分类号：TP391
密　级：公开
学校代码：10363
学　　号：2220920106

安徽工程大学
Anhui Polytechnic University

硕士学位论文

题目　基于问答模式的上下文感知API推荐方法研究

论 文 作 者　方应涛
指 导 教 师　王勇（教授）
学科（专业）　软件工程
研 究 方 向　邻域软件工程

论文提交日期：2025 年 5 月 9 日

分类号：TP391　　单位代码：10363
密 级：公开　　学　号：2220920106

题　目　基于问答模式的上下文感知 API 推荐方法研究

英文并列题　目　**Research on Context-Aware API Recommendation Methods Based on Q&A Patterns**

学生姓名：方应涛
指导教师：王勇（教授）
专　　业：软件工程
研究方向：邻域软件工程

论文答辩日期：2025 年 5 月 10 日

安徽工程大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律后果由本人承担。

学位论文作者签名：方应清

日期：2025年5月9日

安徽工程大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权安徽工程大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密 □，在___年解密后适用本版权书。

本学位论文属于

不保密 ☑。

学位论文作者签名：方云涛　　指导教师签名：

日期：2015年5月9日　　日期：2015年5月9日

基于问答模式的上下文感知 API 推荐方法研究

摘 要

应用程序接口（Application Programming Interface，API）作为现代软件工程体系中的核心中间件组件，其本质是通过接口抽象与实现分离的架构范式，向开发者提供标准化函数调用接口，给开发人员的编程任务带来了极大的便利。然而，随着 API 的迅速发展，其规模呈指数级增长，这使得他们在理解和选择合适的 API 时陷入了认知与决策困境。尽管通过问答社区检索或查阅官方文档能够帮助开发人员理解 API 功能并做出决策，但这些方法的时间代价较高。因此，自动化 API 推荐技术的研究对软件工程领域具有重要意义。现有 API 推荐方法的有效性普遍建立在“用户需求可完整表征”的前提假设上，要求开发者输入的查询语句具备充分的语义完备性。这种假设对新手开发者存在挑战，因其查询语句往往存在显著的表达差异性——这种差异既源于开发者的个体认知差异（如术语使用、表述习惯），也存在于查询文本的描述角度差异（如抽象需求与具体功能的不匹配）。所以本文针对这个问题，主要围绕以下 3 个方面进行展开研究：

（1）问答模式下查询重构的 API 推荐方法研究。用户输入的自然语言查询由于个体认知差异而导致的经验差距现象，不同程序员

对表达同一搜索意图上存在差异，而这种差异现象与编程人员的编程经验有关。对于新手程序员来说，很难准确、完整地表述他们的搜索意图，而有经验的程序员更会描述自己的搜索需求。针对这种现象，本文提出了问答模式下基于查询重构的 API 推荐方法（REAPI）。该方法通过重构编程人员所提问题，来获得更能清晰的意图表达。

（2）问答模式下排序优化的 API 推荐方法。查询文本的描述角度不同而造成知识差异现象，这是由于用户查询输入和官方文档中使用的表达方式不同而产生的现象。用户查询输入主要描述概念和目的，而 API 文档主要描述 API 的功能和结构。针对这种现象，本文提出了问答模式下基于排序优化的 API 推荐方法。该方法通过引入与 API 文档同构的代码注释信息来对 API 候选排序列表进行优化，使得符合用户需求的 API 更容易出现在排序列表中靠前的位置。

（3）问答模式下多源信息集成的 API 推荐方法。自然语言查询的语义表征存在双重维度异质性问题：一方面源于开发者个体间的认知鸿沟，表现为经验水平差异导致的术语使用偏差与语义抽象程度差异，另一方面则表现为查询文本与领域知识库间的语义鸿沟，表现为需求描述与 API 功能介绍间的知识映射失配。本文将这两种问题统称为表达差异现象，为了解决这种现象，本文提出了问答模式下基于多源信息集成的 API 推荐方法（MSEAPIRec）。该方法通过集成 SO 问答网站、Github 代码注释、API 官方文档等多源信息来缓解用户输入的自然语言查询语句存在的表达差异问题，不但有效

的提升了推荐出的 API 的准确性还能优化新手程序员的编程体验，帮助编程新手进行开发任务。

关键词：API 推荐，编程新手，经验差距，知识差距，表达差异，查询重构，多源信息集成

RESERCH ON CONTEXT-AWARE API RECOMMENDATION METHODS BASE ON Q&A

ABSTRACT

Application Programming Interface (API), as a core middleware component in the modern software engineering system, is essentially to provide developers with standardized function call interfaces through the architectural paradigm of separation of interface abstraction and implementation, which brings great convenience to developers' programming tasks. However, with the rapid development of APIs, their size has grown exponentially, which puts them in a cognitive and decision-making dilemma when it comes to understanding and choosing the right APIs. Although searching through Q&A communities or consulting official documentation can help developers understand API functionality and make decisions, these methods are costly in terms of time. Therefore, research on automated API recommendation techniques is of great importance to the software engineering field. The effectiveness of existing API recommendation methods is generally based on the premise that “user requirements can be fully characterized”, which requires developers to input query statements with sufficient semantic completeness. This assumption poses a challenge to novice developers, as their query

statements often have significant expressive variability - a variability that stems from both individual cognitive differences (e.g., terminology usage, presentation habits) and differences in the descriptive perspectives of the query text (e.g., a mismatch between the abstract requirements and the specific functionality). Therefore, this paper focuses on the following three aspects of this problem:

(1) Research on API recommendation methods for query reconstruction in Q&A mode. The natural language query input by the user is due to the experience gap phenomenon caused by individual cognitive differences, different programmers have differences in expressing the same search intent, and this difference phenomenon is related to the programming experience of programmers. It is difficult for novice programmers to express their search intent accurately and completely, while experienced programmers are more likely to describe their search needs. To address this phenomenon, this paper proposes a query reconstruction-based API recommendation method (REAPI) in question-and-answer mode. The method obtains a clearer expression of intent by refactoring the questions asked by programmers.

(2) API recommendation method for sorting optimization in Q&A mode. The phenomenon of knowledge discrepancy due to different descriptive perspectives of the query text is a phenomenon that arises from the difference between the user query input and the expressions used in the

official documentation. The user query input mainly describes the concepts and purposes, while the API documentation mainly describes the functions and structure of the API. To address this phenomenon, this paper proposes an API recommendation method based on sorting optimization in Q&A mode. The method optimizes the API candidate sorting list by introducing code comment information that is isomorphic to the API document, so that the APIs that meet the user's needs are more likely to appear at the top of the sorting list.

(3) API recommendation method for multi-source information integration in Q&A mode. The semantic representation of natural language queries suffers from a two-dimensional heterogeneity problem: on the one hand, it stems from the cognitive gap between individual developers, which is manifested in the terminology usage bias and the difference in semantic abstraction degree caused by the difference in experience level, and on the other hand, it is manifested in the semantic gap between query text and domain knowledge base, which is manifested in the mismatch of knowledge mappings between the requirement descriptions and the API functionality introductions. In this paper, these two problems are collectively referred to as the expression discrepancy phenomenon, and in order to solve this phenomenon, this paper proposes an API recommendation method based on multi-source information integration in Q&A mode (MSEAPIRec). The method mitigates the expression

discrepancy problem of user-input natural language query statements by integrating multi-source information such as SO Q&A websites, Github code comments, and official API documents, which not only effectively improves the accuracy of the recommended APIs but also optimizes the programming experience of novice programmers and helps them to carry out development tasks.

KEY WORDS: API recommendation, novice programmer, experience difference, knowledge difference, expression difference, query reconstruction, multi-source information integration

目录

图录

表录

第1章 绪论

1.1 研究背景及意义

随着云计算和微服务架构的普及，API（Application Programming Interface）已成为现代软件开发中不可或缺的组成部分。API 作为连接不同软件系统的桥梁，允许数据和功能在不同平台、服务和应用之间无缝交互，极大地提高了开发效率和系统的可扩展性。无论是构建移动应用、Web 服务，还是实现企业级系统集成，API 都扮演着关键角色。

API 定义了多个软件中介之间的交互，可以进行的调用或请求的类型、如何进行调用或请求、应该使用的数据格式、要遵循的约定等。其提供了一个预定义函数库，供开发人员在编程期间调用。它不需要开发人员了解内部细节，也不需要访问源代码。由于这些特性，API 可以有效地帮助开发人员提高工作效率。因此，API 在软件开发过程中起着非常重要的作用。事实上，软件开发离不开 API 的支持。例如，每个程序的输入和输出函数实际上都是 API，几乎所有程序都至少使用一个 API。API 已经成为软件开发过程中不可或缺的一部分。

虽然 API 的使用为编程人员带来了极大的便利，但是其带来适用性与有效性的同时也带来了一种重大的挑战——API 的数量飞速增长导致开发人员根本无法熟悉如此庞大的 API 群体[1,2,3]，而通过在问答网站上进行搜索或者查询 API 的官方文档来了解 API 的用法是一件非常耗时的工作，并且对开发人员的经验有很高的要求。缺乏开发经验的人员在搜索或查询时需要花费更多的时间。研究显示，开发人员学习，查找 API 的时间占用了整个开发过程的 40%[4]。

为了节约开发时间，改善编程人员的开发体验，研究人员对自动推荐 API 技术进行了深入的研究。现有的 API 推荐方法可以根据输入信息的不同分为两大类。一类是基于代码上下文进行推荐任务，这类推荐方法主要是利用用户已经编写的代码上下文信息预测该位置应该出现的 API。而另一种推荐方法主要是以问答形式，根据用户的提问进行推荐任务。

基于代码上下文进行推荐的 API 推荐方法首先需要用户客户端编写的代码上下文作为输入信息，而推荐系统则根据输入的结构化的代码信息来推断出上下文中缺失的 API 返回给用户。这种推荐方式快捷且方便，但其准确性较差。

因为这种方式主要依赖代码上下文中的信息，而代码上下文包含的信息十分有限且难以学习。

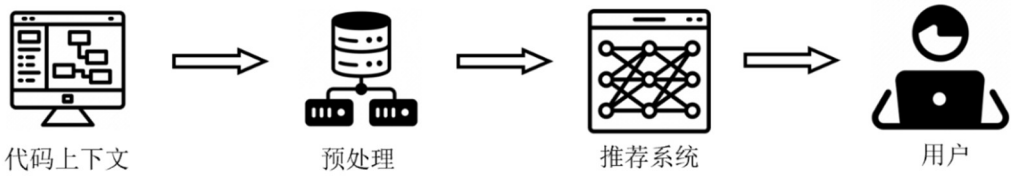

图 1-1 基于代码上下文的 API 推荐方法

Figure 1-1 API recommendation methods based on code context

基于查询的 API 推荐系统需要用户的自然语言查询作为输入信息，推荐系统找到符合用户意图的 API 列表返回给用户。这种推荐方法非常依赖于用户查询语句的质量能全面而又准确的表达自己的需求，这对新手程序员来说是极不友好的。用户给出的查询语句可能存在表达差异的现象，这种表达差异不仅体现在人与人之间（经验表达差距），还体现在文本与文本之间（知识表达差距）。

(1) 经验表达差距：经验差距是指开发人员的编程经验不同，写出的查询语句也会有不同。编程经验越缺乏的开发人员给出的查询语句越模糊。而有经验的程序员对于自身搜索意图的描述则更加得心应手。

(2) 知识表达差距：由于用户查询输入和官方文档中使用的表达方式不同而产生的差异。用户查询输入主要描述概念和目的，而 API 文档主要描述 API 的功能和结构。

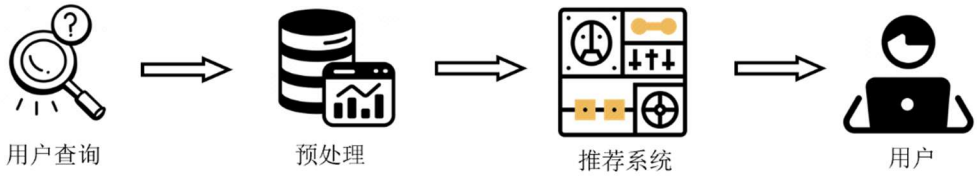

图 1-2 基于查询的 API 推荐方法

Figure 1-2 Query-based API recommendation methods

本文聚焦于基于用户查询的 API 推荐方法，如何弥合用户输入自然语言查询语句的表达差异是本研究的重点问题，在提升 API 推荐性能的同时优化新手程序员的编程体验，节约软件开发时间。所以，基于问答模式的上下文感知 API 推荐方法具有重大的研究价值与现实意义。

1.2 国内外研究现状

软件开发和维护的成本也随着软件系统规模和复杂度的大幅增长而不断增加[5,6]。软件工程领域的一个研究热点是如何高效地重复现有的软件库和框架，并向开发者推荐相关代码。日益兴起的 API 推荐技术，可以对大型程序进行分

析和规律挖掘，减少开发者的工作量，减少开发者对代码的查找，理解，组合和调试，减少人工误差率，提高软件质量。目前相关研究主要集中在基于查询的 API 推荐，基于上下文的 API 推荐。还有一些延伸的 API 推荐方法，例如类库推荐与参数推荐的研究。

1.2.1 API 推荐研究

（1）基于查询的 API 推荐方法

基于查询的 API 推荐系统利用显示查询进行推荐任务，Thung[6]利用用户历史查询与 API 文档描述进行 API 推荐任务。他们提出了一个基于特征请求的方法推荐问题，利用过去类似的关闭或已解决的特征请求的信息，并将特征请求的文本描述与库方法的文本描述进行比较。该技术学习了一个集成的排序函数，然后使用它来推荐在更改后的代码中使用的库方法。Raghothaman[7]与 Wei 等人提出了 SWIM。其引入了结构化调用序列来捕获 API 使用模式。结构化调用序列是方法调用序列的一种通用形式，具有 if 分支和 while 循环来表示条件和重复的 API 使用模式，并且易于提取和合成。zhang[8]提出了一种推荐 API 类的方法 SSAPIR，该方法使用层次聚类算法来提取 API 使用模式,然后通过用户查询与 API 使用模式的相似度值来表示两种信息间的语义相近程度,将相关程度较高的 API 使用方式推荐给开发者。Huang 与 Xia 等人[9]提出了 BIKER，该方法利用多信息源进行推荐任务，引入了 Stack Overflow 来拟合词汇差异与知识差距。引入第三方信息源后，推荐性能得到了质的飞跃。zhou 与 jin 等人[10]提出了基于隐式反馈的推荐方法 BRAID，该方法建立了一个反馈库，利用用户的反馈信息不断的提升推荐的信息。为了缓解用户反馈的冷启动问题，该方法还设计了一个主动学习板块来缓解冷启动问题。wei[11]提出了一种基于对比学习的方法 CLAER，该方法利用联合训练对 BERT 模型进行训练。CLEAR 还采用对比学习拉近相似 SO 帖子距离，远离不相似帖子的距离。Irsan[12]提出了 PICASO。该方法利用用户查询与 Github 注释信息和 API 序列信息进行推荐任务。利用注释信息与 API 序列信息对用户查询进行增强，利用增强后的查询语句可以获得更加准确的推荐性能。随着大语言模型问世以来，其强大的推理能力给 API 推荐领域注入了新的活力。Chen[13]提出了一种上下文增强的方法 APIGen，该方法利用大语言模型强大的表征能力进行 API 推荐任务。Huang[14]提出了一种利用大模型代理澄清用户提问的方法，有效的提升了 API 推荐的性能。

发展趋势可以看出，该研究从开始的与 API 官方文档进行直接匹配的方法，逐渐转化为利用 StackOverflow，GitHub 等第三方信息源进行推荐任务的方法，基于深度学习的语言模型也逐渐代替传统的语言模型。而随着大语言模型的兴起，如何有效激活大语言模型强大的推理能力使其适应与 API 推荐这一下游任务将是未来工作的重点。

（2）基于代码上下文的 API 推荐方法

基于代码上下文的推荐主要从代码中挖掘可用信息。Wang 和 Godfrey[14]研究发现：开发者在构建复杂功能时面临的 API 组合策略选择困境，技术社区中持续涌现大量接口调用顺序相关的技术咨询。以 iOS 开发场景中的 UIScrollView 组件为例，其调用规范相关的技术讨论在 StackOverflow 平台已积累 8422 次知识溯源请求[15]。为应对这一挑战，当前研究趋势聚焦于构建上下文敏感的 API 调用链推理机制，通过深度解析程序执行轨迹与代码语境特征，实现了根据客户端代码上下文语义进行 API 推荐的方法。

在 API 调用序列推荐领域，序列模式挖掘技术扮演着核心角色，始终是学术界关注的焦点。追溯其起源，Agrawal 和 Srikant[16]通过对零售业消费记录的深入剖析，开创性地提出了这一研究课题。随后，众多学者投身该领域，相继开发出多种创新算法，如基于广义序列模式的 GSP[17]、采用前缀投影的 PrefixSpan[18]、利用格点搜索的 SPADE[19]以及基于位向量的 SPAM[20]等。然而，这些算法普遍面临一个棘手的挑战——模式爆炸现象：当最小支持度阈值设置过低时，系统会输出海量的序列模式，其中大量基于频率统计得出的子模式往往缺乏实际意义，严重影响了结果的准确性。为攻克这一难题，研究者们提出了多种优化方案，包括采用双向扩展的 BIDE[21]、基于序列质量的 SQS-search[22]以及运用压缩技术的 GoKrimp[23]等算法。

机器学习的各种方法也被应用到 API 推荐过程中，例如曹[24]等人融合 SOM 功能聚类与 DeepFM 进行 API 推荐任务。Ling 和 Zou 等人[25]提出了一种新的基于图的 API 使用推荐协同过滤方法 GAPI，该方法将 API 调用交互和项目结构集成到一个统一的图中，并利用 GNN 挖掘高阶连通性。并且将客户端方法建模为用户节点，API 被建模为项目节点。图中的边缘包括 API 调用和项目结构之间的关系。然后使用带注意机制的门控循环单元（GRUs）[26]嵌入节点的文本属性，并使用 GNN 有效地利用 API 调用与项目结构的高阶交互。最后通过学习到的嵌入来预测潜在的 API，并从目标库和类似的方法返回 API 的排序列表。杨等学者[27]开创性地将用户反馈信息融入推荐系统，显著提升了 API 推荐的精准度。肖及其团队[28]则另辟蹊径，通过分析代码中的 API 调用序列，构建了 API

关系网络模型，为推荐系统提供了新的视角。段等人[29]提出了一种基于特征表示增强的 Web API 推荐算法，该算法通过增强特征表示，高效地进行 Web API 推荐，从而提升了Mashup创建的效率。奚[30]则从问答平台中挖掘信息，提出了弃用 API 迁移建议推荐技术，这些建议不仅包含替换 API 的文本描述，还提供了实用的代码示例。Chen 和 Peng 等人[31]则提出了一种名为 COOK 的方法，该方法通过三种策略对 API 序列建议进行后处理：首先，将程序语法和推荐多样性注入束搜索过程，以过滤不满意的 API 建议；其次，开发了一种结合文本相似度和 API 嵌入相似度的分层聚类算法，将相似的 API 序列建议分组；最后，根据 API 序列建议中的通用代码结构和关键 API 的功能描述，为每个集群提供摘要，帮助开发人员更好地理解 API 序列建议。这些方法各具特色，共同推动了 API 推荐技术的发展。Chen 和 Gao 等人[32]提出了一种新的基于多视图异构图表示学习的 API 使用推荐方法 MEGA。多视图异构图包括本地视图的方法- API 交互、全局视图的 API-API 共现和外部视图的项目结构。此外，设计了两种新的注意网络，用于在学习 API 和方法表示时编码基于频率的共现信息和基于结构的分层信息。

从基于代码上下文的推荐方法的发展趋势可以看出，基于学习的方法逐渐代替传统的数据挖掘方法。深度学习的方法逐渐代替传统方法，又由于图可以包含更多的节点信息，所以图神经网络成为发展的新趋势。

1.2.2 类库推荐研究

Chen[33]的实证研究揭示开源生态中依赖管理的普遍性：基于 GitHub 平台 1008 个项目的样本分析表明，超过九成项目存在第三方组件集成行为，单个工程平均承载 28 个外部依赖项。这种技术栈的深度嵌套使开发者陷入依赖项管理的决策复杂度之中。在解决此类技术债务方面，Teyton 团队[34]提出基于软件演化规律的依赖迁移策略，通过建立版本迭代路径拓扑模型，实现老旧组件的智能替换。典型案例显示，当代码库中检测到 Log4J 组件时，系统将基于历史迁移数据推荐 SLF4J 作为现代化替代方案。另一维度上，Thung 等研究者[35]通过关联规则挖掘技术，揭示了跨项目依赖组合的潜在模式，建立已集成组件与潜在需求组件间的概率映射关系。为突破技术生态壁垒，Chen与Xing合作开发的SimilarTech框架[36]实现了多语言技术栈的兼容性推荐，覆盖 6 种编程语言 6715 个组件实体。该框架运用语义解析技术，对Stack Overflow技术社区的知识元数据进行深度建模，构建跨平台依赖知识图谱，最终实现基于功能等价性的组件智能匹配。Kawser 等人[37]提出了一种称为 XLibRec 的技术，以推荐跨语言类比库。这种技术依赖于两种不同的信息来源。对四种不同编程语言（Java、Python、C#和 JavaScript）中随机选择的 900 个库进行了手动研究。发现

XLibRec 可以跨不同编程语言推荐类比库，且优于现有的最先进技术（即 SimilarTech）。康等人[38]预训练语言模型进行跨库 API 推荐研究。

从类库推荐研究发展趋势可知，跨语言推荐成为研究的热点问题。加强类库推荐模型的可扩展性与鲁棒性成为主流趋势。

1.2.3 参数推荐研究

在软件开发领域，参数选择是代码实现中的一个关键挑战，不当的参数选择可能导致程序缺陷的产生。针对这一问题，上海交通大学的张团队[39]于 2012 年提出了一种名为 Precise 的 API 参数推荐系统。该系统通过结合当前编码环境与历史代码示例，构建了一个三阶段的推荐机制：首先，Precise 从代码库中提取参数实例的抽象表示和使用上下文特征，构建参数特征库；接着，运用 K 近邻算法在参数库中搜索相似参数，形成候选列表；最后，依据上下文匹配度和使用频率对候选参数进行排序，以辅助开发者做出选择。然而，Precise 在处理布尔型参数及简单实参推荐时表现欠佳，如 frame.setVisible(true)等场景。Asaduzzaman 等人[40]进一步研究指出，参数使用具有局部性特征，通常在代码初始化阶段附近出现。基于这一发现，他们开发了 Parc 系统，通过提取参数前 k 行代码中的关键词（如方法名、类名、接口名等）作为局部特征，构建参数特征库，并采用 simhash 算法计算相似度，从而提供更精准的参数推荐。结果显示：Parc 给出的前 10 推荐结果的准确度达到了 72.06%。Yang[41]通过关键的直觉来进行 API 参数推荐。首先通过切片 API 参数，可以更好地捕获参数的使用模式。第二，深度神经网络可以应用于 API 参数的推荐。这两种见解也是此方法的基础，该方法结合了机器学习和程序分析的技术。Manh 等人[42]提出了一种新的、有效的自动完成 API 实际参数的方法 FLUTE。首先 FLUTE 为了保证推荐的参数在编码程序方面语法正确，应用了程序分析。之后根据形式参数的类型，生成所有潜在的候选项。其次为了解决语义正确性的挑战，训练语言模型来预测每个候选在推荐上下文中的出现概率。张等人[43]提出了基于上下文信息的 API 使用模式和参数推荐方法研究提出了一种基于上下文信息的 API 参数推荐方法。

从 API 参数推荐的发展趋势中可以看出，基于深度学习的方法逐渐代替传统发从历史数据中挖掘的方法。深度学习中的生成模型为参数推荐注入新的活力。

目前，针对 API 推荐的研究已经成为趋势。最新的 API 推荐研究方法 GAPI 和 MEGA 采用了深度学习方法 GNN 进行推荐任务，通过提取项目与项目、项目与 API、API 与 API 之间的联系，挖掘高阶连通性，将 API 调用交互和项目结构集成到一个统一的图中。同时，PICASO 和 CLEAR 也利用了深度学习方法

BERT框架来捕捉语义信息。就API推荐的研究趋势而言，深度学习方法逐渐代替传统的概率模型或基于检索的推荐方法。最近，大语言模型（LLM）成为软件工程领域的热点问题，运用到提示学习与LLM模型进行API推荐任务可能会成为一种新的趋势。而问答模式是大语言模型的一种常见形式，如何将上下文代码中有用的信息与问答模式结合进行推荐任务是一种大幅提升推荐性能的方法，也将是未来研究的趋势与方向。

1.3 主要研究内容

随着 API 的数量呈现爆炸式增长。开发者现在可以从数以万计的公共 API 中选择，涵盖从支付处理、地图服务到人工智能和区块链等各个领域。这种丰富性虽然为开发者提供了更多可能性，但也带来了新的挑战：如何在众多 API 中找到最符合自身需求的选项？开发者不仅需要考虑 API 的功能匹配度，还要评估其性能、可靠性、文档质量、社区支持以及成本等因素。这种复杂性使得手动筛选API变得耗时且低效。

现有的自动化 API 推荐方法虽然有效的缓解了这个问题，但是依然面临着严峻的挑战。即现有的 API 推荐方法过度关注与如何准确全面的表征用户输入，但是都忽略了一个严重的问题——用户输入可能存在问题，用户输入的自然语言查询可能并不表示自身实际的任务需求。在这种情况下，无论表征模型如何的准确也不能推荐出符合任务需求的 API。也就是说现有方法过度依赖于用户能够准确完整的表达自身的任务意图，但是并非所有用户都擅长于表述自身需求。用户描述搜索意图的能力与他们的编程经验密切相关。对于新手程序员来说，很难准确、完整地表述他们的搜索意图，而有经验的程序员则更会描述自己的搜索需求。此外，频繁搜索的开发人员往往是编程新手，他们在开发过程中浪费了大量的时间在搜索步骤上。本文将这种现象称为，经验差距，这种差异是体现在人与人之间的差异现象。

此外，除了人与人之间的差异现象，文本与文本之间也存在这种差异现象，这种现象是由于用户查询输入的自然语言文本和官方文档中 API 描述文本使用的表达方式不同而产生的差异。用户查询文本主要描述概念和目的，而 API 文档描述文本主要描述API的功能和结构。例如：对于一个任务需要的目标API——Arrays.fill。在 SO 网站中的用户是这样提问的：“How to initialize all values in

an array to false?”。而该 API 在 API 文档中的描述是这样的：“Assigns the specified boolean value to each element of the specified array of Booleans.”前者是对搜索目的的描述，而后者是对该 API 功能的描述，这两种描述间并没有共享太多的相同词汇，相似度极低。这种现象极大的限制了 API 推荐结果的准确性。

本文将上述两种现象综合称为表达差异，即编程人员输入的查询语义信息是存在表达差异的挑战，这种表达差异不仅体现在人与人之间（经验差距），还体现在文本与文本之间（知识差距）。经验差距：不同的程序员在表达同一搜索意图上存在差异，而这些差异主要与程序员的编程经验有关。知识差距：由于用户查询输入和官方文档中使用的表达方式不同而产生的差异。用户查询输入主要描述概念和目的，而 API 文档主要描述 API 的功能和结构。本文围绕这种挑战，提出了一种感知用户语义上下文的推荐方法，如图1-3所示。本文的主要研究内容如下：

（1）问答模式下查询重构的 API 推荐算法研究。针对用户输入的查询语句存在的经验差距问题，本文通过提取 Stackoverflow（SO）中高质量的问答对作为丰富的经验先验知识库，基于此知识库与用户显示反馈信息对用户的输入进行重构，重构后的自然语言查询语句能更清晰的反映用户的真实意图。利用重构后的查询语句进行后续推荐任务能显著提升推荐任务的准确性。

（2）问答模式下排序优化的 API 推荐算法研究。针对用户输入的查询语句存在知识差距问题，本文通过利用同样是描述功能与结构的代码注释来对 API 候选列表进行重排。代码注释与 API 官方文档都是用于描述 API 的功能与结构，属于同质信息，计算代码注释与 API 官方文档的相似度来重排候选列表能够使得符合任务需求的答案 API 出现在更靠前的位置。

（3）问答模式下多源信息集成的 API 推荐方法研究。针对用户输入的查询语句存在表述差异的问题，本文通过集合 SO 问答帖子、Github 注释信息与 API 官方文档描述等多源信息来解决。本文引入了更富经验性的注释信息与 SO 问答帖子与原始查询来构建了一个语义增强器。通过语义增强器的原始查询将蕴含更丰富的语义信息与经验信息，这有效缓解了用户输入查询语句不全或存在经验表达差距的问题。此外，本文通过集成 Github 注释信息与 API 官方文档描述与原始查询语句设计了序列优化器，有效的缓解了知识差距的问题并优化了正

确答案的排名。

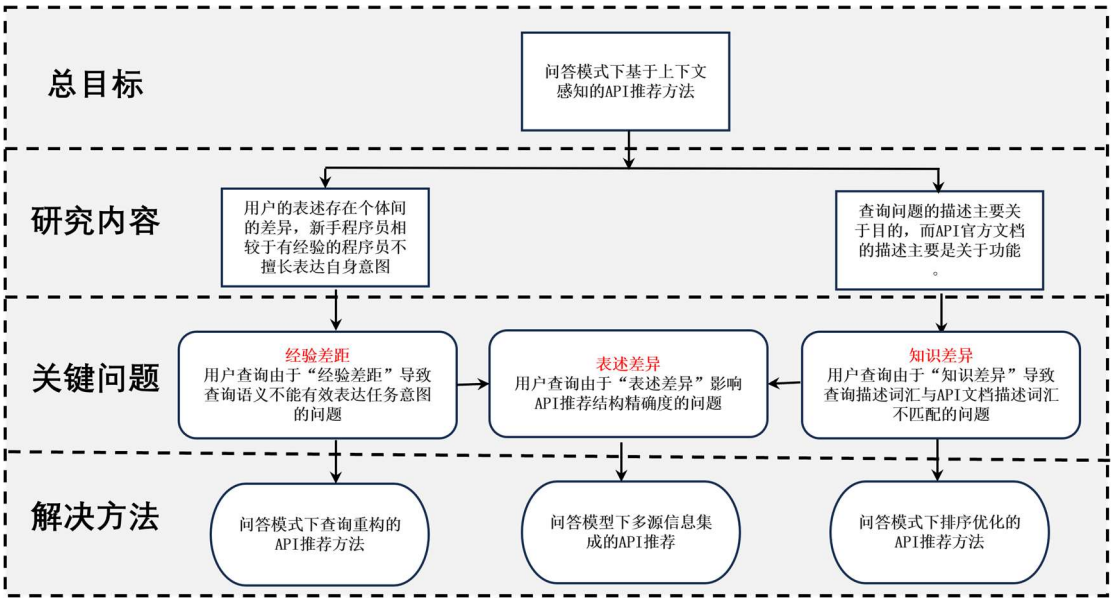

图 1-3 论文研究思路框架图

Figure 1-3 Framework diagram of the research idea of the dissertation

1.4 论文组织结构

本文的第一章是绪论部分，本章节主要介绍了研究背景及意义，国内外研究现状、主要研究内容与论文的组织结构。其中，国内外研究现状包括基于查询的 API 推荐方法、基于代码上下文的 API 推荐方法、类库推荐研究以及参数推荐研究。

第二章为相关技术概述。本章介绍的内容都是本文用用到的技术或概念。其中包括文本表征技术、自然语言处理技术、提示学习、对比学习。关于文本表征技术，本章着重介绍了 TF-IDF 表示、Word2vec 与 BERT 这几种本文用到的表征技术。关于自然语言处理技术，本章介绍了文本清洗、分词、词干提取和词还原与词性标注，这些技术都是本文在数据处理过程中用到的技术内容。

第三章为问答模式下查询重构的 API 推荐方法。本章首先明确研究问题与目标，是针对用户输入的查询语句存在经验差距导致查询语句表述不清的现象。然后详细介绍了模型的设计，其中包括训练语言模型、查询显示化重构与相关 API 推荐。接下来，本章进行了实验验证所提方法的有效性。最后，本章还设计了用户实例研究证明了该方法的现实价值。

第四章为问答模式下排序优化的 API 推荐方法。本章首先介绍了研究问题

与目标，是解决用户输入的自然语言查询语句存在知识差异导致文本词汇不共享的问题。其次，本章介绍了模型的构建过程，包括训练语言模型与候选排序优化。最后，本章进行了实验来测试所提方法的性能。

第五章为问答模式下多源信息集成的 API 推荐方法，本章首先对研究问题与目标进行了介绍。接着，本章详细阐述了模型的设计过程，其中包括训练语言模型、构建语义增强器与相关 API 推荐。最后本章详细介绍了实验的设计过程与对实验结果给进行了全面合理的分析。

第六章为总结与展望，描述了本文的主要工作以及未来的研究方向。

第2章 相关技术概述

2.1 文本表征技术

文本表征是自然语言处理（NLP）与机器学习（ML）领域中的一项至关重要的任务。其主要目的是将非结构化的文本内容转化为数值形式，通常为向量或者矩阵形式。因为计算机可以处理这种数值形式的数据，经过这种变化后可以让计算机能够间接处理非结构化的文本数据，理解其语义与结构信息。文本表征技术是众多自然语言处理任务的基础，包括文本分类、情感分析、信息检索、机器翻译等。本文将介绍 TF-IDF[44]，Word2vec，BERT 这几种在本文中使用到的文本表征技术。

2.1.1 TF-IDF 表示

TF-IDF（Term Frequency-Inverse Document Frequency）是一种经典的文本表示技术[44]。TF-IDF 值是通过将词频和逆文档频率相乘得到的，其数值越大，说明该词在当前文档中越关键，同时在语料库中越不常见。因此，TF-IDF 值的高低可以反映一个词在文档中的重要程度。其计算公式如下所示：

$$TF(c,d)=\frac{\text{文档d中词c出现的次数}}{\text{文档d的总次数}} \tag{2-1}$$

$$IDF(c,d)=\log(\frac{\text{文档集合D中总文档数}}{\text{含有词c的文档数}+1}) \tag{2-2}$$

$$\text{TF}-IDF=TF(c,d)*IDF(c,d) \tag{2-3}$$

TF-IDF 有诸多优势，例如，其计算方法直观且易于理解，实现起来较为简单，适合快速部署和应用。计算速度快，适合处理中小规模的文本数据。此外，通过结合词频（TF）和逆文档频率（IDF），能够有效区分常见词（如停用词）和重要词，突出文档中的关键信息。不需要标注数据，直接基于文本的统计特性进行计算。但是，TF-IDF 也存在很多缺点，其仅基于词频和文档频率进行计算，无法捕捉词汇之间的语义关系（如同义词、一词多义等）。生成的 TF-IDF 向量通常是高维且稀疏的，可能导致计算效率下降，尤其是在处理大规模数据

时。此外，IDF 的计算依赖于语料库的规模和质量，如果语料库不具代表性，可能导致 IDF 值不准确。

2.1.2 Word2vec

Word2Vec[45]核心理念是“具有相似上下文的词语在语义上也相近”，通过大规模语料训练，模型能够生成每个词的向量表示。该技术主要包含以下两种模型结构：

（1）连续词袋模型（CBOW）是一种通过上下文词汇预测目标词汇的架构，尤其适用于高频词的处理[46]，且训练效率较高。该模型的核心目标是根据上下文推断出最可能的中心词。如图 2-1 所示，CBOW 采用三层结构，窗口大小设为 2。输入层（Input）接收上下文词汇 w(t-2)、w(t-1)、w(t+1)、w(t+2)，并将其传递至隐藏层（Projection）进行处理，最终在输出层（Output）生成对中心词 w(t)的预测结果。

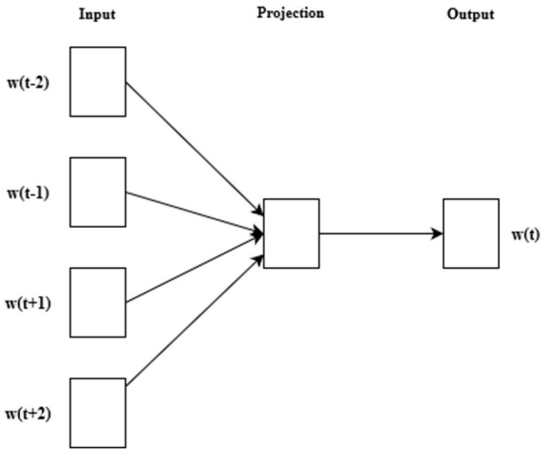

图 2-1 CBOW 示例图

Figure 2-1 CBOW example diagram

（2）Skip-Gram 模型[46]：通过目标词预测其上下文词汇，尤其擅长处理低频词，并能更精细地捕捉语义信息。与 CBOW 不同，Skip-Gram 以当前词为基础推测其周围的上下文。这种架构在处理大规模数据集时表现优异，能够有效挖掘稀疏数据中的关联。如图 2-2 所示，Skip-Gram 同样采用三层结构：输入层（Input）接收目标词 w(t)，经过隐藏层（Projection）处理后，在输出层（Output）预测其上下文词汇 w(t-2)、w(t-1)、w(t+1)、w(t+2)。

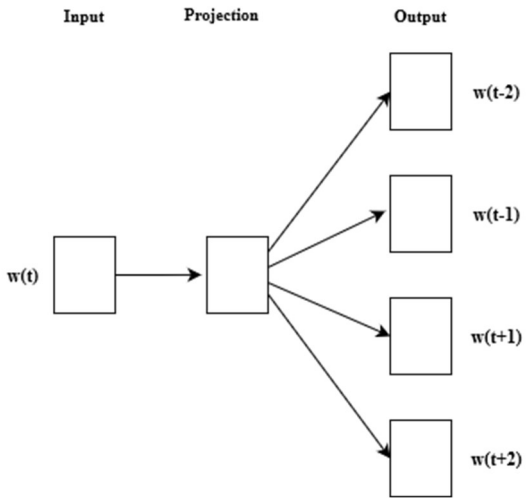

图 2-2 Skip-gram 示例图

Figure 2-2 Skip-gram example diagram

Word2vec 表征模型语义捕捉能力强，能够有效捕捉词语的语义和语法关。此外，相比传统的词袋模型，Word2Vec 生成的词向量维度更低，计算更高效。所以该表征模型常常应用于推荐系统与问答系统中。但是该表征模型也有几点缺点，例如其无法处理多义词，每个词语只有一个向量表示，无法区分多义词的不同含义。依赖大量数据，需要大规模的文本数据才能训练出高质量的词向量。此外，该表征模型无法动态更新，一旦模型训练完成，词向量就固定了，无法动态适应新词或新语义。

2.1.3 BERT

BERT（Bidirectional Encoder Representations from Transformers）是由 Google 于 2018 年提出的一种基于 Transformer 架构的预训练语言模型[47]。其核心目标是通过大规模无监督学习生成高质量的词向量表示，从而为下游自然语言处理任务提供强大的语义理解能力。BERT 的核心创新在于其双向上下文建模机制，即模型能够同时捕捉句子中每个单词左右两侧的上下文信息，从而更全面、准确地理解单词的语义及其在句子中的角色。与传统的单向语言模型不同，BERT 摒弃了解码器部分，仅采用 Transformer 的编码器结构，使其在语义表征上更加高效。

BERT 的训练过程分为两个关键阶段：预训练和微调。在预训练阶段，BERT 通过两个主要任务学习语言的内在规律：

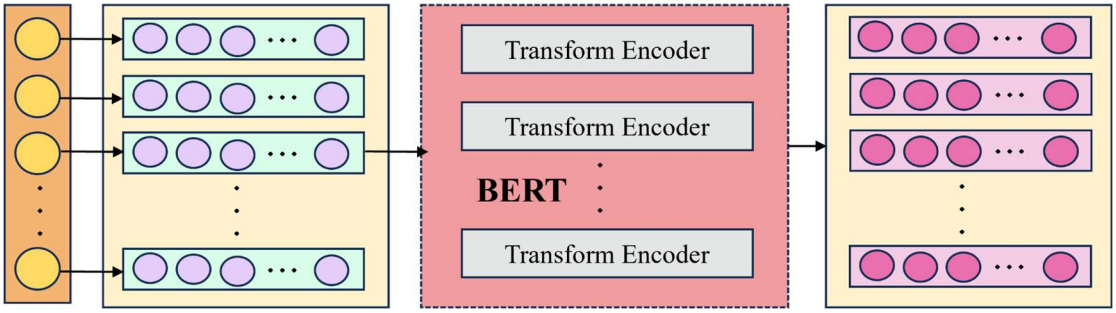

图 2-3 BERT 模型示例图

Figure 2-3 BERT model example diagram

掩码语言模型（Masked Language Model, MLM）：该任务随机掩盖输入句子中的部分单词，并用特殊标记[MASK]替代，随后要求模型基于上下文预测被掩盖的单词。这一机制迫使模型深入理解单词与其上下文之间的关系，从而提升语义表征能力。

下一句预测（Next Sentence Prediction, NSP）：该任务旨在判断两个句子是否为连续的文本片段，帮助模型学习句子之间的逻辑关系和篇章结构。通过这一任务，BERT 能够更好地理解句子级别的语义关联。

在预训练完成后，BERT 进入微调阶段。此时，模型会根据具体的自然语言处理任务（如文本分类、命名实体识别、问答系统等）进行参数调整，以适配特定任务的需求。这种“预训练+微调”的范式使得 BERT 能够快速适应多种下游任务，同时显著减少对标注数据的依赖。

得益于其卓越的双向上下文理解能力，BERT 在文本分类、句子对分类、问答系统等任务中表现优异。此外，通过大规模无监督学习，BERT 能够从海量文本数据中学习到丰富的语言知识，从而在多种任务中展现出强大的泛化能力。然而，BERT 模型也存在一些局限性：首先，其参数量庞大，导致训练和推理成本较高；其次，在数据量较少的下游任务中，模型容易出现过拟合现象。尽管如此，BERT 的出现仍然为自然语言处理领域带来了革命性的进展，并为后续的预训练语言模型研究奠定了重要基础。

2.2 自然语言预处理技术

自然语言预处理技术是自然语言处理（NLP）中的一个重要环节，它为后续的文本分析、特征提取和模型训练做准备。预处理的目的是将原始文本数据转换为更易于处理和分析的形式，同时去除噪声和冗余信息。以下是自然语言

预处理技术的主要步骤和方法：

2.2.1 文本清洗（Text Cleaning）

文本清洗是自然语言预处理的首要步骤，其核心任务是去除文本中的噪声和无关信息，使文本更加干净、简洁，从而为后续的分析和处理提供高质量的数据基础。在实际应用中，文本数据往往来源于多种渠道，如网页、社交媒体、用户评论等，这些数据中常常夹杂着大量无关的符号、标签和格式化内容，这些噪声不仅会增加数据的复杂性，还可能干扰模型对文本核心语义的理解。

（1）去除 HTML 标签：从网页中提取的文本通常包含 HTML 标签，这些标签虽然在网页显示中具有重要作用，但对于文本分析而言，它们往往被视为噪声。HTML 标签的存在不仅会增加数据的复杂性，还可能干扰文本的语义理解。因此，去除 HTML 标签是文本清洗的第一步，也是至关重要的一步。在实际操作中，可以通过正则表达式或使用专门的 HTML 解析工具（如 BeautifulSoup）来实现 HTML 标签的去除。

（2）去除特殊字符和标点符号：文本数据中常常包含各种特殊字符（如@、#、$等）和标点符号。这些符号在某些情况下可能对文本的语义理解没有实质性帮助，甚至可能引入噪声。例如，在情感分析或文本分类任务中，过多的特殊字符和标点符号可能会干扰模型对文本核心语义的把握。因此，去除这些无关的符号是文本清洗的重要环节之一。去除特殊字符和标点符号的操作通常借助正则表达式来完成，正则表达式能够灵活地定义字符模式，从而精准地识别并删除目标字符。

（3）去除停用词：停用词是指在文本中频繁出现但对语义贡献较小的词语，如“的”、“是”、“在”等。这些词语在文本中占据较大比例，但往往缺乏实际的语义价值。在进行文本分析时，保留停用词可能会导致模型过于关注这些高频但无意义的词语，从而降低模型对关键信息的敏感度。因此，去除停用词是提升文本数据质量的有效手段之一。停用词的去除可以通过预定义的停用词列表来实现，这些列表通常包含了各种语言中常见的停用词。在实际应用中，也可以借助 NLP 工具库（如 NLTK、spaCy）提供的停用词列表，这些工具库经过精心设计，能够高效地识别并去除文本中的停用词。

（4）去除空格和换行符：文本数据中多余的空格和换行符可能会对文本的

结构和语义理解造成干扰。例如，过多的空格可能导致文本分词时出现错误，而换行符可能会使文本的逻辑结构变得混乱。因此，去除多余的空格和换行符是文本清洗中不可或缺的一步。去除空格和换行符的操作可以通过字符串的内置方法（如 strip()、replace()）或正则表达式来完成，这些方法能够灵活地处理文本中的空白字符，从而确保文本数据的整洁性和一致性。

通过上述清洗操作，文本数据将变得更加干净、简洁，从而为后续的分析和处理提供高质量的数据基础。高质量的文本数据不仅能够减少模型训练过程中的干扰，还能提高模型的性能和泛化能力。

2.2.2 分词（Tokenization）

分词是自然语言预处理中的关键步骤之一，其目的是将连续的文本字符串分割成有意义的基本单元，如单词、短语或符号。这些基本单元被称为“token”，是后续文本分析和处理的基础。分词的粒度和方法取决于具体任务的需求以及所处理的语言特性。在不同的语言和应用场景中，分词的方法和工具也有所不同。

对于英文等以空格分隔的文本，分词相对简单，通常可以直接根据空格将文本分割成单词。然而，对于中文、日语等没有明显单词分隔符的语言，分词则需要借助专门的算法和工具。例如，中文分词工具（如 jieba）通过统计方法或基于规则的方式，将连续的中文字符序列分割成有意义的词语。分词的准确性直接影响到后续任务的效果，如文本分类、情感分析等。不准确的分词可能导致模型无法正确理解文本的语义，从而影响模型的性能。

在实际应用中，分词不仅需要考虑语言特性，还需要根据具体任务的需求进行调整。例如，在某些任务中，可能需要保留一些特定的短语或符号，而在其他任务中，则需要更细致地分割文本。此外，分词还可以结合上下文信息，以更好地捕捉文本的语义结构。例如，基于深度学习的分词模型（如 BERT）能够利用上下文信息动态地进行分词，从而提高分词的准确性和灵活性。

总之，分词是自然语言预处理中不可或缺的一步，通过将文本分割成有意义的基本单元，为后续的文本分析和处理提供了基础。准确的分词能够显著提高模型对文本语义的理解能力，从而提升模型的性能。

2.2.3. 词干提取（Stemming）和词形还原（Lemmatization）

在自然语言处理中，词干提取和词形还原是两种常见的词汇规范化技术，旨在将单词的不同形态还原为一个统一的基本形式，从而减少词汇的多样性，提高文本数据的一致性。

词干提取（Stemming）是一种基于规则的方法，通过去除单词的前缀和后缀，将其还原为词干形式。例如，将“running”还原为“run”。词干提取算法通常比较简单且高效，常用的算法包括 Porter Stemmer 和 Snowball Stemmer。然而，词干提取的结果可能并不是一个真正的单词，因为它只是基于规则进行简单的字符操作。尽管如此，词干提取在许多应用中仍然非常有效，尤其是在需要快速处理大量文本数据时。

词形还原（Lemmatization）则是一种更为复杂的词汇规范化方法，它不仅考虑单词的形态变化，还结合了语言的词法规则和上下文信息，将单词还原为其基本形式（词形）。例如，将“running”还原为“run”，“are”还原为“be”。词形还原的结果通常是一个真正的单词，因此在需要准确处理文本语义的任务中，词形还原比词干提取更为适用。然而，词形还原的计算成本相对较高，因为它需要借助词典和词法规则来完成。

在实际应用中，选择词干提取还是词形还原取决于具体任务的需求。如果任务对处理速度要求较高，且对词汇的精确还原要求不高，词干提取是一个不错的选择。相反，如果任务需要准确处理文本的语义，如情感分析或机器翻译，词形还原则更为合适。通过将单词还原为基本形式，词干提取和词形还原能够显著减少词汇的多样性，提高文本数据的一致性，从而为后续的文本分析和处理提供更高质量的数据。

2.2.4. 词性标注（Part-of-Speech Tagging）

词性标注是自然语言处理中的一个重要任务，其目的是为文本中的每个单词标注其词性类别，如名词、动词、形容词、副词等。词性标注不仅有助于理解单词在句子中的语法角色，还能为后续的文本分析任务提供重要的语法信息。例如，在句法分析、语义角色标注和信息抽取等任务中，词性标注能够帮助模型更准确地理解句子的结构和语义。

词性标注的方法可以分为基于规则的方法和基于统计的方法。基于规则的

方法通过预定义的词法规则和模式来标注词性，这种方法的优点是规则明确，易于理解和实现。然而，基于规则的方法在处理复杂的语言现象时可能存在局限性，因为语言的规则往往存在例外情况。基于统计的方法则通过训练数据来学习单词的词性分布，这种方法能够更好地处理语言的复杂性和多样性。常用的基于统计的词性标注工具包括 NLTK、spaCy 等，这些工具通过机器学习算法，如隐马尔可夫模型（HMM）或条件随机场（CRF），自动学习单词的词性标注规则。

在实际应用中，词性标注的准确性直接影响到后续任务的效果。例如，在信息抽取任务中，准确的词性标注能够帮助模型更准确地识别关键信息，从而提高信息抽取的准确率。此外，词性标注还可以与其他 NLP 任务结合，如命名实体识别（NER）和依存句法分析，以进一步提升模型的性能。

总之，词性标注是自然语言预处理中的一个重要环节，通过为文本中的每个单词标注其词性类别，能够为后续的文本分析任务提供重要的语法信息，从而提高模型对文本语义的理解能力。

其实关于自然语言预处理还包括命名实体识别（Named Entity Recognition, NER），依存句法分析（Dependency Parsing），向量化（Vectorization）等更多的处理步骤与方法，但是本文不涉及这些步骤，所以并不进行赘述。

2.3 提示学习

提示学习（Prompt Learning）是一种新兴的自然语言处理技术[48]，通过向预训练语言模型（如 GPT-3、GPT-4）提供特定的提示（Prompt），引导模型生成符合任务需求的输出。提示可以理解为一种指令或问题，模型基于这些提示来完成任务，如文本生成、翻译、分类等。这种方法的核心在于，无需对模型进行大规模的重新训练或微调，而是通过设计合适的提示来引导模型的行为。

提示学习的工作原理基于预训练语言模型的强大能力。这些模型在大规模文本语料库上预训练，学习了丰富的语言知识和结构。通过设计合适的提示，可以激活模型的预训练知识，使其能够理解和执行特定的任务。其工作示例图如图 2-4 所示。

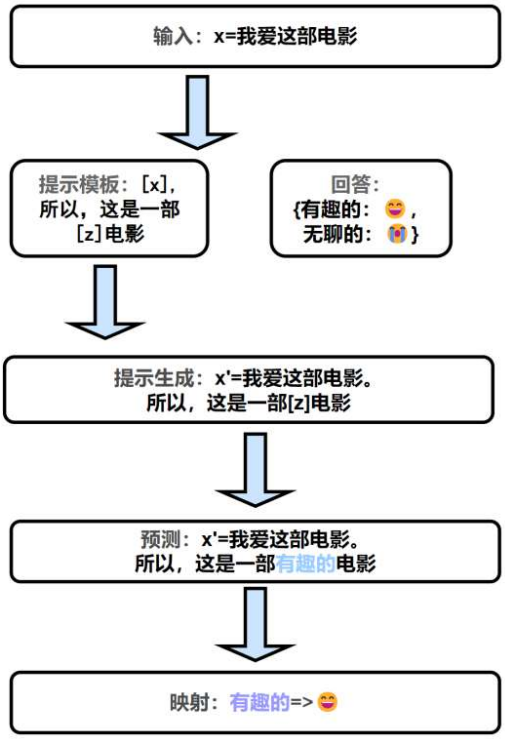

图 2-4 提示学习工作示意图

Figure 2-4 Prompt learning working example diagram

提示学习在自然语言处理中的应用非常广泛，涵盖了文本生成、问题回答、分类与信息提取等多个领域。1.文本生成：通过提示引导模型生成富有创造力的文本，如文章、故事等。2.问题回答：使用明确的提问提示来获取针对性的回答，如基于给定背景内容回答具体问题。3.分类与信息提取：通过精心设计的提示，可以让模型基于输入文本提取特定信息或进行分类。为了适应于这些任务，提示学习有多种提示过程：

上下文学习（In-Context Learning）：通过在对话或输入上下文中为模型提供学习示例，模型能够识别任务模式，并直接在相似输入上模仿这些模式。这种技术在处理少样本学习和零样本学习任务时非常有效。

自动提示生成（Automated Prompt Generation）：利用搜索和优化算法自动生成适合特定任务的高质量提示，降低了人为干预的需求。例如，强化学习可以用于提示生成，通过搜索最优提示来引导模型产生高质量输出。

软提示（Soft Prompting）与提示微调（Prompt Tuning）：软提示通过直接对提示的嵌入向量进行训练，而不是使用普通的自然语言文本。这种方式可以显著减少对模型整体微调的依赖，同时提升性能。

链式提示（Chain-of-Thought Prompting）：在复杂推理任务中，通过逐步提示模型，可以让模型按照逻辑推理一步一步完成任务。这一应用提高了模型的推理深度，使得它能够应对更复杂的逻辑任务。

提示学习的主要优势在于其灵活性和高效性。通过简单地改变提示内容，用户可以有效地重新定义模型的任务。这种方法在低数据资源或无样本场景中尤为重要，使得大模型能够快速适应新任务，提升其实际应用的可能性。此外，提示学习还减少了对大量标注数据的依赖，降低了数据标注的成本。

尽管提示学习取得了显著进展，但仍面临一些挑战。例如，如何构建有效的提示、选择合适的语言模型、构建候选答案空间以及选择最佳的训练策略，仍然是需要持续探索的问题。未来的研究方向可能包括进一步优化提示生成算法、提高模型在复杂任务中的推理能力以及探索更高效的训练策略。

总之，提示学习作为一种新兴的自然语言处理技术，通过设计合适的提示来激活预训练语言模型的知识，使其能够理解和执行特定的任务。这种方法不仅能够充分利用预训练模型的强大能力，还能在资源有限的情况下取得较好的性能表现，具有广阔的应用前景。

2.4 对比学习

对比学习[49]（Contrastive Learning）是一种有效的学习方法，其核心目标是通过分析样本间的相似与差异，从而捕捉数据的内在特征表达。该方法的基本原理是促使相似样本在特征空间中彼此靠近，同时使不相似样本相互远离。这种方法受到人类视觉系统的启发，通过对比不同样本的特征，模型能够学习到数据的内在结构和语义信息。对比学习通过以下几个关键步骤实现其目标：

数据增强：对样本生成不同视角的增强版本，如图像的旋转、裁剪或颜色变换，或文本的同义词替换、句子重组等。这些增强版本作为正样本对，而不同样本之间的组合作为负样本对。

编码器网络：将增强后的样本通过编码器网络映射到特征空间，捕捉样本的高级语义特征。

投影网络：将编码后的特征进一步投影到低维空间，增强特征的判别能力。

损失函数：使用对比损失函数（如 InfoNCE 损失）优化样本间的相似性，确保正样本对的特征更接近，负样本对的特征更远离。

对比学习经常被用于图像分类、目标检测、语义分割等任务，通过学习图像的特征表示，模型能够更好地识别和区分不同物体。此外在句子相似度计算、文本分类、机器翻译等任务也能经常看到它的身影，模型通过对比学习能够捕捉文本的语义信息。

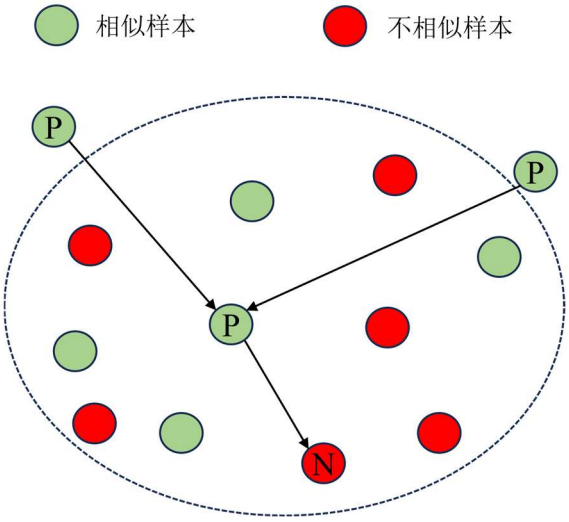

图 2-5 对比学习概念图

Figure 2-5 Contrast learning concept map

对比学习无需大量标记数据，能够在无监督或半监督场景下学习数据的特征表示，可以有效减少对大量标记数据的依赖。对比学习能够提高模型泛化能力，通过学习数据的内在结构和语义信息，对比学习能够提高模型在未见过数据上的泛化能力。此外，其还能增强特征表示，对比学习学习到的特征表示能够更好地捕捉数据的语义相似性和差异性，为下游任务提供高质量的特征。

对比学习作为一种强大的无监督学习方法，通过对比样本之间的相似性和差异性，能够学习到数据的内在结构和语义信息。其在计算机视觉、自然语言处理、推荐系统等多个领域展现了广泛的应用前景，并且随着研究的不断深入，对比学习有望在更多任务中发挥重要作用。

2.5 本章小结

本章分为四个部分，分别介绍了文本表征技术、自然语言预处理技术、提示学习技术和对比学习技术。本章第一部分介绍了传统的文本表征方法 TF-IDF，包括其公式实现及优缺点分析。接着，介绍了基于机器学习的 Word2vec 模型，涵盖连续词袋模型（CBOW）和 Skip-Gram 模型。最后，阐述了基于 Transformer 架构的 BERT 模型及其两种预训练任务：掩码语言模型（MLM）和

下一句预测（NSP）。第二部分介绍了自然语言预处理技术。鉴于本文的研究重点，本章仅对部分预处理步骤进行了介绍，包括文本清洗、分词、词干提取和词性标注。这些步骤旨在净化文本数据、提取关键特征，为后续分析任务提供高质量的数据支持。第三部分对提示学习技术进行了简述，具体来说介绍了提示学习的多种实现方式，包括上下文学习、自动提示生成、软提示和链式提示。这些方法通过设计特定的提示引导预训练语言模型完成任务，减少了对大规模标注数据的依赖，提升了模型的适应性和灵活性。本章第四部分介绍对比学习技术。其中包括对比学习的关键实现步骤，包括数据增强、编码器网络、投影网络和对比损失函数。对比学习通过比较样本的相似性和差异性学习数据的特征表示，广泛应用于计算机视觉、自然语言处理等领域，具有无需大量标注数据、提升模型泛化能力等优点。

第3章 问答模式下查询重构的 API 推荐方法

本章针对用户输入的自然语言查询语句中存在的经验差距问题，提出并实现了一种问答模式下查询重构的 API 推荐方法——REAPI。该方法通过引入 Stack Overflow（SO）作为第三方信息源，有效缓解了经验差距问题。具体而言，REAPI 利用 SO 构建了一个富含经验性知识的先验知识库。对于每个用户查询，REAPI 会基于该知识库匹配出三个语义相似的候选重构语句，并结合用户的显式反馈信息来确定最终的重构语句。此外，REAPI 通过计算重构语句与 SO 帖子标题、重构语句与 API 官方文档描述之间的语义相似度值，生成候选 API 推荐列表。最后，本章在仿真平台上对 REAPI 的性能进行了测试，并通过用户实例研究验证了 REAPI 在实际应用中的价值。

3.1 研究动机

通过对 SO 帖子的观察，本文发现许多提问者在描述查询意图时存在模糊不清甚至逻辑混乱的现象，这表明部分用户缺乏准确表达查询意图的能力。事实上，用户描述搜索意图的能力与其编程经验密切相关：经验丰富的程序员通常能够清晰地表达其需求，而新手程序员则往往难以找到准确且完整的搜索词来描述其意图，本文将此问题定义为经验差距。值得注意的是，频繁进行搜索的开发人员多为编程新手，他们在开发过程中往往需要花费大量时间在搜索环节上。尽管目前已有多种有效的 API 自动推荐方法，但这些方法高度依赖于用户表达意图的能力，因此在为新手程序员提供推荐时往往效果不佳。

为验证这一观点，本文从某大学一年级研究生中收集了相关问题数据。这些学生在完成毕业设计过程中遇到了诸多问题，本文选取了 18 名学生参与 REAPI 的评估，并从中挑选了四个典型问题来展示语句重构的有效性，如图 3-1 所示。例如，第五位新手程序员输入的查询为“How to determine if a Java Class implements a particular Interface.”。在该查询中，“How to decide if”除了“decide”外并无实际意义，属于冗余信息。已有的方法例如 BIKER[9]由于未对用户输入进行任何处理，无法识别并处理此类冗余信息，从而导致推荐错误的 API。相比之下，REAPI 对用户输入进行了重构，生成的重构语句“Check whether a class

implements an interface?”更加简洁明了。

在 API 查询过程中，开发人员可能使用不同的词汇、语法结构或表达方式，这些差异会导致基于文本匹配的推荐系统难以准确推荐正确的 API。REAPI 通过语句重构有效解决了这一问题，其在 API 推荐中的关键作用主要体现在以下几个方面：

（1） 语法结构改进

开发人员在撰写查询时，可能会使用不规则的语法或复杂的表达式，这可能导致推荐系统难以准确理解其意图。语句重构通过将这些查询重写为更标准化的形式，能够显著提升推荐系统的理解能力。例如，如图3-1所示，第三个新手程序员提出的问题与经过 REAPI 重构的问题在结构上存在差异，这种差异正是导致基线方法推荐失败的关键因素。通过语法结构的优化，语句重构能够有效弥合这种差异，从而提高推荐系统的准确性和效率。

（2）消除歧义

开发人员的查询中有时会包含模糊或歧义的表述，这使得推荐系统难以确定最合适的 API。语句重构能够通过语义分析和上下文理解，消除这些歧义，从而提供更清晰的查询意图。例如，第一个新手程序员的查询语句中包含过多冗余信息，经过重构后的句子能够更精准地表达其核心需求。这种重构不仅提高了查询的清晰度，还增强了推荐系统的响应能力，使其能够更准确地匹配和推荐相关 API。

（3）标准化词汇表

在描述相同概念或行为时，开发人员可能会使用不同的词汇。这种词汇多样性可能导致推荐系统在匹配过程中出现偏差。语句重构通过将这些不同的词汇统一为标准化的表示形式，能够显著提高查询与推荐结果之间的匹配准确性。标准化的词汇表不仅减少了因词汇差异导致的误解，还增强了推荐系统的鲁棒性和一致性。因此，语句重构在 API 推荐中发挥着至关重要的作用，能够有效弥合开发人员意图与查询表示之间的知识鸿沟，从而提升整个推荐系统的性能和用户体验。

此外，现有的研究方法大多聚焦于隐式信息的挖掘，通过问答网站等外部资源进行隐式信息的补充挖掘。尽管隐式信息具有一定的信息价值，但其表达往往不够明确，难以清晰地反映用户的实际意图。例如，用户观看某部电影的

行为，并不能直接推断出该用户对该类电影的偏好。因此，为了提升 API 推荐系统的性能，所以需要引入能够明确表达用户意图的显式信息，以增强推荐的准确性和可靠性。为了解决上述问题，本研究提出了一种新的方法，通过语句重构技术弥合开发人员的实际意图与查询表示之间的知识鸿沟。具体而言，本方法包含以下三个关键步骤：

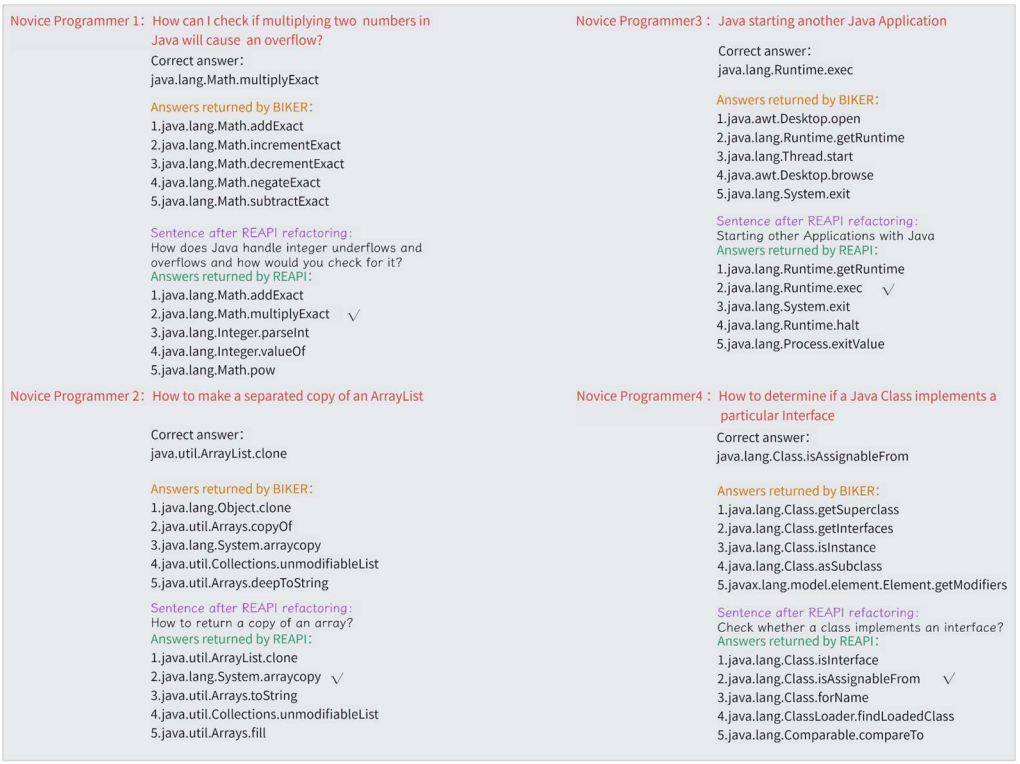

图 3-1 REAPI 重构查询示例图

Figure 3-1 REAPI Refactoring query example diagram

（1）语句重构以减少对初始查询的依赖

通过语句重构，重新描述用户的输入意图，从而降低推荐任务对用户初始查询描述的依赖。重构后的语句能够更准确地反映用户的真实需求，为推荐系统提供更清晰的输入。

（2）解决隐式意图的不可见性问题

本方法通过向用户展示三个重构后的变量描述语句，使用户能够直观地看到其查询背后隐含的意图。这一过程不仅增强了用户对查询意图的理解，还为用户提供了一个选择的机会，从而将隐式意图转化为显式意图。

（3）用户选择作为显式信息的引入

用户从提供的重构语句中选择最符合其意图的语句，这一选择过程为推荐

系统提供了明确的显式信息。具体而言，本方法首先构建一个高质量的问题库，从库中筛选出与用户原始查询语句语义最接近的三个问题作为重构语句的候选。然后，将这些候选重构语句呈现给用户，用户选择最能反映其需求的语句作为最终的重构语句，以此作为后续推荐任务的输入。

通过上述步骤，本方法不仅优化了查询语句的表达，有效的缓解了经验差距的问题，还通过用户的选择引入了显式信息，从而显著提升了 API 推荐系统的性能和用户体验。

3.2 模型设计

REAPI 的构建主要包含三个核心阶段：语言模型训练、查询显示化重构以及相关 API 推荐。其总体框架图如图 3-2 所示。

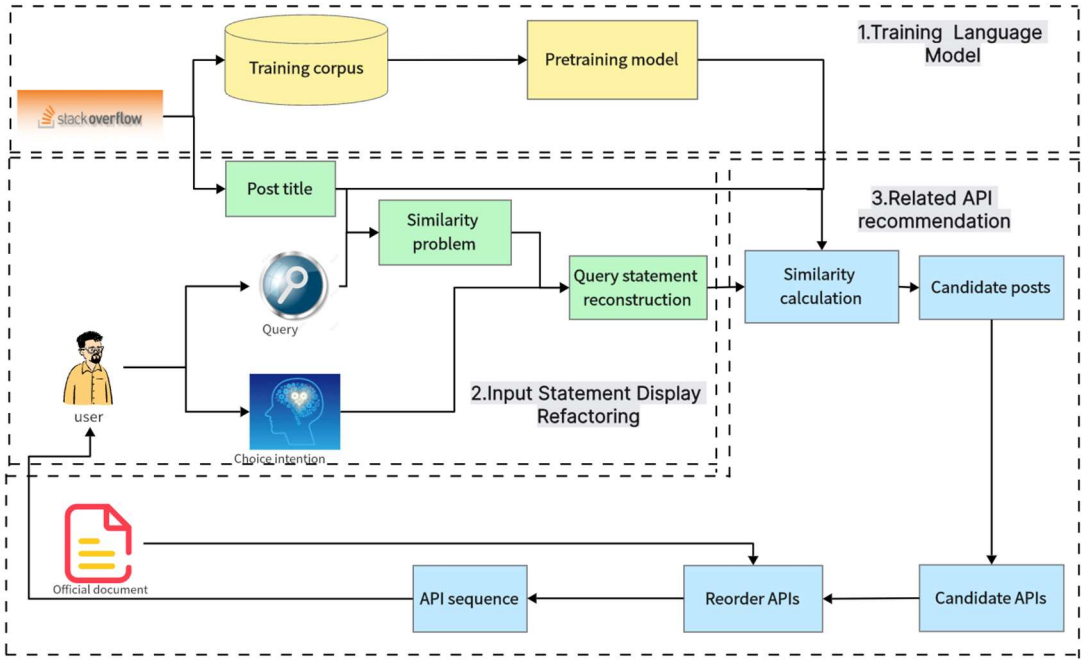

图 3-2 REAPI 的总体框架图

Figure 3-2 Overall framework diagram of REAPI

在第一阶段，即语言模型训练阶段，REAPI 通过 Word2Vec 模型训练词嵌入（Word Embedding），以捕捉词汇的语义信息。同时，利用逆文档频率（IDF）构建词权词汇表，从而量化词汇在语料库中的重要性。这一阶段的目标是为后续任务提供高质量的词汇表示和权重信息。

在第二阶段，即输入表示重构阶段，REAPI 首先基于已经构建好的重构知识库，计算其与用户自然语言查询的语义相似度，并筛选出相似度最高的三个

问题反馈给用户。用户从中选择最符合其意图的问题作为重构后的查询语句。该重构语句将作为后续推荐任务的核心输入，以确保查询的准确性和针对性。

在第三阶段，即相关 API 推荐阶段，REAPI 利用第二阶段生成的重构查询语句，计算其与 SO 帖子的语义相似度，生成 top-k 问题排序列表。这一步骤旨在缩小搜索范围，提升推荐效率。随后，将 top-k 问题对应的答案与 API 官方文档进行比对，筛选出最相关的 API 候选列表，从而为用户提供精准的 API 推荐。

通过上述三个阶段，系统能够从用户输入的自然语言查询中提取关键信息，逐步优化查询表示，并最终推荐符合用户需求的API，实现高效、准确的API推荐服务。

3.2.1 训练语言模型

1.构建语料库

在本节中，将详细介绍语料库的构建方法以及词向量模型的训练过程。语料库的构建是训练高质量词向量模型的基础，而词向量模型则是后续推荐任务的关键工具。为了训练词向量模型，首先需要构建一个高质量的语料库。本研究选择使用 Stack Overflow（SO）的帖子内容作为语料来源。SO 是一个广泛使用的编程问答社区，其中包含大量的编程问题和答案，这些内容为训练模型提供了丰富的语料。本文从 SO 的 HTML 页面中提取文本内容，并对其进行预处理。具体步骤如下：（1）提取文本内容：从HTML页面中提取问题和答案的文本内容。（2）筛选问题：仅保留带有“JAVA”标签的问题，因为本文的推荐任务主要聚焦于 Java API 推荐。（3）处理代码片段：由于 SO 中的代码片段长度不一，过长的代码片段会增加训练负担，因此本方法删除了包含长代码片段的帖子，仅保留包含短代码片段的帖子用于推荐代码片段，并使用 NLTK 对句子进行标记 [8]。

经过上述处理，最终构建了一个包含 1,347,908 对问答的语料库。该语料库将用于训练词嵌入模型，并构建逆文档频率（IDF）词汇表。

2.训练词嵌入模型

对于训练词嵌入模型，REAPI 选择了 Word2vec 模型。Word2vec 是一种基于神经网络的词嵌入模型，能够将每个单词映射到一个低维向量空间中，从而捕捉单词的语义信息。为了高效地实现 Word2vec 模型的训练，REAPI 使用了

Gensim 库，这是一个专门用于实现 Word2vec 的 Python 库。

词嵌入模型为测量词相似度提供了基础。在训练过程中，由于某些高频词（如“the”、“is”等）虽然在文本中频繁出现，但其语义信息相对较少。为了增强模型对重要单词的敏感度，REAPI 引入了逆文档频率（IDF）对预训练的单词进行加权。

逆文档频率（IDF）是信息检索领域中的一个重要概念，用于衡量一个词在文档集合中的重要性。在本研究中，单词的 IDF 值表示为包含该单词的 SO 帖子数的倒数。为了计算 IDF 词汇表，首先对每个单词进行词干化处理，即将单词转换为其原型。例如，单词“sequence”（名词）、“sequenced”（形容词）、“sequencer”（名词）、“sequencing”（名词）等，均用“sequence”这一原型的 IDF 值表示。

训练完成的词嵌入模型和 IDF 词汇表将用于嵌入 SO 帖子标题和 Java API 文档中的单词。由于模型包含的单词数量远远超过实验中使用的帖子标题和官方文档中的单词。因此，API 文档中的单词将直接与训练好的词嵌入模型和 IDF 词汇表一起使用，从而为后续的自然语言处理任务提供高质量的特征表示。

通过上述语料库构建和词向量模型训练过程，不仅为 Java API 推荐任务提供了丰富的语料支持，还通过 IDF 加权机制增强了模型对重要单词的敏感度，从而提升了词嵌入模型的质量和适用性。

3.2.2 查询显式化重构

在本阶段，REAPI 需要对输入的自然语言查询进行重构，使得重构后的查询更能表达用户的任务需求。首先需要从重构库中匹配出与查询语义相匹相似的帖子标题

这需要计算开发人员的自然语言描述和 SO 问题库之间的相似度，定义如下:

$$sim(W_{\mathrm{q}},W_{SO})=\cos(W_{\mathrm{q}},W_{SO})=\frac{W_{\mathrm{q}}^{T}W_{SO}}{\|W_{\mathrm{q}}\|\|W_{SO}\|} \tag{3-1}$$

其中，W_{q}和W_{SO}分别代表查询Q_q和语料库Q_{so}中的单词。使用余弦相似度来计算相似度。余弦相似度通过计算两个向量的内积并使用 L2 范数。为了计算句子之间的相似度，首先使用如下公式来度量句子之间的相似度：

$$sim(W_q, Q_{SO}) = \max_{W_{SO} \subseteq Q_{SO}} (W_q, Q_{SO}) \tag{3-2}$$

然后，需要计算该单词与句子中每个单词的最大相似度，并使用 IDF 词汇表对其进行加权，如下公式所示:

$$sim(Q_q \xrightarrow{sim} Q_{SO}) = \frac{\sum_{W_q \subseteq Q_q} sim(W_q, Q_{SO}) * IDF(W_q)}{\sum_{W_q \subseteq Q_q} IDF(W_q)} \tag{3-3}$$

$$sim(Q_{SO} \xrightarrow{sim} Q_q) = \frac{\sum_{W_{SO} \subseteq Q_{SO}} sim(W_{SO}, Q_q) * IDF(W_{SO})}{\sum_{W_{SO} \subseteq Q_{SO}} IDF(W_{SO})} \tag{3-4}$$

其中 $IDF(W_{SO})$和 $IDF(W_q)$ 分别表示 W_{SO} 和 W_q 两个词的 IDF 权重，将 Q_{SO} 和 Q_q 转换成两个词束，然后分别计算 Q_q 到 Q_{SO} 和 Q_{SO} 到 Q_q 的相似度得分。最后将这两个分数相加，计算出最终的相似度得分，定义如下:

$$sim(Q_{so}, Q_q) = \frac{1}{\frac{1}{sim(Q_q \xrightarrow{sim} Q_{so})} + \frac{1}{sim(Q_{so} \xrightarrow{sim} Q_q)}} \tag{3-5}$$

通过使用上述公式，可以计算出开发人员输入的语句与 SO 中的句子之间的相似度，最后将排名靠前的几个相似句子返回给他们。根据返回的句子，开发人员选择符合自身意愿的句子描述作为重构语句。查询重构算法的伪代码如表 3-1 所示。

表 3-1 查询重构算法

Table 3-1 Query reconstruction algorithm

算法 1: 输入查询显式化.

数据: 输入数据 Q

输出: 重构后的问题语句 Q'

1 语句分割 Q;

2 $Q\_matrix = init\_doc\_matrix(Q, W2V)$ //得到预训练嵌入矩阵;

3 $Q\_idf = init\_doc\_idf(Q, idf)$ //得到 IDF 权重矩阵;

4 $Sim\_q = get\_top\_questions(Q, Q\_matrix, Q\_idf, parent, q)$ //匹配相识问题;

5 $select = input()$;

6 **While** $select != null$ **do**

7 **Switch** $select$

8 **Case 1;** $reQ = Sim\_Q[0]$

9 **Case 2;** $reQ = Sim\_Q[1]$

10 **Case 3;** $reQ = Sim\_Q[2]$

11 **Otherwise;** $reQ = Q$

3.2.3 相关 API 推荐

在重构查询语句后，为缩小搜索范围并减少词汇差异，需进一步搜索相似问题的候选列表。为此，采用 3.2.2 节所述的相似度计算方法，分别设定候选列表长度为 15（方法级别）和 40（类级别），即从语料库中筛选出与目标问题相似度得分最高的前 15 个和前 40 个问题。因为利用这种方式可以剔除大量无关内容与噪声，避免引入过多不重要的信息，从而精准定位相关问题，提升推荐系统的性能。经人工调整验证，上述相似问题数量配置为最优解，可有效平衡搜索范围与推荐质量。

在获取 Stack Overflow（SO）候选列表后，随即开展 API 实体提取工作。API 实体特指 API 方法的全名。通过对 SO 候选列表中答案部分的解析，提取出 API 方法的全名，并将其作为候选 API 实体。在提取过程中，首先检查每个答案中的超链接，利用正则表达式识别指向官方 API 文档站点的超链接。例如，在 Java API 文档中，借助正则表达式从超链接中精准检测出相应的 API 实体（即 API 方法的全名），并将其标记为候选 API。正则表达式作为一种常用技术，可高效提取超链接内容，且在 Python 中可通过调用 re 包实现。然而，超链接内容可能与 API 无关或不包含 API 名称，因此需对提取结果进行进一步过滤。过滤后的候选 API 将用于后续的重新排序，并推荐给用户。

为确保提取的准确性，需构建一个字典，存储从官方文档中抓取的 API 名称。针对每个候选问题，对超链接中的文本内容执行 API 方法名称匹配操作。若文本与字典中的 API 方法名称匹配，则将其添加至新字典，并更新相应的出现次数。最终，依据出现次数与相似度，从候选 API 中筛选出出现频率最高且相似度最优的 API 类，作为推荐结果。若字典中无与文本内容匹配的 API，则将该候选问题过滤掉。

由于上一步生成的 API 序列可能受到多种人为干扰因素的影响，因此需要对序列进行重新排序，以返回 top-K 的 API 推荐结果。为此，采用相似度计算对 API 序列进行重排。具体而言，该相似度计算通过 API 描述与官方文档中查询的相似度值，以及查询与 Stack Overflow（SO）问题的相似度值的调和平均值来表征。

$$sim_{SO} = \min(1, \frac{\sum_{i=1}^{n} sim(Q_{SO}, Q_q^{'})}{n}) * (1 + \log_2 n) \tag{3-6}$$

使用(3-5)计算 $sim(Q_{SO}, Q_q^{'})$，表示 API 问题与查询的相似度。因为相似度的值不能大于 1，所以应该控制在 1 以内。下面的对数函数是为了防止包含这个 API 的问题太多，导致过度增长。sim_{doc} 也可以用式(3-5)计算，最后用于重新排列 API 的综合得分为它们的调和平均值，其公式定义如下：

$$sim_{total} = \frac{2 * sim_{SO} * sim_{doc}}{sim_{SO} + sim_{doc}} \tag{3-7}$$

3.3 实验设计与分析

3.3.1 数据集的设置与构建

为了验证 REAPI 的有效性，进一步的实验和评估是必要的。为此，本研究重用了 BIKER[4]的数据。由于 Stack Overflow（SO）作为一个开放式问答平台，其答案质量参差不齐，因此需要对数据进行严格筛选。首先，构建了一个富有经验知识的重构库，该库的筛选规则如下：1) 库中所有问题的评分需大于 0；2) 每个问题的所有答案中，至少有一个答案包含的 API 实体数量大于或等于 0，且该答案的评分需大于 0。该问题库的主要目的是支持第二阶段的查询重构。此外，为确保实验的准确性，本文还使用了包含问题和正确 API 的数据集，其中 API 作为基准真值。为确保 API 的正确性并避免答案 API 与问题不匹配的情况，本文采用了更为严格的筛选标准：

1) SO 题得分不低于 5 分；

2)这些问题的答案应包括 API 实体；

3)答题得分应大于 0 分。

除了上述筛选条件外，BIKER 的作者还提取了部分问题，并对其标题进行了人工检查，剔除了与编程任务无关的问题。最终，获得了 413 对高质量的问答数据。表 3-2 详细列出了这些测试对中答案 API 所属的库及其对应的 API 方法数量。此外，本研究还从 413 条数据中随机抽取了 5 组数据集，每组包含 20 条数据构建小型数据集来验证用户展示意图信息的有效性。

表 3-2 测试集回答 API 涵盖库范围情况表

Table 3-2 Test set answer API coverage library scope table

项目包	#方法	项目包	#方法
核心库			
java.lang	**271**	java.util	174
java.io	16	java.math	15
java.nio	22	java.net	14
java.awt	18	java.XML	1
非核心库			
Java.text	1	Java.time	6
Java.imageio	2	**总调用数量**	**540**

3.3.2 基线模型简介

本章选择了 BIKER[9]、RACK[50]和 CLEAR[11]作为对比基线。由于训练数据规模不足以充分发挥深度学习算法的性能，现有研究表明，检索算法在当前场景下的表现优于学习算法[51]。例如，BIKER 的性能显著优于 DeepAPI[52]，因此本文未将深度学习算法纳入对比范围。考虑到用户意图重构需要用户从三个相似问题中选择最符合其搜索意图的问题，当测试数据量较大时，这一过程将耗费大量时间。为降低实验复杂度，本文对模型进行了简化：分别基于前三个相似问题生成重构结果，得到三个基线变体 REAPIsim1、REAPIsim2 和 REAPIsim3，并将这些变体与 BIKER 和 RACK 基线进行对比。此外，为测试完整 REAPI 模型的性能，从数据集中随机抽取 20 条数据，邀请两位专家与原作者共同讨论，从三个最相似的问题中选择最符合搜索意图的问题进行重构。最后，本文还开发了一个在线版本，支持用户查询并推荐 API，API 描述及代码片段等信息，不但帮助开发任务进行开发任务，还能有助于新手程序学习该 API 的相关知识。

基线 1 (BIKER): BIKER 为推荐任务使用两个信息源。使用 SO 帖子中的问题作为另一个信息源来缩小词汇差异，最后使用从答案中提取的 API 实体进行推荐任务。本实验在两个细粒度级别（方法和类）上比较了 BIKER 和 REAPI 的性能。

基线 2 (RACK):RACK 使用 SO 中的问答信息构建关键字-API 映射对。RACK 只能推荐类级别的 API，所以只在类级别的细粒度基础上比较它。

基线 3 (CLEAR): CLEAR 利用 BERT 模型和对比学习技术来构建一个保留语

义相关序列信息的语言模型。

基线 4（$REAPI_{sim1}$、$REAPI_{sim2}$和$REAPI_{sim3}$）：在查询重构阶段，分别使用 top-1 相似度问题、top-2 相似度问题和 top-3 相似度问题进行语句重构。其主要目的是验证语句重构的有效性，减少人为的时间成本。

3.3.3 评估指标

为了全面且合理地评估模型性能，本文采用平均倒数秩（MRR）和平均平均精度（MAP）作为主要评价指标。这两种指标在过往研究[14][15][16][17]中被广泛应用。其中，MRR 用于衡量第一个正确 API 出现位置的倒数，其值越大，表明推荐模型的准确性越高；MAP 则综合考虑了所有正确答案的排序情况，MAP 值越大，推荐模型的准确性也越高。

此外，本文还引入了 S@K 指标来进一步评估 REAPI 的性能。S@K 表示返回的 API 列表中前 K 个位置出现正确 API 的概率。各评价指标的具体计算公式如下：

$$MRR = \frac{\sum_{i=1}^{R} \frac{1}{pos_i}}{R} \tag{3-8}$$

式中 pos_i 表示第 i 个问题第一个正确答案在 top-N 中的排名，R 表示所有查询问题的个数

$$MAP = \frac{1}{R}\sum_{i=1}^{R} \frac{\sum_{i=1}^{n} count(hit_i) \times rel(i)}{len(ture\_apis)} \tag{3-9}$$

其中 $count(hit_i)$ 表示 top-i 中正确 API 的数量。注意，$count(hit_i) \times rel(i)$ 只计算该位置的排名 i 与存在正确 API 时正确 API 的数量之间的商。因此 MAP 考虑所有正确 API 的排名：

$$S@K = \frac{\sum_{i=1}^{R} count(rank_i \le K)}{R} \tag{3-10}$$

其中 $rank_i \le K$ 表示推荐列表中前 K 个位置会出现正确的答案。

3.3.4 实验细节设置

本实验在一台搭载 4090GPU 与 32GB RAM 的计算机上进行。在研究过程中

发现，保留的相似问题数量对推荐性能具有显著影响。若保留的相似问题过少，可能导致信息不完整，关键信息缺失，从而影响模型性能；而保留过多的相似问题则会引入大量噪声数据，增加训练成本，进而降低模型性能。因此，在查询重构阶段，通过手动调整相似问题的数量，以确定最优值。

表 3-3 方法基别上的参数实验

Table 3-3 The parameter experiment on the formula base

相似问题数量	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
5	0.569	0.729	0.753	0.615	0.646
10	0.615	0.809	0.859	0.676	0.711
15	**0.634**	0.833	0.901	**0.701**	**0.735**
20	0.622	0.847	0.923	0.698	0.732
30	0.619	0.843	0.947	0.695	0.735
50	0.591	0.840	0.968	0.680	0.721
100	0.557	0.840	0.983	0.654	0.682
200	0.538	**0.852**	0.985	0.636	0.682
300	0.521	0.847	**0.987**	0.666	0.618
500	0.496	0.831	0.978	0.599	0.645

表 3-4 类基别上的参数实验

Table 3-4 Parameter experiments on class bases

相似问题数量	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
15	0.639	0.787	0.823	0.679	0.715
30	0.692	0.864	0.91	0.736	0.777
40	**0.702**	**0.882**	0.925	**0.747**	**0.788**
50	0.697	0.879	0.927	0.746	0.787
60	0.692	0.872	0.937	0.747	0.783
70	0.682	**0.884**	0.942	0.745	0.780
100	0.688	0.879	0.956	0.747	0.782
200	0.668	0.876	0.966	0.744	0.744
500	0.648	0.869	**0.978**	0.731	0.762
1000	0.639	0.864	0.975	0.721	0.753

通过手动调整并设置不同数量的相似问题，实验结果表明（见表 3-3），当保留的相似问题个数为 15 时，模型在方法层面的各项指标几乎均达到最优值；而类水平的最优值则在保留 40 个相似问题时才出现（见表 3-4）。分析两表数据可知，最初保留的相似问题较少时，重要 API 信息大量丢失，导致模型性能欠佳。随着保留的相似问题数量逐渐增加，推荐模型的准确率缓慢提升，当相似问题数量达到一定值时，模型性能达到最优。然而，当相似问题数量继续增加时，大量噪声数据的引入使得模型性能开始下降。

此外，当模型达到最佳性能后，S@3 和 S@10 的值仍随着相似问题数量的增加而继续上升。分析认为，尽管增加更多相似问题会包含更多有用的 API 信

息，但同时也会引入大量噪声，且噪声的比例远大于可用的 API 信息，从而导致 S@1 精度降低。S@3 和 S@10 具有较大的容错空间（top-k 中的容错空间为 k-1，例如 S@1 为 0，即第一个推荐必须是正确答案；S@10 的容错空间为 9，即前十个推荐列表中有一个正确答案即可视为成功，有 9 次犯错的机会），因此能够容忍更多的噪声，从而利用更多有用信息来提升其准确性。当然，当引入的噪声量超过其容错空间时，其性能也会受到影响。

从表中数据可以看出，当数据量继续增加并超出容错空间时，S@3 和 S@10 的性能逐渐下降。这一现象可以通过以下事实加以解释：S@3 比 S@10 更快地到达下降点，因为 S@10 比 S@3 具有更大的容错能力，故 S@3 率先超过其容错能力。由于用户通常更倾向于在推荐列表的最前面看到正确答案，因此本文更关注 S@1 的性能以及模型的整体性能。同时，还注意到，性能的提升是以增加更多时间为代价的。

在研究过程中，本文还发现了一个有趣的现象：类层在保留 40 个相似帖子时达到推荐性能的最优值，而方法层在仅保留 15 个相似帖子时即可获得最准确的推荐结果。对于这一现象，本文推测其原因可能与类级 API 和方法级 API 的特性有关。

类级 API 通常具有更高的抽象性，其语义的理解需要依赖更多的上下文信息。例如，java.util.List 是一个类级 API，它包含 add()、remove()和 get()等方法。要准确理解 List 类的语义，需要全面了解其用途和功能。然而，仅查看一个 Stack Overflow（SO）帖子往往难以提供足够的信息来完整理解 List 类的语义，因此需要更多的 SO 帖子来提供丰富的上下文信息，以帮助准确理解类级 API 的含义。

相比之下，方法 API 则更为具体，其语义的理解所需上下文信息相对较少。以 java.util.List.add()为例，该方法仅接受一个参数，其语义的理解主要依赖于对参数的了解以及该方法的具体作用。通常情况下，仅查看一个 SO 帖子即可获得足够的信息来理解 add()方法的语义，因此不需要更多的 SO 帖子来辅助理解。

基于以上分析，本文认为，在推荐类级 API 时，由于其抽象性较高，需要更多类似的 SO 帖子来提供丰富的上下文信息，以帮助准确理解其含义；而在推荐方法级 API 时，由于其具体性较强，仅需少量类似的 SO 帖子即可满足语义理

解的需求。这一发现为优化推荐系统的相似问题保留策略提供了有益的参考。

3.3.5 研究问题

为了更准确、全面地评价 REAPI 的性能，本章列出了以下研究问题并设计了实验。

RQ1: REAPI 在方法级细粒度上的表现如何？

RQ2: REAPI 在类级粒度上的的表现如何？

RQ3：语句重构如何影响 REAPI？

RQ4：用户显示意图信息的效果如何？

在 RQ1 和 RQ2 中，探讨了 REAPI 在两个不同粒度级别（类和方法）上的性能，并将 REAPI 与高级基线进行了比较，以展示其优点。此外，本实验还探讨了 RQ3 中语句重构的有效性。实验将未重构的模型与重构后的模型进行了比较，以验证其有效性。最后，本章还讨论了 RQ4 中用户显示意图的有效性。

3.3.6 实验结果及分析

RQ1: REAPI 在方法级细粒度上的表现如何？

表3-5展示了REAPI与其他基线方法的比较效果。从表中可以看出，REAPI在大多数评估指标上均优于基线方法。其中，基线方法 CLEAR 的表现优于BIKER。尽管 BIKER 通过词嵌入方法调用双信息源来解决词汇差异问题，但它无法有效捕获顺序语义信息。相比之下，CLEAR 利用新的嵌入模型 BERT 较好地解决了这一问题。然而，这两种基线方法均侧重于生成更准确的词向量表示，而忽视了用户查询语句本身可能存在的问题。

REAPI 通过重构技术生成更精确的表达式语句，从而在一定程度上缓解了这一问题。具体而言，REAPI 在 S@1 评估指标上较 BIKER 高出 32%，较 CLEAR 高出 17%。与 S@3 和 S@10 相比，S@1 在前 k 个指标上的改进更为显著。此外，REAPI 在 S@10 评估指标上较 CLEAR 出现了负向改善（-2%）。这一现象的原因在于，本文更关注前 1 名的成功率以及整体模型性能，而非前 10 名的成功率。通常情况下，用户期望推荐的正确答案位于列表的顶部，理想情况下，第一个位置即为正确答案。因此，在进行参数调优时，本文有意识地牺牲了 S@10 的性能，以优化 S@1 等关键指标。

此外，REAPI 在 MAP 和 MRR 指标上较 BIKER 分别提高了 35%和 29%，

较 CLEAR 分别提高了 8%和 12%。这一结果表明，REAPI 在排名靠前的位置能够推荐更准确的 API 方法，从而在方法粒度级别上实现了更好的性能。

综上所述，REAPI 在方法级细粒度上的推荐性能优于高级基线，特别是 top-1 上的推荐性能有显著提高。

表 3-5 REAPI 在方法级别上的表现

Table 3-5 REAPI performance at the method level

基线方法	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
BIKER	0.428	0.690	0.854	0.519	0.571
CLEAR	0.540	0.788	**0.919**	0.647	0.658
$REAPI_{sim1}$	0.521	0.736	0.821	0.603	0.636
$REAPI_{sim2}$	0.571	0.797	0.874	0.656	0.690
$REAPI_{sim3}$	**0.634**	**0.833**	0.901	**0.701**	**0.735**
Improve.BIKER	32%	21%	6%	35%	29%
Improve.CLEAR	17%	6%	-2%	8%	12%

RQ2: REAPI 在类级细粒度上的表现如何？

表 3-6 展示了 REAPI 在类级别上与其他基准方法的比较结果。分析表明，REAPI 在大多数评估标准上均优于基线方法。基线方法 RACK 的性能最低。RACK 技术利用不同 API 与 Stack Overflow 查询关键字的关联，将用于代码搜索的自然语言查询转换为一组相关 API。这种转换利用了使用特定 API 类和方法的众包知识，但不考虑查询和任务之间的差异。

在三个不同的 top-K 成功标准 S@1、S@3 和 S@10 上，REAPI 分别比 RACK 提高了 115%、72%和 17%。这表明 REAPI 在推荐成功率方面全面优于 RACK，特别是在排名靠前的推荐成功率上有显著提升。与 BIKER 相比，REAPI 在三个成功指标上分别提高了 28%、4%。与 CLEAR 相比，REAPI 在三个前 k 指标上分别提高了 14%、3%和。

值得注意的是，REAPI 在 S@10 上的性能改进为负值。如果需要提高 S@10 的性能，可以通过调优表 III 中所示的参数来实现。然而，由于本文更关注前 1 名的成功率以及整体模型性能，而非前 10 名的成功率，因此在参数调优时牺牲了 S@10 的性能。

在 MAP 和 MRR 指标上，REAPI 较 RACK 分别提高了 78%和 75%，较 BIKER 分别提高了 13%和 13%，较 CLEAR 分别提高了 4%和 8%。这些结果表明，REAPI 在类粒度级别上也实现了更好的性能，能够在排名靠前的位置推荐

更准确的 API。

综上所述，REAPI 的总体改进优于高级基线方法。为了提高整体模型性能和 top-1 性能，本实验在参数调优时牺牲了 S@10 的性能来换取更契合用户心意的 S@1 的性能。

表 3-6 REAPI 在类级别上的性能

Table 3-6 REAPI performance at the class level

基线方法	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
RACK	0.327	0.513	0.787	0.42	0.451
BIKER	0.547	0.850	**0.976**	0.661	0.695
CLEAR	0.617	0.857	0.969	0.721	0.733
$REAPI_{sim1}$	0.615	0.835	0.893	0.683	0.726
$REAPI_{sim2}$	0.634	0.843	0.906	0.711	0.742
$REAPI_{sim3}$	**0.702**	**0.882**	0.925	**0.747**	**0.788**
Improve.RACK	115%	72%	17%	78%	75%
Improve.BIKER	28%	4%	-5%	13%	13%
Improve.CLEAR	14%	3%	-7%	4%	8%

RQ3：语句重构如何影响 REAPI？

语句重构与不进行重构的对比在表 37 和 3-8 中清晰呈现。分析发现，无论采用何种语句进行重构，其性能均优于未进行重构的方法。具体而言，使用最低相似度的 REAPI-sim3 进行的重构，在大多数评价标准上均优于未进行重构的查询。例如，在方法层面，REAPI-sim3 在 S@1、S@3、MAP 和 MRR 指标上的性能分别提升了 22%、7%、16%和 11%。而重构效果最佳的 REAPI-sim1，其性能提升更为显著，分别达到了 48%、21%、6%、35%和 29%。

表 3-7 语句重构在类别上的影响

Table 3-7 Impact of statement reconstruction on category

基线方法	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
$REAPI_{none}$	0.547	0.850	**0.98**	0.661	0.695
$REAPI_{sim3}$	0.615	0.835	0.893	0.683	0.726
$REAPI_{sim2}$	0.634	0.843	0.906	0.711	0.742
$REAPI_{sim1}$	**0.702**	**0.882**	0.925	**0.747**	**0.788**
Improve	48%	21%	-6%	35%	29%

在类级别细粒度方面，经过重构的模型在大多数评价指标上同样表现出色，优于未进行重构的模型。这一结果表明，语句重构在推荐系统中发挥着至关重要的作用。重构时所使用的语句与原始查询的相似度越高，模型的准确率也越

高。这是因为更相似的语句能够更贴近地表达原始查询的语义信息，从而更好地捕捉用户的意图，进而提升推荐系统的整体性能。

表 3-8 语句重构在方法级别上的影响

Table 3-8 Impact of statement refactoring at the method level

基线方法	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
$REAPI_{none}$	0.428	0.690	0.854	0.519	0.571
$REAPI_{sim3}$	0.521	0.736	0.821	0.603	0.636
$REAPI_{sim2}$	0.571	0.797	0.874	0.656	0.690
$REAPI_{sim1}$	**0.634**	**0.833**	**0.901**	**0.701**	**0.735**
Improve	28%	4%	6%	13%	13%

RQ4：用户显示意图信息的效果如何？

现有的 API 推荐方法在挖掘隐式信息方面表现出色，隐式信息来源广泛且具有一定指向性。然而，隐式信息也存在诸多不足，可能对推荐性能产生负面影响。例如，隐式信息可能包含误导性内容，无法清晰地反映用户的实际意图。鉴于此，REAPI 引入了用户明确的选择意图信息，以验证显式信息对推荐性能的影响。具体而言，通过比较添加显式信息前后的推荐性能变化，来评估显式信息的作用。

结果如表3-9所示。表中前三行数据对应未添加显式意图信息的情况，而最后一行数据则对应添加显式意图信息的情况。分析发现，在大多数评价指标上，REAPI-user（添加显式意图信息）的性能均优于前三种未添加显式信息的方法。这一结果表明，显式信息在 REAPI 中发挥了积极作用，显著提升了推荐系统的性能。

表 3-9 显示信息对 REAPI 性能的影响

Table 3-9 Impact of display information on REAPI performance

基线方法	评估指标				
	S@1	**S@3**	**S@10**	**MAP**	**MRR**
$REAPI_{sim3}$	0.532	0.704	0.754	0.589	0.631
$REAPI_{sim2}$	**5.581**	**0.815**	0.856	0.654	0.689
$REAPI_{sim1}$	0.582	0.778	0.840	0.648	0.681
$REAPI_{user}$	**0.595**	0.791	**0.864**	**0.659**	**0.694**
Improve	12%	-3%	15%	12%	10%

3.4 用户实例研究

3.4.1 实验设计

参与者：由于该方法主要面向新手程序员设计，因此需要编程初学者协助评估模型的有效性。编程新手被定义为处于学习编程语言阶段但尚未具备独立开发软件系统能力的程序员。为此，本实验随机选取了一所大学的四年级学生作为样本群体，该群体具备一定的编程基础但实践经验相对有限。最终选取了 18 名学生参与研究，其中仅 1 名学生拥有一年以上的编程经验，其余大部分学生的 JAVA 编程经验均不足一年，具体分布如表 3-10 所示。

表 3-10 参与人员编程经验年限表

Table 3-10 Table of participants' years of programming experience

编程年限	1 年以内	1 年以上
	17	1
社会背景	大四学生	其他
	18	0

实验问题与答案 API：首先，收集了学生在毕业设计中遇到的相关问题，随后通过人工筛选去除与 API 无关的问题，保留有效问题。鉴于即使经验丰富的程序员也难以记忆所有 API，为确保 API 的真实性，将剩余问题与测试集中的相似问题进行匹配，并由领域专家判断与剩余问题目标一致的相似问题。通过这一过程，收集了 9 个实验问题及其对应的真实 API。为避免团队背景对实验结果的影响，本实验从测试集中随机选取另外 9 个问答对，由团队成员进行交叉验证。最终，共收集了 18 个实验问题及其对应的真实 API，如图 3-3 所示。

ID	用户提问	答案
Q1	What does it look like to write a Java program that splits a file name into a base part and an extension part?	java.lang.String.split
Q2	How do I add new system properties in Java?	java.lang.String.matches
Q3	How do I find Spaces in a string?	java.lang.System.setProperty
Q4	How do I get the MAC address of my local machine in Java?	java.lang.String.indexOf
Q5	How do I find before and after substrings in a string?	java.lang.System.nanoTime
Q6	How to calculate the runtime of a Java program?	java.awt.MouseEvent.getPoint
Q7	How do I get the position of the mouse relative to the panel	java.net.NetworkInterface
Q8	Given a string, how do I take only the first X letters?	java.lang.String.substring
Q9	How do I get my Android app to generate random numbers?	java.util.Random.nextInt
Q10	How to convert binary string value to decimal?	java.lang.Integer.toString
Q11	How can I use html styles in strings with format arguments?	java.lang.String.format
Q12	Converting a sentence string to a string array of words in Java	java.lang.String.split
Q13	How can I get a Unicode character's code?	java.lang.Character.codePointAt
Q14	How to timeout a thread	java.lang.Thread.interrupted
Q15	How to modify a Collection while iterating using for-each loop without ConcurrentModificationException?	java.lang.Thread.join
Q16	How to hide cursor in a Swing application?	java.awt.createCustomCursor
Q17	How to generate a random permutation in Java?	java.util.Collections.shuffle
Q18	Java generating Strings with placeholders	java.lang.String.format

图 3-3 用户实例实验所用的问题与其对应的答案

Figure 3-3 Questions used in the user example experiment with their corresponding answers

实验设计：将 18 名学生均分为三组，分组方式如下：1) 第一组学生仅依赖网络资源搜索合适的 API 方法；2) 第二组学生使用 BIKER 方法搜索 API；3) 第三组学生采用 REAPI 方法搜索合适的 API。由于 RACK 无法推荐方法级 API，且 CLEAR 未提供在线提问工具，因此未对这两种方法进行评估。若参与者认为所提供的工具无法解决问题，仍可通过互联网搜索资源。实验记录了每名学生完成每个问题所需的时间，最终以中位数表示完成时间，以避免异常值的影响，例如已预先了解 API 的学生或因网络故障等临时原因中断答题的学生，这些情况可能导致时间记录过短或过长。

为消除不同团队背景对实验结果的潜在影响，首先使用前 9 个问题进行实验，随后交换第二组和第三组的成员，利用剩余的 9 个问题重新进行实验。仅交换这两组成员的原因如下：1) 使用自动化 API 推荐工具的效率普遍高于网络引擎检索，后续实验结果也验证了这一点；2) 这两组成员均来自使用自动化 API 推荐工具的组，其间的切换可避免因对实验工具不熟悉而影响结果。

3.4.2 结果分析

本实验采用两个指标对用户研究结果进行分析：

正确性：用于评估参与者是否能够找到正确的 API。若参与者提交的 API 正确，则正确性记为 1；若提交错误，则记为 0。对于每个问题，若一组 6 名参与者中有 4 人及以上答对，则认为该组答对该问题，否则记为 0。该设计旨在避免异常值对结果的影响。

完成时间：该指标用于评估参与者解决问题所花费的时间。以每组参与者回答时间的中位数作为该组完成时间的代表值。

最终实验结果如表 3-11 与 3-12 所示。

表 3-11 用户实例研究结果表

Table 3-11 Table of results of the user case study

评估	分组	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	平均
正确性	网络引擎搜索	0	1	0	1	1	1	0	1	0	0.56
	BIKER	1	0	0	1	0	1	1	1	0	0.56
	REAPI	1	0	1	1	1	1	1	1	0	0.78
耗时性	网络引擎搜索	332s	216s	162s	74s	239s	148s	182s	266s	65s	187.1s
	BIKER	265s	192s	95s	56s	220s	96s	99s	130s	152s	145s
	REAPI	222s	54s	235s	176s	89s	83s	54s	229s	54s	132.9s

表 3-12 交换组员实验结果表

Table 3 -12 Table of Experimental Results for Swap Crews

评估	分组	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	平均
正确性	BIKER	1	1	0	1	0	1	0	1	1	0.67
	REAPI	1	1	0	1	0	1	1	1	1	0.78
耗时性	BIKER	144s	165s	171s	77s	146s	71s	167s	67s	85s	121.4s
	REAPI	128s	163s	184s	97s	144s	74s	64s	71s	64s	109.9s

通过表 3-11 可以看出，本方法相较于直接使用 BIKER 工具或网络资源搜索，在准确率上表现更优，同时耗时更低。值得注意的是，使用 REAPI 工具的小组在九个问题中仍有两个问题回答错误。将这两个错误答案对应的原始查询重新输入 REAPI 工具后，发现工具能够为这两个问题推荐正确答案（一个位于推荐列表的第二位，另一个位于第三位）。对于 BIKER 组回答错误的问题，将原始查询重新输入 BIKER 工具后，仅最后一个问题未能推荐正确的 API。这表明新手程序员在使用编程辅助工具方面经验不足，若熟悉这些工具，其表现可能优于直接使用网络资源搜索。此外，在交换第 2 组和第 3 组成员后，REAPI 在耗时和准确率方面仍优于 BIKER，如表 3-12 所示。

通过对参与者的反馈分析，发现 REAPI 在大多数情况下能够将正确答案推荐至 top-1 位置，而 BIKER 的正确答案排名较低或未出现在推荐列表中，这显著增加了筛选时间。实验结果表明 REAPI 能够有效的帮助编程新手进行开发任务，无论在推荐结果的准确性上还是在耗时上都取体现了其优越性。

3.4.3 用户反馈

在所有参与者完成问题回答后，本实验设计了一份问卷，以收集 12 名使用 REAPI 工具的参与者对工具的反馈。问卷包含三个问题：

1. 重构的语句是否对您有帮助？
2. 附加信息是否对您有帮助？
3. REAPI 是否对您有帮助？

对于第一个问题，9 名新手程序员表现出积极态度，他们认为一些重新表述的语句比原始实验问题更专业且更简洁。然而，有 3 名参与者指出，对于某些问题，重新表述的陈述可能改变了原始实验问题的语义。确实，REAPI 重新表述的问题可能会扭曲原始查询的意图，但这种情况较为罕见。此外，用户可以

选择不使用重新表述的语句，而是继续使用原始查询进行推荐任务。

对于第二个问题，10 名参与者表现出积极态度。部分参与者认为，根据推荐的文档信息可以快速确定 API 的功能，从而节省了查询 API 功能的时间；另一些参与者则认为代码片段有助于他们学习如何使用 API。然而，也有 2 名参与者认为代码片段过于简短，无法提供更全面的内容，甚至在某些问题中未提供任何代码片段。实际上，REAPI 有时无法为某些 API 提供代码片段，原因在于：在筛选 top-k 个相似问题时，这些问题的答案可能本身缺少该 API 的代码片段，或者在删除过长的代码片段时将其遗漏。未来可以通过构建从 API 到相应代码片段的映射数据库来解决这一问题。

对于第三个问题，所有参与者均表现出积极态度。值得注意的是，即使有 1 名参与者对前两个问题持消极态度，但他仍然认为 REAPI 是有帮助的。经过进一步询问，该参与者表示，REAPI 作为 API 推荐工具，能够帮助他快速找到合适的 API，相比直接在 Web 资源上查询，节省了大量时间。

3.5 本章小结

本章对问答模式下基于查询重构的 API 推荐方法进行了详细的介绍，首先本章对研究问题与目标进行了介绍，解释了为什么设计该研究方法，该方法的作用等。随后本章对模型 REAPI 的设计进行了详细的介绍，包括语言模型的训练，相关 API 推荐与最重要的步骤—查询显示化重构。接着，本章介绍了实验如何设计与对实验结果进行了分析。实验设计包括数据集的设置与构建、基线模型的介绍、评估指标的介绍、实验中的一些细节设置，其中包括实验的参数设置以及为什么使用该参数，参数变化影响实验结果的原因等。此外，本章设置了 4 个研究问题来全面测评 REAPI 的性能，其中包含两个细粒度级别上的对比试验、查询重构模块的消融实验与用户显示反馈信息的作用的实验。实验结果证明了 REAPI 由于最先进的基线方法，并且证明了查询重构模块对 REAPI 起着至关重要的作用。最后，本章还设计了用户实例研究来证明 REAPI 的现实价值，详细介绍了用户实例研究的设计、实验结果的分析与用户反馈的搜集等等。研究结果证实了 REAPI 的现实价值，能够有效的帮助编程新手进行开发任务，无论在推荐结果的准确性和开发耗时上都体现出其优越性。

第4章 问答模式下排序优化的 API 推荐方法

本章针对用户输入的自然语言查询与 API 官方文档间的描述存在的知识差距的问题，提出了一种问答模式下排序优化的 API 推荐方法。该方法通过引入与 API 官方文档描述同质的注释信息重排 API 推荐候选列表，有效的弥合了知识差异问题，提升了推荐结果的准确性。具体来说，本章首先根据问题回答中的 API 与注释的代码片段中的中的 API 的重叠度（重叠度大于 0.75 则认为相似）得到语义相似的<查询问题，正注释>对， 加入重叠度最低的注释组成三元组<查询问题，正注释，负注释>。本文利用该三元组使用对比学习训练 BERT 模型，训练后的 BERT 模型能有效捕获与查询问题语义相关的注释信息。所以，当对于用户输入的查询问题，推荐系统会返回一个 API 候选推荐列表时，本方法会利用训练好的 BERT 模型为查询问题匹配一个语义相似的注释，利用注释与 API 文档描述，查询问题与 API 文档描述的相似度值相结合来对 API 候选推荐列表重排。

4.1 研究动机

开发人员用于表达自身搜素任务需求的自然语言查询的描述与 API 官方文档间的描述存在知识差异的现象。这些差异主要表现在用户意图描述和官方文档之间的词汇不匹配。例如，考虑用户查询 "How to initialize all values in an array to false?" 对应的 API 为 Array.fill()，该 API 有一个官方文档描述：“"Assigns the specified boolean value to each element of the specified array of booleans。显而易见，用户查询与官方文档描述都是对 API—Array.fill()的描述，但是它们几乎没有共享任何重要的词汇，这两个描述间的相似度极低。遇见这种情况时，基于检索的 API 推荐方法会无法推荐出正确的 API。

本章将这种用户意图和官方文档之间的差异称为知识差距。知识差距的产生主要是因为用户意图描述往往是非结构化的自然语言，专注于概念或目标，而官方文档通常是结构化的，强调 API 的功能和结构。这些不同的描述可以被认为是异构数据。虽然现有的方法认识到查询和文档之间存在知识差距，并试图通过合并堆栈溢出（Stack Overflow， SO）信息来弥合这些差距。然而，在

推荐相关 API 的阶段，仍然存在意图描述和文档描述之间的知识鸿沟问题。因为再次阶段，已有的方法都是根据 SO 问题与 API 文档间的相似度来排序 API 候选推荐列表，当时 SO 中的问题与用户的自然语言提问本质没有区别，他们都是描述 API 的概念与目的的同构数据。这些方法没有从根本上解决知识差异的问题。

因此本文提出了一种排名优化的 API 推荐方法来解决知识差异的问题。该方法的目标是优化推荐的 API 序列，以减轻知识差距的问题。具体来说，该方法通过利用官方文档描述和代码注释信息来构建排名优化器。官方文档描述和代码注释都是同质的信息，阐明了 API 的功能和结构。利用同质信息对 API 序列进行重新排列，可以有效地解决知识差异问题。因此，当对于用户输入的查询问题，推荐系统会返回一个 API 候选推荐列表时，本方法会为查询问题匹配一个语义相似的注释，利用注释与 API 文档描述，查询问题与 API 文档描述的相似度值相结合来对 API 候选推荐列表重排。

4.2 模型设计

问答模式下排序优化的 API 推荐方法由两个关键模块组成，最终将输入查询转换为适合其特定需求的 API。阐明每个模块的功能以及它们如何相互作用以最终生成推荐的 API 是必要的。这两个模块分别是：训练语言模型模块与候选序列优化模块。

训练语言模型：在语言模型的训练过程中，本章采用 RoBERTa[53]为基本的骨架，采用对比学习的方式对其进行联合微调，微调后的模型可以更加有效的捕获与用户查询相似的代码注释信息。

候选序列优化：用户输入自然语言查询给推荐模型后，推荐模型会理解查询的语义信息并给出答案 API 的候选列表。本方法会利用微调后的 RoBERTa 模型得到语义一致的注释。最后根据描述候选 API 的功能和结构的注释信息与候选 API 相应的官方文档描述的相似度，来对候选 API 列表重新排序。

4.2.1 训练语言模型

本文利用 Github 中的注释构建三元组用于训练模型—注释三元组 $(Q,P,N)_{ann}$。其中，Q 表示用户输入的查询语句，P 表示与查询语句具有相同语义信息的代码注释，N 表示与查询没有语义相关性的代码注释。为了识别语义

上相似的注释和帖子，本文通过计算 Github 中注释对应的代码片段中包含的 API 与 SO 答案中提到的 API 之间的重叠比例，并假设重叠比例大于 0.75 的帖子在语义上与注释相似。最终将类似的注释视为积极的样本帖子。在构造三元组时，当找到所有的正样本注释后，选择重叠率最小的注释作为负样本帖子。最终得到查询、正样本注释和负样本注释，形成注释三人组进行对比训练。

例如，如图 4-1 所示的一个问题：“Could the assignment value of a variable of double type be different from the print out value of that variable in Java?”，其答案中包含的 API 序列为：Double.compare, String.equals, Double.parseDouble.

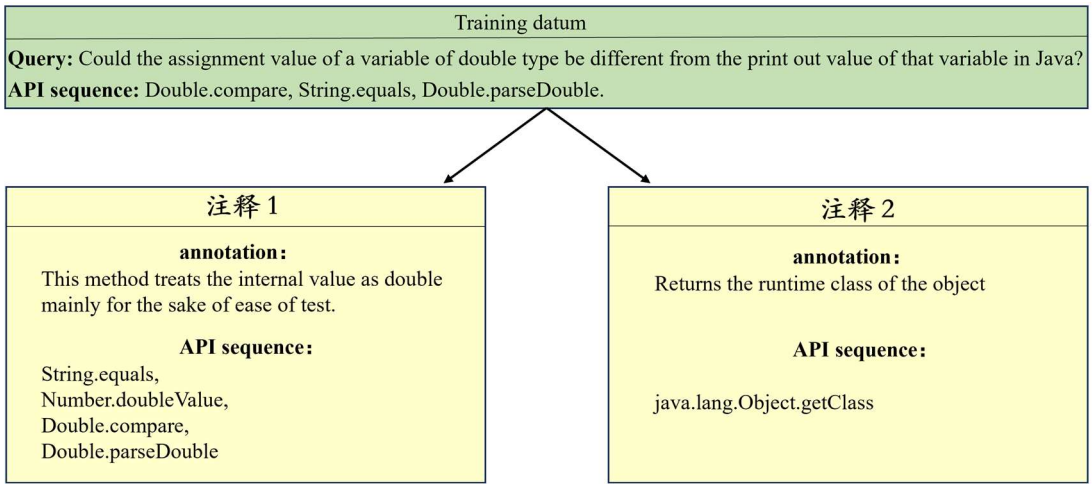

图 4-1 正负样本选择示例图

Fig. 4-1 Example diagram of positive and negative sample selection

现在，有两个注释，即注释 1：“This method treats the internal value as double mainly for the sake of ease of test”代码片段中包含的 API： String.equals、Number.doubleValue,、Double.compare,与 Double.parseDouble. 注释 2: “Returns the runtime class of the object” 其代码片段中包含的 API 为：: java.lang.Object.getClass。

可以观察到，与问题相对应的 API 序列中的 3 个 API，都可以在注释 1 中找到，并且重叠率为 1。而问题和注释之间的重叠率为 0。本文将重叠率设置为 0.75 作为阈值，这意味着如果注释的代码片段中的 API 与问题的答案中 API 的重叠率的相似度大于 0.75，则为正样本，如果低于此阈值，即为负样本。因为这个 0.75 的阈值可以捕获相关性相对较高的注释，同时确保有足够的注释作为阴性样本。因此，在上述示例中，注释 1 将被视为正样本注释，而注释 2 将被认为是负样本注释。在构建三元组时，该方法发现所有正样本注释后，将重叠率最小的注释作为负样本注释。最终，本方法将获得问题、正样本注释和负样

本注释，以形成注释三元组。最后，本章利用得到的注释三元组微调 RoBERTa 模型。训练目标是微调网络，使问题 Q 和正样本 P 之间的距离小于负样本 N 之间的距离。训练后的模型用于为每个用户查询输入匹配语义相似的代码注释。

4.2.2 候选排序优化

该阶段的核心目标是对推荐模型生成的候选 API 序列进行重新排序，以优化最终推荐结果。本方法采用了一种灵活的框架模型设计，其特点在于与底层推荐模型解耦，能够在不干扰推荐模型正常工作的情况下，对推荐结果进行后处理优化。这种设计使得该方法具备高度的通用性和可扩展性，能够与任何现有的 API 推荐模型无缝结合。此外通过引入与 API 描述文档同质的注释信息拟合了知识差异的问题，从而保证可以有效的提升推荐系统的整体性能。

具体而言，在这个阶段，本方法需要构建一个排名优化器来重新排序候选 API 列表，目的是提高正确 API 的排名。首先，将第三节 C 部分中获得的注释信息用作排名优化器的数据。该排名优化器在注释和对应于候选 API 列表中的 API 的官方文档描述之间进行相似性计算，从而获得每个 API 官方文档的注释 Q_A 和描述 Q_{doc} 之间的相似性得分 S_{A-doc} 。

$$S_{A-doc}= cos(E(Q_A), E(Q_{doc})) \tag{4.1}$$

其中，E(-) 是嵌入表示，Q_A 表示用户查询，Q_{doc} 表示 API 文档。同理，本阶段还对用户查询和 API 官方文档注释间进行相似性计算，获得查询 Q 和相关文档 Q_{doc} 之间的相似性得分 S_{Q-doc} 。最终，该排名优化器将 S_{A-doc} 和 S_{Q-doc} 的调和均值作为排名优化器的最终输出 S，具体公式如下：

$$S = \frac{2 \times S_{A-doc} \times S_{Q-doc}}{S_{A-doc} + S_{Q-doc}} \tag{4.2}$$

对于每个候选 API 列表，排序优化器为其中每个 API 生成评分 S，然后，会根据该分数对候选 API 列表进行重新排序，并最终向用户推荐重新排序的 API 序列。鉴于 GitHub 注释和官方文档都是描述 API 功能和结构的同质信息，使用注释对候选 API 列表进行重新排序可以显著缓解知识差异问题，最终达到优化正确 API 排名的目的。

4.3 实验设计与分析

4.3.1 数据集的设置与构建

为了采用对比学习训练 BERT 模型，除了用户的提问问题外需要保证有足够的注释来构建注释三元组。用户的提问本文用第 3 章中介绍的 SO 帖子标题来代替。而注释数据集来源于 Irsan 等人[12]的研究，该数据集基于 Martin 等人[52]的公开数据集构建。 它由注释和 API 序列对组成，包含 196,276 对注释和 API 序列，用于后续的模型训练与验证。

为了评估本章所提方法的性能，本研究采用了 BIKER 提供的测试数据集。该测试数据集经过严格筛选，筛选方法如下：

1.筛选出 Stack Overflow（SO）问题中得分大于

2.筛选出的问题答案中必须包含 API 实体，且答案得分大于 0。

此外，BIKER 的作者还对这些问题进行了人工检查，排除了与非编程任务搜索相关的问题。通过上述筛选方法，测试数据集能够有效反映编程场景下 API 推荐的实际需求，为模型性能评估提供了高质量的基准。

4.3.2 基线模型简介

本章使用 3 种先进的 API 推荐方法作为基线方法。其中包括一种深度学习方法与两种检索的方法。这 3 种方法的具体描述如下：

DeepAPI[52]：将为给定的自然语言查询生成 API 使用序列的任务建模为机器翻译问题。它采用循环神经网络 (RNN) 编码器-解码器神经网络模型从用户的自然语言查询中学习单词序列。DeepAPI 将单词序列编码为固定长度 7 上下文向量，并根据该向量生成相关的 API 序列。

BIKER[9]：使用 StackOverflow 和 Java JDK 官方文档进行推荐任务。它使用经过训练的 CBOW[46]模型和 TF-IDF 文档来查找与给定输入相关的候选 SO 帖子，并使用 Java JDK 官方文档描述与输入的相似度得分来细化最终答案。

BRAID[10]：利用用户的反馈信息来提升 API 推荐的性能。此外，还加入了主动学习模块来解决冷启动问题。

4.3.3 评估指标

为了科学、全面地评估模型的性能，本章采用了信息检索领域中广泛使用的评估指标，包括平均准确率（MAP, Mean Average Precision）、平均倒数排名

（MRR, Mean Reciprocal Rank）以及 SuccessRank@K。这些指标从不同角度衡量了推荐结果的准确性和排名质量，能够为模型性能提供多维度的评估依据。MAP 是一种基于排序质量的评估指标，其核心思想是综合考虑所有正确答案在推荐结果列表中的排名情况。具体而言，MAP 计算每个查询的平均准确率（AP），然后对所有查询的 AP 值取平均。MAP 的值域为 [0, 1]，其值越大，表明模型在整体排名性能上表现越好。MAP 的优势在于能够反映模型对多个正确答案的识别和排序能力，适用于需要返回多个相关结果的场景。MRR 是一种基于单答案排名的评估指标，重点关注推荐结果列表中第一个正确答案的位置。具体计算方法为：对每个查询，取第一个正确答案的排名的倒数，然后对所有查询的结果取平均。MRR 的值域为 [0, 1]，其值越大，表明正确答案在推荐列表中的排名越靠前。MRR 的优势在于能够直观反映模型对最相关结果的识别能力。SuccessRank@K 是一种更为细致的评估指标，用于衡量正确答案是否出现在推荐结果列表的前 K 个位置。具体而言，SuccessRank@K 关注正确答案在推荐列表中的第一次出现位置，若其排名位于前 K 位，则视为成功。该指标的值域为 [0, 1]，其值越大，表明模型在前 K 个结果中返回正确答案的能力越强。SuccessRank@K 的优势在于能够直接反映模型在实际应用中的实用性，尤其适用于需要快速返回高质量结果的场景。

通过结合 MAP、MRR 和 SuccessRank@K 三个指标，本方法能够从多个维度全面评估模型的性能，MAP 反映了模型对多个相关结果的综合排序能力。MRR 反映了模型对最相关结果的快速定位能力。SuccessRank@K 反映了模型在实际应用场景中的实用性和鲁棒性。本章采用的评估指标体系（MAP、MRR 和 SuccessRank@K）能够从不同角度全面衡量模型的性能，为推荐系统的优化和改进提供了科学依据。

4.3.4 实验细节设置

在本研究中，为了评估基于序列优化的 API 推荐方法的性能，本文选择了三种高级 API 推荐方法作为基线，分别是 DeepAPI，BIKER 和 CLEAR。这些基线方法的实现均基于其作者公布的代码或由其他研究人员复制的版本。对于 MSEAPIRec，本文通过参数检验来检查其与各种 API 推荐基准测试在性能上的统计显著差异。

在构造带注释的三元组时，本章计算带注释的代码片段中的 API 与 SO 帖子答案中的 API 的重叠率，以判断注释和 SO 帖子的语义是否相似。这种处理方法可能会引入噪音。本章将重叠率的阈值设置为 0.75，在确保有足够训练数据的情况下，尽量避免噪声带来的影响。

在统计检验方面，本文采用了 Wilcoxon 符号秩检验。 Wilcoxon 符号秩检验是一种非参数检验方法，已广泛应用于众多软件工程研究-。 该检验方法的优点在于不要求数据遵循任何特定的分布，因此适用于本研究中 API 推荐性能的评估。 在检验中，p 值小于 0.05 被认定为两个基线之间的性能差异具有统计显著性。

为了确保实验的顺利进行并获得可靠的性能评估结果，本文在两台配备 4090 GPU 的服务器上完成了所有部署。 这种硬件配置为模型训练和评估提供了强大的计算支持，确保了实验的高效性和结果的准确性。

4.3.5 研究问题

为了更准确、全面地评价 REAPI 的性能，本章列出了以下研究问题并设计了实验。

RQ1: 该方法在方法级与类级别细粒度上的表现如何？

RQ2: 序列优化模块是否有用？

4.3.6 实验结果分析

RQ1: 该方法在方法级与类级别细粒度上的表现如何？

为了全面评估基于序列优化的 API 推荐方法的性能，本文在两种细粒度级别（方法级别和类级别）上对其进行了系统性测试。测试过程中，本文选取了三种先进的基线方法（BIKER、BRAID 和 DeepAPI）作为对比对象，并在相同的数据集上进行了实验，以确保实验的公平性和可重复性。基线方法的实验复现均基于其作者公布的代码包，从而保证了实验结果的可信度。

实验结果如表 4-1 所示，对比分析表明，本章所提方法在绝大多数情况下均优于基线方法，充分证明了基于序列优化的 API 推荐方法的优越性。进一步分析表 4-1 中的数据可以发现。

DeepAPI 的性能相对较差，其性能落后于其他基线方法的主要原因在于，DeepAPI 作为一种深度学习方法，对训练数据的规模和质量有较高要求。然而，

收集足够的高质量训练数据在实际应用中具有较大挑战性，这限制了其性能表现。BRAID 的性能优于 BIKER，CLEAR 采用了 BERT 框架，能够有效捕获序列语义信息并表示句子级特征，而 BIKER 则忽略了这一关键点，导致其性能相对较弱。所提方法的性能优势，尽管 BRAID 在基线方法中表现最佳，但其仍然受限于原始查询本身的局限性，例如语义信息不足和知识差异等问题。本章所提方法通过引入序列优化机制，有效缓解了这些问题，从而实现了更准确的推荐性能。

实验结果表明，本章所提的方法由于先进的基线模型方法，在 S@1 评估指标上，相比 BIKER 实现了 36%的改进，相比 BRAID 实现了 23.6%的改进。在 MAP 与 MRR 等多种评估指标上都实现了巨大的性能提升。

本章所提方法在方法级别和类级别上均显著优于基线方法，尤其是在处理语义信息不足和数据存在知识差异时方面表现突出。这一结果验证了基于序列优化的 API 推荐方法的有效性和优越性，为 API 推荐领域的研究和实践提供了新的思路和技术支持。

表 4-1 基于序列优化算法的实验性能表

Table 4-1 Experimental performance table based on sequence optimization algorithm

基线	方法级别					类级别				
	S@1	S@3	S@5	MAP	MRR	S@1	S@3	S@5	MAP	MRR
DeepAPI	0.013	0.016	0.023	0.010	0.031	0.060	0.101	0.122	0.081	0.092
BIKER	0.427	0.690	0.804	0.519	0.571	0.547	0.849	0.952	0.662	0.695
BRAID	0.440	0.671	0.781	0.565	0.582	0.563	0.817	0.906	0.689	0.704
Ours	0.544	0.739	0.809	0.649	0.639	0.707	0.896	0.969	0.819	0.809
Improve.BIKER	36.0%	11.3%	5.7%	31.6%	17.3%	37.8%	10.4%	4.3%	27.6%	19.9%
Improve.BRAID	23.6%	10.2%	3.6%	15.3%	10.1%	25.6%	9.6%	7.0%	18.9%	14.9%

RQ2: 序列优化模块是否有用？

为了验证本章所设计的序列优化算法在缓解用户输入的自然语言查询描述与 API 官方文档描述之间知识差异方面的有效性，本节对基于序列优化的 API 推荐方法进行了消融实验。消融实验通过对比加入序列优化模块与去除序列优化模块的实验结果，深入分析了序列优化模块对推荐性能的影响。实验结果如表4-2所示，表中分别展示了在方法级别和类级别两个细粒度上，加入与去除序列优化模块的性能对比。

实验结果表明，序列优化模块对 API 推荐性能的提升具有显著作用。具体而言，加入序列优化模块：在方法级别和类级别上，推荐性能均得到了显著提升，评估指标（如 MAP、MRR 和 SuccessRank@K）的值均有明显提高。去除

序列优化模块，推荐性能出现显著下降，尤其是在正确答案的排名上表现较差，导致评估指标值大幅降低。

序列优化模块通过以下机制提升推荐性能：1.语义匹配优化：利用用户查询与 API 官方文档描述之间的语义相似度计算，重新评估候选 API 的相关性。2.排名提升：通过优化算法将正确答案的排名提升至更靠前的位置，从而使其进入评估指标的考虑范围（如前 K 个结果）。3.知识差异缓解：通过结合多源信息（如用户查询、API 官方文档和注释信息），缓解了用户输入与 API 描述之间的知识差异问题。

在实验过程中，本章进一步研究了候选 API 列表长度对推荐性能的影响。研究发现：1.较短的候选列表：可能导致正确答案未被包含在候选列表中，从而无法通过序列优化模块提升其排名。2.适当增加候选列表长度：能够提高正确答案被包含在候选列表中的概率，为序列优化模块提供更多的优化空间。因此，在实际应用中，建议在使用推荐模型生成候选 API 列表时，适当增加候选列表的长度，以提升正确答案被推荐出来的可能性。

表 4-2 序列优化模块的实验结果表

Table 4-2 Table of experimental results for the sequence optimization module

消融实验		评估指标				
		S@1	**S@3**	**S@5**	**MAP**	**MRR**
方法级别	Ablation	0.4749	0.719	0.8116	0.5982	0.5812
	Ours	0.5444	0.7392	0.8087	0.6498	0.6397
类级别	Ablation	0.6023	0.861	0.9498	0.7434	0.7251
	Ours	0.7066	0.8958	0.9691	0.8197	0.8078

消融实验结果表明，序列优化模块在提升 API 推荐性能方面具有重要作用。其核心机制在于通过语义匹配缓解知识差异现象和排名优化能力，将正确答案提升至更靠前的位置，从而显著改善推荐结果的准确性。此外，适当增加候选 API 列表的长度能够进一步提高序列优化模块的有效性。

4.4 本章小结

本章详细阐述了一种基于排序优化的 API 推荐方法。 首先，本章明确了研究问题与目标，针对用户输入的自然语言查询与 API 官方文档之间存在的知识差距问题，提出了一种基于序列优化的算法。 该算法旨在通过优化查询与 API 文档之间的语义匹配，有效缓解知识差距对推荐性能的影响。

接着，本章深入介绍了基于排序优化的 API 推荐方法模型的实现细节。 具体而言，模型的实现包括两个核心部分：语言模型的训练和候选序列的优化。在语言模型训练方面，本章采用了对比学习微调 RoBERTa 模型的方法，通过对比学习增强模型对查询语义的理解能力。 在候选序列优化方面，本章引入了注释信息，利用注释与 API 文档之间的语义关联来优化候选 API 的排名，从而提升正确 API 在推荐列表中的位置。

随后，本章详细介绍了实验设计，包括数据集的构建、基线模型的选择以及评估指标的设定。 实验设计旨在全面评估基于排序优化的 API 推荐方法的性能，并与现有的先进方法进行对比分析。

最后，本章对实验结果进行了深入分析。 实验结果表明，基于排序优化的 API 推荐方法在多个关键指标上均优于现有的基线模型，展现了其优越的推荐性能。 同时，实验结果也验证了序列优化模块的有效性，证明了其在缓解知识差距问题、提升推荐准确性和效率方面的显著优势。

第5章 问答模式下多源信息集成的 API 推荐方法

本章针对用户提出的自然语言查询语句中存在的表达差异问题，提出了一种问答模式下多源信息集成的 API 推荐方法——MSEAPIRec。用户输入的查询语句的表达差异主要体现在两个方面：一是开发人员之间的经验表达差异，二是文本与文本之间的知识表达差异。为了解决经验表达差异问题，本章设计了一种语义增强器，用于增强原始查询的语义信息。 具体而言，MSEAPIRec 利用 Stack Overflow 的讨论和 GitHub 的代码解释，构建了语义增强器，从而能够生成具有更丰富语义信息的自然语言查询。 针对知识表达差异问题，本章设计了一种排名优化器，用于重排推荐列表。 MSEAPIRec 通过利用 GitHub 的代码解释和官方文档的描述，构建了排名优化器，从而优化推荐结果。

为了验证 MSEAPIRec 的性能，本章在公开数据集上进行了广泛的实验，并将其与三种基线方法进行了比较。 实验结果表明，在 SuccessRank@1 评估指标中，MSEAPIRec 在方法级别和类级别上分别比最先进的基线方法 CLEAR 提高了 7.9%和 7.3%。 此外，在衡量模型整体性能的 MAP 和 MRR 指标方面，MSEAPIRec 也优于最先进的基线方法。 这些结果充分证明了 MSEAPIRec 在提升 API 推荐准确性和效率方面的显著优势。

5.1 研究动机

在问答推荐系统中，推荐结果的准确性高度依赖于输入查询的质量。构建能够全面准确传达用户搜索意图的高质量查询面临显著挑战，这一现象在 API 推荐领域尤为突出。用户输入的查询表达差异对推荐准确性形成双重制约：既存在编程人员个体间的经验表达差距，也包含文本间的知识表达差距。这种表达差异的根源可归结于以下两个客观因素：

首先，个体间的意图表达能力呈现系统性差异。实验观测表明，不同经验水平的开发者在表达查询意图时存在显著区别。经验丰富的开发者通常能精确描述功能需求，而新手开发者往往难以完整表达其搜索意图。本研究将此类个体表达差异定义为"经验表达差距"。需特别指出的是，这种差距本质上是相对概念，主要取决于用户对特定功能模块的熟悉程度而非绝对编程经验水平。在

API 搜索场景中，提问者通常处于功能认知的初级阶段，而注释者则需要具备完整的代码功能理解才能生成有效注释。这种认知层级的差异使得经验丰富的程序员撰写的代码注释天然包含优化查询的关键信息。基于此发现，本章提出通过整合 GitHub 平台的专业级代码注释来解决用户输入中的经验差距问题。

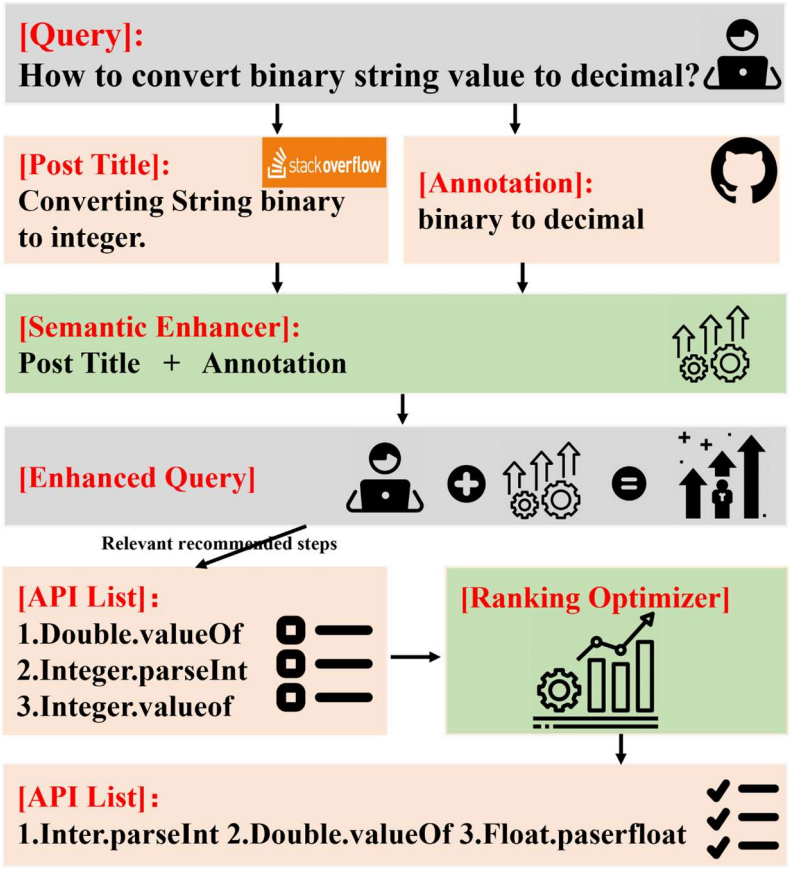

图 5-1 MSEAPIRec 推荐流程示例图

Figure 5-1 MSEAPIRec Recommendation Process Example Diagram

以典型应用场景为例（如图 5-1 所示），专业注释"binary to decimal"较之用户原始查询"How to convert a binary string value to a decimal."具有更高的语义密度和准确性，后者包含冗余成分如"How"、"to"等非必要元素。本方法的核心在于将此类经过专家验证的优质注释信息融合至原始查询中，从而有效弥合经验表达差距。

在 API 推荐系统中，用户意图与官方文档间的表达差异构成另一关键挑战，主要表现为二者间的系统性词汇失配现象。以典型查询语句"How to initialize all values in an array to false?"为例，其对应 API——Arrays.fill 的官方文档描述为"Assigns the specified boolean value to each element of the specified array of booleans"，两者在核心词汇层面存在显著差异。本研究将此现象定义为"知识表达差距"，

其根源在于用户意图描述的非结构化特征与官方文档的结构化特性之间的根本性矛盾：前者多采用目标导向的自然语言表述，后者则侧重 API 功能与结构的规范化描述。这种异构数据特征对语义映射形成实质性障碍。

为验证本方法的有效性，图 5-2 展示了典型实例分析结果。针对查询"Converting a sentence string to a string array of words in Java."，现有最优基线方法 CLEAR 的 top-3 推荐结果均未包含正确 API。相较之下，MSEAPIRec 通过双通道语义融合机制实现精准推荐：首先基于深度语义匹配模型检索相关开发帖与专业评论，继而构建语义增强的提示模板。在此过程中，专业注释如"for space-separated text-based processing"发挥了关键作用——该表述不仅通过"space-separated"明确指定字符串分割规则，更利用"text-based processing"界定方法的应用场景，有效弥补用户原始查询中领域知识的缺失。

特别值得关注的是，专业注释中"word arrays"、"space-separated"等规范化术语的应用，相较于用户查询中的功能性描述，能更精确地映射至官方文档的语义空间。本方法通过构建注释特征向量与 API 文档向量的联合嵌入空间，成功实现跨域语义对齐，最终准确推荐目标 API:"java.lang.String.split"。

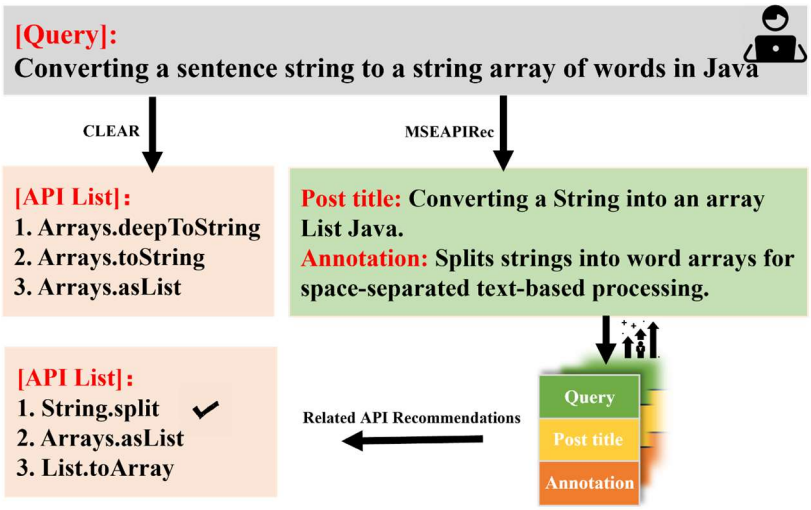

图 5-2 MSEAPIRec 效果示例图

Figure 5-2 Example diagram of MSEAPIRec effect

5.2 模型设计

图 5-3 展示了 MSEAPIRec 框架的体系架构，该架构通过三个核心处理模块实现用户查询到定制化 API 的转换。各模块的技术实现流程如下：

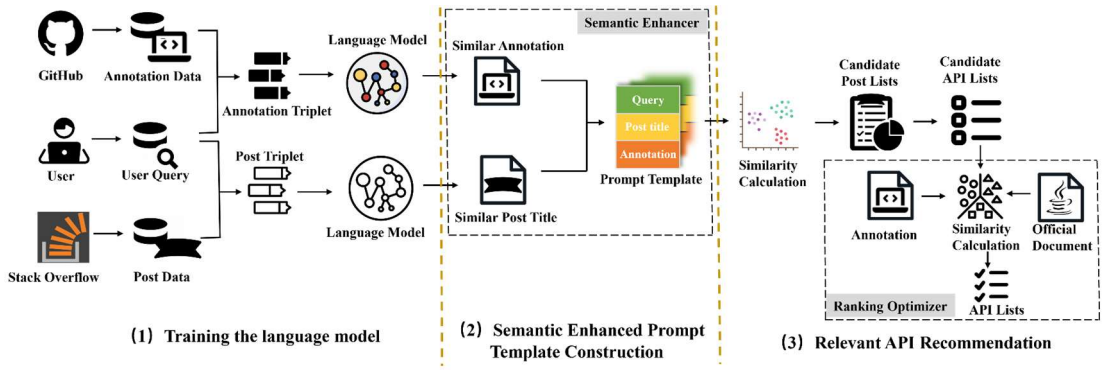

图 5-3 MSEAPIRec 的总体流程图

Figure 5-3 Overall flow diagram of MSEAPIRec

（1）训练语言模型：本方法基于 RoBERTa 基础架构，采用对比学习策略进行语言模型的领域适配训练。通过引入 CLEAR[11]提出的对比训练范式，对预训练语言模型实施精细化微调，使模型具备跨词汇的深度语义表征能力。经过优化的模型能够有效捕获句子级语义特征，为后续模块提供高精度的语义相似度计算基础，尤其擅长识别词汇差异但语义相近的文本实例。

（2）构建语义增强器：本模块构建具有多源知识融合能力的提示模板生成机制。首先通过微调后的语言模型提取查询语句的深度特征嵌入，随后在 Stack Overflow 技术问答库和 GitHub 代码库的元数据空间中进行跨模态相似度检索，系统化筛选与查询语义最相关的技术讨论标题和代码注释片段。最后通过特征空间的线性插值方法，将检索所得的多源知识要素融合构建结构化提示模板。此过程显著扩展原始查询的语义表征维度，形成包含领域知识的增强型查询表达。

（3）相关 API 推荐：基于增强型查询输入，本模块采用两阶段排序机制实现精准 API 推荐。第一阶段通过神经匹配模型生成初步候选 API 列表，第二阶段引入基于文档结构的排序优化器进行结果校准。该优化器通过计算候选 API 官方文档描述与其对应代码库功能注释之间的语义相似度，构建基于知识一致性的重排序权重矩阵。通过线性组合初始匹配分数与知识一致性分数，实现候选 API 列表的最终优化排序。这种双阶段架构有效缓解了代码语义与文档描述间的知识断层问题，显著提升了推荐结果的可解释性和准确性。

5.2.1 训练语言模型

构建三元组：为了利用对比训练方法获得能够全面表示句子级特征的语言模型，需要建立训练的输入数据—三元组[54]。 在本章中，需要利用训练好的语

言模型来识别与查询语义相似的帖子和注释，因此需要构建两类三元组作为训练输入。

第一类三元组称为查询三元组$(Q,P,N)_{title}$。其中，Q表示用户输入的查询语句，P表示与Q具有相似语义信息的正样本帖子，而N表示与Q不相关的负样本帖子。正样本表示具有与Q相同答案的帖子标题，本章将具有与Q不同的答案的帖子标题视为负样本。为了进行训练，本章为Q取M个正样本帖子和K个负样本帖子，总共生成$M\times K$个三元组用于训练

本章将 Github 中的注释视为查询语句，从而构成第二类三元组——注释三元组$(A,P,N)_{ann}$。其中，A表示 Github 中的注释，P表示与注释承载相同语义信息的帖子标题，N表示与注释无语义相关性的帖子标题。为了识别语义相似的注释和帖子作为正样本对，本文遵循 Irsan 等人[12]所做的工作。计算 Github 中注释对应的代码片段所包含的 API 与 SO 答案中提到的 API 的重叠率，并假设重叠率大于 0.75 的帖子与注释在语义上相似。最终，将相似的帖子视为正样本帖子。在构建三元组时，当发现所有的正样本帖子时，将选择重叠率最小的帖子作为负样本帖子。最终会得到标注、正样本帖子、负样本帖子，形成标注三元组。

训练语言模型：基于上述两类三元组，本方法采用双通道对比学习框架对 CrossEncoder[62]模型进行领域适配。具体实施步骤如下：模型架构，以 RoBERTa[63]为基座网络，构建双塔式编码器分别处理查询三元组和注释三元。损失函数设计，定义基于余弦距离的对比损失函数：

$$\Im = \sum_{i=1}^{M}\sum_{j=1}^{K}\max(0,\text{ə}+\cos(E_Q,E_{N_j})-\cos(E_Q,E_{P_j})) \tag{5-1}$$

其中E_Q、E_{P_j}、E_{N_j}分别表示查询、正样本、负样本的嵌入向量，ə=1 为边界超参数，强制约束正负样本间距至少为 1 个余弦单位。训练后获得两个专用编码器——查询编码器$\Im_Q$用于检索相关 SO 帖子，注释编码器$\Im_A$用于关联代码注释。

5.2.1 构建语义增强器

此阶段旨在设计一个语义增强器，用于扩展用户输入的原始查询的语义。每个经过语义增强器的用户查询都会引出一个包含更丰富语义信息的新查询。

语义增强器的建立包括三个步骤，即 （1）相关帖子检测、（2）相关注释检测和 （3）提示模板生成。

（1）相关帖子检测。对于给定的用户查询，MSEAPIRec 首先使用基于查询三元组训练的嵌入模型$\Im_Q$来嵌入用户的原始查询和 SO 帖子标题。由于该嵌入模型是使用同一领域的同质信息训练的，其生成的嵌入空间具有领域内语义一致性特征。在获得查询和帖子的嵌入之后，使用余弦相似度来计算用户嵌入与 SO 帖子标题之间的相似度值，最终选择相似度值最高的帖子标题用于构建提示模板。

（2）相关注释的检测。与（1）中的步骤类似，对于用户输入，MSEAPIRec 使用基于注释三元组训练的嵌入模型$\Im_A$来嵌入用户输入和 GitHub 注释。由于 GitHub 注释与代码相关，通常包含代码的描述和使用方法，而 SO 问题与编程相关，通常涉及问题的描述和解决方案。因此，该模型是使用异构信息进行训练的。通过对比学习的方法将相似的注释和SO帖子标题一起纳入句子嵌入中，可以获得对这两类异构信息具有更强表示能力的模型。同理，在获得注释和问题的文本嵌入后，利用余弦相似度来识别与用户输入在语义上相似的注释，并利用获取的注释构建提示模板。

（3）生成提示模板。在此阶段，MSEAPIRec将步骤 （1） 和（2） 中获得的语义相关的帖子和注释组合成一个模板。例如，如果 （1） 中获得的帖子为$P(p_1,p_2,p_3,\cdots,p_n)$，（2） 中获得的注释为$N(n_1,n_2,n_3,\cdots,n_n)$。则最终的提示模板为$T=(p_1,p_2,p_3,\cdots,p_n,n_1,n_2,n_3,\cdots,n_n)$。通过上述方法，可以构造一个语义增强器。对于每一个用户查询Q作为输入，语义增强器都会生成一个提示模板T，MSEAPIRec将Q与T结合起来，形成一个拥有更丰富语义信息的新查询Q'，最终Q'将替代Q纳入到后续的推荐任务中。

5.3.1 相关 API 推荐

（1）候选 API 列表的获取：在此阶段，目标是将经过语义增强器处理后的新增强查询Q'作为输入，获取候选 API 列表。首先，需要找出与Q'最相似的 k 个相关帖子，并利用得到的 k 个相关帖子提取相关 API 实体。此步骤主要是为

了缩小相关 API 实体的搜索范围。关于 API 实体的提取方法，遵循前人的工作[8]，利用正则表达式[18]识别每个候选帖子答案中的超链接，这些超链接通常指向官方 API 文档，通过正则表达式提取超链接中包含的 API 实体。当然，也会出现超链接中的内容不包含 API 实体或与 API 毫无关系的情况，因此需要对提取的内容进行过滤。本章为此构建了一个字典，存储从官方文档中检索到的 API 方法名称。对于候选帖子答案中提取出的内容，将其与字典中的 API 方法名进行匹配，如果有与提取出的内容相匹配的API方法，则将其添加到候选API列表中，否则，将该内容过滤掉。

（2）构建排名优化器：此阶段构建排名优化器，对候选 API 列表进行重新排序，以提高正确 API 的排名。首先，将获得的注释信息作为排名优化器的数据。MSEAPIRec 对候选 API 列表中 API 对应的注释与官方文档描述进行相似度计算，得到注释 Q_A 与每个 API 官方文档描述 Q_{doc} 之间的相似度得分 $S_{\mathrm{A}-doc}$。MSEAPIRec 还对用户查询与上一步中的相关帖子进行相似度计算，得到查询 Q 与相关帖子 Q_{so} 之间的相似度得分 S_{Q-SO}。最终，MSEAPIRec 将 $S_{\mathrm{A}-doc}$ 和 S_{Q-SO} 的调和平均值作为排序优化器的最终输出 S：

$$S=\frac{2\times S_{\mathrm{A}-doc}\times S_{Q-SO}}{(S_{\mathrm{A}-doc}+S_{Q-SO})} \tag{5-2}$$

对于每个候选 API 列表，排序优化器会为每个 API 生成一个分数，MSEAPIRec 利用该分数对候选 API 列表进行重新排序，最终将重新排序后的 API 序列推荐给用户。由于 GitHub 注释和官方文档都是描述 API 功能和结构的同类信息，因此利用注释对候选 API 列表进行重新排序可以大大缓解知识差异问题，最终达到优化正确 API 排序的目的。

5.3 实验设计与分析

5.3.1 数据集设计与构建

为了全面评估 MSEAPIRec 的性能，研究采用了 CLEAR 这一先进方法的数据集来构建 SO 数据集，该数据集在相关领域被广泛使用。此数据集最初由 BIKER 的研究团队从 SO 的官方数据转储中收集并整理。数据集的构建遵循以下原则：首先，所有 SO 问题必须与 Java JDK 相关，并包含“JAVA”标签；其次，

所有问题的分数必须为正值；最后，每个问题的答案中至少需包含一个 API 实体。基于上述标准，构建的 BIKER 数据集共包含 33K 个与 Java 相关的 SO 问题。

在注释数据方面，研究采用了 Irsan 等人[12]的数据集，该数据集源自 Martin 等人[52]的公开数据集，由注释和 API 序列对组成。该注释数据集共包含 196,276 对注释和 API 序列。

为了评估 MSEAPIRec 的性能，研究采用了 BIKER 提供的测试数据集。该测试集通过严格的筛选流程获得，具体筛选标准如下：首先，筛选出得分大于或等于 5 的 SO 问题；其次，筛选出的问题的答案需包含 API 实体，且该答案的得分应大于 0。此外，BIKER 的研究团队还对这些筛选后的问题进行了人工检查，排除了不符合编程任务搜索要求的问题。最终，研究获得了 413 对问答对作为测试数据，用于评估本方法的性能。

5.3.2 基线模型简介

本章使用三种最先进的 API 推荐技术作为基线方法，并将这些基线方法与 MSEAPIRec 进行了比较。这三种基线方法如下：

DeepAPI[52]：将为给定的自然语言查询生成 API 使用序列的任务建模为机器翻译问题。它采用循环神经网络 (RNN) 编码器-解码器神经网络模型从用户的自然语言查询中学习单词序列。DeepAPI 将单词序列编码为固定长度的上下文向量，并基于该向量生成相关的 API 序列。

BIKER[9]：将 StackOverflow 与官方 Java JDK 文档结合使用，完成推荐任务。它使用经过训练的 CBOW[46]模型和 TF-IDF 文档来查找与给定输入相关的候选 SO 帖子，并使用 Java JDK 官方文档描述与输入的相似度得分来细化最终答案。

CLEAR[11]：使用对比学习技术微调 ROBERTa 模型，以构建表征序列语义信息的语言模型。对于用户输入，CLEAR 利用训练好的语言模型进行嵌入，并根据相似性找到候选 SO 列表，然后使用联合训练的基于 BERT 的分类模型对它们进行重新排序，最后根据重新排序的帖子向用户推荐相关的 API。

LLM[55,56,57]：大型语言模型自推出以来就受到了广泛关注。在本文中，部署了几个开源模型来与 MSEAPIRec 进行比较。其中包括 Meta AI 推出的

LLaMA[55] 系列开源模型，本实验使用 LLaMA3_3b、8b 进行比较实验。例外还包括阿里云发布的 Qwen2_14b[56] 和谷歌发布的 gemma2_9b[57]，它们也被用作基线方法。

5.3.3 评估指标

为了合理地衡量模型的性能，采用了 MAP（平均准确率）和 MRR（平均倒数排名），这两个指标在信息检索中被广泛使用[58-61]。MAP 需要考虑所有正确答案的排名，其值越大，模型性能越好。而 MRR 则表示推荐结果列表中第一个正确答案排名的倒数，值越大表示正确答案的排名越靠前。此外，本文还使用更细致的评估指标 SuccessRank@K 来评估 MSEAPIRec。SuccessRank@K 考虑正确答案的第一次出现是否在 K 范围内有排名。所有评估指标的详细信息如下：

MAP：考虑所有正确答案的排名，可以全面考虑模型的整体性能

$$AP = \frac{\sum_{i=1}^{n} count(hit_i) \times rel(i)}{m} \tag{5-3}$$

$$MAP = \frac{\sum_{i=1}^{R} AP_i}{R} \tag{5-4}$$

其中，$count(hit_i)$ 表示 top-i 中正确的 API 数量，m 表示真实组 API 数量。注意，当推荐结果列表中的正确 API 数量大于 0 时，$count(hit_i) \times rel(i)$ 只计算该位置的排序倒数与当前位置出现过的正确 API 数量的乘积。R 表示该测试集中所有问题的数量。

MRR：表示从推荐结果列表遍历多远才能找到正确答案的 API。

$$MRR = \frac{\sum_{i=1}^{R} \frac{1}{fisthit_i}}{R} \tag{5-5}$$

其中 $fisthit_i$ 表示在推荐结果列表中找到第一个正确答案时的排名。如果遍历所有列表后仍未找到正确答案，则 $\frac{1}{fisthit_i}$ 为 0。

SuccessRank@K：要求正确答案 API 的第一个出现在 K 之内。本文中，K 取值为 1、3 和 5。

$$SuccessRank@K = \frac{\sum_{i=1}^{R} firsthit_i \le K}{R} \tag{5-6}$$

若正确答案的位置在 K 个列表范围内 $firsthit_i \le K$ 取 1，否则取 0。其中，$SuccessRank@1$是一个极其严格的评估指标。

5.3.4 实验细节设置

对于作为基线的三种高级 API 推荐方法，即 DeepAPI[19]、BIKER[3]和 CLEAR[4]，本章直接使用了提出这些方法的作者公布的或其他人员复制的复制包。由于保留过多的类似问题会引入大量的噪音数据，而保留的类似问题太少会忽略有效的语义。因此，在方法级别保留的问题数量为 41 个，而在类级别保留的问题数量为 85 个。此外，为了成功进行实验，本实验在配备 4090 个 GPU 的两台服务器上进行了所有部署。

5.3.4 研究问题

为了对 MSEAPIRec 的性能进行全面的评估，本章设计了大量的实验。这些实验都围绕着以下几个研究问题。

RQ1：与先进的基线方法相比，MSEAPIRec 的性能如何？

RQ2：MSEAPIRec 在不同参数下的性能。

RQ3：语义增强器和排序优化器对 MSEAPIRec 的性能有什么影响？

RQ4：语义增强器中提示模板的数量和位置是否会对 MSEAPIRec 产生影响。

5.3.4 实验结果与分析

RQ1：与先进基线方法相比，MSEAPIRec 的性能表现如何？

为验证 MSEAPIRec 在 API 推荐任务中的有效性及其性能优势，本研究设计了 RQ1，旨在通过测试数据集对比本方法与现有先进基线方法的性能差异。

如表 5-1 所示，实验结果表明，MSEAPIRec 在多数评测指标上均优于基线方法，展现出显著的推荐性能优势。进一步分析表 2 中数据可发现，基线方法 DeepAPI 的性能显著落后于其他方法，其原因可能在于其深度学习框架对大规模高质量训练数据的依赖性较高，而此类数据的获取与标注成本在实际场景中具有较大挑战性。值得注意的是，尽管 LLM（大语言模型）具备丰富的先验知识，但其表现未达预期（见图 5-4）。初步分析表明，这一现象可能归因于实验设计中使用的提示词（例如“Please help me to suggest the correct API for the

complaint”）过于简化，未能有效激活 LLM 的深层知识推理能力。

在基线方法中，CLEAR 取得了次优性能，其采用的 BERT 框架能够有效捕捉序列语义信息并表征句子级特征，而这一能力恰为另一基线方法 BIKER 所欠缺。然而，CLEAR 与 BIKER 均未充分考虑原始查询（Query）的固有局限性，例如语义信息不足或用户经验差异对推荐性能的影响。相比之下，MSEAPIRec 通过缓解上述问题，实现了更精准的 API 推荐。

表 5-1 MSEAPIRec 的对实验结果表

Table 5-1 Experimental results of MSEAPIRec

基线	方法级别					类级别				
	S@1	S@3	S@5	MAP	MRR	S@1	S@3	S@5	MAP	MRR
DeepAPI	0.013	0.017	0.023	0.010	0.031	0.060	0.100	0.122	0.081	0.092
BIKER	0.429	0.690	0.804	0.519	0.571	0.547	0.849	0.952	0.662	0.695
CLEAR	0.541	**0.788**	**0.884**	0.658	0.648	0.703	0.923	0.977	0.818	0.808
MSEAPIRec	**0.583**	0.768	0.850	**0.683**	**0.669**	**0.754**	**0.938**	**0.992**	**0.844**	**0.833**
Improve.BIKER	36.0%	11.3%	5.7%	31.6%	17.3%	37.8%	10.4%	4.3%	27.6%	19.9%
Improve.CLEAR	7.9%	-2.5%	-3.8%	3.8%	3.4%	7.3%	1.7%	1.6%	3.2%	3.1%

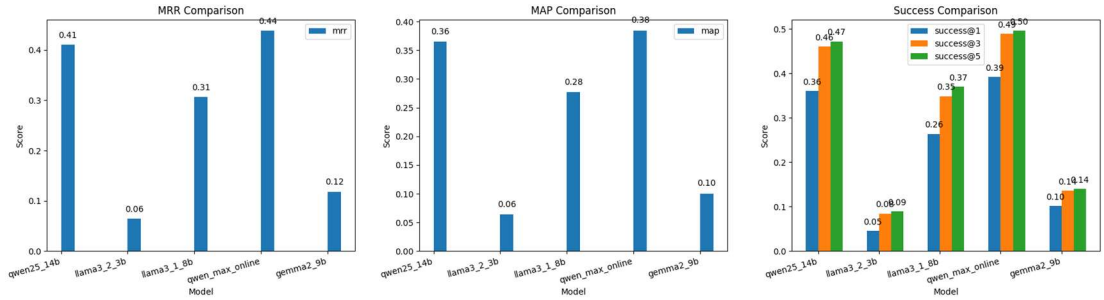

图 5-4 LLM 在简单提升时在 API 推荐任务上的性能

Figure 5-4 Performance of LLM on API recommendation task at simple boosting

具体而言，在方法层面，MSEAPIRec 相比最先进的基线方法 CLEAR，其 SuccessRank@1 指标提升 7.9%，衡量整体性能的 MAP 与 MRR 分别提升 3.8%和 3.4%。在类层面，本方法在 SuccessRank@1、MAP 和 MRR 指标上较 CLEAR 分别提升 7.3%、3.2%和 3.1%。尽管 MSEAPIRec 在 SuccessRank@3 和 SuccessRank@5 指标上略逊于基线方法，但考虑到用户通常期望推荐结果中正确答案位于列表顶端，因此 SuccessRank@1 及整体性能指标（MAP、MRR）具有更高的实际意义。这一结果进一步验证了 MSEAPIRec 在现实推荐场景中的技术优势。

RQ2：MSEAPIRec 在不同参数下的性能。

本实验旨在探究候选帖子列表长度（即检索过程中保留的候选帖子数量）对 MSEAPIRec 推荐性能的影响。该参数直接决定模型从 Stack Overflow 问答对

中提取的 API 实体数量，进而影响后续推荐步骤的输入质量。

表 5-2 MSEAPIRec 的在不同参数下的表现情况表

Table 5-2 Performance of MSEAPIRec under different parameters Table

方法基本						类级别					
Value	S@1	S@3	S@5	MAP	MRR	Value	S@1	S@3	S@5	MAP	MRR
10	0.5405	0.695	0.7259	0.6108	0.6227	20	0.7297	0.9035	0.9382	0.8223	0.8111
30	0.5675	0.749	0.8108	0.6668	0.6573	40	0.7413	**0.9421**	0.9653	0.8406	0.8276
35	0.5677	0.7568	0.8069	0.6701	0.6573	60	0.7413	0.9382	0.9768	0.8421	0.8294
40	**0.5792**	0.7683	0.8263	**0.6795**	**0.6662**	80	0.7413	0.9344	0.9884	0.8406	0.8293
45	0.5714	0.7761	0.8417	0.6770	0.6639	**85**	**0.7542**	0.9382	**0.9923**	**0.8444**	**0.8329**
50	0.5752	0.7761	0.8301	0.6702	0.6662	90	0.7374	0.9343	0.9846	0.8393	0.8278
100	0.5405	0.8069	0.8726	0.6612	0.6478	300	0.7336	0.8880	0.9575	0.8226	0.8129
500	0.5367	**0.8069**	**0.8803**	0.6540	0.6463	500	0.7297	0.8880	0.9614	0.8209	0.8115

实验首先采用步长为 5 的网格搜索策略，对参数进行初步探索。如表 5-2 所示，基于 8 组参数值的实验结果表明：在方法层面，当候选列表长度接近 40 时，模型性能达到峰值（SuccessRank@1、MAP 和 MRR 等指标最优），且性能随参数值变化呈现先升后降的单峰趋势。这一现象可归因于候选列表长度的双重作用：当参数值较低时（如<30），候选列表过短可能导致关键 API 实体未被充分捕获，从而限制推荐性能；而当参数值超过临界值（如>40），候选列表过长会引入冗余或噪声信息，导致模型准确率显著下降。类似规律在类层面参数分析中同样存在，其最优值出现在 85 附近，进一步验证了上述推断的普适性。

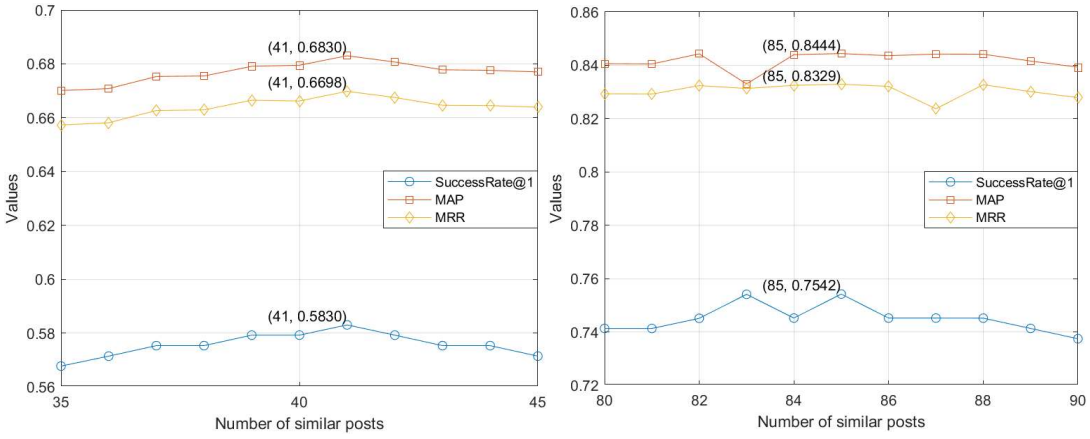

图 5-5 方法（左）与类（右）级别的最优参数图

Figure 5-5 Optimal Parameter Map at the Method (left) and Class (right) Levels

为精确确定最优参数，研究进一步在初步最优值附近（±10 区间）开展精细化网格搜索，将步长缩减至 1。如图 5-5 所示，实验最终确定方法级参数最优值为 41，类级参数最优值为 85。这一结果表明，候选列表长度需在信息完整性与噪声抑制之间实现平衡，而参数敏感性分析为模型优化提供了重要依据。

RQ3：语义增强器和排序优化器对 MSEAPIRec 的性能有什么影响？

为验证语义增强器（弥补经验差距、增强原始查询）与排序优化器（缩小知识差距）对 MSEAPIRec 性能的贡献，本研究通过消融实验构建两个变体模型：

$MSEAPI\,\mathrm{Re}c_{SE}$：仅保留语义增强器，移除排序优化器；

$MSEAPI\,\mathrm{Re}c_{Ro}$：仅保留排序优化器，移除语义增强器，直接使用原始查询作为输入。

如表 5-3 所示，两个变体模型在方法级与类级推荐任务中的性能均显著低于完整模型，表明两个模块对系统性能均具有关键作用。

表 5-3 不同模块对 MSEAPIRec 的影响

Table 5-3 Effect of Different Modules on MSEAPIRec

消融实验		评估指标				
		S@1	**S@3**	**S@5**	**MAP**	**MRR**
方法级别	$MSEAPI\,\mathrm{Re}c_{SE}$	0.4749	0.7190	0.8116	0.5982	0.5812
	$MSEAPI\,\mathrm{Re}c_{Ro}$	0.5444	0.7392	0.8087	0.6498	0.6397
	MSEAPIRec	**0.5830**	**0.7683**	**0.8301**	**0.6830**	**0.6698**
类级别	$MSEAPI\,\mathrm{Re}c_{Ro}$	0.6023	0.8610	0.9498	0.7434	0.7251
	$MSEAPI\,\mathrm{Re}c_{Ro}$	0.7066	0.8958	0.9691	0.8197	0.8078
	MSEAPIRec	**0.7542**	**0.9382**	**0.9923**	**0.8444**	**0.8329**

语义增强器的影响分析：移除语义增强器后（$MSEAPI\,\mathrm{Re}c_{Ro}$），模型性能全面下降。在方法级任务中，SuccessRank@1、MAP 和 MRR 分别降低 6.6%、4.9%和 4.5%；在类级任务中，上述指标分别降低 6.3%、2.9%和 3.0%。这一结果表明，语义增强器通过扩展原始查询的语义信息，有效缓解用户经验差距，从而提升推荐准确性。

排序优化器的影响分析：移除排序优化器后（$MSEAPI\,\mathrm{Re}c_{SE}$），模型性能下降更为显著。在方法级任务中，SuccessRank@1、MAP 和 MRR 分别下降 18.5%、12.4%和 13.2%；类级任务中则下降 20.1%、12.0%和 14.9%。进一步分析发现，尽管未使用排序优化器时模型仍能生成包含正确答案的候选列表，但正确答案的排名普遍靠后（如超出 Top-3 范围），导致其未被评估指标覆盖。此现象表明，排序优化器通过重排机制将正确答案提升至列表前列，是模型实现高排名指标的核心组件。

模块协同效应：实验表明，语义增强器与排序优化器具有协同作用：前者

通过丰富查询语义提高候选答案的召回率，后者通过知识驱动排序策略优化答案的可见性。二者缺一不可，共同保障 MSEAPIRec 在复杂场景下的推荐性能。

RQ4：语义增强器中提示模板的数量和位置是否会对 MSEAPIRec 产生影响。

本研究进一步探究提示模板数量及其位置对 MSEAPIRec 性能的作用机制。如表 5-4 所示，实验从以下两个维度展开分析：

1.模板数量的影响：假设提示模板数量增加可通过引入更多语义信息提升模型性能，但实验结果表明该假设存在局限。数据显示，当模板数量超过阈值时（如>3），模型在 SuccessRank@1、MAP 等指标上出现显著波动甚至下降。通过案例分析发现，低排名的参考帖子生成的模板质量存在较大不确定性，可能包含语义偏差或噪声信息。例如，针对输入问题“How to convert binary string value to decimal”，推荐系统基于 top-3 和 top-4 帖子生成的模板分别为“how to convert from decimal to ASCII characters”和“Convert a number to a String with exactly 2 decimal places”，其语义与原始问题无关。此类低质量模板会误导语义增强过程，导致推荐性能下降。此现象表明，模板质量而非数量是决定增强效果的核心因素。

表 5-4 语义增强器中提示模板的设置对 MSEAPIRec 的影响

Table 5-4 How the number and position of cue templates in the semantic enhancer affects MSEAPIRec performance

数量	位置	方法基本					类级别				
		评估指标					评估指标				
		S@1	S@3	S@5	MAP	MRR	S@1	S@3	S@5	MAP	MRR
1	后	**0.583**	**0.768**	**0.830**	**0.683**	**0.669**	**0.754**	**0.938**	**0.992**	**0.844**	**0.833**
	前	0.552	0.745	0.791	0.650	0.639	0.745	0.903	0.953	0.832	0.821
2	后	0.544	0.745	0.799	0.652	0.643	0.718	0.911	0.953	0.822	0.813
	前	**0.563**	**0.764**	**0.814**	**0.66**	**0.649**	**0.752**	**0.915**	**0.980**	**0.844**	**0.832**
3	后	0.540	0.768	0.818	0.655	0.646	0.730	0.930	0.980	0.831	0.822
	前	**0.540**	**0.783**	**0.834**	**0.654**	**0.644**	**0.749**	**0.907**	**0.984**	**0.842**	**0.831**
4	后	0.528	0.741	0.818	0.644	0.632	0.718	0.926	0.973	0.823	0.813
	前	**0.559**	**0.756**	**0.837**	**0.662**	**0.650**	**0.745**	**0.911**	**0.973**	**0.837**	**0.826**
5	后	0.540	0.741	0.826	0.646	0.637	0.722	0.918	0.969	0.826	0.816
	前	**0.552**	**0.745**	**0.834**	**0.649**	**0.641**	**0.741**	**0.919**	**0.963**	**0.828**	**0.823**

2. 模板位置的影响：基于 RoBERTa 微调语言模型的特性（可捕捉词序与位置语义特征），实验对比了模板置于问题前、后两种配置的差异。结果表明，除单一模板场景外，将模板前置至问题前端的配置在多数情况下性能更优。这一结果与语言模型的注意力机制特性相符：前置模板可为模型提供更早的上下

文线索，从而更有效地引导语义理解过程。

综上，提示模板的应用需权衡数量与质量间的平衡，同时需根据模型特性优化模板位置策略。实验结果为语义增强器的工程实现提供了重要设计准则。

5.4 本章小结

本章提出并验证了一种问答模式下基于多源信息集成的 API 推荐方法 MSEAPIRec，旨在解决自然语言查询中存在的语义表达差异问题。研究首先界定了两类核心挑战：1）用户间经验差异导致的语义表达模糊性；2）文本间知识差异引发的语义信息不匹配。针对上述问题，本方法提出双阶段优化框架：语义增强器，通过多源语义信息融合机制扩展原始查询的语义表征，缓解用户经验差异导致的表达局限性。排名优化器，基于知识对齐策略对候选 API 列表进行重排序，解决文本间知识差异带来的语义鸿沟问题。

方法构建包含三个核心阶段：基于领域语料的预训练语言模型微调，语义增强器的上下文感知模板生成，多维度特征融合的 API 推荐决策。随后，通过系统性实验方案验证方法有效性。在方法级别，SuccessRank@1 指标较最优基线 CLEAR 提升 7.9%，MAP 与 MRR 分别提升 3.8%和 3.4%；在类级别，对应指标分别提升 7.3%、3.2%和 3.1%。此外，消融实验表明，语义增强器与排名优化器分别使 SuccessRank@1 降低 6.6%和 18.5%，验证双模块协同必要性。最后，参数敏感性分析确定方法级/类级最优候选列表长度为 41/85，模型在±10 参数区间内保持稳定性（性能波动<1.2%）。

实验结果表明，MSEAPIRec 通过语义增强与排序优化的双重优化机制，显著提升了复杂查询场景下的 API 推荐精度与可靠性。

第6章 总结与展望

6.1 工作总结

API 的使用能显著的提升开发人员的开发效率、给开发人员带来便捷的同时还能促进代码复用，具有重要的现实价值。针对现有的 API 推荐方法都关注于如何设计能全面表示用户自然语言查询的表征模型，而忽视了用户输入的自然语言查询本身的问题。因此，本文提出了 3 个递进性的解决方法来解决这种问题，从而提升 API 推荐结果的准确性，优化开发人员的编程体验。

（1）针对用户输入的自然语言查询由于经验差异导致的语义表达模糊性问题，设计并实现了一种问答模式下查询重构的 API 推荐方法 REAPI。该方法通过利用 Stack overflow 构建先验知识库，基于此先验知识库来重构用户查询的方式，使得重构后的查询语句更能清晰表达用户的任务需求，有效的缓解了经验差距的现象。仿真实验结果证明了 REAPI 的有效性，REAPI 具有更加准确的推荐性能。此外，用户实例研究的结果验证其在实际编程任务中的辅助价值，证明其能有效降低新手开发者的表达门槛。

（2）针对用户输入的自然语言查询由于知识差异引发的语义信息不匹配问题，提出了一种问答模式下序列优化的 API 推荐方法。该方法通过引入与 API 官方文档同质的注释信息来对 API 候选进行优化，有效的缓解了 API 官方文档与用户查询语句间的知识差异问题。

（3）用户输入的自然语言查询存在的表述差异问题，这种差异不仅体现在开发人员人与人之间（经验表达差异），还体现在文档与文档间（知识表达差异）。针对上述问题，提出一种问答模式下多源信息集成的 API 推荐方法 MSEAPIRec，其核心思想包括两个模块：（1）语义增强器，通过多源语义信息融合机制扩展原始查询的语义表征，缓解用户经验差异导致的表达局限性。（2）排名优化器，基于知识对齐策略对候选 API 列表进行重排序，解决文本间知识差异带来的语义鸿沟问题。实验结果证明了 MSEAPIRec 相较于最优秀的基线方法，拥有更加准确的推荐性能。

6.2 工作展望

API 推荐依然是软件工程领域中不和或缺的热点问题，虽然本文针对现有 API 对用户查询语句的质量具有高依赖性的痛点问题，提出了一种问答模式下基于上下文感知的 API 推荐方法，并通过仿真实验与用户实证研究验证了其有效性，但方法仍存在以下局限性，亟待后续研究突破：

（1） 模型泛化能力的系统性提升

当前实验验证集中于 Java 语言场景，其性能优势依赖于特定编程语言的语法特征与 API 生态。虽然理论上可通过替换训练语料实现跨语言适配，但实际应用中面临两大挑战：

跨语言语义迁移障碍：不同编程语言（如 Python 与 C#）的 API 调用范式、社区术语体系存在显著差异，需构建通用型语义表征框架以捕捉跨语言共性特征。

低资源语言适配难题：新兴或小众编程语言（如 Rust）的标注数据稀缺性可能制约模型性能，需探索半监督迁移学习策略或元学习机制。

未来研究需建立多语言基准测试集，评估方法在跨语言场景下的鲁棒性，并开发可扩展的领域自适应技术。

（2） 推荐结果的工程实用性增强

API 推荐与实际开发环境的兼容性问题尚未得到充分解决，具体表现为：

版本兼容性风险：推荐 API 可能因版本迭代（如 Java 8 至 Java 17 的 Stream API 变更）与用户环境不匹配，需构建动态版本知识图谱并集成依赖管理工具（如 Maven）进行实时校验。

上下文代码适配性：推荐 API 与开发者现有代码的接口兼容性（如参数类型匹配、异常处理机制）缺乏自动化检测，可结合静态代码分析工具（如 SonarQube）实现上下文感知的兼容性评估。

后续工作需设计端到端的兼容性验证模块，融合环境元数据与代码静态分析技术，形成闭环推荐系统。

（3） 大语言模型驱动的适应性优化

尽管大语言模型（LLM）在代码理解任务中展现出潜力，但其在 API 推荐中的有效应用仍存在技术瓶颈：

提示工程敏感性：现有实验表明，简单提示难以激活 LLM 的深层 API 知识，需研究结构化提示模板与链式推理（Chain-of-Thought）策略，提升意图-API 的映射精度。

计算效率与成本：LLM 的高推理延迟与资源消耗限制其在实时推荐场景的应用，需探索知识蒸馏技术或轻量化适配器（Adapter）实现效率与性能的平衡。

与传统方法的融合机制：如何将 LLM 的语义泛化能力与本方法的结构化知识推理优势相结合，构建混合增强型推荐框架，是值得探索的方向。

（4） 推荐内容的针对性扩展

长尾 API 推荐优化：针对低频 API（使用率<0.1%）的冷启动问题，研究基于图神经网络的异构信息传播机制；

多模态输入支持：扩展方法对代码片段、流程图等非文本输入的理解能力，构建多模态 API 推荐系统；

开发者画像构建：通过历史行为分析建立开发者能力模型，实现个性化推荐策略。

本章展望了 API 推荐领域的核心问题并提供了粗浅的解决方案，但上述开放性挑战揭示了该方向仍具有广阔的研究纵深。后续工作需在理论创新与工程落地的交叉点上持续探索，推动智能化编程辅助工具向实用化、普适化方向演进。

参考文献

[1] Z. Yu, C. Bai, L. Seinturier, et al. “Characterizing the usage, evolution and impact of java annotations in practice,” IEEE Transactions on Software Engineering, vol. 47, no. 5, pp. 969–986, 2019.

[2] Hou D, Yao X. Exploring the intent behind api evolution: A case study[C]//2011 18th Working Conference on Reverse Engineering. IEEE, 2011: 131-140.

[3] Yu Z, Bai C, Seinturier L, et al. Characterizing the usage, evolution and impact of java annotations in practice[J]. IEEE Transactions on Software Engineering, 2019, 47(5): 969-986.

[4] 李正,吴敬征,李明树.API 使用的关键问题研究[J].软件学报,2018,29(06):1716-1738.

[5] 方波,赵锐.大数据背景下软件开发与维护对策分析[J].网络安全技术与应用,2024,(03):47-49.

[6] F. Thung, S. Wang, D. Lo, et al. “Automatic recommendation of api methods from feature requests,” in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE,2013, pp. 290–300.

[7] M. Raghothaman, Y. Wei, and Y. Hamadi. “Swim: synthesizing what i mean: code search and idiomatic snippet synthesis,” in Proceedings of the 38th International Conference on Software Engineering, 2016, pp.357–367.

[8] 张云帆,周宇,黄志球.基于语义相似度的 API 使用模式推荐[J].计算机科学,2020,47(03):34-40.

[9] Q. Huang, X. Xia, Z. Xing, et al. “Api method recommendation without worrying about the task-api knowledge gap,” in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 293–304.

[10] Y. Zhou, H. Jin, X. Yang, et al. Narasimhan, and H. C. Gall, “Braid: an api recommender supporting implicit user feedback,” in Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the F oundations of Software Engineering, 2021, pp. 1510–1514.

[11] M. Wei, N. S. Harzevili, Y et al. “Clear: Contrastive learning for api recommendation,” in 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 2022, pp.376–387.

[12] I. C. Irsan, T. Zhang, F. Thung, et al. “Picaso: Enhancing API recommendations with relevant stack overflow posts,” in 20th IEEE/ACM International Conference on Mining Software Repositories, MSR 2023, Melbourne, Australia, May 15-16, 2023, pp. 92–103, IEEE,2023.

[13] Y. Chen, C. Gao, M. Zhu, et al. “Apigen: Generative API method recommendation,” in IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2024, Rovaniemi, Finland, March 12-15, 2024. IEEE, 2024, pp. 171–182.

[14] Q. Huang, Z. Li, Z. Xing, et al. “Answering uncertain, under-specified API queries assisted by knowledge-aware human-ai dialogue,” IEEE Trans. Software

Eng., vol. 50, no. 2, pp. 280–295, 2024.
[15] Wang W, Godfrey M W. Detecting api usage obstacles: A study of ios and android developer questions[C]//2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 61-64.
[16] Wang J, Dang Y, Zhang H, et al. Mining succinct and high-coverage API usage patterns from source code[C]//2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 319-328.
[17] Agrawal R, Srikant R. Mining sequential patterns[C]//Proceedings of the eleventh international conference on data engineering. IEEE, 1995: 3-14.
[18] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements[C]//International conference on extending database technology. Springer, Berlin, Heidelberg, 1996: 1-17.
[19] Pei J. Mining sequential patterns efficiently by prefix-projected pattern growth[C]//International Conference of Data Engineering (ICDE2001), April. 2001.
[20] Ayres J, Flannick J, Gehrke J, et al. Sequential pattern mining using a bitmap representation[C]//Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. 2002: 429-435.
[21] Wang J, Han J. BIDE: Efficient mining of frequent closed sequences[C]//Proceedings. 20th international conference on data engineering. IEEE, 2004: 79-90.
[22] Tatti N, Vreeken J. The long and the short of it: summarising event sequences with serial episodes[C]//Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. 2012: 462-470.
[23] 贾倩倩. AUPLearner：上下文敏感的自更新 API 推荐方法[D].南京大学,2021
[24] 曹步清, 肖巧翔, 张祥平等.融合 SOM 功能聚类与 DeepFM 质量预测的 API 服务推荐方法[J].计算机学报,2019,42(06):1367-1383.
[25] C. Ling, Y. Zou, and B. Xie, “Graph neural network based collaborative filtering for api usage recommendation,” in 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2021, pp. 36–47.
[26] Li Y, Wang S, Wang W, et al. Rap4DQ: Learning to recommend relevant API documentation for developer questions[J]. Empirical Software Engineering, 2022, 27(1): 1-34.
[27] 杨忻莹. 基于用户反馈的 API 推荐方法研究[D].南京航空航天大学,2023.
[28] 肖海涛, 王鹏, 包义祥等. 一种基于调用序列网络的 API 推荐方法[J].软件导刊,2018,17(07):79-82.
[29] 段云浩,武浩.基于特征表示增强的 Web API 推荐[J].云南大学学报(自然科学版),2021,43(05):877-886.
[30] 奚耀国, 沈立炜, 赵文耘. 基于问答平台的弃用 API 迁移建议推荐技术[J].计算机应用与软件,2022,39(05):.
[31] Chen C, Peng X, Chen B, et al. “More Than Deep Learning”: post-processing for API sequence recommendation[J]. Empirical Software Engineering, 2022, 27(1): 1-32.

[32] Chen Y, Ren X, Gao C, et al. API Usage Recommendation via Multi-View Heterogeneous Graph Representation Learning[J]. arXiv preprint arXiv:2208.01971, 2022.
[33] Thung F. API recommendation system for software development[C]//2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2016: 896-899.
[34] Teyton C, Falleri J R, Blanc X. Mining library migration graphs[C]//2012 19th Working Conference on Reverse Engineering. IEEE, 2012: 289-298.
[35] Thung F, Lo D, Lawall J. Automated library recommendation[C]//2013 20th Working conference on reverse engineering (WCRE). IEEE, 2013: 182-191.
[36] Chen C, Xing Z. Similartech: automatically recommend analogical libraries across different programming languages[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016: 834-839.
[37] Nafi K W, Asaduzzaman M, Roy B, et al. Mining Software Information Sites to Recommend Cross-Language Analogical Libraries[C]//2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2022: 913-924.
[38] 康雨宁. 基于预训练语言模型的跨库 API 推荐技术的研究[D].天津大学,2021.
[39] Zhang C, Yang J, Zhang Y, et al. Automatic parameter recommendation for practical API usage[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 826-836.
[40] Asaduzzaman M, Roy C K, Monir S, et al. Exploring API method parameter recommendations[C]//2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2015: 271-280.
[41] Yang W. API Parameter Recommendations Made Effective[C]//2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2020: 675-676.
[42] Manh C T, Trung K T, Nguyen T M, et al. API parameter recommendation based on language model and program analysis[C]//2021 28th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2021: 492-496.
[43] 张云帆. 基于上下文信息的 API 使用模式和参数推荐方法研究[D].南京航空航天大学,2021. 000333.
[44] N. Azam and J. Yao, “Comparison of term frequency and document frequency based feature selection metrics in text categorization,” ExpertSyst. Appl., vol. 39, no. 5, pp. 4760–4768, 2012.
[45] T. Mikolov, I. Sutskever, K. Chen, et al. “Distributed representations of words and phrases and their compositionality,” in Proc. 26th Int. Conf. Neural Inf. Process. Syst., 2013, pp. 3111–3119.
[46] Y. H. Lee, D. W. Kim, and M. T. Lim, “A two-level recurrent neural network language model based on the continuous bag-of-words model for sentence classification,” Int. J. Artif. Intell. Tools,vol. 28, no. 1, pp. 1 950 002:1–1 950 002:16, 2019.
[47] J. Devlin, M. Chang, K. Lee, et al. “BERT: pre-training of deep bidirectional transformers for language understanding,” in Proceedings of the 2019 Conference

of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp.4171–4186.

[48] N. Ding, S. Hu, W. Zhao, et al. “Openprompt: Anopen-source framework for prompt-learning,” in Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022 - System Demonstrations, Dublin, Ireland, May 22-27, 2022, V. Basile, Z. Kozareva, and S. Stajner, Eds. Association for Computational Linguistics, 2022, pp. 105–113.

[49] P. Khosla, P. Teterwak, C. Wang, et al. A. Maschinot, C. Liu, and D. Krishnan, “Supervised contrastive learning,” in Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[50] M. M. Rahman, C. K. Roy, and D. Lo, “Rack: Automatic API recommendation using crowdsourced knowledge,” in Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reengineering, 2016, pp. 349–359.

[51] Y. Peng et al., “Revisiting, benchmarking and exploring API recommendation: How far are we?” IEEE Trans. Softw. Eng., vol. 49, no. 4, pp. 1876–1897, Apr. 2023

[52] J. Martin and J. L. C. Guo, “Deep API learning revisited,” in Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022, A. Rastogi,R. Tufano, G. Bavota, V . Arnaoudova, and S. Haiduc, Eds. ACM, 2022, pp. 321–330.

[53] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized BERT pretraining approach,” CoRR, vol. abs/1907.11692, 2019.

[54] E. Hoffer and N. Ailon, “Deep metric learning using triplet network,” in Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3. Springer, 2015, pp. 84–92.

[55] H. Touvron and e. Thibaut Lavril, “Llama: Open and efficient foundation language models,” 2023.

[56] A. Yang and e. Baosong Yang, “Qwen2 technical report,” 2024.

[57] G. Team and e. Morgane Riviere, “Gemma 2: Improving open language models at a practical size,” 2024.

[58] A. N. Lam, A. T. Nguyen, H. A. Nguyen, et al. “Combining deep learning with information retrieval to localize buggy files for bug reports (N),” in 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015, M. B. Cohen, L. Grunske, and M. Whalen, Eds. IEEE Computer Society, 2015, pp. 476–481.

[59] M. B. Zanjani, H. H. Kagdi, and C. Bird, “Automatically recommending peer reviewers in modern code review,” IEEE Trans. Software Eng., vol. 42, no. 6, pp. 530–543, 2016.

[60] M. Wen, R. Wu, and S. Cheung, “Locus: locating bugs from software changes,” in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016, D. Lo, S. Apel, and S. Khurshid, Eds. ACM, 2016, pp. 262–273.

[61] R. K. Saha, M. Lease, S. Khurshid, et al. “Improving bug localization using structured information retrieval,” in 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon V alley, CA, USA, November 11-15, 2013, E. Denney, T. Bultan, and A. Zeller, Eds. IEEE, 2013, pp. 345–355.

[62] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th In ternational Joint Conference on Natural Language Processing, EMNLP IJCNLP 2019, Hong Kong, China, November 3-7, 2019, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 3980–3990.

[63] Y. Liu, M. Ott, N. Goyal, et al. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized BERT pretraining approach,” CoRR, vol. abs/1907.11692, 2019.

攻读学位期间获得的成果

[1] Wang Yong, **Fang Yingtao**, Gao Cuiyun, Chen Linjun. API Recommendation for Novice Programmers: Build a Bridge of Query-Task Knowledge Gap, **IEEE Transactions on Reliability**, 2024 (导师一作学生二作，SCI，中科院 2 区)

[2] Wang Yong, Chen Linjun, Gao Cuiyun, **Fang Yintao**, Li Yong. Prompt enhance API recommendation: visualize the user's real intention behind this query, **Automated Software Engineering**, 2024, 31(27) (第四作者，SCI，中科院 2 区， CCF B 类)

[3] **Fang Yingtao,** Wang Yong, Gao Cuiyun, Shuai Meng, Li Zeng. API Recommendation via Multi-source Information Integration: Without Worrying about Expression Differences, **IEEE Transactions on Services Computing**. (under review，第一作者，CCF-A 类)

[4] **Fang Yingtao,** Wang Yong, Gao Cuiyun, Linjun Chen, Yifan Zhang. Similarity Pattern Enhancement for API Recommended Methods via Graph Representation Learning, **IEEE Transactions on Reliability**. (under review，第一作者，中科院 2 区)

致谢

白驹过隙，三年的研究生生涯如同一幅绚丽画卷，即将缓缓合上。这段旅程虽转瞬即逝，却色彩斑斓，满是难忘的回忆。回首往昔，怀揣着梦想启程，在奋斗中砥砺前行，凭借着坚韧不拔的毅力攻克一个又一个难关，那些挥洒汗水与泪水的日子，都化作了如今成长的基石，让我深知，所有的辛勤付出都必将换来累累硕果。值此毕业之际，心中满是感恩，我要向滋养我成长的母校献上最诚挚的谢意，也要向每一位陪伴我走过这段难忘岁月、给予我关心与支持的人，表达我衷心的感激与祝福。

首先，我要把最深切的感激之情献给我的导师—王勇导师。何其有幸，能在求学生涯中遇到您这样一位良师。初入校园，面对未来的学业方向，我满心迷茫。听闻您治学严谨却又平易近人，怀着忐忑的心情与您联系，承蒙您的接纳，我得以成为您门下的弟子。从那时起，您便如明灯照亮我前行的道路。

在学术研究的征程上，您始终耐心地引领着我。还记得初涉学术领域，面对浩如烟海的文献，我不知所措。是您，手把手教导我如何甄别优质论文，如何提炼核心观点，如何构建自己的研究思路。在您的悉心指导下，我逐渐掌握了学术研究的方法，从青涩走向成熟。读研期间，您为我制定了系统的学习计划，涵盖了专业领域内的前沿知识，督促我深入学习机器学习、模式识别等关键内容。每当我因难题后不知所措时，您会温和的鼓励我，随后以极大的耐心为我讲解，让我重燃斗志。这些知识在我后续的学术研究和项目实践中，犹如基石，支撑着我不断向上攀登。我还要感谢崔琳老师在我读研生涯中的帮助，您不仅给予我生活中的支持，更以身作则地传递了求真务实的学术品格。

其次，我要向 PRISE 小组的各位老师和同学们致以衷心的感谢。每周六的组会，是知识的盛宴，也是成长的阶梯。在论文分享与进度汇报中，我感受到了浓厚的学术氛围，这不仅督促我不断学习新知识，更让我在交流中收获了许多宝贵的见解。感谢卢老师、李老师，唐老师等各位老师在组会上给予的专业指导，你们丰富的学识和独到的见解，拓宽了我的学术视野，让我在迷茫时找到方向。感谢课题组的同学们，与你们共同学习、共同进步的日子，是我研究

生生活中最美好的回忆。我们一起探讨学术难题，分享生活趣事，营造了一个温馨、和谐、积极向上的学习氛围。你们的知识分享与无私帮助，让我收获颇丰，衷心感谢你们的陪伴与支持。

此外，我要感谢我的朋友们，感谢你们出现在我的生命里。这三年，我们一起笑过、闹过、奋斗过、迷茫过。是你们的陪伴，让我的研究生生活充满阳光；是你们的支持，让我有勇气面对困难；是你们的鼓励，让我不断成长进步。毕业在即，虽有万般不舍，但我坚信，我们的情谊不会因距离而疏远。愿大家在未来的日子里，都能得偿所愿，熠熠生辉。无论相隔多远，无论岁月如何变迁，这份情谊都如陈酿的美酒，愈发香醇浓厚。

接下来，我要感谢我的同门师兄弟们，毕业的钟声即将敲响，我们也即将奔赴各自的前程。但我深知，这段同门情谊将永远铭刻在我心中。感谢你们在这三年里的陪伴、帮助与支持，是你们让我的研究生生活变得如此丰富多彩。愿我们在未来的日子里，都能在各自的领域发光发热，实现自己的理想与抱负。

最后，我要感谢我的家人。父母那无私的爱与默默的支持，是我前行的动力源泉。无论我遇到什么困难，他们总是在背后默默鼓励我，给予我温暖的港湾。他们的辛勤付出与殷切期望，让我能够心无旁骛地追求学业。

毕业是一段旅程的结束，更是新征程的开始。感恩这一路的相遇与陪伴，我将带着这份温暖与力量，勇敢地迈向未来。祝愿母校越办越好，桃李满园；祝愿各位老师身体健康，学术有成；祝愿同学们前程似锦，一帆风顺。