|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 论文题目 | Summarizing source code with hierarchical code representation | | | |
| Paper URL | [Summarizing source code with hierarchical code representation | Semantic Scholar](https://www.semanticscholar.org/paper/Summarizing-source-code-with-hierarchical-code-Zhou-Yu/8fe2af818da11a39d3fa8f6176861e2b0182e7af) | | | |
| Project URL | https://github.com/zy-zhou/HACS-release | | | |
| 综述/背景介绍 | 发展状况 | 原因 | 意义 | 关键词（速记词汇、信息索引词汇） |
| 高质量的代码摘要，以注释或文档等形式，不仅可以帮助开发人员理解程序，而且有助于代码搜索和代码分类等重要任​​务。随着软件规模和复杂性的不断增加，代码摘要在软件开发和维护的生命周期中发挥着至关重要的作用。代码摘要旨在为代码自动生成自然语言描述，并已成为一个快速扩展的研究领域。基于神经网络的数据驱动代码摘要模型近年来激增。几乎所有现有的神经模型都建立在token或 AST 节点的粒度上。这有几个缺点： a) 代码摘要需要高级代码知识，而token表示仅限于提供全局视图； b) 这种方法很难对代码的层次结构进行建模； c) 由于大量的token和 AST 节点，长输入代码挑战这些模型来处理远程依赖关系。  我们提出了一个新的框架来利用代码的层次表示来生成更好的摘要。我们考虑两个级别的代码层次结构：token级别和语句级别。我们的框架包含一对定制的编码器-解码器模型，分别用于token和代码 AST。它们中的每一个都有一个分层编码器，旨在提取token和语句级代码特征，以及一个能够在解码过程中处理这些不同表示级别的注意力解码器。然后将它们组合起来，通过集成学习预测摘要。 | | | Code summarization  Program comprehension  Deep learning  Hierarchical attention |
| 假设 | **考虑token→语句→代码的层次结构**。这遵循人类参与者的自然行为来总结代码片段：首先扫描代码以找到方法头、API 调用和控制流等关键语句，然后在这些语句中收集更具体的信息，例如标识符和字符串，以提供简要说明。 | | |  |
| 方法描述(含图) | 我们提出了一个新的框架来利用代码的层次表示来生成更好的摘要。在我们的框架中，输入代码的token序列和 AST 按语句拆分，然后输入一对定制的编码器-解码器模型。每个编码器-解码器模型都配备了一个分层编码器，旨在提取token（节点）和语句级代码特征，以及一个能够在解码期间关注这些不同表示级别的注意力解码器。我们的编码器设计参考了用于文档分类的分层注意网络（HAN）。最后，将这两个模型结合起来，通过集成学习来预测摘要。 | | |  |
| 实验设计 |  | | |  |
| 数据处理 | 输入 | 处理方式 | | 关键词（速记词汇、信息索引词汇） |
| 源代码和AST | 源代码拆分：通常每行编写一个语句，而大括号（例如 Java）和复杂表达式的样式可能会有所不同。如果直接按行拆分代码会丢失一些结构信息，例如嵌套。因此，我们首先取消长语句中的断点，在无括号的单行条件语句中添加大括号，然后按照 Allman 样式格式化代码。之后，我们将代码逐行拆分以获得语句序列。然后将每个语句拆分为一个token序列。标识符和字符串根据snack\_case 和camelCase 进一步拆分为子token。  Ast拆分：我们按顺序遍历 AST 以确定节点应该属于哪个语句。如果一个节点有一个子命名块（例如 WhileStatement 和 ForStatement）或主体（例如 MethodDeclaration和 ClassDeclaration），它将引导一组嵌套语句。它也是诸如方法头之类的语句的根，因此它的后代（不包括其块或主体中的后代）将首先被收集为语句子树。其块或主体节点的（直接）子节点是其后续嵌套语句的根。对于这些孩子中的每一个，如果它还包含块或主体，那么它就像以前一样被对待。否则，它将与它的所有后代一起包含在一个新的子树中。相应地可以得到语句子树的序列。这些子树的出现顺序与代码中它们对应的语句的顺序相同，因此我们使用前序节点序列来表示子树。终端 AST 节点有一个值字段，它是代码中某处的token，并且可能包含丰富的语义。类似地，如果它是标识符或字符串，我们进一步拆分token，然后将子token视为该节点的孩子。这些子token的顺序在遍历期间保持不变。 | |  |
| 结论 | 我们提出了一种新的编码器-解码器架构用于代码摘要，其中编码器提取代码的层次表示，解码器充分利用这些特征来更好地生成摘要。我们通过集成解码利用token序列和 AST 的输入。我们还提供了详细的数据处理，包括输入的构建。总之，关注代码的层次表示可以显着提高神经模型的性能。与基于token或 AST 节点的模型不同，我们的方法可以更好地学习代码的全局信息，并在摘要生成期间在重要语句之间转移注意力。在intra attention和decoder attention的配合下，他们能够以自上而下的方式更准确地定位关键词。此外，集成学习被证明是一种从多个输入源中受益的强大方法，并且可以被视为现有组合策略的替代方案。对于未来的工作，需要研究更复杂的代码层次结构。我们还将尝试将我们的方法扩展到高级输入形式和神经网络，例如 GNN 和 TreeRNN。 | | | |
| 局限性分析 |  | | | |

**（论文名）：**基于分层代码表示的代码摘要生成

**（题目）：**《Summarizing source code with hierarchical code representation》

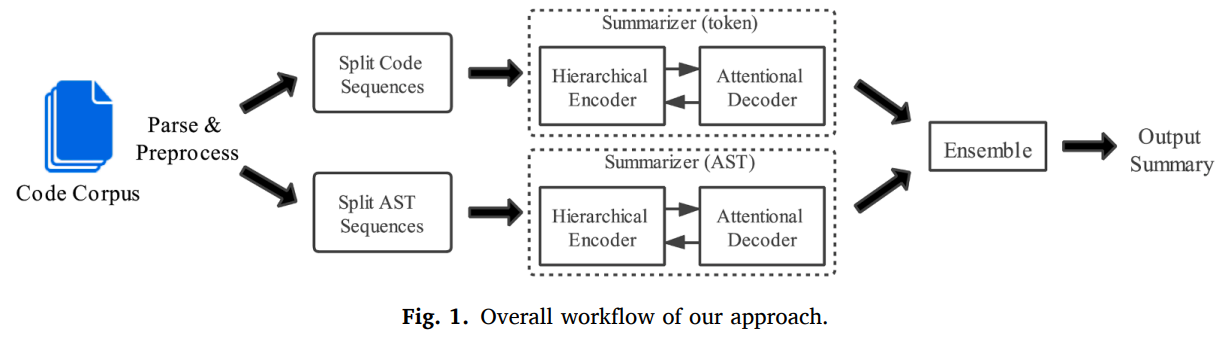
**（论文URL）：**论文地址: [Summarizing source code with hierarchical code representation | Semantic Scholar](https://www.semanticscholar.org/paper/Summarizing-source-code-with-hierarchical-code-Zhou-Yu/8fe2af818da11a39d3fa8f6176861e2b0182e7af)

**（总结）：**我们提出了一种新的编码器-解码器架构用于代码摘要，其中编码器提取代码的层次表示，解码器充分利用这些特征来更好地生成摘要。我们通过集成解码利用token序列和 AST 的输入。具体的，在我们的框架中，输入代码的token序列和 AST 按语句拆分，然后输入一对定制的编码器-解码器模型。每个编码器-解码器模型都配备了一个分层编码器，旨在提取token（节点）和语句级代码特征，以及一个能够在解码期间关注这些不同表示级别的注意力解码器。实验结果表明，关注代码的层次表示可以显着提高神经模型的性能。与基于token或 AST 节点的summarizer模型不同，我们的方法可以更好地学习代码的全局信息，并在摘要生成期间在重要语句之间转移注意力。在intra attention和decoder attention的配合下，他们能够以自上而下的方式更准确地定位关键词。此外，集成学习被证明是一种从多个输入源中受益的强大方法，并且可以被视为现有组合策略的替代方案。

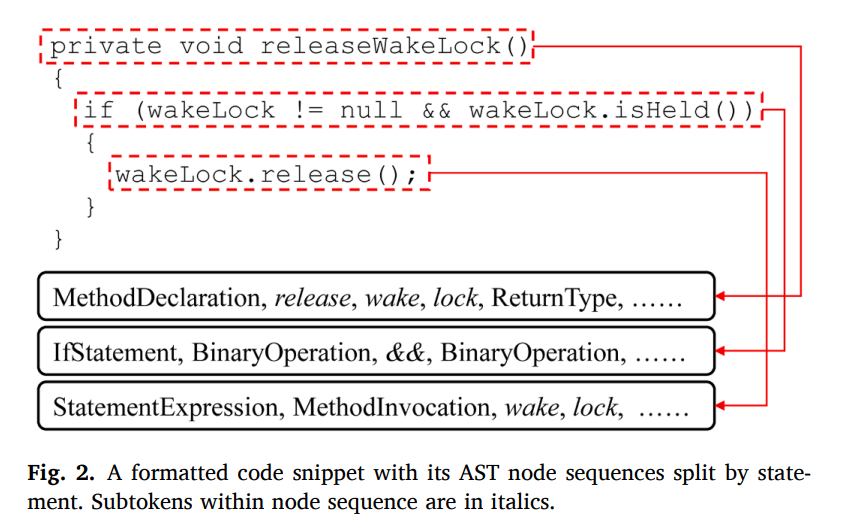
**（附图）：**

**我们最终模型的整体工作流程（预测期间）。如下：它由两个独立的编码器-解码器模型组成，分别接收由语句拆分的token序列和 AST 节点序列，称为summarizer（token）和summarizer（AST）。它们中的每一个都有一个分层编码器来更好地建模代码结构和一个解码器，它能够参考学习到的不同级别的代码表示。最终模型是它们的集合，即它们在每个时间步的目标词的预测概率在推理过程中被组合。**

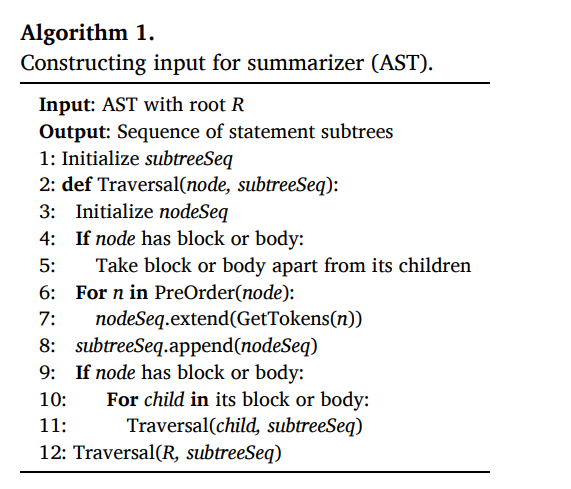
**集成：**为了将它们结合起来以从两种输入类型中受益，我们只需在每一步平均它们的输出概率向量，并将当前选择的目标词作为下一步的输入反馈给它们。

****

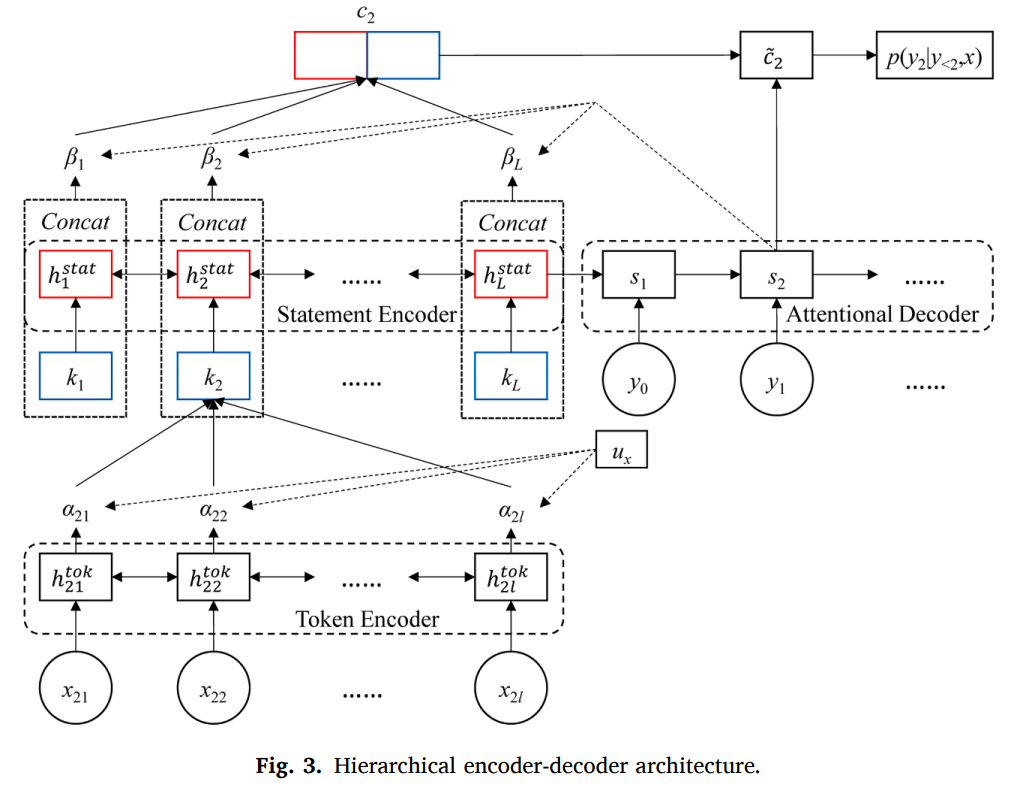
通常每行编写一个语句，而大括号（例如 Java）和复杂表达式的样式可能会有所不同。直接按行拆分代码会丢失一些结构信息，例如嵌套。因此，我们首先取消长语句中的断点，在无括号的单行条件语句中添加大括号，然后按照 Allman 样式格式化代码。之后，我们将代码逐行拆分以获得语句序列。然后将每个语句拆分为一个token序列。标识符和字符串根据snack\_case 和camelCase 进一步拆分为子token。



AST划分的伪代码。按顺序遍历 AST 以确定节点应该属于哪个语句。如果一个节点有一个子命名块（例如 WhileStatement 和 ForStatement）或主体（例如 MethodDeclaration 和 ClassDeclaration），它将引导一组嵌套语句。它也是诸如方法头之类的语句的根，因此它的后代（不包括其块或主体中的后代）将首先被收集为语句子树。其块或主体节点的（直接）子节点是其后续嵌套语句的根。对于这些孩子中的每一个，如果它还包含块或主体，那么它就像以前一样被对待。否则，它将与它的所有后代一起包含在一个新的子树中。相应地可以得到语句子树的序列。这些子树的出现顺序与代码中它们对应的语句的顺序相同，我们使用前序节点序列来表示子树。终端 AST 节点有一个值字段，它是代码中某处的token，并且可能包含丰富的语义。类似地，如果它是标识符或字符串，我们进一步拆分token，然后将子token视为该节点的孩子。这些子token的顺序在遍历期间保持不变。

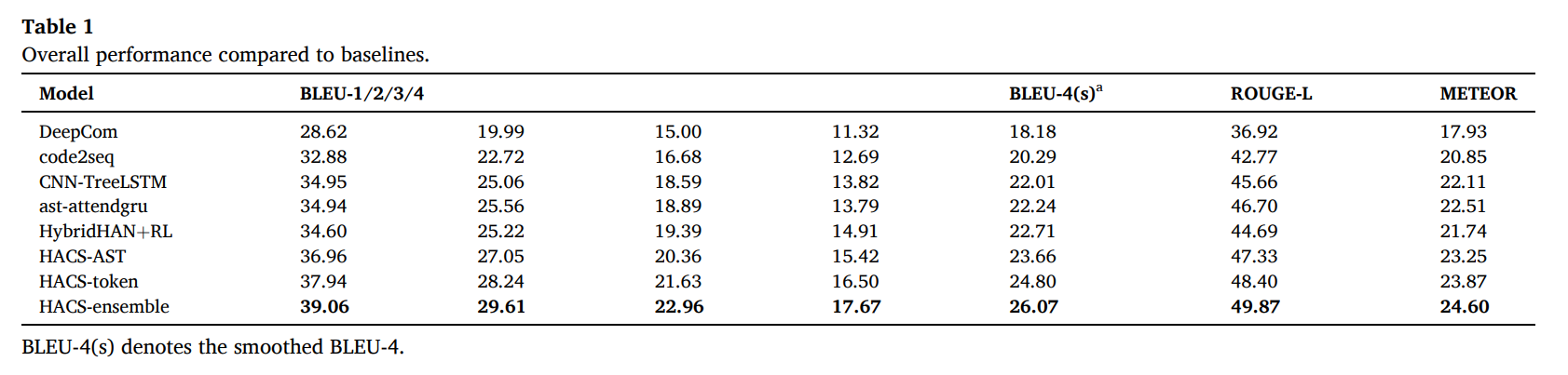


分层编码器由一个token编码器、一个语句编码器和一个token级内部注意力层组成，如图 3 所示。并非所有token都对语句的表示做出同等贡献，我们使用类似于的注意机制来提取重要部分并将低级表示聚合到语句向量 。解码器是一个单向 LSTM。它的初始状态是语句编码器的反向最终状态。我们让解码器在每个解码时间步 t 处理每个语句的token级和语句级表示，即 和 。具体来说，我们使用它们的串联 = [; ] 作为注意力机制的键和值元素。除了代码编码器，我们任务中最重要的部分是摘要生成，因此让解码器从编码结果中执行细粒度信息选择是直观的。在内部注意力和解码器注意力的配合下，解码器可以有效地评估编码结果 和 的两个级别。



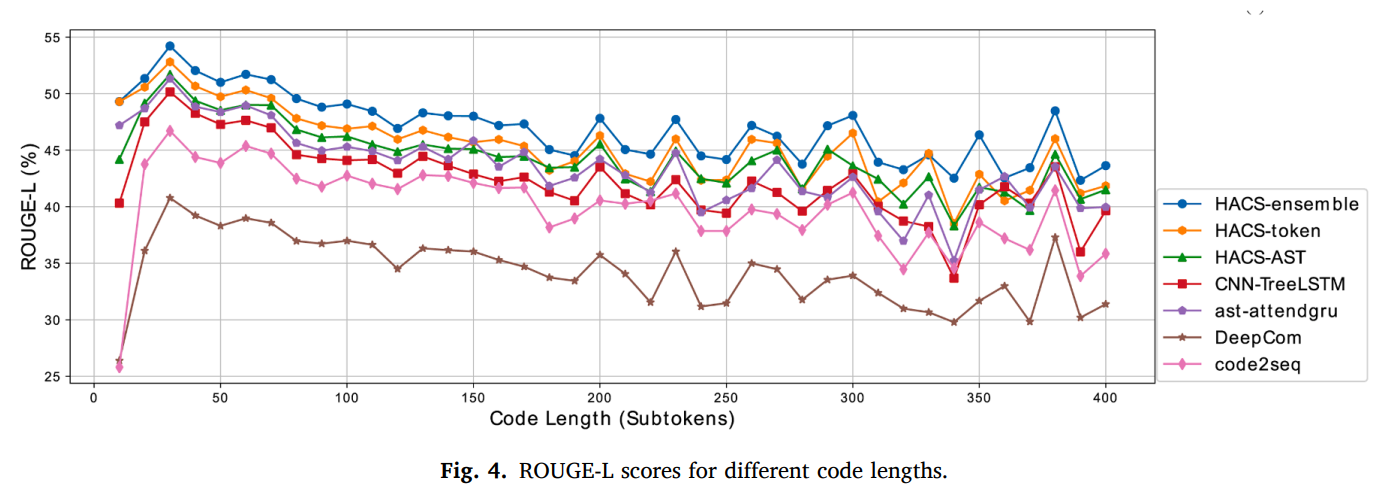
实验结果：

1、与基线相比，我们提出的方法表现如何？



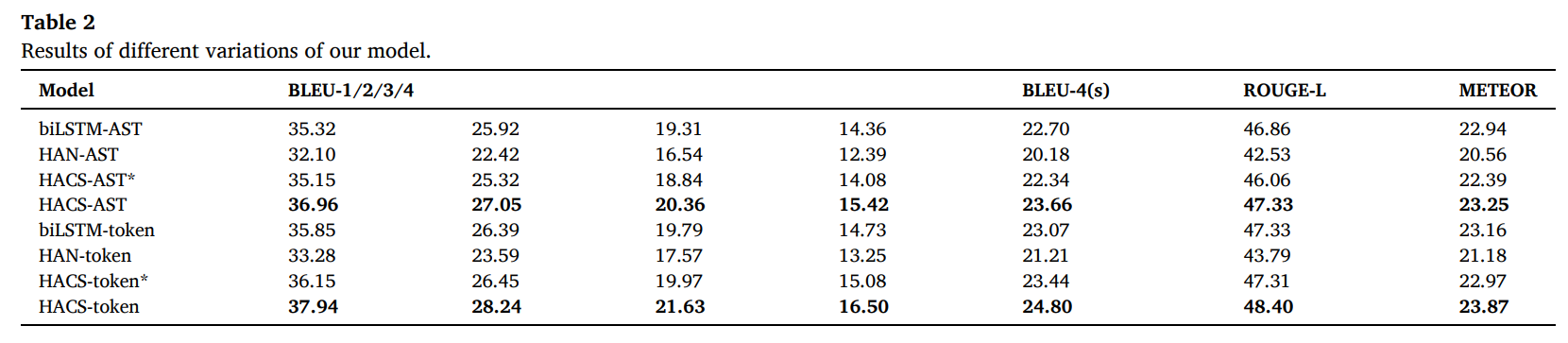
从表中我们可以看出，在大多数指标上，ast-attendgru 是基线中的竞争模型。这主要是因为它同时使用了来自token序列和 AST 的信息。处理token对于代码摘要至关重要，因为摘要中的许多关键字也可能出现在源代码中。很明显，多个输入的组合可以提高生成结果的质量，这就是我们引入 HACS-ensemble 的原因。 CNN-TreeLSTM 是具有巨大潜力的基线。虽然它只将 AST 作为输入，但与 ast-attendgru 相比，它获得了相似的 BLEU-1 和 BLEU-4 分数。这表明语法树中还包含丰富的词汇语义，因此找到对这两种信息进行编码的有效方法是合理的。我们的任何一个模型在所有指标上都优于所有基线，但有一定的差距。集成模型中的各个摘要器已经显示出它们的强大功能。这表明我们的方法可以更好地利用 AST 中的词汇和结构特征。本质上，分层输入明确地反映了token序列中的结构，并且我们的模型能够在解码过程中提取这些不同级别的特征。 HACS-ensemble 获得了最好的分数，表明我们的组合方法是有意义的。

2、我们的模型在多大程度上可以处理代码中的长期依赖关系？



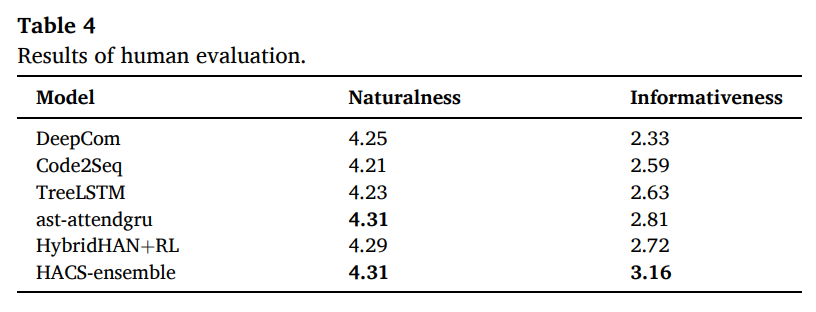
我们可以观察到，所有模型的峰值性能都在 30 到 70 个子token的代码长度内达到，并且对于较长的代码，得分有自然下降的趋势。特别是，基于 AST 的基线在代码长度为 10 时得分非常低。一个可能的原因是这种短代码仅包含少数 AST 节点或路径，因此这些模型难以从输入中获取有用的特征，而词汇信息可以在这种情况下更有帮助。HACS 将输入代码分解为短片段，减轻模型记忆长序列的负担，从而可以通过不同级别的编码器更好地建模代码中的远程依赖关系。

3、我们的模型的不同配置有什么影响，对代码表示的不同关注变体？

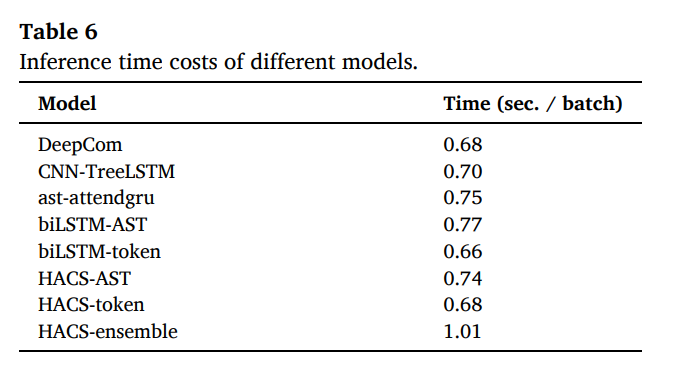


很明显，HANAST 和 HAN-token 表现出最差的分数，并且严重降低了标准 Seq2Seq 基线。原因是对于这些模型中的每一个，解码器只使用了一个编码向量，并且失去了其信息选择和token复制的能力。换句话说，HAN计算的注意力权重在整个生成过程中是固定的。HAN 最初是为文本分类而设计的，并不旨在指导文本生成任务，而代码摘要更多地依赖于输入中的细节和词汇资源，而不是分类。我们的模型在所有指标上始终优于其他模型，这证明了我们方法的有效性。这再次表明仅使用高级代码表示不足以使模型产生良好的摘要，而首选对token的引用。

4、人工评估

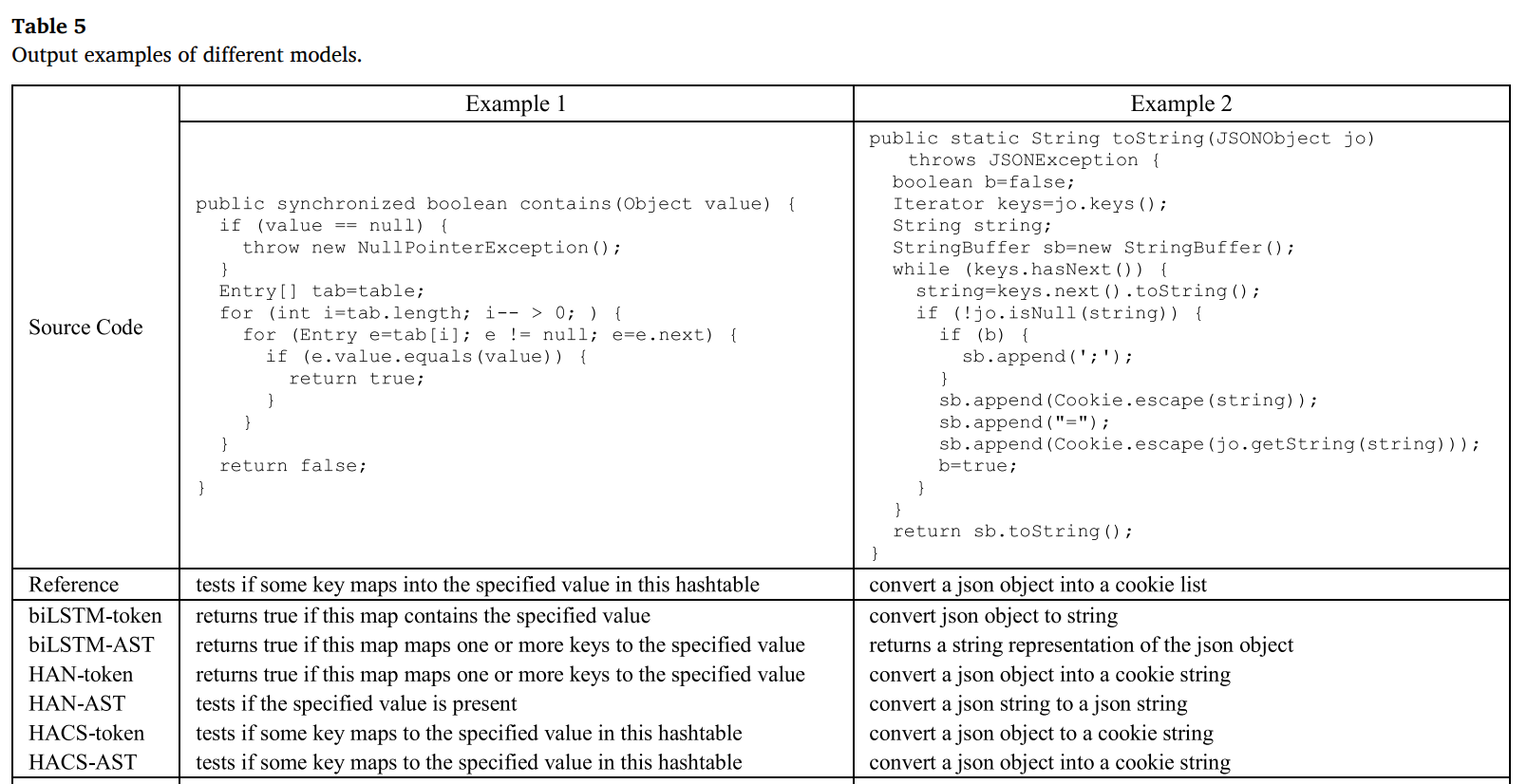


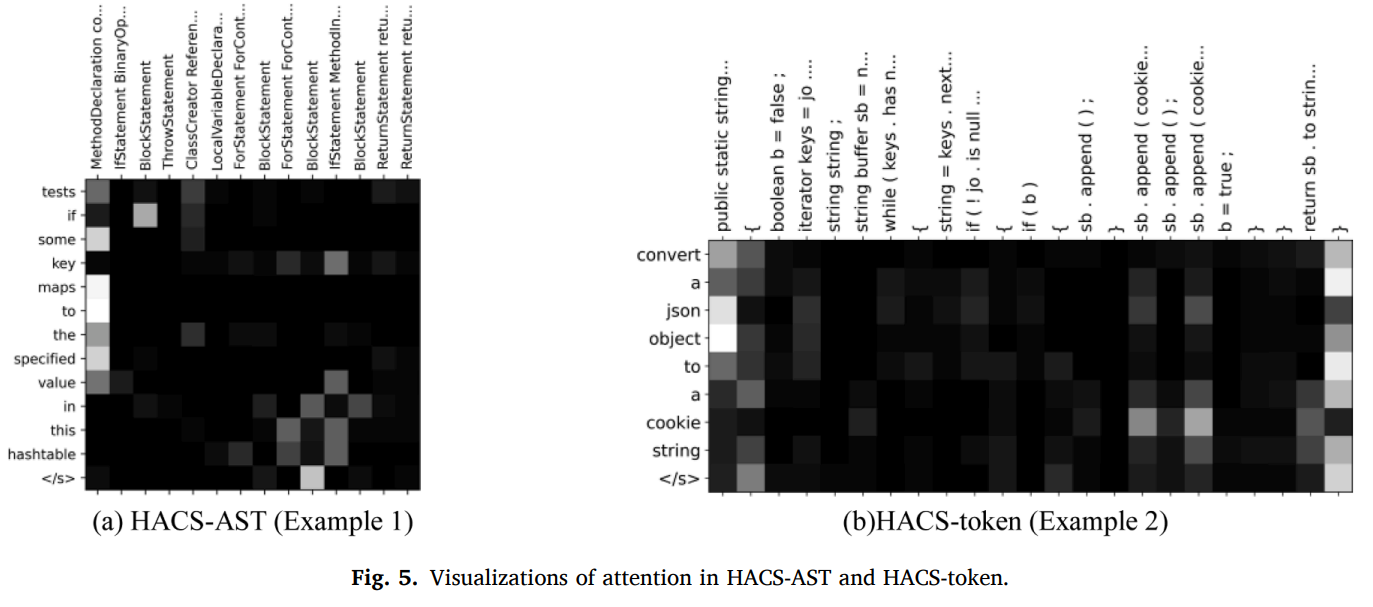
从表中可以看出，所有模型都能够生成流畅的摘要，并且在自然度上取得了接近的分数。这主要是由于它们相同的波束搜索设置。然而，他们的信息量得分差异是显而易见的。 HACS 在信息量方面超过了所有基线，这意味着它能够生成比其他人更相关和更准确的摘要。通过志愿者的反馈，我们发现当引用包含给定代码片段之外的信息时，模型总是无法生成此类内容，例如，变量或方法名称出现在类的上下文中。我们将尝试通过在未来的工作中包含更高级别的代码层次结构来消除此限制。

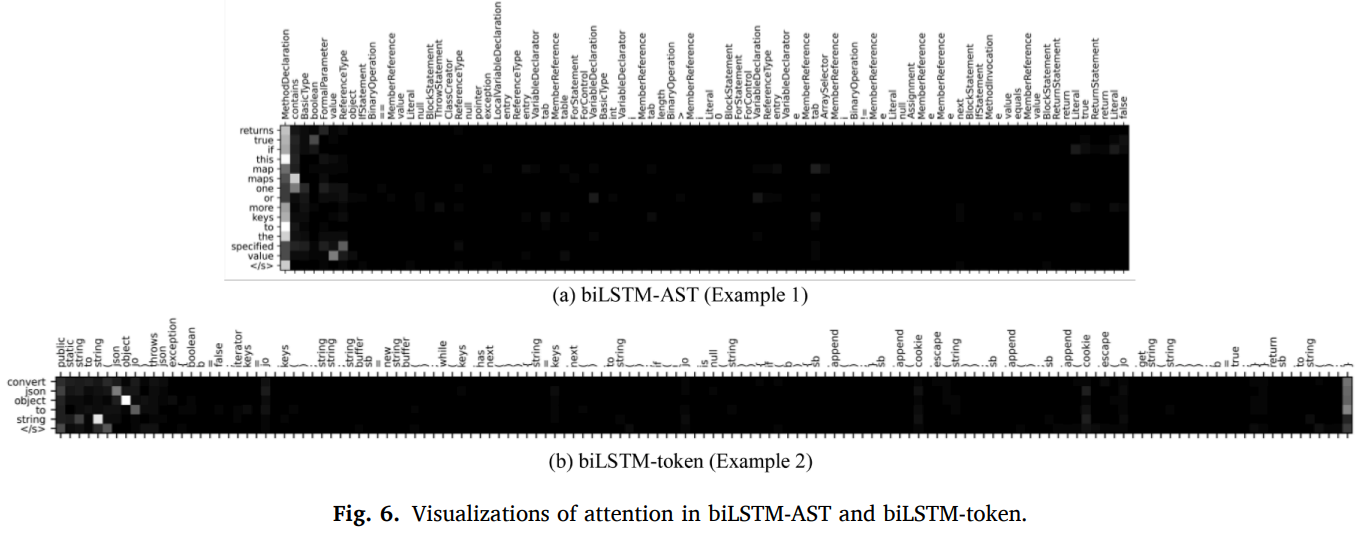


时间成本；我们从表 6 中可以看出，HACS-AST/token 与 biLSTM-AST/token 一样快。原因如下。首先，尽管 HACS 有两级编码器，但每一级都需要更短的序列：对于任何代码片段，token编码器的输入长度是语句中token（AST 节点）的数量，而语句编码器是代码中的语句数。此外，token编码器可以同时编码不同的语句。因此，HACS 的效率仍然很高。但是，我们可以看到 HACS-ensemble 比它的双输入对应物（即 ast-attendgru）慢得多。主要原因之一是 ast-attendgru 的 GRU 比 biLSTM 快得多。另一方面，由于我们的实验只使用了一个 GPU，我们需要依次计算 HACSAST 和 HACS-token 的输出以获得集成结果。HACSensemble 中个体的训练和推理是完全可并行的，因此如果将它们分配给多个 GPU，可以显着降低时间成本。总的来说，我们的方法是适用的。

实验案例: 模型的输出示例和可视化，以讨论在解码过程中关注分层代码表示的有效性。







如表 5 所示，HAN-AST 和 HAN-token 有时可以产生很好的总结，但它们的许多输出存在缺陷。这表明来自 HAN 的表示可以提供一些有用的特征，但最终模型不能像 HACS 那样充分利用它们。主要原因是 HAN 的顶层特征向量信息量较少，其配对解码器无法从输入中进行细粒度特征选择。

根据可视化，HACS 的优点之一是它拥有代码的全局视图，并且能够在生成过程中的不同语句之间转移注意力。由于基于token的基线主要关注本地信息并且缺乏高级知识，因此它们可能难以在长代码中定位关键字。另一方面，HACS 还保留了传统注意力的能力，即复制特定令牌，尽管该过程是间接的。由于 HACS 以分层方式定位重要令牌，因此结果可以更准确。