从业者对代码补全的预期

Chaozheng Wang Junhao Hu Cuiyun Gao Yu Jin哈尔滨工业大学，北京大学，哈尔滨工业大学，腾讯，深圳，中国 北京，中国 深圳，中国 广州，中国 wangchaozheng@stu.hit.edu.cn junhaohu@stu.pku.edu.cn gaocuiyun@hit.edu.cn lenajin@tencent.com

Tao Xie Hailiang Huang Zhenyu Lei Yuetang Deng北京大学，腾讯，腾讯，腾讯，北京，中国 广州，中国 广州，中国 广州，中国 taoxie@pku.edu.cn eraserhuang@tencent.com rainylei@tencent.com yuetangdeng@tencent.com

arXiv:2301.03846v1 [cs.SE] 10 Jan 2023

摘要—**代码补全已成为程序员日常编程活动中的常见做法。它旨在自动预测程序员倾向于使用的下一个词元或代码行。优秀的代码补全工具能够显著减少程序员的击键次数，提高编程效率。近年来，研究人员提出了各种可供实际使用的代码补全技术。然而，从业者对代码补全有何预期，以及现有研究是否满足了他们的需求，仍不清楚。为弥补这一空白，我们首先访谈了 15 位从业者，随后对来自 18 家 IT 公司的 599 位从业者开展调查，以了解他们对代码补全的预期。接着，我们通过文献综述，对比从业者的需求与当前研究，梳理了 2012 至 2022 年发表于顶级出版渠道的代码补全论文。基于这一对比，我们指出了研究人员值得投入精力的方向，以开发满足从业者预期的代码补全技术。**

索引词—**代码补全、实证研究、从业者预期**

I. 引言

代码补全旨在预测程序员在日常编程活动中通常会输入的代码词元或代码行。它可以大幅减少程序员的击键次数并提高编程效率。正如 [1],报告的那样，使用代码补全时，程序员平均每完成一个词元可节省 3.65 次击键。代码补全已成为现代集成开发环境（IDE）中使用最频繁的功能 [2], ，并日益受到学术界和工业界的关注 [3]–[13]。

传统代码补全技术 [14]–[16] 采用静态分析，并根据预定义规则提供补全候选项，通常需要投入大量人工工作，且效果不佳。为实现整个过程的自动化，研究者提出了基于机器学习（ML）的方法 [17]–[19] ，例如 N 元语法模型，用于学习代码统计特征，从而能够泛化到新的代码。随着深度学习（DL）的发展，许多基于 DL 的方法 [3], [20]相继提出，它们能够隐式学习代码中的模式，并达到最先进的性能。一些基于 DL 的技术 [5], [8], [10], [21]采用预训练语言模型（PLM），例如 BERT 和 GPT [22]–[24], 用于代码补全，因为 PLM 是在大型未标注语料库上训练的，能够将大量代码知识编码到大规模参数中。除了准确预测下一个词元外，基于 PLM 的方法在预测接下来的代码行方面也表现出色。

尽管针对代码补全开展了大量研究，但遗憾的是，尚无先前研究探讨从业者对相关技术的预期。目前尚不清楚从业者是否认可现有的代码补全技术，以及他们在采用过程中最关注哪些方面（例如，补全效率和有效性）。从业者的视角对于揭示关键问题至关重要，也能为软件工程研究人员提供指导，帮助他们创造出满足程序员需求的解决方案。

在本文中，我们采用混合方法研究 [25],[26] ，深入了解从业者对代码补全的预期。首先，我们对 15 位专业人士进行了半结构化访谈，他们的软件编程经验平均为 8.4 年。通过访谈，我们定性考察了代码补全实践的现状、受访者在使用代码补全工具时遇到的问题，以及他们对代码补全的预期。随后，我们对来自 18 家 IT 公司的 599 位专业人士开展探索性调查，以定量验证访谈中发现的从业者预期。最后，我们回顾了 2012 年至 2022 年间发表于顶级会议和期刊的研究论文，并将论文中提出的技术与从业者的预期进行比较。

具体而言，我们探讨以下四个研究问题：

RQ1：代码补全实践的现状是什么?

这个研究问题探讨代码补全实践，包括代码补全工具和使用场景（从业者使用代码补全的场景）。对于补全工具，分别有 72% 和 54% 的参与者表示，他们在日常编程中经常使用代码补全，并且期待更好的代码补全工具。与此同时，集成开发环境中的内置工具远比第三方插件式补全工具更受欢迎，例如 Copilot [11] 和 IntelliCode [8], 。

参与者中分别有 96% 和约 13% 使用了这些工具。

关于使用场景，为便于调查，我们根据补全粒度对代码补全技术进行了粗略分类，包括词元级和语句级。平均而言，81%的参与者采用词元级补全，明显高于采用语句级补全的参与者（仅32%）。对于词元级补全，分别有84%和85%的参与者经常使用工具来补全标识符和推荐API。对于语句级补全，补全当前编辑行和预测 API 参数是最受欢迎的两种使用场景。

研究问题2：代码补全对从业者重要吗？存在哪些问题?

在这一研究问题中，我们调查从业者如何看待代码补全的重要性，并研究从业者在编程过程中面临的问题。对于从业者对其重要性的看法，87%的人非常同意或同意代码补全在软件开发过程中的重要性。对于代码补全的问题，只有36%的参与者认为当前的代码补全工具令人满意。分别有56%和58%的参与者认为错误补全和繁琐的工具安装是主要问题。

研究问题3：从业者对代码补全工具有哪些预期?

本研究问题聚焦于从业者对不同粒度代码补全工具的预期，包括词元级补全和语句级补全。我们从多个方面调查这些预期，包括使用场景、评估指标、服务访问（在线或离线）、补全有效性和效率。平均而言，88%和57%的参与者分别期待词元级补全场景和语句级补全场景。对于词元级补全，约90%的参与者期待工具能够补全标识符、推荐 API，以及补全路径（即所需词元与文件/目录路径相关时）。如果补全工具能在前3个候选项中提供合适的词元，79%的参与者会感到满意。大多数参与者（85%）期待工具在200毫秒内生成补全结果。对于语句级补全，最受期待的场景是骨架补全 （预测类和方法的骨架）、*API* 参数推荐 （推荐被调用 API 的参数）以及当前编辑行补全。最受青睐的补全延迟是不超过2秒。对于评估指标，大多数参与者在采用词元级和语句级补全技术时，分别关注补全准确率（79%）和语法正确性（86%）。**研究问题4：当前研究在多大程度上满足了从业者的需求?**

这个研究问题旨在探讨代码补全研究，并分析研究与从业者预期之间的差距。我们找到了 26 篇关于代码补全技术的论文，其中 17 篇提出了词元级补全，10 篇涉及语句级补全。关于词元级补全的研究，大多聚焦于*API* 推荐 和标识符补全 场景，这与从业者的预期一致。然而，没有论文涉及路径补全场景，而绝大多数从业者都期待这一场景。关于语句级补全的研究，大多数论文聚焦于下一行补全，但只有少数论文探讨从业者最期待的骨架补全 和*API* 参数预测 。此外，大多数论文都使用重叠 *N* 元语法 和编辑相似度来衡量补全代码与人工编写的代码之间的差异，但大多数参与者并不偏好这些指标。

我们的研究旨在为代码补全领域未来的研究指明方向，并帮助研究人员在研究代码补全技术时考虑从业者的预期。我们希望研究人员能够提出更好的代码补全工具，最终将其应用于编程实践，满足广大程序员的需求。

总之，我们做出了以下贡献:

- 我们是首批关注从业者对代码补全工具预期的研究者。我们先访谈了 15 位从业者，随后对来自 18 家 IT 公司的 599 位从业者开展了调查。我们探究了他们对代码补全工具的看法、使用代码补全的场景，以及他们对代码补全工具的预期。
- 我们全面回顾了过去十年发表于软件工程和人工智能领域顶级会议与期刊的论文。随后，我们将已发表的论文与从业者的预期进行比较，指出研究人员应在哪些方面改进代码补全，以满足从业者的需求。

论文结构：第二节介绍本研究的方法。第三节呈现本研究的结果。第四节讨论研究结果的意义。第五节讨论相关工作。

II. 研究方法

本节介绍我们的整体研究方法。我们的研究分为三个主要阶段。**阶段 1：** 访谈专业人士，了解他们在使用代码补全时的实践、遇到的问题以及对代码补全工具的预期。**阶段 2：** 开展在线调查，以验证并拓展基于访谈得出的从业者预期。**阶段 3：** 开展文献综述，分析现有技术是否以及在多大程度上满足了从业者的需求。访谈和调查均已获得相关机构审查委员会（IRB）的批准。

A. 阶段 *1*：访谈

1）流程：两位作者依据访谈指南开展一系列面对面、半结构化、深入的访谈，以了解参与者在代码补全方面的实践、问题及

[1]在 [10]中提出的方法可以进行词元级和语句级代码补全。

```
from sklearn.ensemble import RandomForestClassifier
X_train = [1,2,3,4]
Y_train = [1,0,1,0]
model = RandomForestClassifier(n_estimators=2)
model.fit(X_train, Y_train)
```

图 1：用于说明调查中*API* 参数预测场景的示例。

预期。我们邀请了来自全球七家 IT 公司的 15 位编程专业人士参与访谈。每次访谈持续 45–60 分钟。

在每次访谈中，我们首先会询问受访者一些人口统计学问题，了解他们的背景，包括工作角色和工作经验。然后，我们请受访者自由谈谈他们认为优秀的代码补全工具应具备哪些特点，以及他们对代码补全技术的预期。最后，我们了解他们的代码补全实践以及曾遇到的问题。

2）受访者：我们邀请了来自 IT 行业人脉网络的受访者。受访者均为全职从业者，担任开发人员和人工智能算法设计人员等不同角色。最终，来自七家 IT 公司的 15 名受访者同意参加访谈。受访者的编程经验为 5 至 13 年，平均为 8.4 年。

3）数据分析：第一作者将访谈内容转录下来，并将受访者的观点归纳为观点卡片。随后，另一位作者核查归纳后的观点卡片，并提出改进建议。采纳这些建议后，两位作者分别分析并整理观点卡片，将其归纳为问卷的潜在描述。两位作者之间的科恩卡帕系数为 0.67，表明双方具有较高的一致性。两位作者讨论了彼此意见不一致之处，以达成共识。为减少两位作者在整理描述时产生的偏差，另外两位作者也审阅并确认了最终的调查描述。

访谈结束时，我们归纳出四种代码补全实践和六个问题。我们分别总结了从业者对词元级和语句级代码补全的预期所涉及的三种和六种使用场景。此外，我们分别归纳出八个和十三个会具体影响词元级补全和语句级补全采用的因素，并总结了通用代码补全工具的另外三个方面。

B. 阶段 *2*：在线调查

1) 调查设计：调查包含不同类型的问题，包括单选题/多选题、李克特量表（选项：非常不同意、不同意、中立、同意、非常同意）以及简答题。对于每个问题，我们都添加了“我不理解”这一选项，以筛除不理解我们描述的参与者。

该调查由六个部分组成：

- **人口统计信息**。该调查首先询问参与者的人口统计信息，包括主要工作角色和编程体验。
- **代码补全工具**。我们询问参与者有关代码补全工具的相关因素。具体而言，我们调查了他们曾经使用过哪些代码补全工具，以及他们使用代码补全工具的实践。
- **代码补全使用场景**。本部分旨在调查从业者会在什么场景下使用代码补全工具。我们总结了词元级补全的三种场景和语句级补全的六种场景。对于每种场景，我们都使用截图或 GIF 进行说明，图 1 展示了一个示例。三种词元级补全场景包括标识符补全、*API* 推荐和路径补全（即补全所引用文件或目录的路径）。研究的语句级补全场景包括下一行预测、当前编辑行补全、*API* 参数推荐、字符串补全（例如补全日志）、骨架预测和代码块内容预测（例如预测一个方法块）。

- **工具的重要性**。在这一部分，我们调查从业者对代码补全工具的态度，并询问参与者如何看待这些工具的重要性。
- **代码补全问题**。这一部分调查参与者在使用代码补全工具时遇到的问题，例如错误补全、补全延迟过高、高资源消耗以及对数据泄露的担忧。

- **从业者的预期**。在这一部分，我们研究从业者对两种粒度级别的代码补全工具的预期，即词元级补全和语句级补全。针对每种粒度级别，我们会询问多个方面的预期，包括使用场景、评估指标、服务访问（在线或离线）、补全有效性和效率。此外，我们还会调查影响从业者采用代码补全工具意愿的一般因素，例如工具安装耗时。

在调查问卷末尾，我们允许参与者自由提供关于代码补全和本调查的评论、建议和意见。

为了了解参与者对调查时长和描述清晰度的看法，我们在大规模发放前，对一小部分与访谈对象及调查参与者不同的从业者开展了预调查。根据收到的反馈，我们对调查做了少量修改，并形成最终版本。我们利用广泛使用的问卷网站 [27], [28] 发放调查。

2）参与者招募：我们通过社交网络联系在 IT 公司全职工作的专业人士，请他们协助填写并传播这份调查问卷。具体而言，我们向在 Microsoft、Intel、腾讯、阿里巴巴及其他 IT 公司工作的专业人士发送邀请。

表 I：参与者的角色与工作经验

Role	人数	<1y	1-3y	3-5y	5-10 年	>10y
开发	345	32	102	121	65	15
算法设计	173	10	46	81	42	4
测试	27	6	8	4	6	3
架构师	4	0	1	2	1	0
项目经理	2	0	0	0	1	1
其他	48	9	15	17	7	0
	599	57	172	225	122	23

我们最终共收到 611 份问卷回答，平均完成时间为 8.2 分钟。其中，我们剔除了 12 份在两分钟内完成的答复。本文呈现的调查结果基于剩余的 599 份有效响应进行分析。表 I 展示了受调查参与者的角色及其体验概况。大多数参与者从事软件开发，并拥有 3 至 5 年的编程体验。

3）结果分析：我们根据问题类型分析结果。对于多项选择题和单项选择题，我们报告每个选项被选中的比例。对于李克特量表题，我们绘制柱状图来展示李克特评分的分布。对于开放式简答题，我们对结果进行定性分析。此外，我们剔除“I don’tunderstand”评分，因为这类评分在所有收到的评分中占少数（不到 1%）。

C. 第三阶段：文献综述

关于代码补全的论文通常发表于软件工程和人工智能领域。因此，我们查阅了 2012 年至 2022 年发表在 ICSE、ESEC/FSE、ASE、ICPC、SANER、MSR、ICSME、PLDI、OOSPLA、TSE、TOSEM、EMSE、ACL、EMNLP、NAACL、IJCAI、ICLR、NeurIPS 和 AAAI 上的研究论文。我们选择上述会议和期刊上的论文，是因为它们是软件工程和人工智能领域的顶级发表渠道。

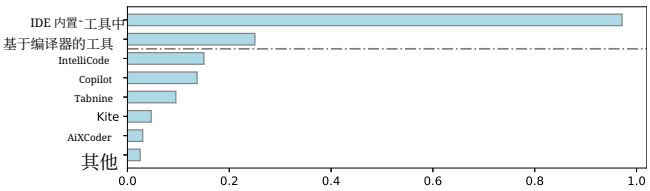
我们首先阅读论文的标题和摘要，以确认其是否与代码补全有关。对于每篇代码补全论文，两位作者会阅读其内容，并从补全粒度、应用场景、评估指标、代码补全服务的访问方式以及效率等方面分析所提方法的能力。例如，Izadi等人 [10] 声明，他们利用多任务学习训练模型，以预测下一个词元和下一条语句。因此，我们推断该论文同时研究词元级和语句级补全。Svyatkovskiy等人 [3] 声称提供网页补全服务和客户端补全模型，因此我们推断其服务既可在线访问，也可离线访问。两

作者讨论能力分析中的差异，并通过进一步阅读论文确认最终结果。

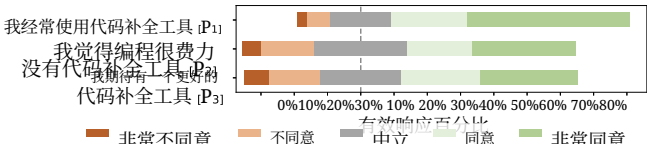
III. 结果分析

A. RQ1：代码补全实践

在这个研究问题中，我们探讨从业者的代码补全实践，包括所使用的代码补全



(a) 从业者使用过的代码补全工具统计。



(b) 代码补全工具的使用情况。

图2：代码补全工具。

工具和使用场景。参与者对一些与代码补全实践相关的描述的评分结果如图2和图3所示。

1) 代码补全工具：我们首先询问参与者使用过哪些代码补全工具，结果如图2(a)所示。结果显示，96%的参与者表示他们采用了 IDE 内置代码补全工具。基于编译器的工具（如 *CCLS* 和 *Clangd* [14] ）是受访从业者中第二受欢迎的工具。然而，第三方插件工具并不普及，例如，约13%的参与者使用过 Copilot [11]和 IntelliCode[8]。

随后，我们调查了从业者使用代码补全工具的情况，结果如图2(b)所示。尽管大多数从业者（72%）表示他们在日常编程中经常使用代码补全，但只有50%的参与者同意，没有代码补全时编程会很费力。此外，54%的参与者会主动寻找更好的代码补全工具。一位参与者表示：“我一直关注新代码补全工具的发布，并渴望成为第一个使用它们的人。”

发现1： 最常用的代码补全工具是 IDE 内置工具。其他第三方插件工具，如 IntelliCode [7]和 Copilot [11]，远不如它受欢迎，仅约 13% 的参与者使用过。此外，大多数参与者表示他们经常使用代码补全工具，其中 54% 的参与者期待更好的代码补全工具。

2）使用场景：本节探讨从业者在什么场景下使用代码补全。根据 [29],，我们从两种粒度研究这些场景，分别是词元级补全和语句级补全。词元级补全旨在补全开发人员当前正在输入的词元以及下一个要使用的词元。语句级补全则能够预测多个词元，甚至多行代码。为帮助参与者理解，我们在调查中给出了这两种粒度级别补全的定义。词元级补全和

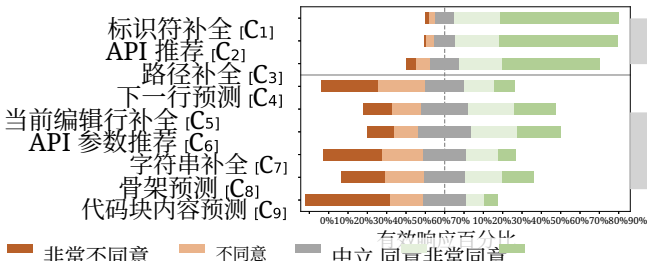


图 3：从业者使用代码补全的场景。

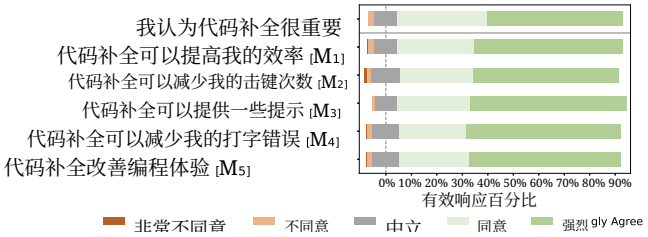


图4：代码补全的重要性。

语句级补全分别涵盖三种和六种场景，如第二节-B所述。参与者对使用场景的评分如图3所示。我们观察到，语句级补全场景的整体受欢迎程度明显低于词元级补全场景（平均分别为32%和81%）。

词元级补全。如图3上半部分所示，三种使用场景在从业者中都很受欢迎。具体而言，约85%的参与者使用代码补全来补全标识符和推荐 API。超过70%的受调查从业者在日常编程中采用代码补全进行路径补全。

发现2： 三种词元级代码补全使用场景都很受参与者欢迎，其中标识符补全 和*API* 推荐 是使用最广泛的场景。

语句级补全。如图3下半部分所示，最受欢迎的三种语句级补全场景是*API* 参数推荐、当前编辑行补全和骨架预测，使用者比例分别为46%、45%和36%。此外，约25%的参与者采用代码补全来预测下一行代码和字符串内容。只有16%的受调查参与者在代码块内容预测场景中采用代码补全。一位参与者表示：“现有补全工具的性能不足以预测整个代码块内容。在某些情况下，预测结果可能会产生严重误导，从而降低我的编程效率。”

发现3： 采用最多的语句级代码补全场景包括*API* 参数推荐、当前编辑行的补全和骨架预测。

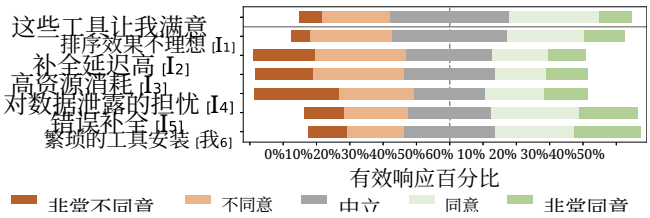


图5：代码补全的问题。

B. 研究问题*2*：代码补全的重要性与问题 *ues*

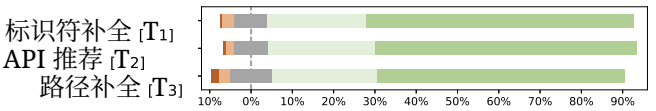
1）代码补全的重要性：我们询问受访者是否认为代码补全对开发很重要，以及他们是否同意代码补全重要的可能原因。结果如图4所示。

从结果中可以发现，大多数参与者（88%）认为代码补全发挥着重要作用。90% 的参与者同意或非常同意代码补全能够为代码实现提供一些提示。一位从业者这样描述：“有时我会忘记如何输入较长的标识符或方法名。这种情况下，代码补全可以显示候选项，并为完成代码提供一些提示。”。此外，超过 85% 的参与者认为，补全工具有助于减少打字错误、提高编程效率、减少击键次数并改善编程体验。

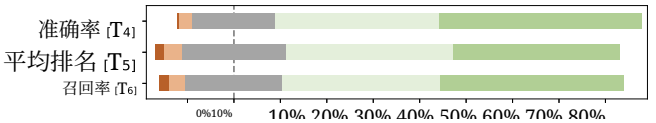
发现4： 88% 的调查参与者同意代码补全工具在编程中发挥着重要作用。大多数参与者（90%、87% 和 86%）认为，这些工具能够提供一些提示、提高编程效率并改善编程体验。

2）代码补全问题：图5展示了参与者对他们在编程过程中遇到的代码补全相关问题的评分。尽管代码补全很重要，但只有 36% 的参与者认为他们目前使用的工具令人满意。分别有 45% 和 44% 的参与者认为，错误补全和繁琐的工具安装是主要问题。错误补全是指补全代码中出现了一些错误，需要从业者进行修改。正如一位参与者所说：“补全结果通常并不完全符合我的需求，而且纠正其中的错误让我很恼火。”。此外，分别有 34% 和 31% 的参与者对候选项中合适词元的排序不满意，以及担心工具会造成数据泄露。约四分之一的参与者认为，高资源消耗（即内存和 CPU）以及高补全延迟令人无法接受。一位从事大型 C++ 项目的参与者告诉我们：“*C++* 代码补全工具（如 *CCLS* 和 *Clangd*）会在编程时解析已打开的 *cpp* 文件并为其建立索引，这可能会耗尽我的 *CPU* 和内存。”。

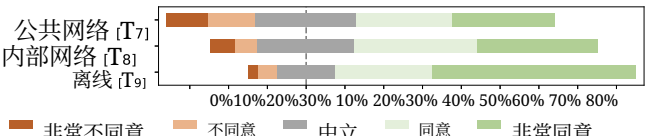
发现 5： 从业者最关注的问题是错误的代码补全、令人头疼的工具安装以及不尽如人意的排序。



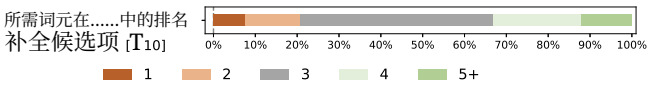
（a）对使用场景的预期。



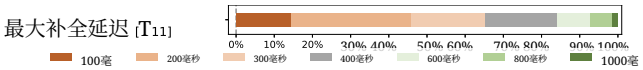
（b）对评估指标的预期。



（c）对获取代码补全服务的预期。



（d）对代码补全效果的预期。



（e）对代码补全效率的预期。

图6：对令牌级代码补全的预期，横轴表示有效答复的百分比。

C. 研究问题*3*：从业者的预期

在这个研究问题中，我们全面调查了从业者对代码补全的预期。我们探讨的方面包括使用场景、评估指标、服务访问、补全有效性和效率。我们报告了从业者对代码补全两种粒度层级的预期，即词元级和语句级补全。我们还调查了从业者对影响其使用体验的通用补全工具其他三个方面的预期，包括安装耗时、个性化补全和附加信息显示。

1) 研究问题 *3.1* 对词元级补全的预期：参与者对令牌级代码补全预期的评分如图6所示。

使用场景。 从图 6 (a) 可以看出，大多数参与者对这三种场景持积极态度。超过 88% 的参与者希望在标识符补全和*API* 推荐场景中使用代码补全。此外，86% 的参与者希望代码补全工具能够补全他们正在输入的路径。某位参与者认为，当他/她在大型项目中工作时，路径补全 非常重要，因为没有路径补全时，文件名很容易出错。

发现 6： 对于令牌级代码补全，超过 80% 的参与者同意，工具应该支持标识符补全、*API* 推荐和路径补全。

评估指标。我们在图6 (b) 中研究了参与者对评估词元级代码补全工具的看法。我们观察到，准确率获得的支持率最高（79%）。此外，超过70%的参与者也关注所需词元在候选项中的平均排名（例如平均倒数排名 MRR）以及工具的召回率。

服务访问。图6 (c) 显示，82%的参与者希望离线使用词元级补全工具。一位参与者表示：“离线工具最稳定，不会受到网络状态的影响，而且我可以在任何地方使用。”相比之下，在线工具的支持率较低。分别有62%和50%的参与者愿意在内部网络和公共网络中使用在线工具。

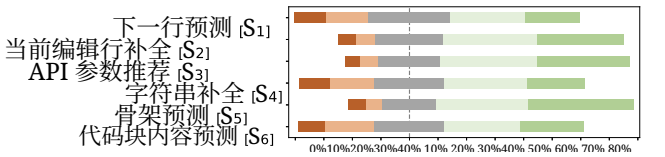
有效性。 图6（d）展示了从业者对所需词元补全候选项不同排名的满意度比例。根据我们的调查，如果代码补全工具能将所需词元排在前3个候选项之中，就能满足约80%的从业者。

效率。 我们向参与者展示了7个 GIF 动图，分别对应不同的补全延迟：100、200、300、400、600、800 和 1000 毫秒。从图6（e）可以看出，如果词元级代码补全工具能在200毫秒内提供补全候选项，就能满足85%的从业者。

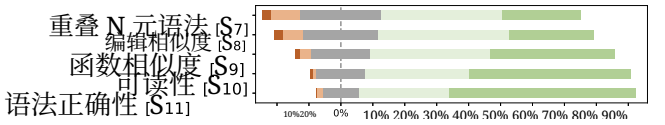
2）RQ 3.2 对语句级补全的预期：参与者对语句级补全的预期结果见图7。

使用场景。 图7（a）展示了从业者对语句级代码补全场景的预期。我们发现，从业者的预期趋势与第三节 A 部分所示的各使用场景的受欢迎程度相似。同样，最受期待的三个场景是骨架预测、当前编辑行补全和*API* 参数推荐，这三者的支持率均至少达到60%，其中骨架预测 最受期待。此外，51%的参与者同意，语句级补全工具应当预测代码块内容，例如方法块和循环块。53%的参与者希望使用代码补全工具预测字符串。有趣的是，尽管预测下一行代码与补全当前编辑行有些相似，但只有43%的参与者认为下一行预测 是重要场景，而当前编辑行补全 的支持率则达到62%。正如一位参与者所评论：“完成一行代码后，我会立即开始输入下一行，而不是等到行末才查看补全结果。”。

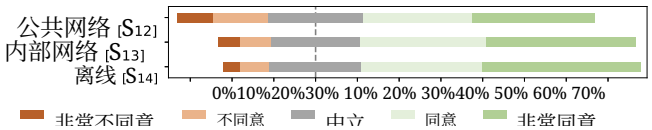
发现 7： 骨架预测、当前编辑行补全 和*API* 参数推荐被认为是最受期待的语句级代码补全场景，与各使用场景的受欢迎程度相似。



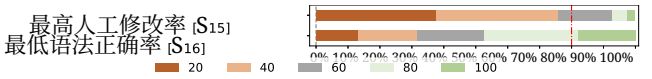
(a) 对使用场景的预期。



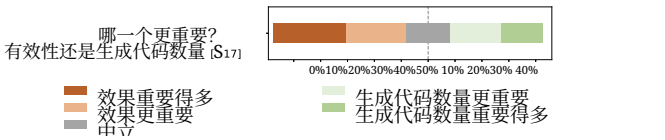
(b) 对评估指标的预期。



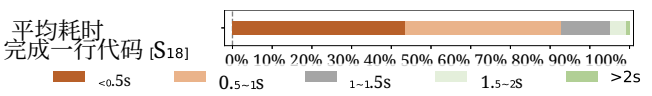
(c) 对获取代码补全服务的预期。



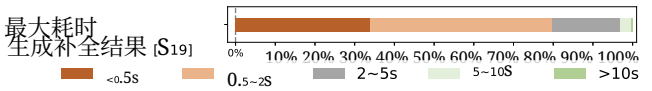
(d) 对代码补全效果的预期。



(e) 有效性与生成代码数量。



(f) 对生成一行代码的平均耗时的预期。



（g）对代码补全最大耗时的预期结果。

图7：对语句级代码补全的预期，其中横轴表示有效回答的百分比。

评估指标。我们在图7（b）中展示了从业者对语句级代码补全方法所采用的不同评估指标的期望。最受青睐的三项评估指标是语法正确性、可读性和功能/结构相似度，获得了超过 80% 的参与者支持。然而，重叠 n 元语法 （例如 BLEU 分数 [30]）和编辑相似度 获得的支持率最低（分别为 65% 和 70%）。

服务访问。 图7（c）展示了使用语句级代码补全工具时，不同服务访问方式的可能性。尽管与词元级补全相比，参与者对离线服务的支持率下降了15%（从82%降至67%），离线工具仍是最受期待的。不过，参与者采用在线语句级代码补全工具的意愿有所上升。例如，66%的参与者同意或非常同意在内部网络中调用在线补全服务（词元级补全的这一比例为62%）。此外，参与者对公共网络工具的支持率也从50%上升至58%。一位参与者告诉我们：“目前，几乎所有语句级补全工具都基于人工智能，这意味着其计算复杂度很高。因此，使用在线代码补全服务是可以接受的。”此外，我们观察到参与者对部署在公共网络和内部网络中的工具的预期存在差异（分别为58%和66%）。根据我们的调查，这种差异可归因于参与者对公共网络中数据泄露和网络稳定性的担忧。一位参与者表示，在其所在部门，由于 Copilot 会将代码上传到云端，可能导致机密代码泄露，因此不允许安装。

有效性。 从图7（d）中我们发现，只有26%的参与者能够接受人工修改超过补全代码60%的情况。此外，我们的参与者中有35%和39%的人希望最大修改比例分别不超过20%和40%。对于语法正确性，如果语句级代码补全工具能够确保至少80%的补全代码在语法上正确，就可以满足82%的从业者。此外，18%的参与者对生成代码的正确性有严格要求，即他们希望工具在代码语法方面不出任何错误。

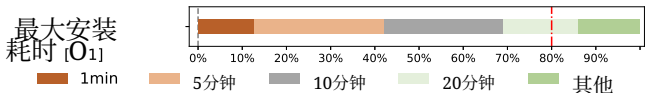
根据现有工作 [9], [21],，补全的代码越多，出现的错误就越多。我们进一步调查了从业者在补全效果（生成更少的代码，但准确率更高）和生成代码数量（生成更多代码，但需要进行更多人工修改）之间的偏好。结果如图7（e）所示。根据我们的调查，两者之间没有明显的胜者（有效性的支持率为46%，生成代码数量的支持率为40%），并且有略多一些参与者认为有效性更重要。在这个问题上，参与者主要有两种观点：

1) **有效性更重要**。对于重视有效性的从业者来说，有错误的补全结果可能会影响他们的效率。一位参与者分享了自己使用语句级代码补全工具的体验：“修改补全结果对我来说非常烦人。.....我宁愿少补全一些代码，也不愿修改补全结果。”

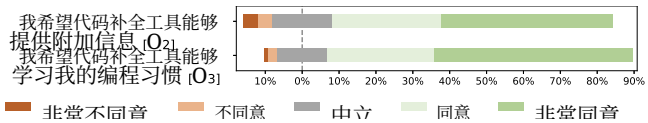
2) **代码数量更重要**。重视代码数量的从业者期望语句级补全工具一次预测更多代码，即使这些代码还需要修改。一位参与者表示，只要工具能够按照正确的逻辑预测代码（例如，正确的代码框架），在他看来，这个工具就是有效的。

效率。我们从两个方面研究从业者对语句级代码补全工具效率的预期：补全一行代码的平均耗时（见图7 (f)）以及生成补全结果的最大延迟（见图7 (g)）。

根据我们的调查，我们发现，平均每行代码生成耗时 0.5 秒的速度支持率最高，



(a) 对工具安装耗时的预期。



(b) 对附加信息显示和个性化补全的预期。

图 8：对通用代码补全工具其他方面的预期，横轴表示有效响应百分比。

达到 44%；39% 的参与者希望耗时不超过 1 秒。对于最大补全延迟，希望代码补全工具能在 2 秒内生成补全结果的从业者占比最高（即 47%）。如果语句级代码补全工具能在 2 秒内生成补全结果，就能满足 67% 的从业者。此外，只有少数参与者（即 4%）表示愿意等待超过 5 秒来获取补全结果。

发现 8： 在补全效率方面，大多数从业者预期语句级代码补全工具平均用时少于 0.5 秒即可补全一行代码。此外，为满足大多数（67%）从业者的需求，工具应在 2 秒内生成补全结果。

3) RQ 3.3 对通用代码补全工具其他方面的预期：本节介绍从业者对代码补全一般方面的预期，结果如图 8 所示。

图 8(a) 展示了从业者对安装代码补全工具所需时间的预期。如果能在 5 分钟内安装好工具，83% 的参与者会感到满意。此外，我们发现 14% 的参与者愿意花 20 分钟以上安装工具。正如一位参与者所说：“只要代码补全工具确实能提高我的效率，我愿意花一整天来安装它。”

从图 8 (b) 可以看出，大多数（76%）从业者同意或非常同意代码补全工具应当学习他们的编程习惯（例如，对函数名和变量的偏好）。此外，83% 的从业者希望代码补全工具在展示预测代码的同时提供一些附加信息（例如，在推荐 API 时向用户展示 API 定义或文档）。一位参与者告诉我们：“有时我会忘记 *API* 中参数的顺序，因此向我展示 *API* 的定义可以大大减少我的编程错误。”

表 II：当前研究的能力。李克特量表得分表示参与者评分的加权平均分（1 至 5 分分别对应非常不同意至非常同意）。

	描述		李克特量表得分	论文
Token-level completion	使用场景			
	标识符补全	[T1]	4.50	[5], [10], [31]–[35]
	API 推荐	[T2]	4.50	[3], [17], [31], [36]–[38] [4]–[6], [32], [33], [35] [7], [10], [19], [34], [39]
	路径补全	[T3]	4.39	-
	评估指标			
	准确率	[T4]	4.19	[10], [17], [32], [34]
	平均排名	[T5]	4.00	[3], [5], [7], [36], [38]
	召回率	[T6]	4.05	[19], [33], [34], [36] [7], [31], [35], [39] [3], [6]
	其他			[17], [32], [36], [37]
	服务访问			
	在线	[T7]	3.43	[3], [10]
	离线	[T9]	4.20	[3], [7], [17]
	耗时			
	提及			[6], [32]–[34], [38], [39] [3], [7], [10], [17], [36]
Statement-level completion	使用场景			
	下一行预测	[S1]	3.27	[8]–[10], [21] [20], [37], [40], [41]
	当前编辑行补全	[S2]	3.74	[8], [20], [21]
	API 参数推荐	[S3]	3.81	[8], [21], [34]
	字符串补全	[S4]	3.32	-
	骨架预测	[S5]	3.88	-
	代码块内容预测	[S6]	3.43	[21], [40], [42]
	评估指标			
	重叠 N 元语法	[S7]	3.72	[9], [21], [37], [40]
	编辑相似度	[S8]	3.81	[8], [37], [40], [42]
	函数相似度	[S9]	4.09	[9]
	可读性	[S10]	4.31	-
	语法正确性	[S11]	4.42	-
	其他			[8], [20], [40]–[42]
	服务访问			
	在线	[S12]	3.54	[8], [10], [42]
	离线	[S14]	3.89	[41]
	耗时			
	提及			[8], [20], [40], [41]

D. RQ4：当前最先进的研究

文献回顾后，我们从软件工程和人工智能领域的顶级会议和期刊中共筛选出 26 篇论文。表 II 展示了受调查的代码补全技术所具备的功能。我们可以看到，关于词元级补全的研究远多于语句级补全（17 篇论文，相比之下为 10 篇）。

1）词元级补全：**使用场景**。从表 II 可以看出，收集到的 17 篇词元级补全论文均支持*API* 推荐。然而，只有少数论文研究标识符补全，且没有论文提及路径补全。这可能是因为对于所提出的技术而言，大多数标识符和路径词元都是词汇表外（OoV）词。

评估指标。在本部分中，准确率等同于 top-1 准确率和错误率，召回率指更一般的 top-k 准确率，而平均排名指预测候选项中适当词元的平均排名（例如，平均倒数排名 MRR）。在表 II 中，我们发现大多数论文都使用召回率和平均排名来评估其方法的有效性。此外，研究人员还探索了其他评估指标 [3], [7], [17], [36]。一些研究人员尝试探索更好的指标，作为衡量用户生产力的代理，例如节省的击键次数 [17]。

服务访问。很少有论文关注将这些技术应用于工业产品。

根据部署方式，我们将论文分为在线和离线两类。Svyatkovskiy等人 [3], [7] 将其系统开发为 Visual Studio Code集成开发环境中 IntelliCode 扩展的一部分，[43]，使程序员能够离线使用该系统。Izadi等人 [10] 采用 Transformer架构，并称他们将模型部署在云端，提供代码补全网络服务。此外，Hindle等人 [17] 提到，他们的工具集成在离线 Eclipse插件中。

效率。 在收集的令牌级代码补全文献中，有 11 篇明确考虑了耗时。 [3], [6], [7], [10], [17], [32], [33]–[36], [38] 提出的 9 种方法预测一个词元所需时间少于 200 毫秒。根据我们的结果，具有此类延迟的技术能够满足 85% 的从业者需求。然而，另外两种方法完成单个词元需要数秒 [34], [39]，这对于词元级补全而言被认为是不可接受的。

2）语句级补全： **使用场景**。如表 II 所示，大多数论文聚焦于下一行预测，但该场景仅获得 43% 的支持率；而更受期待的当前编辑行补全和*API* 参数推荐却较少受到关注。有 3 篇论文针对代码块内容预测 场景提出了方法。此外，没有论文提出预测方法骨架（骨架预测）或完成字符串内容（字符串补全）的技术。

发现 9： 大多数论文聚焦于下一行预测场景，而只有少数论文探索当前编辑行补全 和*API* 参数推荐。此外，还没有论文针对骨架预测提出相关技术，而该场景被认为是最受期待的场景。

评估指标。 从表 II 可以看出，大多数论文通过重叠 *N* 元语法 （如 BLUE 和 ROUGE）以及编辑相似度来评估生成代码；根据我们的调查，这两项指标的支持率最低（分别为 65% 和 70%）。只有一篇论文通过函数/结构相似度，将生成代码与人工编写的代码进行比较评估。此外，没有论文提及生成代码的可读性 和语法正确性。一些论文还提出了定制指标，以全面评估其语句级补全方法，例如点击率 [8], [20], [40]–[42]。

发现 10： 大多数论文都侧重于衡量生成代码与人工编写的代码之间的重叠 *N* 元语法 和编辑相似度 ，而绝大多数参与者并不青睐这些指标。没有论文通过语法正确性 和可读性来评估生成代码，而这两项正是从业者最看重的。

服务访问。文献 [8] 提出了一款名为 Intellicode Compose 的工具，该工具将模型部署在云端，并支持客户端缓存。此外，Wen等人 [42] 将一个 Android Studio 插件作为网络服务提供。另外， [41] 中提出的工具以 Eclipse插件的形式提供，并提供离线补全服务。

效率。在收集到的、提出语句级代码补全方法的论文中，只有四篇明确讨论了其技术的耗时。例如， [8], [21], [41] 中提出的方法完成代码语句的耗时不到 1 秒，而 [20] 中的工作平均需要 5.5 秒以上才能完成预测。

发现 11： 耗时是影响代码补全工具采用的关键因素，但在我们收集的词元级和语句级论文中，有 40% 未提及耗时。

IV. 讨论

A. 启示

我们的调查结果为研究社区带来了几点启示：

1) 对代码补全场景的启示：对于词元级代码补全工具，除了标识符补全和*API* 推荐外，人们也期望这类工具支持路径补全，86% 的参与者对此有所期待。

对于能够预测代码行的代码补全工具，程序员最看重的场景是骨架预测、当前编辑行补全和*API* 参数推荐。然而，目前大多数关于语句级补全的论文都关注较少受期待的场景，例如下一行预测 [9], [10]。

2) 对代码补全工具的启示：绝大多数程序员希望代码补全工具变得更加智能。例如，76% 的参与者希望工具能够学习他们的编程习惯，从而保持编程风格一致并减少修改。此外，他们中的大多数人也希望获得有关预测代码的附加信息，例如 API 定义或文档（81%）。一位参与者表示：“我很难记住项目中数不清的 *API*。当我使用某个 *API* 时，如果代码补全工具能同时定位并显示该 *API* 的文档，就能显著节省搜索 *API* 的时间。”

此外，大多数从业者希望工具安装和配置耗时不超过 20 分钟。如果需要花费大量时间安装工具，他们可能会因此打消使用该工具的念头。

3）对评估指标的启示：评估指标是另一个需要符合从业者预期的重要因素。现有关于语句级代码补全的大多数研究，都是通过比较生成代码与人工编写的代码在重叠 *n* 元语法（例如 BLEU 分数和 ROUGE） [21] 以及编辑相似度 [29], [40]方面的表现来评估生成代码。然而，在我们的调查中，这两项指标在所有评估指标中获得的支持率最低。从业者更希望工具使用函数/结构相似度指标进行评估。一位参与者告诉我们：“……*BLEU* 分数或许是评判生成的自然语言文本质量的良好标准，但它很难评估代码片段是否令人满意。”此外，参与者最看重的指标，例如语法正确性

正确性和可读性，在当前的出版物中都未得到体现。

4) 平衡有效性与代码数量： 根据我们的调查，46%的从业者认为语句级代码补全的有效性比预测代码的数量（39%）更为重要。考虑到两种补全策略都受到一定数量参与者的青睐，如果工具能够提供可配置选项，让用户决定预测代码的数量，将会更好。

此外，我们还观察到，大多数从业者希望补全代码不需要进行大量人工修改。例如，74%的参与者无法接受超过40%的补全代码需要人工修改。因此，代码补全工具可以初步估计预测代码中是否有超过40%需要修改的概率。如果该概率较高，工具可以终止当前的补全过程并生成补全结果。

5) 对代码补全延迟的影响： 补全延迟是显著影响从业者采用代码补全工具意愿的一个关键方面。根据我们的调查，仅有15%的参与者能够接受工具耗时超过400毫秒来预测一个词元。此外，83%的参与者希望生成一行代码的平均耗时少于1秒，80%的参与者期望语句级补全延迟不超过2秒。然而，在我们收集的论文中，近一半都没有考虑耗时因素。研究人员应更加关注代码补全延迟。考虑到离线工具最受期待，如何有效压缩补全模型是未来研究的一个重要方向。

6）提升代码补全工具的鲁棒性：在我们的调查中，许多参与者提到，代码补全工具的鲁棒性对用户体验也至关重要。鲁棒性要求补全结果不受输入中轻微扰动的影响。在调查中，一位参与者分享了自己使用 Tabnine 的体验：“如果变量命名为‘*X train*’，*Tabnine* 就能成功预测 *API* 参数。然而，如果我将名称改为‘*x train*’，补全结果就会截然不同。”除了标识符更改之外，代码补全工具也应该能够适应语句更改。一位参与者表示：“*AiXCoder* 能很好地预测方法的代码块内容，但插入一条无关的赋值语句后，结果就变得很糟糕。”

B. 效度威胁

我们在调查中面临的一项威胁是，部分参与者可能无法完全理解问题。例如，一些参与者从未使用过语句级代码补全工具，因此可能不熟悉有关语句级代码补全的问题。为降低这一威胁，我们使用一两张清晰的图片或 GIF 动图描述每种情境，帮助他们更好地理解问题。此外，一些参与者没有认真作答，因此调查结果无法反映他们的看法。因此，我们剔除了用时少于两分钟的答复。这是以往研究中常见且可以接受的效度威胁，例如 [44]。

另一个威胁是，我们的参与者可能无法代表典型的程序员。为此，我们广泛调查了在多家 IT 公司工作的从业者。我们认为，这一威胁对调查结果的影响已降至最低。

V. 相关工作

A. 代码补全

传统的代码补全侧重于静态分析技术，并结合人工定义的规则来推荐代码[15], [16], [45], [46]。例如，研究人员利用类型信息 [16], [47], 、相似代码片段 [48]和历史数据 [49]来预测所需的代码词元。

随着机器学习的应用，研究人员提出了一系列采用统计语言模型的代码补全方法 [17]–[19], [34], [36], [38]。例如，Hellen等人[36]明确将嵌套作用域等技术纳入考量，从而提升了 n 元语法模型的性能。此外，Raychev等人结合决策树与领域知识，提出了一种概率模型 [33]。

随着深度学习的发展，循环神经网络（RNN） [50] 和 Transformer [51] 等神经网络展现出从源代码中学习特征的强大能力 [5], [31], [36]。例如，Li等人 [31] 提出了一种点混合网络，以缓解词汇表外问题。Kim等人 [5]将代码解析为抽象语法树，并将这些树输入基于 Transformer 的模型。近年来，预训练语言模型已被用于预测多个代码词元 [8], [10]。Svyatkovskiy等人 [8] 提出了 GPT-C，该模型能够预测代码语句。Izadi等人 [10] 提出了 CodeFill，该方法结合了代码的类型信息和语义信息，进一步提升了补全性能。

B. 代码补全实践研究

除代码补全研究外，还有一些其他工作专注于研究代码补全实践 [52]–[55]。例如，Vaithilingam 等人 [52]要求参与者在使用或不使用 Copilot 的情况下完成编程任务 [11]，并判定 Copilot 是否有用。此外，Ziegler 等人 [53]专注于研究代码补全的评估指标。作者比较了可测量的用户数据（客观数据）与用户报告的生产力（主观数据），并确定了最具代表性的生产力代理指标。该研究 [54], [55] 既指出了合成数据与真实世界数据之间的差异，也确定了需要补全的最重要词元。

然而，此前尚无研究探讨从业者对代码补全的预期。本文开展了一项大规模用户研究，探究从业者对代码补全工具多个方面的预期。此外，我们还进行了全面的文献综述，以揭示当前技术与从业者预期之间的差距，并为研究人员指明未来方向。

VI. 结论与未来工作

本文访谈了 15 位专业人士，并调查了 599 位从业者的代码补全实践、他们面临的问题以及对代码补全工具的预期。从业者期望代码补全工具能够针对不同粒度提供代码建议。他们还期望代码补全工具能够满足以下方面的要求：使用场景、评估指标、服务访问、有效性 和效率。我们还通过文献综述，将当前研究的能力与从业者的预期进行比较，指出了为满足从业者需求而有待改进的方面。

参考文献

[1] J. Li、R. Huang、W. Li、K. Yao 和 W. Tan，“通过接受度和排序模型降低代码补全的隐性成本”，载于*IEEE*国际软件维护与演化会议，*ICSME 2021*，卢森堡，*2021*年*9*月*27*日至*10*月*1*日。IEEE，2021年，第195–205页。[2] S. Amann、S. Proksch、S. Nadi 和 M. Mezini，“Visual Studio 实际使用情况研究”，载于*IEEE*第*23*届软件分析、演化与再工程国际会议，*SANER 2016*，日本大阪吹田，*2016*年*3*月*14*日至*18*日——第*1*卷。IEEE 计算机学会，2016年，第124–134页。[3] A. Svyatkovskiy、Y. Zhao、S. Fu 和 N. Sundaresan，“Pythia：人工智能辅助代码补全系统”，载于第*25*届*ACM SIGKDD*知识发现与数据挖掘国际会议，*KDD 2019*，美国阿拉斯加州安克雷奇，*2019*年*8*月*4*日至*8*日，A. Teredesai、V. Kumar、Y. Li、R. Rosales、E. Terzi 和 G. Karypis 编。ACM，2019年，第2727–2735页。[4] Y. Wang 和 H. Li，“将扁平化抽象语法树建模为图以实现代码补全”，载于第三十五届*AAAI*人工智能会议，*AAAI 2021*；第三十三届创新人工智能应用会议，*IAAI 2021*；第十一届人工智能教育进展研讨会，*EAAI 2021*；线上活动，*2021*年*2*月*2*日至*9*日。AAAI Press，2021年，第14 015–14 023页。[5] S. Kim、J. Zhao、Y. Tian 和 S. Chandra，“通过向 Transformer 输入树进行代码预测”，载于第*43*届*IEEE/ACM*软件工程国际会议，*ICSE 2021*，西班牙马德里，*2021*年*5*月*22*日至*30*日。IEEE，2021年，第150–162页。[6] F. Liu、G. Li、B. Wei、X. Xia、Z. Fu 和 Z. Jin，“一种基于多任务学习的代码补全自注意力神经网络架构”，载于*ICPC '20*：第*28*届程序理解国际会议，大韩民国首尔，*2020*年*7*月*13*日至*15*日。ACM，2020年，第37–47页。[7] A. Svyatkovskiy、S. Lee、A. Hadjitofi、M. Riechert、J. V. Franco 和 M. Allamanis，“快速且节省内存的神经代码补全”，载于第*18*届*IEEE/ACM*软件仓库挖掘国际会议，*MSR 2021*，西班牙马德里，*2021*年*5*月*17*日至*19*日。IEEE，2021年，第329–340页。[8] A. Svyatkovskiy、S. K. Deng、S. Fu 和 N. Sundaresan，“Intellicode Compose：使用 Transformer 进行代码生成”，载于*ESEC/FSE '20*：第*28*届*ACM*欧洲软件工程会议与软件工程基础研讨会联合会议，线上活动，美国，*2020*年*11*月*8*日至*13*日，P. Devanbu、M. B. Cohen 和 T. Zimmermann 编。ACM，2020年，第1433–1443页。[9] D. Guo、A. Svyatkovskiy、J. Yin、N. Duan、M. Brockschmidt 和 M. Allamanis，“学习借助代码草图补全代码”，载于国际学习表征会议，2021年。[10] M. Izadi、R. Gismondi 和 G. Gousios，“CodeFill：通过联合学习结构和命名序列实现多词元代码补全”，载于第*44*届*IEEE/ACM*软件工程国际会议，*ICSE 2022*，美国宾夕法尼亚州匹兹堡，*2022*年*5*月*25*日至*27*日。ACM，2022年，第401–412页。[11] “Copilot”，https://github.com/features/copilot/。[12] “AiXCoder”，https://aixcoder.com/。[13] “Tabnine”，https://www.tabnine.com/。[14] “Clangd”，https://clangd.llvm.org/。[15] D. Mandelin, L. Xu, R. Bod´ık, and D. Kimelman, “Jungloid 挖掘：帮助在 API丛林中导航”，载于*ACM SIGPLAN 2005*年编程语言设计与实现会议论文集，芝加哥（美国伊利诺伊州），*2005*年*6*月*12*日至*15*日，V. Sarkar and M. W. Hall 编。ACM，2005年，第48–61页。[16] T. Gvero, V. Kuncak, I. Kuraj, and R. Piskac，“利用类型和权重实现完整补全”，载于*ACM SIGPLAN*编程语言设计与实现会议，*PLDI '13*，西雅图（美国华盛顿州），*2013*年*6*月*16*日至*19*日，H. Boehm and C. Flanagan 编。ACM，2013年，第27–38页。[17] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu，“论软件的自然性”，载于第*34*届国际软件工程大会，*ICSE 2012*，*2012*年*6*月*2*日至*9*日，苏黎世，瑞士，M. Glinz, G. C. Murphy, and M. Pezz`e 编。IEEE 计算机学会，2012年，第837–847页。[18] S. Proksch, J. Lerch, and M. Mezini，“利用贝叶斯网络实现智能代码补全”，载于*2016*年软件工程会议，德国信息学会软件技术分会会议，*2016*年*2*月*23*日至*26*日，维也纳，¨奥地利，LNI丛书，J. Knoop and U. Zdun 编，第P-252卷。GI，2016年，第25–26页。[19] A. T. Nguyen and T. N. Nguyen，“用于代码的基于图的统计语言模型”，载于第*37*届*IEEE/ACM*国际软件工程大会，*ICSE 2015*，佛罗伦萨，意大利，*2015*年*5*月*16*日至*24*日，第*1*卷，A. Bertolino, G. Canfora, and S. G. Elbaum 编。IEEE 计算机学会，2015年，第858–868页。[20] S. V. Nguyen, T. N. Nguyen, Y. Li, and S. Wang，“结合程序分析与统计语言模型进行代码语句补全”，载于第*34*届*IEEE/ACM*自动化软件工程国际会议，*ASE 2019*，圣迭戈（美国加利福尼亚州），*2019*年*11*月*11*日至*15*日。IEEE，2019年，第710–721页。[21] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. D. Penta, and G. Bavota，“关于BERT模型在代码补全中应用的实证研究”，载于第*18*届*IEEE/ACM*国际软件仓库挖掘大会，*MSR 2021*，马德里，西班牙，*2021*年*5*月*17*日至*19*日。IEEE，2021年，第108–119页。[22] J. Devlin, M. Chang, K. Lee, and K. Toutanova，“BERT：用于语言理解的深度双向变换器预训练”，载于*2019*年计算语言学协会北美分会年会论文集：人类语言技术，*NAACL-HLT 2019*，*7*，*2019*，*2 1*，明尼阿波利斯（美国明尼苏达州），*6*月——第*1*卷（长短篇论文），J. Burstein, C. Doran, and T. Solorio 编。计算语言学协会，2019年，第4171–4186页。[23] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell等，“语言模型是少样本学习者”，神经信息处理系统进展，第33卷，第1877–1901页，2020年。[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever 等，“语言模型是无监督多任务学习者”，*OpenAI*博客，第1卷，第8期，第9页，2019年。[25] X. Hu, X. Xia, D. Lo, Z. Wan, Q. Chen, and T. Zimmermann，“从业者对自动代码注释生成的预期”，载于第*44*届*IEEE/ACM*第*44*届国际软件工程大会，*ICSE 2022*，匹兹堡（美国宾夕法尼亚州），*2022*年*5*月*25*日至*27*日。ACM，2022年，第1693–1705页。[26] P. S. Kochhar, X. Xia, D. Lo, and S. Li，“从业者对自动故障定位的预期”，载于第*25*届国际软件测试与分析研讨会论文集，*ISSTA 2016*，萨尔布吕¨肯，德国，*2016*年*7*月*18*日至*20*日，A. Zeller and A. Roychoudhury 编。ACM，2016年，第165–176页。[27] “腾讯问卷”，https://wj.qq.com/。[28] “Google 表单”，https://docs.google.com/forms。[29] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu，“Codexglue：用于代码理解与生成的机器学习基准数据集”，载于神经信息处理系统数据集与基准专题论文集*1*，*NeurIPS* 数据集与基准 *2021*，*2021*年*12*月，线上，J. Vanschoren and S. Yeung 编，2021年。[30] K. Papineni, S. Roukos, T. Ward, and W. Zhu，“BLEU：一种用于机器翻译自动评估的方法”，载于计算语言学协会第*40*届年会论文集，*2002*年*7*月*6*日至*12*日，费城（美国宾夕法尼亚州）。ACL，2002年，第311–318页。

[31] J. Li, Y. Wang, M. R. Lyu, and I. King，“结合神经注意力和指针网络的代码补全”，载于第二十七届国际人工智能联合会议论文集，*IJCAI 2018*，*2018*年*7*月*13*日至*19*日，斯德哥尔摩，瑞典，J. Lang 编。ijcai.org，2018年，第4159–4165页。[32] P. Bielik, V. Raychev, and M. T. Vechev，“PHOG：代码概率模型”，载于第*33*届国际机器学习会议论文集，*ICML 2016*，纽约市（美国纽约州），*2016*年*6*月*19*日至*24*日，《机器学习研究杂志》研讨会与会议论文集，M. Balcan and K. Q. Weinberger 编，第48卷。JMLR.org，2016年，第2933–2942页。[33] V. Raychev, P. Bielik, and M. T. Vechev，“基于决策树的代码概率模型”，载于*2016*年*ACM SIGPLAN*面向对象编程、系统、语言与应用国际会议论文集，*OOPSLA 2016*，*SPLASH 2016*组成部分，阿姆斯特丹，荷兰，*2016*年*10*月*30*日至*11*月*4*日，E. Visser and Y. Smaragdakis 编。ACM，2016年，第731–747页。[34] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen，“用于源代码的统计语义语言模型”，载于欧洲软件工程会议与*ACM SIGSOFT*国际软件工程基础研讨会联合会议，*ESEC/FSE'13*，圣彼得堡，俄罗斯联邦，*2013*年*8*月*18*日至*26*日，B. Meyer, L. Baresi, and M. Mezini 编。ACM，2013年，第532–542页。[35] F. Liu, G. Li, Y. Zhao, and Z. Jin，“基于多任务学习的代码补全预训练语言模型”，载于第*35*届*IEEE/ACM*自动化软件工程国际会议，*ASE 2020*，墨尔本，澳大利亚，*2020*年*9*月*21*日至*25*日。IEEE，2020年，第473–485页。[36] V. J. Hellendoorn and P. T. Devanbu，“深度神经网络是源代码建模的最佳选择吗？”，载于*2017*年第*11*届软件工程基础联合会议论文集，*ESEC/FSE 2017*，帕德博恩，德国，*2017*年*9*月*4*日至*8*日，E. Bodden, W. Sch¨afer, A. van Deursen, and A. Zisman 编。ACM，2017年，第763–773页。[37] S. Lu, N. Duan, H. Han, D. Guo, S. Hwang, and A. Svyatkovskiy，“Reacc：检索增强代码补全框架”，载于计算语言学协会第*60*届年会论文集（第*1*卷：长篇论文），*ACL 2022*，都柏林，爱尔兰，*2022*年*5*月*22*日至*27*日，S. Muresan, P. Nakov, and A. Villavicencio 编。计算语言学协会，2022年，第6227–6240页。[38] Z. Tu, Z. Su, and P. T. Devanbu，“论软件的局部性”，载于第*22*届*ACM SIGSOFT*国际软件工程基础研讨会论文集，（*FSE-22*），香港，中国，*2014*年*11*月*16*日至*22*日，S. Cheung, A. Orso, and M. D. Storey 编。ACM，2014年，第269–280页。[39] V. Raychev, M. T. Vechev, and E. Yahav，“基于统计语言模型的代码补全”，载于*ACM SIGPLAN*编程语言设计与实现会议，*PLDI '14*，爱丁堡，英国，*2014*年*6*月*9*日至*11*日，M. F. P. O'Boyle and K. Pingali 编。ACM，2014年，第419–428页。[40] C. B. Clement, S. Lu, X. Liu, M. Tufano, D. Drain, N. Duan, N. Sundare- san, and A. Svyatkovskiy，“利用EwASH（基于语法层级的扩展窗口访问）对源代码文件进行长距离建模”，载于*2021*年自然语言处理实证方法会议论文集，*EMNLP 2021*，线上活动／蓬塔卡纳，多米尼加共和国，*2021*年*11*月*7*日至*11*日，M. Moens, X. Huang, L. Specia, and S. W. Yih 编。计算语言学协会，2021年，第4713–4722页。[41] Y. Yang, Y. Jiang, M. Gu, J. Sun, J. Gao, and H. Liu，“软件代码语句的语言模型”，载于第*32*届*IEEE/ACM*自动化软件工程国际会议论文集，*ASE 2017*，厄巴纳（美国伊利诺伊州），*2017*年*10*月*30*日至*11*月*3*日，G. Rosu, M. D. Pe nta, and T. N. Nguyen 编。IEEE 计算机学会，2017年，第682–687页。[42] F. Wen, E. Aghajani, C. Nagy, M. Lanza, and G. Bavota，“Siri，写出下一个方法”，载于第*43*届*IEEE/ACM*国际软件工程大会，*ICSE 2021*，马德里，西班牙，*2021*年*5*月*22*日至*30*日。IEEE，2021年，第138–149页。[43] “Visual Studio Code”，https://code.visualstudio.com/。[44] M. Kim, T. Zimmermann, and N. Nagappan，“Microsoft重构挑战与收益的实证研究”，*IEEE*软件工程汇刊，第40卷，第7期，第633–649页，2014年。[45] D. Perelman, S. Gulwani, T. Ball, and D. Grossman，“部分表达式的类型引导补全”，载于*ACM SIGPLAN*编程语言设计与实现会议，*PLDI '12*，北京，中国，*2012*年*6*月*11*日至*16*日，J. Vitek, H. Lin, and F. Tip 编。ACM，2012年，第275–286页。

[46] S. Thummalapenta 和 T. Xie，《Parseweb：一种帮助程序员在网络上重用开源代码的助手》，载于第*22*届 *IEEE/ACM* 国际自动化软件工程会议（*ASE 2007*），*2007*年*11*月*5*日至*9*日，美国佐治亚州亚特兰大，R. E. K. Stirewalt、A. Egyed 和 B. Fischer 编。ACM，2007年，第204–213页。[47] D. Hou 和 D. M. Pletcher，《通过 API 分组、筛选和基于流行度的排序，打造更好的代码补全系统》，载于第二届软件工程推荐系统国际研讨会论文集（*RSSE 2010*），*2010*年*5*月*4*日，南非开普敦，R. Holmes、M. P. Robillard、R. J. Walker 和 T. Zimmermann 编。ACM，2010年，第26–30页。[48] M. Bruch、M. Monperrus 和 M. Mezini，《从示例中学习以改进代码补全系统》，载于欧洲软件工程会议与 *ACM SIGSOFT* 国际软件工程基础研讨会第七届联合会议论文集，*2009*年*8*月*24*日至*28*日，荷兰阿姆斯特丹，H. van Vliet 和 V. Issarny 编。ACM，2009年，第213–222页。[49] R. Robbes 和 M. Lanza，《程序历史如何改进代码补全》，载于第*23*届 *IEEE/ACM* 国际自动化软件工程会议（*ASE 2008*），*2008*年*9*月*15*日至*19*日，意大利拉奎拉。IEEE 计算机学会，2008年，第317–326页。[50] A. Sherstinsky，《循环神经网络（RNN）与长短期记忆（LSTM）网络基础》，*CoRR*，第 abs/1808 03314 卷，2018年。。[51] A. Vaswani、N. Shazeer、N. Parmar、J. Uszkoreit、L. Jones、A. N. Gomez、L. Kaiser 和 I. Polosukhin，《Attention Is All You Need》，载于神经信息处理系统进展第*30*卷：*2017*年神经信息处理系统年会，*2017*年*12*月*4*日至*9*日，美国加利福尼亚州长滩，I. Guyon、U. von Luxburg、S. Bengio、H. M. Wallach、R. Fergus、S. V. N. Vishwanathan 和 R. Garnett 编，2017年，第5998–6008页。[52] P. Vaithilingam、T. Zhang 和 E. L. Glassman，《预期与体验：评估由大语言模型驱动的代码生成工具的可用性》，载于 *CHI '22*：*CHI* 计算系统中的人为因素会议，*2022*年*4*月*29*日至*5*月*5*日，美国路易斯安那州新奥尔良，扩展摘要，S. D. J. Barbosa、C. Lampe、C. Appert 和 D. A. Shamma 编。ACM，2022年，第332:1–332:7页。[53] A. Ziegler、E. Kalliamvakou、X. A. Li、A. Rice、D. Rifkin、S. Simister、G. Sittampalam 和 E. Aftandilian，《神经代码补全的生产力评估》，载于 *MAPS@PLDI 2022*：第六届 *ACM SIGPLAN* 机器编程国际研讨会，*2022*年*6*月*13*日，美国加利福尼亚州圣迭戈，S. Chaudhuri 和 C. Sutton 编。ACM，2022年，第21–29页。[54] V. J. Hellendoorn、S. Proksch、H. C. Gall 和 A. Bacchelli，《代码补全何时失效：真实场景中的代码补全案例研究》，载于第*41*届国际软件工程会议论文集（*ICSE 2019*），*2019*年*5*月*25*日至*31*日，加拿大魁北克省蒙特利尔，J. M. Atlee、T. Bultan 和 J. Whittle 编。IEEE / ACM，2019年，第960–970页。[55] G. A. Aye、S. Kim 和 H. Li，《从真实世界数据集中学习自动补全》，载于第*43*届 *IEEE/ACM*软件工程国际会议：软件工程实践（*ICSE (SEIP) 2021*），*2021*年*5*月*25*日至*28*日，西班牙马德里。IEEE，2021年，第131–139页。