arXiv:2301.03987v1 [cs.SE] 10 Jan 2023

通过动态提示调优语言模型从文本中联合抽取 API 实体与关系

QING HUANG,江西师范大学，计算机信息工程学院，中国YANBANG SUN*, 江西师范大学，计算机信息工程学院，中国ZHENCHANG XING, CSIRO Data61 与澳大利亚国立大学，工程与计算机科学学院，澳大利亚MIN YU†, 江西师范大学，计算机信息工程学院，中国XIWEI XU,CSIRO Data61，澳大利亚QINGHUA LU, CSIRO Data61，澳大利亚

从非结构化文本（例如 Stack Overflow）中抽取应用程序编程接口（API）及其语义关系，是软件工程任务（例如 API推荐）的一项基础工作。然而，现有方法基于规则或序列标注。它们必须针对大量句式手动枚举规则或标注数据，这需要付出大量人力成本，而且会受到形态歧义和常用词歧义的进一步影响。本文不采用匹配或标注 API实体及关系的方式，而是将异构 API 抽取和 API关系抽取任务表述为序列到序列生成任务，并提出 AERJE——一种基于大型预训练语言模型的 API实体关系联合抽取模型。在少量模糊但标注正确的数据上进行训练后，AERJE 构建了一种多任务架构，利用动态提示从非结构化文本中抽取 API实体及关系。我们在一组来自 Stack Overflow 的长句和模糊句子上系统地评估了 AERJE。实验结果表明，即使在零样本或少样本微调的情况下，AERJE 在 API实体关系联合抽取方面仍具有较高的准确率和区分能力。

A附加关键词与短语：API 实体、API关系、联合抽取、动态提示

ACM 引用格式：
Qing Huang、Yanbang Sun、Zhenchang Xing、Min Yu、Xiwei Xu 和 Qinghua Lu。2023。基于动态提示调优语言模型的文本 API实体与关系联合抽取。1, 1（2023年1月），20 页。
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 引言

应用程序编程接口（API）是重要的软件工程产物，广泛存在于各种自然语言文本中，从官方API参考文档和教程到非正式的在线论坛。同时，这些文本中也包含 API关系。例如，Stack Overflow（SO）帖子中的文本“Tomanipulate datayou actuallyneed *executeUpdate()* ratherthan *execute-Query()*” [1] 描述了函数替换关系 [1]，即*executeUpdate()* 与*executeQuery()*之间的关系。这种 API关系表明，为了解决帖子中的问题——“为什么无法使用 *executeQuery()* 来执行数据操作语句”，我们应该用 *executeQuery()*替换 *executeQuery()*。从非结构化文本中抽取 API 实体与关系是

*Y. Sun 和 Q. Huang 为共同第一作者。
†M. Yu 为通讯作者。[1]

https://stackoverflow.com/questions/1905607

作者地址：Qing Huang，江西师范大学计算机信息工程学院，中国江西南昌，qh@whu.edu.cn；Yanbang Sun，江西师范大学计算机信息工程学院，中国江西南昌，ybsun@jxnu.edu.cn；Zhenchang Xing，CSIRO Data61 和澳大利亚国立大学工程与计算机科学学院，澳大利亚堪培拉，zhenchang.xing@data61.csiro.au；Min Yu，江西师范大学计算机信息工程学院，中国江西南昌，myu@jxnu.edu.cn；Xiwei Xu，CSIRO Data61，澳大利亚悉尼，xiwei.xu@data61.csiro.au；Qinghua Lu，CSIRO Data61，澳大利亚悉尼，qinghua.lu@data61. csiro.au。

表 1。API实体和关系的三种歧义类型。

帖子 ID	句子
#47871272	您需要在迭代器中重写 remove()。
#14200489	这段代码无效，因为在遍历 l 时调用了 l.remove()。
#60017952	你可能多次调用了 iterator.remove。
#34682267	默认情况下，printWriter 会在 println 中调用刷新，但不会在 print 中这样做。
#703396	如果你的想法是 print 以双精度浮点数形式存储的整数.....
#322715	链表和数组列表是列表接口的两种不同实现。
#33405095	nextline() 会读取整行，而 next() 只会读取下一个单词。
#355089	StringBuffer 是同步的，StringBuilder 则不是。

注意：API 提及以下划线标记，普通词语以波浪线标记。

高效获取 API 知识并将其应用于各种软件工程任务的基础。抽取出的实体和关系可以组织成结构化知识（尤其是知识图谱），以支持各类软件工程任务，例如 API链接 [2, 3], API推荐 [4, 5],和 API 比较 [6]。

目前，从非结构化文本中抽取 API 实体的方法主要有两种。第一种是基于规则的方法，例如基于语言约定的正则表达式 [7, 8]、、岛屿解析 [9, 10] 和启发式规则匹配 [1, 6, 11]。由于不可能手动穷举适用于所有句式的规则，因此这种方法会产生规则设计开销。第二种是基于序列标注的方法，例如条件随机场 [2, 12]和双向长短期记忆条件随机场 [13]。由于不可能为大量句子手动标注实体，因此这种方法会产生数据标注开销。与 API 实体抽取相比，从软件文本中进行关系抽取还相当初级，主要依赖 API 语法（例如，类声明方法） [6], 、特殊标签标注关系（例如，“另见”关键词和基于超链接的方法） [14], 或某些临时关系短语（例如，“不同于”和“类似于”） [1]。与基于规则的实体抽取一样，这些关系抽取方法也会产生规则设计开销。在本研究中，我们将规则设计开销和数据标注开销统称为人工开销。

这项人工开销因三类歧义而加剧，因此需要人工设计更多规则，或标注更多数据，以区分模糊句子。形态歧义是其中一种，包括缩写、同义词和拼写错误 [12]。这种歧义在非正式讨论中很常见，因为人们很少会写出与库中 API 名称完全一致的完整 API 名称 [15]。例如，表1第一行中的三个句子展示了 API*java.util.iterator.remove()*的三种形态变体：它们要么省略部分前缀和特殊符号（例如包名、类名和“*()*”），要么在前面加上用户定义的变量名。第二类是常用词与 API 提及之间的常用词歧义 [12], ，这是因为人们经常在书写 API 方法名时省略必要的标点、括号和大写字母。例如，如表1第二行所示，尽管单词*print* 出现在两个句子中，*print* 在第一个句子中指的是 API*java.io.printwriter.print()*，而*print* 在第二个句子中则只是一个动词。最后一类是 API 语义关系的表达式歧义，其成因是句式发生变化。一般而言，同一种 API 关系可以用多种句式表达。例如，如表1第三行所示，三个句子采用了三种不同句式，但都表达了 API 实体之间的行为差异关系。它们分别是“API1 和 API2 不同”“API1 做一件事，但 API2 做另一件事”以及“API1 是（形容词），API2 不是”。

为降低人工开销，我们提出一种新颖的方法，利用大型预训练语言模型（大语言模型）提取 API 实体和关系。大语言模型存储了大量先验知识，可充当现实世界实体和关系的神经知识库 [16]。此外，大语言模型还能提供更好的模型初始化 [17] 和强大的学习能力。通过使用少量领域专用训练数据对大语言模型进行微调，我们可以促使其尽可能多地识别 API 实体和关系。为了提高大语言模型的判别能力，训练数据应包含足够多的形态歧义和常用词歧义，并且应正确标注 API 实体和关系。为减少人工标注，我们设计了形态和基于动词的数据增强策略，以生成更多含有歧义但标注正确的句子，用于微调大语言模型。

现有工作 [18] 将 API实体抽取和关系抽取分为两项任务，使关系抽取严重依赖实体抽取结果，从而导致错误传播 [19]。与此不同，我们将 API实体抽取和关系抽取视为相互关联的任务，并受近期通用信息抽取（UIE [20]）研究的启发，采用统一模型联合抽取实体与关系。然而，UIE 中的大语言模型（即 T5 [21]）在处理复杂句子时往往表现不佳，尤其是包含 API实体和多种关系的长句与模糊句子，因为它仅使用一个静态提示来识别所有类型的 API关系。为解决这一问题，我们受输入依赖的提示调优方法 [22],启发，设计了一个动态提示生成器。该方法在推理时根据实际输入句子，为少量可能相关的关系生成动态提示，而不是对所有输入都依赖同一个静态提示。面对复杂句子时，需要识别的关系类型越多，受到的噪声干扰就越大，大语言模型也就越难理解复杂句子在何种程度上包含特定关系。由于我们的动态提示减少了需要识别的关系类型数量并缓解了噪声干扰，因此提高了 API关系的抽取准确率。

本文提出了一种 API 实体关系联合抽取框架，称为 AERJE。该框架由动态提示生成器和实体关系联合抽取器组成。提示生成器的核心是一个基于 BERT 的文本分类器，用于对输入文本进行分类。每个类别代表一种 API关系，提示生成器根据排名前 N 的可能 API关系生成动态提示。生成的动态提示与输入句子一同输入实体关系联合抽取器，以抽取文本中包含的 API实体和关系。在当前实现中，实体关系联合抽取器采用基于 Transformer 的大语言模型（T5）。提示生成器和实体关系抽取器以端到端方式进行微调。

据我们所知，目前还没有模型能够同时抽取 API实体和关系。相比之下，AERJE 在 API实体抽取上的 F1 分数达到 96.51%，比当前最先进的 API 实体识别模型 ARCLIN [13] 高约 6%，比 APIReal [2]，高 7%；其 API关系抽取的 F1 分数为 81.20%。随后，我们评估了内在因素（两种数据增强策略以及动态提示中 API关系的数量）对性能的影响。实验发现，数据增强有助于提升 AERJE 区分 API实体和关系的能力，而包含四种 API关系的动态提示则能显著提高 AERJE 的抽取准确率。最后，我们评估了 AERJE 的泛化能力，以及其在低资源场景（即微调数据少于 0.8%）中抽取 API实体和关系的能力。结果发现，即使在低资源条件下，AERJE 仍具有很强的抽取能力，表现优于 APIReal [2] 和 ARCLIN [13]。

本文的主要贡献如下：

- 从概念上讲，我们首次将异构 API 抽取和 API关系抽取任务统一建模为序列到序列生成任务，并提出了 AERJE——一个基于预训练大语言模型的 API 实体关系联合抽取框架。

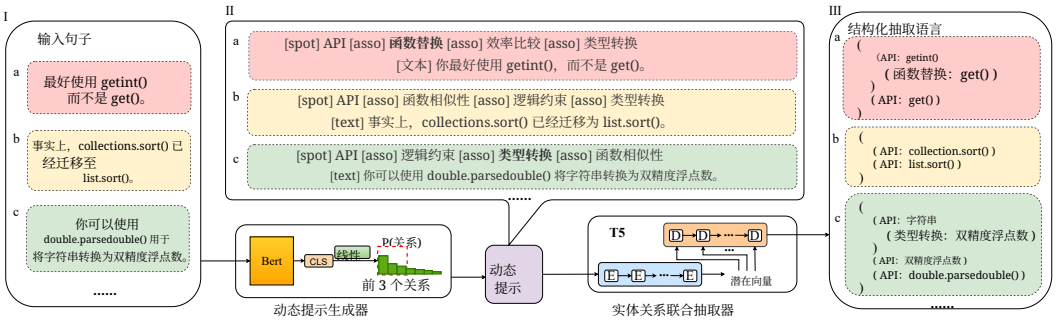


图 1. 整体 AERJE 框架。动态提示中的粗体字表示待抽取的语义关系
抽取自 输入句子。II.b 中没有粗体字，因为 I.b 中不包含语义关系。

- 我们设计了两种数据增强策略，以获得更多具有歧义但标注正确的句子。通过学习这类句子，AERJE 能够更具判别力地识别 API实体和关系。
- 不同于单任务模型，我们构建了一种多任务架构，将实体与关系抽取的结构编码为统一的结构化语言，以同时抽取 API 实体和关系。
- 我们没有为所有句子统一使用包含所有类型 API 关系的单个静态提示，而是设计了一种基于关系分类的动态提示，从而减少需要识别的关系类型数量、消除噪声干扰，并降低关系抽取的难度。
- 我们系统地评估了 AERJE 的内在因素、性能、泛化能力和少样本学习能力。这是首个能够同时抽取 API 实体和关系的方法，其性能优于独立的 API 抽取和 API 关系抽取。我们的数据包可在此处获取[2]，论文被接收后将发布代码。

2 方法

我们将异构 API 抽取和 API 关系抽取任务统一表述为序列到序列生成任务，并提出了一种新模型 AERJE 来完成该任务。如图所示 1，AERJE 由动态提示生成器和实体关系联合抽取器组成。动态提示生成器根据输入文本（每次处理一段）生成动态提示。随后，将输入文本附加到提示中，构成完整输入并送入实体关系联合抽取器，由其生成包含 API 实体和关系的结构化抽取语言序列。

2.1 动态提示生成器

本节介绍如何构建一个统一异构 API 抽取和 API关系抽取任务的提示，随后讨论如何设计动态提示，以提升 AERJE 的 API关系抽取性能。

2.1.1 多任务处理的提示构建。为了从输入文本中同时提取 API 实体和 API关系，提示由 API 实体类型、API 关系类型和输入文本组成，分别用 [spot], [asso]and [text],标记。例如，“[spot] API [asso]函数

[2]https://anonymous.4open.science/r/AERJE-6DBF/README.md

替换 [asso] 效率比较 [text] 最好使用 getint() 而不是 get()”表示一个 API 实体类型“API”、两个关系类型“函数替换”和“效率比较”，以及输入文本“You better using getint() instead of get()”。在这项工作中，我们考虑一种通用的 API 实体类型“API”，以及 [1], 中定义的七种关系类型，包括“函数相似性”“行为差异”“逻辑约束”“类型转换”“函数协作”“效率比较”和“函数替换”。请注意，也可以使用更细粒度的 API 实体类型，例如“类”“方法”“字段” [23], 但我们将其留待未来工作。

2.1.2 动态提示生成。 如第 1 节所述，关系类型越多，T5 就越难确定输入文本中的API实体属于哪种关系类型，尤其是在句子较长且存在歧义时。如果我们采用包含全部七种关系类型的静态提示，模型的关系抽取性能就会下降（参见 RQ3）。因此，我们设计了一种动态提示生成方法，使提示内容对于复杂输入文本更加准确且具有指导性。如图 1 的 II.a 所示，动态提示仅包含排名前 N 的关系，能为后续由 T5 支持的实体关系联合抽取器提供更好的指导。这里，提示生成器被实现为一个文本分类器，用于预测输入文本中出现的 API关系。我们使用基于BERT的分类器，因为 BERT 的预训练任务（即下一句预测） [24] 与我们的任务一致，两者都是分类任务。给定一个包含 API实体的句子（见图 1 的 I.a），基于BERT的分类器会输出该句子属于每种语义关系的概率；随后选取排名前三的关系作为候选关系。最后，通过标签（即 [spot], [asso], [text]）将实体类型、候选关系和输入句子连接起来，生成动态提示（见图 1 的 II.a）。

请注意，我们当前实现中的基于BERT的分类器旨在缩小范围并提供候选关系，不能取代 API关系抽取器。当分类器判定的候选关系与句中的这些实体不匹配时，抽取器不会强行从错误的候选关系中选出一种关系，而是假定这些实体之间不存在关系。例如，给定一个不包含 API实体之间关系的句子（见图1中的 I.b），动态提示生成器会生成一个动态提示（见图1中的 II.b）。基于这样的动态提示，后续抽取器不会从该句子中抽取关系，因为没有任何候选关系类型适用于输入句子。

总而言之，候选关系过多可能会降低抽取器识别它们的能力，而候选关系过少则可能导致抽取器遗漏正确的关系。因此，我们应探究候选关系的适当数量（参见 RQ3）。

2.2 API 联合实体-关系抽取器

我们采用一种结构化抽取语言（SEL） [20] ，将实体抽取和关系抽取的结构编码为统一表示，从而能够在序列到序列生成框架中对异构 API 抽取和 API关系抽取任务进行统一建模。第一个序列指动态提示，第二个序列指 SEL序列。

2.2.1 结构化抽取语言。 SEL序列通过分层定位关联结构，旨在编码不同的信息抽取结构。图 2.a 展示了其通用格式。“标注名称：信息跨度”表示各种实体类型及特定实体类型的对象；“关联名称：信息跨度”表示各种关系类型及特定关系类型所关联的对象。图 2.b 展示了我们工作中的具体 SEL 序列。“API：getint()”表示“getint()”是一个 API 实体；“函数替换：get()”表示“getint()”与“get()”之间的关系

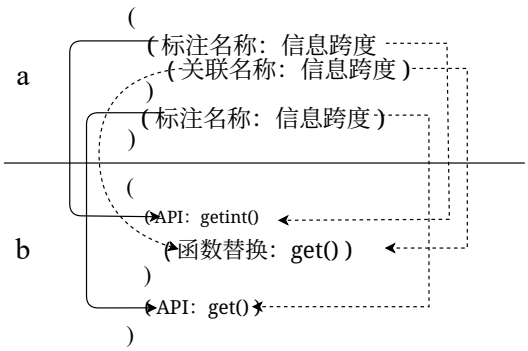


图 2. 通用结构化抽取语言的专门化。

是“函数替换”。通过这一具体的 SEL 序列，我们可以同时抽取 API 实体和关系，因为它统一了 API 实体和关系的结构。

2.2.2 SEL 序列生成。我们将 API 实体关系联合抽取器实现为序列到序列生成框架：

$$\left[y_1, \ldots, y_{|y|}\right] = \mathrm{JE}(\left[p_1, \ldots, p_{|p|}\right]) \tag{1}$$

其中，JE 是一个基于 Transformer 的大语言模型，$\left[p_1, \ldots, p_{|p|}\right]$是动态提示，$\left[y_1, \ldots, y_{|y|}\right]$是线性化 SEL 序列，其中包含待抽取的 API实体和关系。在该框架中，我们将动态提示输入大语言模型（如图 1.II 所示），大语言模型生成 SEL 序列（如图 1.III 所示），据此我们可以获得 API实体和关系。向 JE 提供的动态提示也可以采用第 2.1.1 节所述的格式：

$$\begin{aligned}\left[p_1, \ldots, p_{|p|}\right] = & [[\text{ spot }], \ldots [\text{ spot }] \ldots, \\ & [\text{ asso }], \ldots, [\text{ asso }] \ldots, \\ & [\text{ text }], x_1, x_2, \ldots, x_{|x|}\right]\end{aligned} \tag{2}$$

其中，$x = \left[x_1, \ldots, x_{|x|}\right]$表示输入文本。

为了更好地说明该框架的内部机制，我们引入了一种编码器-解码器式架构。给定动态提示 p，JE 计算每个词元的隐藏表示 $\mathbf{H} = \left[\mathbf{p}_1, \ldots\ldots, \mathbf{p}_{|p|}\right]$:

$$\mathbf{H} = \mathrm{Encoder}\left(p_1, \ldots, p_{|p|}\right) \tag{3}$$

其中，Encoder(·) 是一个 Transformer 编码器。随后，JE 以自回归方式将提示解码为 SEL序列。在解码步骤 i，JE 生成 SEL序列中的第 i 个词元 y_i 以及解码器状态$\mathbf{h}_i^d$，如下所示：

$$y_i, \mathbf{h}_i^d = \mathrm{Decoder}\left(\left[\mathbf{H}; \mathbf{h}_1^d, \ldots, \mathbf{h}_{i-1}^d\right]\right) \tag{4}$$

Decoder(·) 是一个 Transformer 解码器，它预测词元的条件概率 $p\left(y_i \mid y_{<i}, p\right) y_i$，直至输出结束符号 <eos> 。

2.3 模型训练

本节介绍数据收集与增强、模型训练，其中包括训练基于BERT的分类器以及微调基于 Transformer 的大语言模型。

2.3.1 数据收集。鉴于我们考虑的关系类型均来自 Java API 的知识图谱 [1], , 我们从 Stack Overflow 数据转储中随机选取了 5,000 篇带有 Java 标签的帖子 [3]。每篇帖子都附有其回答和帖子标签（例如“java”、“arrays”、“java.lang”）。为确保训练数据的质量，我们选取了这些帖中得票最多的回答，但由于本研究关注非正式文本，因此排除了代码片段和所有 HTML 标签。随后，我们使用 spaCy [4] 将所有回答拆分成句子，共得到 28,140 个句子。每个句子都附有其所属帖子的多个类别标签。然后，我们使用软件专用分词器 [12] 将每个句子解析为词元，该分词器能够保留 API 提及的完整性。例如，*iterator.remove()* 被视为一个单独的词元。最后，我们爬取了 JDK 1.8 [5] 中的所有 API，并使用这些 API 筛选出包含 API 实体的句子，筛选依据参考了一项先前研究 [13], , 具体标准如下：

- 由于形态歧义较多，如果某个词元与爬取到的任一 API（例如 *remove()* 和 *java.util.Iterator.remove()*）部分匹配，就可能是一个 API 实体。
- 由于 API 提及通常以“*()*”结尾，如果词元包含“*()*”，就将其视为 API 实体。
- API提及通常包含“.”以表示函数调用（例如 *iterator.remove()* 或 *l.remove()*）；因此，如果词元包含“.”，我们就将其视为 API 实体。

经过筛选，我们得到 9,111 个可能包含 API 实体的句子。然而，这只是粗略的句子筛选。为了准确筛选句子，我们邀请了 12 名硕士生（均拥有五年以上的 Java 使用经验）检查这些 API实体，并标注 API 之间的语义关系，以进一步验证这些句子是否包含 API 实体，以及是否包含我们旨在抽取的七种 API关系。标注前，我们对标注人员进行了培训，确保他们能够识别文本中的这些 API关系。培训结束后，标注人员被分为六组，每组两名学生对相同内容进行标注。标注完成后，我们安排两位作者处理标注结果中的冲突，Cohen’s Kappa [25] 系数为 0.859（即几乎完全一致）。最终，我们得到共计 2917 个句子，其中 2471 个仅包含实体，446 个同时包含实体和关系。

2.3.2 数据增强。为提升 AERJE 从较长且模糊的句子中识别 API 实体及关系的能力，我们设计了两种数据增强策略，以生成更多模糊句子用于模型训练。

基于形态的变异。受 [13], 启发，我们改变句子中每个 API 实体的形式。具体而言，我们用其全限定名称的最后一部分替换 API 实体本身。例如，*iterator.remove()* 会被替换为 *remove()* 或*remove*。

基于动词的变异。我们使用 spaCy 定位每个 API 实体所依赖的动词，然后用同义词替换这些动词，就像 Liu 等人 [26] 为生成相似的问题标题所做的那样。如表 1的第七个句子所示，我们将“read”替换为“load”。然而，由于 spaCy 可能无法正确识别 API 实体，我们必须确定 API 实体的子词元与动词之间的依存关系，以确保变异质量。例如，“*nextline*”和“read”之间存在依存关系，因此我们可以可靠地用同义词替换“read”。

我们的数据增强策略不包括句式变异 [26], , 这种变异通过不同的句式来表达同一 API 实体之间的相同 API 关系。与基于形态和基于动词的变异不同，这种变异在软件文本中并不可靠，因为软件文本对语义的要求比一般文本更严格。句式变异可能导致句子结构重构，从而很可能改变句子语义，污染

[3]检索于2022年6月6日，网址：https://archive.org/download/stackexchange/ [4]https://spacy.io [5] https://docs.oracle.com/javase/8/docs/api

训练数据，并损害 AERJE 训练。例如，原句“StringBuffer 是同步的，StringBuilder 不是”可能会变异为“StringBuffer 和 StringBuilder 在同步方面有所不同”。原句表明 *StringBuffer* 是同步的，而 *StringBuilder*是异步的，但变异后的句子没有说明谁是同步的，谁是异步的。

我们从收集的 2,917 个句子中，按 8:2 的比例获取 2,334 个句子作为初始训练集，583 个句子作为初始测试集。对初始训练集和测试集应用两种数据增强策略后，句子数量分别增至 10,678 和 2,686，分别称为最终训练集和最终测试集。最终训练集用于微调基于大语言模型的抽取器，最终测试集则用于测试微调后的抽取器。这里，我们先将句子划分为训练集和测试集，再对其进行变异。这样可以确保变异前后的句子属于同一个集合，避免训练数据泄漏到测试集（例如，原句在训练集中，而其变异句在测试集中）。此外，我们从最终训练集中获取 1,639 个同时包含实体和关系的句子，作为分类器训练集。同样，我们从最终测试集中获取 387 个同时包含实体和关系的句子，作为分类器测试集。

2.3.3 基于*BERT*的关系分类器训练。 我们选择 BERT [24] 作为关系分类器，因为其预训练任务（即下一句预测）与我们的任务一致，二者都是分类任务。不过，关系分类器的实现并不限于 BERT，我们也可以使用 TextCNN [27] 和 FastText [28]。在当前实现中，我们使用 BERT-base分类器将每个输入句子分类为 N 种关系类型之一。动态提示生成器根据这 N 种关系类型生成相应的动态提示。

掩码语言模型（BERT） [24] 和一个线性层共同构成分类器。由于有七种 API关系类型，线性层的输出维度设为 7。将句子输入 BERT 后，我们从 CLS标记处获取潜在向量。与其他位置相比，从 CLS标记处获取的潜在向量能更好地表征句子特征，从而获得更好的分类性能。随后，潜在向量被输入线性层，生成一个七维向量，其中每一维对应一种关系类型。最后，使用分类器训练集对分类器进行训练。在反向传播过程中，我们使用交叉熵损失计算分类器的损失，并调整 BERT 和线性层的参数。损失函数公式如下，其中 $z = [z_0, \ldots, z_{C-1}]$ 表示线性层的输出结果，C 表示句子的标签。

$$\mathrm{Loss}(z, c) = -z[c] + \log\left(\sum_{j=0}^{C-1} \exp(z[j])\right) \tag{5}$$

2.3.4 基于大语言模型的抽取器微调。 由于 T5 的训练目标与我们将 API实体与关系抽取任务定义为序列到序列生成任务的表述完全一致，因此在当前实现中，我们使用预训练的 T5-v1.1-large 模型 [21] 作为大语言模型。此外，研究 [29, 30] 证实，T5 能够捕获丰富的文本信息，并证明了其在各种下游自然语言处理任务中的有效性。我们的方法并不局限于 T5，也可以使用任何基于 Transformer 的大语言模型。

为了微调 T5，我们将最终训练集中的每个已标注句子转换为 SEL 序列（y），然后将其输入动态提示生成器以获得其动态提示（p），最后构建标注语料库：$\mathrm{D_e} = \{(p, y)\}$。我们在该标注语料库上使用 Adam优化器对 T5 进行微调，训练 50 个 epoch，批次大小为 10，学习率为 1e-4，采用线性调度，预热比例为 6%，并使用教师强制交叉熵损失：

$$\mathcal{L}_{\mathrm{FT}} = \sum_{(p,y)\in\mathcal{D}_e} -\log P\left(y \mid p; \theta_e, \theta_d\right) \tag{6}$$

其中， θ_e 和 θ_d 分别是编码器和解码器的参数。

3 实验设置

本节首先提出五个有关 AERJE 性能的问题，随后介绍实验设置，包括数据集、基线和评估指标。

3.1 研究问题

- RQ1：数据增强的有效性
- RQ2：动态提示的最佳关系类型数量
- RQ3：AERJE 的联合抽取性能
- RQ4：AERJE 的泛化能力
- 研究问题5：AERJE 在低资源场景中的性能

3.2 数据集

如第 2.3.2 节所述，数据集分为三组。第一组是最初收集的句子，其中一些仅包含实体，另一些则同时包含实体和关系。

- 初始训练集由 2,334 个句子组成，其中 362 个同时包含实体和关系。
- 初始测试集包含 583 个句子，其中 84 个同时包含实体和关系。

第二组指应用两种数据增强策略后的句子，其中一些仅包含实体，另一些则同时包含实体和关系。

- 最终训练集共包含 10,678 个句子，其中 1639 个同时包含实体和关系。
- 最终测试集共包含 2,686 个句子，其中 387 个同时包含实体和关系。

第三组指最终训练集和测试集中同时包含实体和关系的句子。

- 分类器训练集共包含 1,639 个句子。
- 分类器测试集共包含 387 个句子。

3.3 基线方法

我们的 AERJE 能够联合抽取 API 实体和关系。然而，据我们所知，先前没有研究同时从非结构化文本中抽取 API 实体和关系。因此，我们只能将 AERJE 与 API 实体抽取和 API 关系抽取各自领域的现有研究进行比较。

对于 API实体抽取，现有方法包括基于规则的方法（如正则表达式 [7, 8]）、启发式规则匹配方法 [1, 6, 11], 以及基于序列标注的方法（如 AR- CLIN [13]，使用双向长短期记忆网络作为编码器、条件随机场作为解码器；APIReal [2]仅使用条件随机场）。由于前两类方法的性能不如最后一类方法 [13],，我们选择 ARCLIN 和 APIReal 作为基线方法。我们从 GitHub [6] [7] 获取它们的源代码，并使用“BIO”标签标注句子中的 API实体和非实体（即，“B”：API实体片段的开头，“I”：API实体片段内部，“O”：非实体）。然后，我们创建原始句子与标注句子配对的数据，用于训练这两个基线方法。最后，我们在最终测试集上评估训练好的模型，并据此根据“BIO”标签获取 API实体。

[6] https://github.com/YintongHuo/ARCLIN [7] https://github.com/baolingfeng/APIExing

对于 API关系抽取，现有方法只有依赖 API 语法的规则匹配方法 [6],特殊标签标注关系 [14], 或一些临时关系短语 [1]。由于规则设计开销较大，重新实现这些方法非常困难。此外，鉴于我们假设输入为不带任何特殊标注的纯文本，应用这些方法也不切实际。因此，我们实现了一个变体 $AERJE_{w/oDPG}$ (DPG 意为动态提示生成器），它使用包含全部 7 种关系类型的静态提示来评估 AERJE 完整版本的性能。

此外，我们还实现了 AERJE 的另外两个变体，作为基线。其中一个是 $AERJE_{single}$，它将 API 实体抽取和关系抽取拆分为两个独立任务。对于 API 实体抽取，提示仅包含“[spot] API”。对于 API 关系抽取，提示仅包含“[asso] relation type”。 $AERJE_{single}$ 在关系抽取中仍使用动态提示。关系抽取完成后，我们将抽取出的实体和关系合并为 $AERJE_{single}$ 的最终结果。例如，抽取出的实体 getint()、get() 和关系函数替换会合并为 (API: getint() (function replace: get()))。我们将 $AERJE_{single}$ 与 AERJE 进行比较，以了解实体-关系联合抽取的效果。同时， $AERJE_{single}$ 的实体抽取结果等同于对预训练模型进行实体抽取微调，其最终结果则等同于对预训练模型进行关系抽取微调。因此， $AERJE_{single}$ 也体现了分别对预训练模型进行实体和关系抽取微调的能力。另一个变体是$AERJE_{base}$，它将 AERJE 中的 T5-v.1.1-large 替换为较小的模型骨干网络，即 T5-v1.1-base。我们使用它来探究大型预训练语言模型对 AERJE 性能的影响。所有变体均使用与 AERJE 相同的超参数，并在不同实验场景中保持不变。请注意，已有研究表明，AERJE 中使用的 SEL 在抽取任务中有效 [20]。因此，我们不再验证 SEL 在 AERJE 中的有效性。

3.4 评估指标

我们采用精确率、召回率和 F1 分数作为指标，评估 AERJE 和基线模型在测试集上的表现。精确率表示抽取出的 API 实体和关系中有多大比例是正确的；召回率表示真实 API 实体和关系中有多大比例被抽取出来；F1 分数则是精确率和召回率的调和平均数。需要注意的是，只有关系类型和对应实体都正确时，关系才算正确。就我们的研究而言，我们并不关注前 N 个关系分类准确率。只要前 N 个结果中包含相关关系类型，抽取器就不在意这些关系类型的顺序。此外，一个句子可能包含 2 个或更多关系，因此前 1 位准确率并不适用。最后，由于抽取器能够在提示中排除不相关的关系类型，因此也无需评估前 N 位的分类精确率和召回率。

4 实验结果

本节将探讨五个研究问题，以评估并讨论 AERJE 的表现。

4.1 RQ1：数据增强的有效性

4.1.1 研究动机。 为了减少人工标注工作量并改进模型训练，我们设计了两种数据增强策略。我们希望探究通过这两种数据增强策略获得的、含义模糊但标注正确的句子，是否能够提升 AERJE 提取 API实体和关系的判别能力，从而证明这两种数据增强策略的有效性。

4.1.2 方法。 我们设置了两种场景 $AERJE_{w/oDA}$ 和 $AERJE_{wDA}$ （DA 表示数据增强）。 $AERJE_{w/oDA}$ 在初始训练集上进行训练，而 $AERJE_{wDA}$ 在

表 2. 数据增强策略对 AERJE 的影响

策略	实体			关系		
	P	R	F1	P	R	F1
$AERJE_{wDA}$	**97.57**	**95.48**	**96.51**	**86.54**	**76.48**	**81.20**
$AERJE_{w/oDA}$	95.11	92.19	93.63	77.71	75.66	76.67

最终训练集上进行训练。 $AERJE_{w/oDA}$ 和 $AERJE_{wDA}$均在同一个最终测试集上进行测试。该设置使我们能够比较数据增强的有效性。

4.1.3 结果。表 2 展示了实验结果。在 API 实体抽取方面， $AERJE_{wDA}$的精确率、召回率和 F1 分数分别为 97.57 %、95.48% 和 96.51%，而 $AERJE_{w/oDA}$ 的精确率、召回率和 F1 分数分别为 95.11%、92.19% 和 93.63%。 $AERJE_{wDA}$的精确率、召回率和 F1 分数均高于 $AERJE_{w/oDA}$的，其中 $AERJE_{wDA}$的召回率和 F1 分数高出约 3%。

在 API 关系抽取方面， $AERJE_{wDA}$的精确率、召回率和 F1分数分别为 86.54%、76.48% 和 81.20%，而 $AERJE_{w/oDA}$ 的精确率、召回率和 F1分数分别为 77.71%、75.66% 和 76.67%。$AERJE_{wDA}$的精确率、召回率和 F1分数均高于 $AERJE_{w/oDA}$。$AERJE_{wDA}$的精确率比 $AERJE_{w/oDA}$高 8.83%， $AERJE_{wDA}$的 F1分数比 $AERJE_{w/oDA}$高 4.53%。这表明，使用大量包含 API关系的模糊句子对 $AERJE_{wDA}$进行微调，有助于 $AERJE_{wDA}$ 区分关系与非关系，以及正确关系与错误关系。相比之下， $AERJE_{w/oDA}$ 并未使用模糊句子进行微调，因此表现不如 $AERJE_{wDA}$。例如，在模糊句子“你想了解一下 processbuilder，以启动 exe 文件，然后使用 waitfor() 等待进程完成”中， $AERJE_{wDA}$ 正确地从句子中抽取出两个 API实体：*ProcBuilder* 和*waitfor()*，以及它们之间的“逻辑约束”关系。相比之下， $AERJE_{w/oDA}$ 仅从句子中抽取出一个 API，即*waitfor()*。这表明，通过使用增强数据进行微调，可以提升 AERJE 从模糊句子中抽取 API实体和关系的能力。

> 通过使用数据增强策略获得的、含糊但标注正确的句子对 AERJE 进行微调，可以提升其区分 API实体和关系的能力。

4.2 RQ2：动态提示的最佳关系类型数量

4.2.1 动机。 如第 2.1.2 节所述，给定一个输入句子，动态提示生成器会使用基于BERT的分类器预测一组候选关系类型，并将其纳入动态提示，以指导后续的实体关系联合抽取器。在本项研究问题中，我们希望探究多少个候选关系类型（即分类器排名前 N 的结果）能够为抽取器提供最有效的指导。

4.2.2 方法。 我们穷举动态提示生成器中 N 值（从 1 到 6）的所有情况，然后在相同的最终训练集上对 AERJE 进行微调，并在相同的最终测试集上对其进行测试，根据实验结果选出最合适的 N 值。我们不测试 N=7，因为它实质上就是包含全部七种关系类型的静态提示（即 $AERJE_{w/oDPG}$，将在 RQ3 中研究）。

4.2.3 结果。 如表 3所示，改变 N 值对实体抽取的影响很小，因为 N 表示动态提示中的关系类型数量，不会直接影响实体抽取。当 N=3时，AERJE 的 API 实体抽取 F1分数略高，为 96.51%。

表 3. 不同 top-N 值下的模型结果 N

top-N	实体			关系		
	P	R	F1	P	R	F1
1	96.88	94.23	95.54	75.92	71.92	73.87
2	97.04	95.18	96.10	77.75	73.80	75.72
3	97.57	**95.48**	**96.51**	**86.54**	**76.48**	**81.20**
4	**97.84**	94.39	96.08	83.51	73.22	78.03
5	96.72	94.39	95.54	77.90	73.30	75.53
6	96.44	94.75	95.59	75.35	72.61	73.95

对于关系抽取，改变 N 值对精确率和召回率的影响更大。随着 N 增大，精确率和召回率都会提升，直到 N=3。当 N=3时，AERJE 的精确率、召回率和 F1分数分别达到最高值 86.54%、76.48% 和 81.20%。这意味着，分类器预测的排名前三的候选关系中最有可能包含正确的 API 关系类型。然而，当 N 小于 3 时，AERJE 在关系抽取上的 F1分数会下降，因为排名前 N 的候选关系可能遗漏正确的关系类型。以下是一个例子：“TreeMap 存在相同的局限性（ HashMap 也如此；如果其元素的哈希码在插入后发生变化，HashMap 也会失效）”。当 N=2 时，分类器预测 *TreeMap* 和 ***HashMap***之间存在两种关系，包括“行为差异”和“逻辑约束”，但忽略了“函数相似性”关系。分类器预测的第三种关系正是这一被忽略的关系。然而，当 N 大于 3 时，AERJE 在关系抽取上的 F1分数会下降，因为动态提示可能包含一些错误的关系类型，从而误导抽取器。这种误导对精确率的影响大于对召回率的影响。

> 动态提示的最佳关系类型数量应设置为 3。这不仅能确保大多数正确的关系类型出现在动态提示中，还能防止动态提示包含过多噪声关系类型，因为这可能会使模型以牺牲精确率为代价来提高召回率。

4.3 RQ3：AERJE 的联合抽取性能

4.3.1 动机。 我们希望评估 AERJE 在 API 实体与关系联合抽取方面的性能，并将其与用于 API 实体抽取和 API关系抽取的最先进方法进行比较。请注意，只有我们的 AERJE 能够实现 API 实体与关系的联合抽取。

4.3.2 方法。 对于 API实体抽取，我们将 AERJE 与 APIReal 和 ARCLIN 进行比较；对于 API 实体与关系抽取，则将其与三个变体模型（即 $AERJE_{w/oDPG}$、 $AERJE_{single}$和 $AERJE_{base}$）进行比较。请注意， $AERJE_{single}$ 的实体与关系抽取结果体现了分别针对这两项任务对预训练模型进行微调的能力。所有模型均在相同的最终训练集和测试集上进行训练和测试。配置详情见第 3.3 节。

4.3.3 结果。表4 展示了 AERJE 和五种基线方法在最终测试集上的评估结果。可以看出，在 API 实体抽取方面，AERJE 的 F1 分数比 APIReal 高 7.5%，比 ARCLIN 高 5.7%。与三种变体模型相比，AERJE 在 API 实体抽取方面的 F1分数仅略低于表现最佳的模型（低 0.18 %；即 $AERJE_{single}$的 96.69%），但在关系抽取方面，AERJE 的 F1分数比表现第二好的模型（ $AERJE_{single}$的 74.37%）高 6.83%。

表4. 总体性能比较

模型	实体			关系		
	P	R	F1	P	R	F1
APIReal	89.13	88.90	89.01	-	-	-
ARCLIN	94.76	87.17	90.81	-	-	-
AERJE	97.57	95.48	96.51	**86.54**	**76.48**	**81.20**
$AERJE_{single}$	**98.03**	95.38	**96.69**	82.83	67.47	74.37
$AERJE_{w/oDPG}$	97.52	**95.78**	96.64	75.38	70.62	72.92
$AERJE_{base}$	96.39	95.15	95.77	75.97	70.28	73.01

在 API 实体抽取方面，AERJE 及其变体模型的表现都远超 APIReal 和 ARCLIN。这种优异表现得益于 AERJE 所采用的预训练大型语言模型（T5）。在预训练过程中，T5 学习文本中的语言知识和语义知识，并具备强大的词语和句子表征能力。通过微调，T5 中蕴含的语义知识可以迁移到下游任务中，从而提升 API 实体抽取效果。

T5 中的知识量也会影响 AERJE 在 API 实体与关系抽取任务上的性能。与 AERJE 相比，$AERJE_{base}$ 在 API实体抽取和 API关系抽取中的 F1分数分别降低了 0.74% 和 8.19%。与 API关系抽取相比，$AERJE_{base}$ 在 API实体抽取上的 F1 分数降幅非常小。这是因为最终训练集中包含 API实体的句子数量是同时包含 API实体和关系的句子数量的 6 倍（即 10,678 条对 1,639 条）。充足的 API实体抽取微调数据使基础 T5 模型在 API实体抽取上的性能达到与大型 T5 相当的水平。相比之下，关系抽取比实体抽取更加复杂，且微调数据量更少。在这种情况下，基础 T5 无法与大型 T5 竞争。

对于 $AERJE_{single}$ 和 AERJE，它们的实体抽取性能几乎相同。然而，在 API关系抽取方面，AERJE 的 F1分数（81.20%）、精确率（86.54%）和召回率（76.48%）分别远高于 $AERJE_{single}$ 的 F1分数（74.37%）、精确率（82.83%）和召回率（67.47%）。这表明，单独对预训练模型进行 API实体抽取微调，或将其与 API关系抽取联合微调，都不会影响 API实体抽取的质量。但与仅针对关系抽取任务微调模型相比，联合进行实体与关系抽取对关系抽取任务更为有效。

对于 $AERJE_{w/oDPG}$ 和 AERJE，它们的实体抽取性能也几乎相同。这是因为动态提示只影响关系类型，而不影响实体类型。在 API关系抽取方面，AERJE 的精确率（86.54%）、召回率（76.48%）和 F1分数（81.20%）分别高于 $AERJE_{w/oDPG}$的精确率（75.38%）、召回率（70.62%）和 F1分数（72.92%）。这是因为 $AERJE_{w/oDPG}$ 对所有输入句子使用相同的静态提示，其中包含全部七种关系类型。提示中包含的关系类型越多，提示中的噪声就越大，AERJE 就越难识别并抽取输入句子中的正确关系。相比之下，AERJE 使用动态提示，减少了需要识别的关系类型数量，从而提升了其抽取 API关系的能力。

借助大型预训练语言模型（T5），AERJE 在 API 实体抽取方面优于传统序列标注模型。动态提示对 API 实体抽取没有影响，但能大幅提升 API 关系抽取的性能。联合微调预训练模型，远比仅针对单项任务微调模型有效，这使实体-关系联合抽取在两项任务上的准确度都高于分别进行实体抽取和关系抽取。

4.4 RQ4：AERJE 的泛化能力

4.4.1 研究动机。 每个 API 都有其对应的 API 包，而这些 API 包的形式往往各不相同。此外，不同 API 包中的 API 支持的功能各异，因此它们出现的文本在内容和语言特征上可能有所不同。AERJE 在微调期间不可能见到所有 API 包。在本研究问题中，我们希望探究 AERJE 是否能够识别来自微调期间未见过的 API 包的 API 及其关系。

*4.4.2 方法。*为了尽可能收集来自不同包的更多数据，我们将最终训练集和最终测试集合并为一个包含 13,364 个句子的新数据集。如第 2.3.1 节所述，每个句子都带有多个帖子标签，其中一些标签表示该句子与 API 包之间的关系。例如，标签“io”与包名“java.io”相关联。因此，我们将句子的每个标签与任意一个 JDK 1.8 包名进行匹配，以筛选出带有包名的句子。这里采用部分匹配，即只需匹配包名的一部分，例如，“swing”可以匹配“javax.swing”。然后，我们汇总与句子一起出现的包名，并选出出现频率最高的三个包名（即 javax.swing、java.io 和 java.util）。最后，我们收集了 1651 个标签与这三个包名相匹配的句子。

为确保通过近似匹配获得的句子的正确性，我们邀请了六名学生（此前曾参与标注），并将他们分为三组，分别对来自三个不同包的句子进行标注。每组的两名学生对相同的句子进行标注。他们独立判断每个句子中的 API实体 是否来自指定的包（即 java.io、java.util、javax.swing）。以下是一个针对 java.io 包的示例：“you can use lines() method in BufferedRead”。该句子被标注为 True，因为其中的 API实体*line()* 和 *BufferedRead* 仅对应于 java.io。相反，如果句子中的任何 API实体不属于指定的包，则该句子被标注为 False。随后，我们指派一名作者处理组员之间的冲突。最终，我们获得了 999 个与这些包名称严格匹配的句子。Cohen’s Kappa [25] 系数为 0.795（即达成了实质性一致）。每个包的数据详情如下：

- java.io 数据集包含 235 个句子，其中 51 个同时包含实体和关系。这 51 个句子中有 12 个是未增强句子。
- javax.swing 数据集包含 435 个句子，其中 76 个同时包含实体和关系。这 76 个句子中有 14 个是未经增强的句子。
- java.util 数据集包含 329 个句子，其中 68 个同时包含实体和关系。这 68 个句子中有 18 个是未经增强的句子。

我们的 AERJE 和基线模型均在三个数据集中的一个上训练，并在另外两个数据集上测试。由于 AERJE 的表现优于其变体，因此我们在此不考虑这些变体。

*4.4.3 结果。*表5 展示了反映各模型泛化能力的结果。对于 API 实体抽取，AERJE 在 java.util 数据集上训练时，F1分数达到 95.05%，远高于 APIReal 的 F1分数（61.27%）和 ARCLIN 的 F1分数（58.50%）。我们认为，这归功于 AERJE 所基于的大语言模型。正如 Qiu 等人 [17] 所指出的，大语言模型能够提供更好的模型初始化，这通常会带来更好的目标任务泛化性能。在 java.io 数据集和 javax.swing 数据集上训练模型，也能观察到类似现象。总体而言，ARCLIN 和 APIReal 可能在精确率或召回率其中一项上表现良好，但无法兼顾，因此 F1分数较低。相比之下，AERJE 的表现非常稳定，精确率和召回率都显著更高，且各项实验中的 F1分数仅有小幅波动。

对于 API 关系抽取，当分别在 java.io、java.util 和 javax.swing 数据集上训练时，AERJE 的 F1分数分别为 40.98%、79.99% 和 68.48%。在跨包训练-测试

表 5. 泛化能力比较

模型	java.io						java.util						javax.swing					
	实体			关系			实体			关系			实体			关系		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
APIReal	85.02	36.97	51.53	-	-	-	98.11	44.54	61.27	-	-	-	**98.99**	25.62	40.70	-	-	-
ARCLIN	**95.93**	70.84	81.50	-	-	-	**98.55**	41.60	58.50	-	-	-	98.64	56.57	71.90	-	-	-
AERJE	92.00	**89.35**	**90.66**	45.87	37.03	40.98	95.17	**94.93**	**95.05**	78.68	81.35	79.99	93.89	**89.91**	**91.86**	96.92	52.94	68.48

表6. 低资源场景下的实验结果

模型	单样本						五样本						十样本					
	实体			关系			实体			关系			实体			关系		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
APIReal	**80.30**	15.92	26.57	-	-	-	**86.17**	60.67	71.20	-	-	-	**83.94**	68.59	75.49	-	-	-
ARCLIN	55.07	62.38	58.50	-	-	-	74.64	72.91	73.76	-	-	-	83.58	75.52	79.34	-	-	-
AERJE	72.76	**85.06**	**78.43**	9.34	44.49	15.44	79.09	**91.62**	**84.90**	31.68	65.96	42.80	82.47	**93.74**	**87.74**	35.00	72.94	47.30

设置下，与非跨包设置相比，AERJE 的性能有所下降（见表 4）。然而，在 java.util 数据集上训练时，AERJE 的 F1分数（79.99%）仅比非跨包设置（81.20%）低约 1%。这表明 AERJE 能够应对不同包之间的数据漂移。此外，不同的跨包训练-测试设置也会产生不同的结果。使用 java.util 训练 AERJE 时，其 F1分数比使用 java.io 训练的 AERJE 高约 39%。首先，这是因为 java.io 中包含关系的句子少于 java.util （ 51 对 68 ）。其次，java.io 数据中包含关系的非增强句子也少于 java.util（12 对 18），因此 java.io 数据的多样性不如 java.util。

我们的 AERJE 面对不同 API 包之间的数据漂移时，具有很强的泛化能力。这种能力源自底层大语言模型的泛化能力。

4.5 研究问题5：AERJE 在低资源场景中的表现

4.5.1 研究动机。 人工开销意味着可用于训练的数据有限。在本研究问题中，我们希望考察使用极少量训练数据进行训练时，AERJE 的表现如何。

4.5.2 研究方法。 我们开展 K-shot 实验，其中 K 可以取 1、5 或 10。首先，在 K-shot 实验中，我们从最终训练集中为每种关系类型随机选取 K 个句子。随后，我们从最终训练集中随机选取 K 个仅包含实体、不包含关系的句子。由此得到一个包含 8*k 个句子的训练集。最后，我们使用该训练集训练 AERJE 和基线模型，并在最终测试集上进行测试。请注意，为避免随机抽样的影响，我们使用不同样本将每个 K-shot 实验重复十次。

4.5.3 实验结果。 对于 API实体抽取，表 6 展示了各模型在三种低资源场景（即单样本、五样本和十样本）下的表现，其中 AERJE 显著优于 APIReal 和 ARCLIN。尤其是在单样本场景下，AERJE 的 F1分数为 78.43%，显著高于 APIReal（26.57%）和 ARCLIN（58.50%）。与 APIReal 和 ARCLIN 相比，基于大语言模型的 AERJE 通过大语言模型预训练获得了大量先验知识。随着微调样本数增加，AERJE 的准确率迅速提升，尤其是 F1分数，在五样本时达到 84.90%，在十样本时达到 87.74%。

对于 API 关系抽取，在单样本场景下，AERJE 的表现不佳，但仍出人意料地达到了 44.49% 的召回率。然而，只需再增加 4 个示例（即五样本），AERJE 的 F1分数便从单样本时的不到 16% 大幅提升至五样本时的约 43%。这表明，底层大语言模型能够迅速适应其在预训练期间未见过的 API关系抽取任务

，而且只需少量示例。在少样本设置下，我们发现，精确率比召回率更难提升。这可能是由于关系存在歧义，即相同类型的关系可以用截然不同的形式表达（如表 1 所示），而不同类型的关系也可能采用相似的形式表达（例如，“API1 be ADJ to API2”既可以表示函数相似性，也可以表示函数相反关系）。每种关系类型只有少量示例，使得模型更难学会区分它们。此外，AERJE 在单样本下的实体抽取 F1分数为 78.43%，而关系抽取 F1分数仅为 15.44%。其主要原因是，单样本训练集几乎涵盖了所有 API 实体歧义类型，却只涵盖了少数几种 API 关系歧义类型。因此，关系抽取比实体抽取更难。

AERJE 只需少量微调数据，便能快速将底层大语言模型适配到 API 实体和关系抽取任务。大语言模型中的先验知识使这种快速适配成为可能。在少样本设置下，关系抽取比实体抽取困难得多。

5 讨论

对内部效度的主要威胁来自训练和测试数据集的人工标注。错误的人类标注可能会影响模型的训练和测试。为降低这一威胁，我们邀请两名学生对相同内容进行标注，并指定一名作者解决标注结果中的分歧。然而，即使是人类，有时也无法判断某个词元是否指代 API，尤其是指代基本计算概念的常见名词，例如 *policy* 和 *time*，它们既可能是基本名词概念，也可能是 API（*java.security.policy* 类、*java.time* 包）。我们采取保守策略：将常见名词标注为 API 实体，除非两位标注者达成一致。

外部效度面临三方面威胁。第一项外部威胁是，我们仅从 Stack Overflow 收集数据。尽管我们的模型在 SO 数据集上表现良好，但我们计划进一步验证其在其他数据源（例如，Java Tutorial[8], SitePoint[9],和 Reddit[10]）上的泛化性能。第二项外部威胁是，AERJE 仅在 Java 包上进行过测试。我们选择 Java，是因为先前的研究 [1, 6, 11] 已证明，从 Java 中抽取这些 API实体和关系有多么困难。未来，我们计划将 AERJE 扩展到其他编程语言（如 Python 和 C#）。第三项外部威胁源于 AERJE 的两个组件：基于BERT的分类器和基于 T5 的抽取器。这两个模型组件都有许多替代模型。TextCNN [27] 和 FastText [28]可用于构建分类器；也可以使用 BART [31] 和 GPT-3 [32] 来实现抽取器。未来，我们将把 AERJE 的两个组件与替代模型进行比较，以确定性能最佳的模型。

6 相关工作

API 实体与关系抽取是软件工程中的一项基础工作。它有助于构建知识图谱；抽取出的结构化 API 知识能够支持许多软件工程任务，例如 API链接 [2, 3, 8, 33], API 误用检测 [11], API推荐 [4, 5], 和 API 比较 [6]。本节介绍从非结构化文本中抽取 API实体和关系的方法。

Bacchelli [7, 34] 和 Dagenais [3] 利用不同正字法特征的正则表达式，从开发者电子邮件、文档和论坛帖子中检测类和方法提及。Ren [11],、Huang [1], 和 Liu [6] 利用 HTML <code> 标签从 Stack Overflow 帖子中抽取实体。Bacchelli 等人 [10] 利用岛屿

[8] http://www.java2s.com [9] https://www.sitepoint.com/ [10] https://www.reddit.com

解析。Huang [1], 和 Liu [6] 基于句法模式抽取实体之间的语义关系。然而，他们从自然语言文本中抽取 API 实体和关系的方法依赖于 API 独特的正字法特征，并且规则设计开销较大。

为减轻规则设计带来的开销，研究人员采用机器学习方法抽取 API实体。Ye等人 [2] 提出了 APIReal，该方法使用条件随机场识别 API实体。他们用“BIO”标签标注句子中的 API实体和非实体，并将标注序列与句子组成配对。随后，他们使用这些配对训练条件随机场，再用训练好的条件随机场标注输入文本，从而提取带有“BI”或“B”标签的 API实体。另一方面，Huo等人 [13], 提出了 ARCLIN，该方法使用双向长短期记忆网络作为编码器、条件随机场作为解码器来识别 API实体，而非仅使用条件随机场。然而，这些方法需要付出数据标注的开销，因为要为这些序列标注模型准备大量高质量训练数据并不现实。

为解决上述两种开销问题，研究人员使用大语言模型抽取实体。Li等人 [35] 使用 BERT，Yan等人 [36] 使用 XLNet [37] 抽取自然语言领域中的实体。然而，这些模型仅限于单一的自然语言处理任务，即实体抽取。为实现多个任务的联合抽取，研究人员提出了基于大语言模型的统一架构模型，例如 UIE [20] 和 OpenUE [38]。具体而言，UIE提出了 SEL，通过分层定位关联结构对不同的信息抽取结构进行编码。受此启发，我们考虑将 UIE调整用于API实体与关系联合抽取。然而，UIE不擅长处理复杂句子，尤其是包含API 实体和多种关系的长句和模糊句子，因为UIE仅使用一个静态提示来识别所有类型的API关系。因此，面对模糊句子时，需要识别的关系类型越多，噪声干扰就越强，UIE的识别率也越低。相比之下，我们提出了基于大语言模型的AERJE，可同时从非结构化复杂句子中抽取API实体和关系。与UIE不同，我们的动态提示设计能够为输入文本生成少量可能相关的关系，从而消除噪声干扰并降低API关系抽取的难度。

7 结论与未来工作

本文首次将异构 API 抽取和 API 关系抽取任务表述为序列到序列任务，并提出 AERJE，利用预训练大语言模型和动态提示学习，从非结构化文本中同时抽取 API实体及其关系。我们使用 Stack Overflow 中一组较长且模糊的句子，对 AERJE 进行了系统评估。实验结果表明，只需少量数据即可激发 AERJE 抽取 API实体及其关系的能力，使其能够从微调过程中从未见过的复杂文本中准确识别 API实体及其关系。未来，我们将落实讨论中提及的计划，并将 AERJE 应用于由 API 实体与关系抽取所支持的各类软件工程任务，例如 API链接、API搜索和 API推荐。

致谢

本研究部分获得以下项目资助：国家自然科学基金项目（编号：62262031、61902162）、江西省自然科学基金（20202BAB202015）、中央引导地方科技发展专项（20222ZDH04090）以及江西省研究生创新专项资金项目（YC2021-S308、YC2022-S258）。

参考文献

[1] Qing Huang、Zhiqiang Yuan、Zhenchang Xing、Zhengkang Zuo、Changjing Wang 和 Xin Xia。1+1>2：编程知识与实践知识融合、语义丰富及连贯应用。IEEE服务计算汇刊，页码1–14，2022年。[2] Deheng Ye、Lingfeng Bao、Zhenchang Xing 和 Shang-Wei Lin。APIReal：一种面向在线开发者论坛的API识别与链接方法。实证软件工程，23(6):3129–3160，2018年。[3] Barthélémy Dagenais 和 Martin P. Robillard。恢复API与其学习资源之间的可追溯性链接。2012年第34届国际软件工程会议（ICSE），页码47–57，2012年。[4] Qiao Huang、Xin Xia、Zhenchang Xing、David Lo 和 Xinyu Wang。无需担忧任务与API之间的知识鸿沟的API方法推荐。载于2018年第33届IEEE/ACM国际自动化软件工程会议（ASE），页码293–304。IEEE，2018年。[5] Wenkai Xie、Xin Peng、Mingwei Liu、Christoph Treude、Zhenchang Xing、Xiaoxin Zhang 和 Wenyun Zhao。通过显式匹配功能动词短语进行API方法推荐。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论文集，2020年。[6] Yang Liu、Mingwei Liu、Xin Peng、Christoph Treude、Zhenchang Xing 和 Xiaoxin Zhang。利用知识图谱生成基于概念的API元素比较。载于第35届IEEE/ACM国际自动化软件工程会议论文集，页码834–845，2020年。[7] Alberto Bacchelli、Michele Lanza 和 Romain Robbes。关联电子邮件与源代码制品。载于第32届ACM/IEEE国际软件工程会议论文集（第1卷），页码375–384，2010年。[8] Christoph Treude 和 Martin P. Robillard。利用 Stack Overflow 中的信息增强API文档。2016年IEEE/ACM第38届国际软件工程会议（ICSE），页码392–403，2016年。[9] Peter C. Rigby 和 Martin P. Robillard。从非正式文档中发现关键代码元素。2013年第35届国际软件工程会议（ICSE），页码832–841，2013年。[10] Alberto Bacchelli、Anthony Cleve、Michele Lanza 和 Andrea Mocci。利用岛屿解析从自然语言文档中抽取结构化数据。2011年第26届IEEE/ACM国际自动化软件工程会议（ASE 2011），页码476–479，2011年。[11] Xiaoxue Ren、Xinyuan Ye、Zhenchang Xing、Xin Xia、Xiwei Xu、Liming Zhu 和 Jianling Sun。由细粒度API约束知识图谱驱动的API误用检测。2020年第35届IEEE/ACM国际自动化软件工程会议（ASE），页码461–472，2020年。[12] Deheng Ye、Zhenchang Xing、Chee Yong Foo、J. Li 和 Nachiket Kapre。学习从非正式自然语言讨论中抽取API提及。2016年IEEE国际软件维护与演化会议（ICSME），页码389–399，2016年。[13] Yintong Huo、Yuxin Su、Hongming Zhang 和 Michael R. Lyu。ARCLIN：针对未格式化文本的自动API提及消歧。2022年IEEE/ACM第44届国际软件工程会议（ICSE），页码138–149，2022年。[14] Hongwei Li、Sirui Li、Jiamou Sun、Zhenchang Xing、Xin Peng、Mingwei Liu 和 Xuejiao Zhao。通过挖掘API注意事项知识图谱，提升API注意事项的可访问性。2018年IEEE国际软件维护与演化会议（ICSME），页码183–193，2018年。[15] Chunyang Chen、Zhenchang Xing 和 Ximing Wang。从非正式讨论中推断软件特定形态形式。2017年IEEE/ACM第39届国际软件工程会议（ICSE），页码450–461，2017年。[16] Fabio Petroni、Tim Rocktäschel、Patrick Lewis、Anton Bakhtin、Yuxiang Wu、Alexander H. Miller 和 Sebastian Riedel。语言模型能否充当知识库？arXiv，abs/1909.01066，2019年。[17] Xipeng Qiu、Tianxiang Sun、Yige Xu、Yunfan Shao、Ning Dai 和 Xuanjing Huang。自然语言处理预训练模型综述。arXiv，abs/2003.08271，2020年。[18] Suyu Ma、Zhenchang Xing、Chunyang Chen、Cheng Chen、Lizhen Qu 和 Guoqiang Li。通过多层级特征嵌入和迁移学习实现易于部署的API抽取。IEEE软件工程汇刊，47(10):2296–2311，2019年。[19] Zexuan Zhong 和 Danqi Chen。一种出乎意料地简单的实体与关系抽取方法。arXiv，abs/2010.12812，2021年。[20] Yaojie Lu、Qing Liu、Dai Dai、Xinyan Xiao、Hongyu Lin、Xianpei Han、Le Sun 和 Hua Wu。用于通用信息抽取的统一结构生成。载于ACL，2022年。[21] Colin Raffel、Noam M. Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li 和 Peter J. Liu。探索统一文本到文本转换器的迁移学习极限。arXiv，abs/1910.10683，2020年。[22] Yoav Levine、Itay Dalmedigos、Ori Ram、Yoel Zeldes、Daniel Jannai、Dor Muhlgay、Yoni Osin、Opher Lieber、Barak Lenz、Shai Shalev-Shwartz、Amnon Shashua、Kevin Leyton-Brown 和 Yoav Shoham。站在巨型冻结语言模型的肩膀上。arXiv，abs/2204.10019，2022年。

[23] Deheng Ye、Zhenchang Xing、Chee Yong Foo、Zi Qun Ang、Jing Li 和 Nachiket Kapre。《软件工程社交内容中的软件专用命名实体识别》。载于*2016*年*IEEE*第*23*届国际软件分析、演化与再工程会议（*SANER*），第1卷，第90–101页。IEEE，2016年。[24] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。《BERT：用于语言理解的深度双向 Transformer 预训练》。*arXiv*，abs/1810.04805，2019年。[25] J. R. Landis 和 G. Koch。《层次型卡帕统计量在评估多名观察者之间多数一致性方面的应用》。*Biometrics*，33卷2期：363–74，1977年。[26] Mingwei Liu、Xin Peng、Andrian Marcus、Christoph Treude、Jiazhan Xie、Huanjun Xu 和 Yanjun Yang。《如何在软件开发中提出具体的操作指南类问题？》。载于*FSE*，2022年。[27] Yoon Kim。《用于句子分类的卷积神经网络》。载于*EMNLP*，2014年。[28] Armand Joulin、Edouard Grave、Piotr Bojanowski 和 Tomas Mikolov。《高效文本分类的实用技巧集》。载于*EACL*，2017年。[29] Kelvin Jiang、Ronak Pradeep、Jimmy J. Lin 和 David R. Cheriton。《探索使用 T5 进行事实核验的列表式证据推理》。载于*ACL*，2021年。[30] Diedre Carmo、Marcos Piau、Israel Campiotti、Rodrigo Nogueira 和 Roberto de Alencar Lotufo。《PTT5：基于巴西葡萄牙语数据对 T5 模型进行预训练与验证》。*arXiv*，abs/2008.09144，2020年。[31] Mike Lewis、Yinhan Liu、Naman Goyal、Marjan Ghazvininejad、Abdelrahman Mohamed、Omer Levy、Veselin Stoyanov 和 Luke Zettlemoyer。《BART：用于自然语言生成、翻译和理解的去噪序列到序列预训练》。载于*ACL*，2020年。[32] Tom B. Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell、Sandhini Agarwal、Ariel Herbert-Voss、Gretchen Krueger、T. J. Henighan、Rewon Child、Aditya Ramesh、Daniel M. Ziegler、Jeff Wu、Clemens Winter、Christopher Hesse、Mark Chen、Eric Sigler、Mateusz Litwin、Scott Gray、Benjamin Chess、Jack Clark、Christopher Berner、Sam McCandlish、Alec Radford、Ilya Sutskever 和 Dario Amodei。《语言模型是少样本学习者》。*arXiv*，abs/2005.14165，2020年。[33] Hang Yin、Yuanhao Zheng、Yanchun Sun 和 Gang Huang。《面向经验不足的开发者、基于 API 知识图谱的 API 学习服务》。载于*2021*年*IEEE*国际*Web*服务会议（*ICWS*），第251–261页。IEEE，2021年。[34] Alberto Bacchelli、Marco D'Ambros、Michele Lanza 和 Romain Robbes。《用于关联电子邮件与源代码的轻量级技术基准评测》。*2009*年第*16*届逆向工程工作会议，第205–214页，2009年。[35] Xiaoya Li、Jingrong Feng、Yuxian Meng、Qinghong Han、Fei Wu 和 Jiwei Li。《用于命名实体识别的统一机器阅读理解框架》。*arXiv*，abs/1910.11476，2020年。[36] Rongen Yan、Xue Jiang 和 Depeng Dang。《使用 XLNet-BiLSTM-CRF 进行命名实体识别》。*Neural Process. Lett.*，53：3339–3356，2021年。[37] Zhilin Yang、Zihang Dai、Yiming Yang、Jaime G. Carbonell、Ruslan Salakhutdinov 和 Quoc V. Le。《XLNet：用于语言理解的广义自回归预训练》。载于*NeurIPS*，2019年。[38] Ningyu Zhang、Shumin Deng、Zhen Bi、Haiyang Yu、Jiacheng Yang、Mosha Chen、Fei Huang、Wei Zhang 和 Huajun Chen。《OpenUE：文本通用抽取的开放工具包》。载于*EMNLP*，2020年。



QING HUANG 于2009年获得南昌大学计算机应用与技术硕士学位，并于2018年获得武汉大学计算机软件与理论博士学位。目前，他是中国江西师范大学计算机与信息工程学院的助理教授。他的研究兴趣包括信息安全、软件工程和知识图谱。



Yanbang Sun 是中国江西师范大学计算机与信息工程学院的一名硕士二年级学生。他的研究兴趣包括软件工程和知识图谱。



Zhenchang Xing 是澳大利亚联邦科学与工业研究组织 Data61 位于澳大利亚新南威尔士州伊夫利的高级研究科学家。此外，他还是澳大利亚国立大学计算机科学研究院的副教授。此前，他于 2012 至 2016 年在新加坡南洋理工大学计算机科学与工程学院担任助理教授。他的主要研究领域包括软件工程、应用数据分析和人机交互。



MIN YU是江西师范大学通信、电子工程与计算机科学专业的教授，曾任美国加州大学欧文分校访问学者，研究兴趣包括分布式计算、无线传感器网络和室内定位。



Xiwei Xu是澳大利亚联邦科学与工业研究组织 Data61 架构与分析平台团队的高级研究科学家，也是新南威尔士大学的联合讲师。她自 2015 年起开始从事区块链研究。她的主要研究兴趣是软件架构，也研究服务计算、业务流程、云计算和可信赖性。



Qinghua Lu 是澳大利亚新南威尔士州伊夫利的澳大利亚联邦科学与工业研究组织 Data61 高级研究科学家。她已在国际期刊和会议上发表 100 多篇学术论文。她的研究兴趣包括软件架构、区块链、人工智能软件工程和人工智能伦理。