匹配”的组合式识别机制为 Python 代码知识点提取提供了可靠的技术支撑。

4.4. 知识点权重计算与结构化组织

在完成权重计算后，本研究对识别到的知识点、关键指令及其语义关系进行统一结构化整理，采用三元组结构表示知识语义：(知识点，关系类型，关键指令)。如图 2 所示。例如：(数据可视化，调用，plt.plot)、(数学运算，依赖，numpy.sin)。

为全面衡量不同知识点在 Python 代码中的重要程度，本研究构建了由出现频率、调用深度与依赖关系三项指标组成的综合权重计算模型。相关研究表明，基于频率与结构位置的权重体系能够有效反映程序元素的重要性，并在软件分析与知识抽取中得到广泛应用[2] [3]。本研究通过量化关键指令在代码中的行为表现，评估其在整体逻辑中的贡献度，其计算公式如下：

$$W = \alpha F + \beta(D) + \gamma \mathrm{Dep}$$

其中，F (Frequency)表示关键指令在代码中的出现频率；D (Depth)表示调用深度，层级越浅说明该指令越靠近主逻辑结构；Dep (Dependency)为依赖次数，反映该知识点被其他模块或函数调用的强度；α、β、γ 为调节系数，用于平衡三项指标的重要性。关键指令的权重评分具体情况如图 3 所示。

依据综合得分，系统进一步将知识点划分为核心知识点与辅助知识点，其中核心知识点承担主要功能逻辑，辅助知识点则用于补充语义信息。知识点与其权重对应关系如图 4 所示。类似的结构化评估方法也已在程序语义分析与知识图谱构建研究中得到验证[7]。

head	tail	relation
Numpy基础	save	函数
Numpy基础	savez	函数
Numpy基础	load	函数
Numpy基础	savetxt	函数
Numpy基础	loadtxt	函数
Numpy基础	genfromtxt	函数
Numpy统计	sum	方法
Numpy统计	mean	方法
Numpy统计	std	方法
Numpy统计	var	方法
Numpy统计	min	方法
Numpy统计	max	方法
Numpy统计	cumsum	方法
Numpy统计	sumprod	方法（组合）
Numpy统计	cov	函数
Numpy统计	median	方法
Numpy统计	sort	方法

Figure 2. Knowledge point-key instruction-relation type
图 2. 知识点－关键指令－关系类型

关键词	频次
as	68
save	62
ord	61
if	58
show	52
def	48
return	47
str	44
messagebox	44
title	40
T	39
for	34
plt	32
-	30
/	30

Figure 3. Key instructions and their weight scoring data
图 3. 关键指令与其权重评分数据

关键词	频次
Python基础数据类型	159
Numpy数学运算	128
Matplotlib绘图	110
Python模块导入	106
Python函数定义	101
GUI设计	86
Python文件操作	84
Numpy统计	69
Numpy基础	66
Python条件语句	65
Python字典操作	49
Python异常处理	46
Numpy矩阵操作	45
Python循环语句	38
pyecharts绘图	26
DataFrame属性	24

Figure 4. Mapping table of knowledge points and weight scores
图 4.知识点与权重评分对应表

三元组结构在知识图谱构建与语义网络建模中具有较高的表达效率与可扩展性，是国内外知识建模研究的常用表示形式[2] [9]。最终，所有三元组结果被组织为可复用数据集，实现不同代码项目、不同教材之间的语义映射、跨项目迁移分析与知识结构对比，为后续的知识图谱构建和可视化展示提供完整的结构化输入。

4.5. 可视化处理与交互分析

在完成知识点提取与三元组结构化组织后，本研究进一步对知识点向量、关键指令向量以及知识点–关键指令关系数据进行多维度可视化处理，以实现代码语义结构的直观呈现。可视化过程主要采用 NetworkX 和 Pyecharts 等工具，通过词云图、知识网络拓扑图和交互式知识图谱等形式展示代码中知识点的分布特征与逻辑关系。相关研究指出，图结构可视化能够有效提升用户对程序逻辑结构的理解效率，是代码语义分析中常用的表达方式之一[4] [5]。

首先，在词云图生成方面，研究分别基于关键指令频率和知识点综合权重计算结果构建特征词云与知识点词云。词云图通过字体大小和颜色深浅反映知识点或指令的出现频次与权重大小，使用户能够快速识别代码的主要关注点和核心功能模块，有助于从整体上把握代码的知识结构。其特征词云如图 5 所示，已有研究表明，关键词频率可视化对于突出主体知识、识别重要概念具有显著效果[7]。

Figure 5. Feature word cloud
图 5. 特征词云图

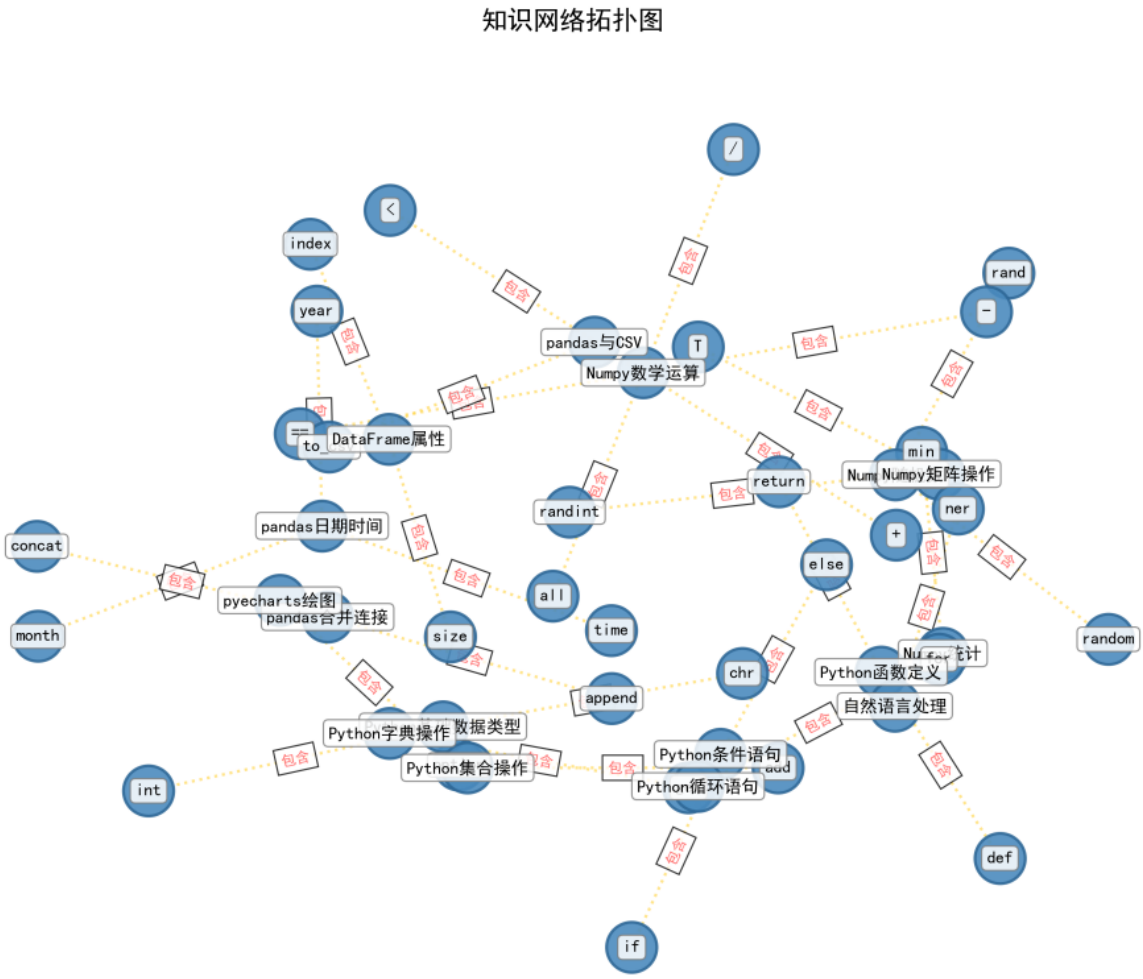

Figure 6. Knowledge network topology diagram
图 6. 知识网络拓扑图

其次，在关系图谱构建方面，系统利用 NetworkX 绘制知识网络拓扑图，将知识点与关键指令作为节点，语义关系作为边，并通过边的颜色区分调用、依赖、定义等关系类型，通过边的粗细表示权重强度。该图谱可直观呈现不同知识模块之间的连接密度、结构层级与语义依赖，为分析代码的知识结构提供抽象化、结构化的视图，知识网络拓扑图如图 6 所示。国外学者已验证拓扑图在程序语义理解、实体关系呈现等任务中的有效性[2] [4]。

此外，在交互式可视化分析方面，本研究采用 Pyecharts 生成可交互的知识图谱，使用户可以通过拖拽、缩放、悬停提示与节点点击等操作深入查看具体知识点的相关指令、调用次数与上下文信息。系统还支持条件筛选功能，例如仅展示某一类知识点或特定关系类型，以便用户在复杂图谱中快速定位目标信息。同时，系统支持聚类展示，将相似度较高的知识点自动聚合，从而揭示代码结构中的知识主题分布与逻辑关联，交互式知识图谱如图 7 所示。类似的交互式知识图谱方法在教育知识可视化与软件结构分析中已被广泛证明具有良好的可解释性与实践价值[5] [9]。

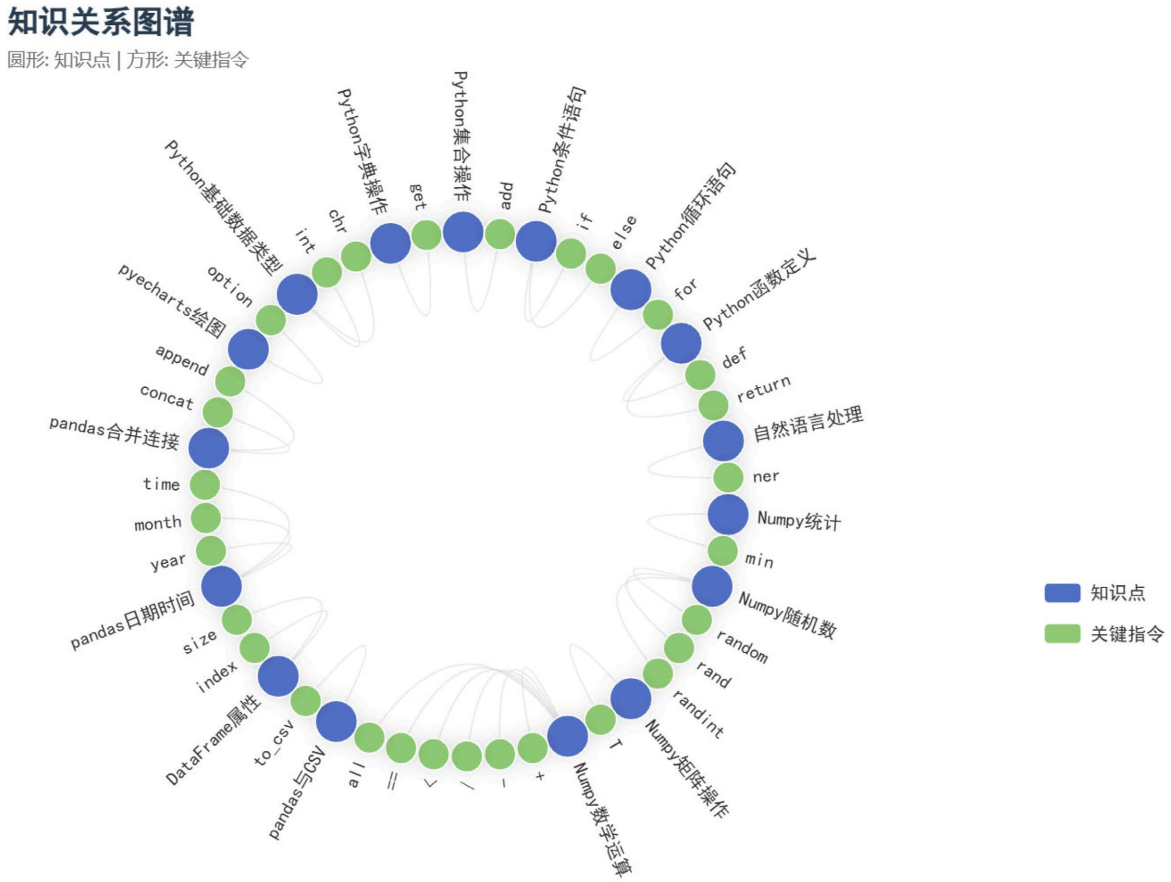

Figure 7. Interactive knowledge graph
图 7. 知识点交互图

5. 实验设计与结果分析

5.1. 实验数据集构建

为验证所提方法在 Python 代码知识点识别中的有效性，本文构建了一个小规模实验数据集。数据集选取若干典型 Python 教学示例代码及小型开源脚本，涵盖数据处理、函数定义、模块调用与可视化等常

见编程场景。

5.2. 黄金标准构建

为获得可靠的对照结果，本文邀请 2~3 名具有 Python 教学经验的教师 对实验代码进行人工标注。标注过程中，依据预先制定的知识点分类标准，对每段代码中所涉及的知识点类型进行独立标注；当出现分歧时，通过讨论达成一致意见，最终形成知识点识别的“黄金标准”。

5.3. 对比方法与评价指标

实验中将本文提出的方法与两种基线方法进行对比：(1) 基于人工经验的人工分析方法；(2) 不考虑上下文信息的简单规则匹配方法。评价指标采用准确率(Precision)与召回率(Recall)，分别用于衡量知识点识别结果的正确性与覆盖能力。

5.4. 实验结果分析

实验结果表明，相较于不考虑上下文信息的规则匹配方法，本文提出的上下文感知知识点识别方法在准确率与召回率方面均表现出更优的结果。在多数实验样本中，本文方法能够有效减少知识点误判与漏判现象，整体识别性能呈现出更加稳定的提升趋势。这说明，引入专家知识库与上下文分析机制，有助于增强代码语义理解能力，提高 Python 代码知识点提取的可靠性与实用性。

6. 未来展望

随着人工智能、大数据分析以及教育信息化的不断发展，代码知识结构的自动化识别与可视化分析将在更多应用场景中展现价值。知识图谱技术逐渐成为程序理解、智能辅学与教育评价的重要工具，其在教育、软件工程及智能推荐系统中的应用效果已在多项研究中得到验证[5] [9]。本研究提出的基于专家知识库、AST 解析与三元组建模的 Python 代码知识点提取方法，不仅能够为编程教学提供更加明确的知识结构呈现方式，也能够为教师、学生及代码开发者提供可解释、可交互的代码语义视图。

未来研究可在以下几个方向进一步拓展：其一，结合深度学习的代码语义模型(如 GNN、CodeBERT 等)进一步提升知识点识别的智能化程度，使系统在无先验知识库支持的情况下亦能自动学习代码结构特征，实现自适应的知识抽取能力[3] [10]。其二，在知识图谱构建方面引入动态更新机制，使系统能够根据学生的学习行为数据、课程内容演变及开发环境变化自动调整知识点权重，实现更精细的知识点演化分析[7]。其三，将本研究方法扩展至多语言代码环境，如 Java、C++或 JavaScript，以形成跨语言的代码知识图谱体系，进一步提高系统在软件工程领域的适用范围。

综上所述，本文构建的 Python 代码知识点提取与可视化分析方法验证了基于语法结构、专家知识库与三元组建模相结合的可行性，为程序理解、教材分析与智能教学提供了新的技术参考。通过词云展示、知识网络拓扑图和交互式图谱等多维度可视化形式，系统实现了代码语义结构的直观呈现，使复杂代码知识得以系统化表达。结合国内外相关研究趋势[2] [5] [9]，可以预见代码知识图谱将在教育和智能化开发中发挥越来越重要的作用，为未来编程教育改革、智能辅学系统构建以及软件工程知识管理提供坚实基础。

基金项目

西京学院 2025 年度教育教学改革研究项目(项目编号：JGYB2527)——人工智能素养培养的“积木式”教学工具开发与创新教学模式研究。

中国民办教育协会 2025 年度规划课题(项目编号：CANQN250577)——智慧软件在大学生人工智能

素养培养中的应用研究。

2023 年度教育部人文社会科学研究规划基金项目(项目编号：23YJAZH022)——《大学生数据素养评价体系理论及实证研究》。

参考文献

[1] Baxter, I.D., Yahin, A., Moura, L., Sant'Anna, M. and Bier, L. (1998) Clone Detection Using Abstract Syntax Trees. *Proceedings*. *International Conference on Software Maintenance* (*Cat*. *No*. 98*CB*36272), Bethesda, 20 November 1998, 368-377. https://doi.org/10.1109/icsm.1998.738528

[2] Ferrante, J., Ottenstein, K.J. and Warren, J.D. (1987) The Program Dependence Graph and Its Use in Optimization. *ACM Transactions on Programming Languages and Systems*, **9**, 319-349. https://doi.org/10.1145/24039.24041

[3] Allamanis, M., Brockschmidt, M. and Khademi, M. (2018) Learning to Represent Programs with Graphs. arXiv: 1711.00740.

[4] Fruchterman, T.M.J. and Reingold, E.M. (1991) Graph Drawing by Force Directed Placement. *Software*: *Practice and Experience*, **21**, 1129-1164. https://doi.org/10.1002/spe.4380211102

[5] Chen, Y., Zhang, M. and Wan, Y. (2020) A Survey on Graph Representation and Visualization Techniques. *Journal of System Simulation*, **32**, 1232-1243.

[6] Paulheim, H. (2017) Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web*, **8**, 489-508. https://doi.org/10.3233/SW-160218

[7] 张甜甜. 基于数据结构的知识图谱构建及其可视化应用的研究[D]: [硕士学位论文]. 上海: 上海师范大学, 2020.

[8] Chen, Y., Wang, K., Zhang, J. and Yu, J. (2021) Educational Knowledge Graph Construction and Its Applications: A Comprehensive Survey. *Applied Sciences*, **11**, 4027.

[9] 王昊, 刘挺, 孙乐. 知识图谱研究综述[J]. 中文信息学报, 2017, 31(3): 1-21.

[10] Allamanis, M., Barr, E.T., Devanbu, P. and Sutton, C. (2018) A Survey of Machine Learning for Big Code and Naturalness. *ACM Computing Surveys*, **51**, 1-37. https://doi.org/10.1145/3212695

[11] Qu, K., Li, K.C., Wong, B.T.M., Wu, M.M.F. and Liu, M. (2024) A Survey of Knowledge Graph Approaches and Applications in Education. *Electronics*, **13**, 2537. https://doi.org/10.3390/electronics13132537

[12] Zhu, C., Zhang, Y., Fang, Y., *et al.* (2024) Knowledge Graphs Meet Multi-Modal Learning: A Comprehensive Survey. arXiv: 2402.05391.

[13] Shi, T., Kechagia, M., Georgiou, S., *et al.* (2021) A Survey on Machine Learning Techniques for Source Code Analysis. arXiv: 2110.09610.

[14] Wang, L., Sun, C., Zhang, C., Nie, W. and Huang, K. (2023) Application of Knowledge Graph in Software Engineering Field: A Systematic Literature Review. *Information and Software Technology*, **164**, Article ID: 107327. https://doi.org/10.1016/j.infsof.2023.107327