arXiv:2304.04812v1 [cs.PL] 10 Apr 2023

Scallop：一种神经符号编程语言

ZIYANG LI*， 宾夕法尼亚大学JIANI HUANG*， 宾夕法尼亚大学MAYUR NAIK， 宾夕法尼亚大学

我们介绍 Scallop，这是一种结合深度学习与逻辑推理优势的语言。Scallop 使用户能够编写广泛的神经符号应用，并以数据和计算高效的方式对其进行训练。它通过三个关键特性实现这些目标：1）基于关系数据模型的灵活符号表示；2）基于 Datalog的声明式逻辑编程语言，支持递归、聚合和否定；以及3）基于来源半环理论的自动且高效的可微推理框架。我们在文献中的八种神经符号应用组成的测试套件上对Scallop 进行了评估。我们的评估表明，Scallop 能够在多样且具有挑战性的人工智能任务中表达算法推理，为机器学习程序员提供简洁的接口以整合逻辑领域知识，并且在准确率方面产生可比或优于最先进的模型的解决方案。此外，Scallop 的解决方案在运行时间、数据效率、可解释性和泛化能力等方面优于这些模型。

1 引言

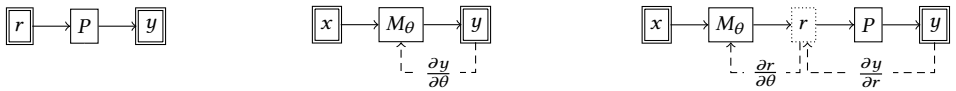
经典算法和深度学习体现了现代程序设计的两种主流范式。经典算法非常适合处理定义明确的任务，例如对数字列表进行排序，或在图中寻找最短路径。另一方面，深度学习非常适合处理使用经典算法无法切实完成或不可行的任务，例如检测图像中的对象或解析自然语言文本。这些任务通常使用一组输入-输出训练数据进行指定，而解决这些任务则涉及使用基于梯度的方法学习深度神经网络的参数，以拟合数据。

这两种范式在本质上互为补充。例如，逻辑程序等经典算法 P 如图 1a 所示，具有可解释性，但只能处理有限的（例如，结构化）输入r。相比之下，深度神经网络，例如 M_θ 图 1b 所示的网络，能够处理丰富的（例如，非结构化）输入 x，但不具备可解释性。现代应用需要兼具这两种范式的能力。示例包括问答 [Rajpurkar 等人， 2016], 代码补全 [Chen 等，2021b], 以及数学问题求解 [Lewkowycz 等人， 2022],等等。例如，代码补全需要借助深度学习从代码上下文中理解程序员的意图，并借助经典算法确保生成的代码正确。因此，一个自然且根本的问题是，如何通过整合这两种范式来编写此类应用。

神经符号编程是一种新兴范式，旨在实现这一目标 [Chaudhuriet al. 2021]。它旨在将符号知识和推理与神经网络架构相结合，相较于单独使用神经或符号方法，实现更高的效率、可解释性和泛化能力。考虑手写公式求值这一任务 [Li etal. 2020], ，该任务以图像形式的公式作为输入，并输出一个与对其求值结果相对应的数字。该任务的一个输入-输出示例是 $\langle x = , y = 1.6\rangle$。用于此任务的神经符号程序如图 1c所示的程序，可能会首先应用卷积神经网络 M_θ 处理输入图像，以获得一种结构化的中间形式 r，即由符号组成的序列 ['1', '+', '3', '/','5'], ，随后使用经典算法 P 解析该序列、计算解析后的公式，并输出最终结果 1.6。

*两位作者对本文的贡献相同

作者地址：Ziyang Li，宾夕法尼亚大学，liby99@seas.upenn.edu；Jiani Huang，宾夕法尼亚大学，jianih@seas.upenn.edu；Mayur Naik，宾夕法尼亚大学，mnaik@seas.upenn.edu。



(a) 逻辑程序。 (b) 神经模型。 (c) 基本的神经符号程序。

图 1. 不同范式的比较。逻辑程序 P 仅接受结构化输入 r，而带有参数的神经模型 M_θ θ可以处理非结构化输入 x。双框标出的数据表示提供了监督。在算法监督下，神经符号程序必须在 r上没有监督的情况下学习 θ。

尽管个别神经符号应用已取得显著进展 [Chen 等人，2020；Li 等人，2020；Mao 等人，2019；Minervini 等人，2020；Wang 等人，2019；Yi 等人， 2018],目前仍缺少一种具备编译器支持的语言，使神经符号范式的优势能够更广泛地惠及各方。我们着手开发这样一种语言，并确定了它要满足的五项关键标准，才能具备实用性。图1c中神经符号程序的各个组成部分标注了这些标准，具体如下：

(1) 一种用于 r 的符号数据表示，支持多种数据类型，例如图像、视频、自然语言文本、表格数据及其组合。(2) 一种用于 P 的符号推理语言，可表达递归、否定和聚合等常见推理模式。(3) 一种自动且高效的可微推理引擎，用于在算法监督下进行学习（$\frac{\partial y}{\partial r}$），即基于可观测输入-输出数据（$x$， y）提供监督，而不基于 r提供监督。(4) 能够根据各个应用的特点定制学习（$\frac{\partial y}{\partial r}$），因为逻辑程序的损失曲面并非连续，这妨碍了使用一种通用方法进行学习。(5) 一种利用现有训练流程（$\frac{\partial r}{\partial \theta}$）并与之集成的机制，以及神经网络架构和模型 M_θ 的实现与硬件（例如 GPU）优化。

本文介绍了 Scallop，这是一种满足上述标准的语言。Scallop 的核心理念在于选择了三个相互依赖的设计决策：用于符号数据表示的关系模型、用于符号推理的声明式语言，以及用于可微推理的溯源框架。我们将逐一详细阐述这些决策。

关系可以表示任意图，因此非常适合在 Scallop 应用中表示符号数据。例如，它们可以表示 场景图，一种图像的典型符号表示 [Johnson 等人 2015], 或抽象语法树，即自然语言文本的符号表示。此外，它们还可以组合成 关系数据库 以表示多模态数据。关系也天然适用于概率推理 [Dries 等人 2015], ，这是必要的，因为 r 神经模型生成的 M_θ 符号可能带有关联概率。

接下来，Scallop中的符号推理程序 P 采用声明式逻辑编程语言进行指定。该语言扩展了Datalog [Abiteboul等人 1995] ，其表达能力足以让程序员指定复杂的领域知识模式，而神经模型 M_θ 可能难以处理这些模式。Datalog 的实现可以利用关系数据库系统相关文献中的优化。这反过来又能够实现高效的推理与学习，因为当前任务的逻辑领域知识规范有助于减轻 M_θ的负担，而其职责现在变得不那么复杂，也更加模块化。最后，Datalog 基于规则，这使程序更易于编写、调试和验证。它还通过借鉴程序综合 [Gulwani等人 2017] 和 ILP [Cropper 等人 2022]的技术，促进了对这些程序的推断。

虽然符号推理具有上述诸多优势，但它给参数 θ 的学习带来了根本性挑战。深度学习依赖基于梯度的方法；这些方法之所以可行，是因为构成 M_θ 的底层激活函数具有可微性，从而能够获得 $\frac{\partial r}{\partial \theta}$。

关键挑战

于是，问题在于如何支持对高层逻辑程序进行自动且高效的微分P以获得 $\frac{\partial y}{\partial r}$，并将其与 $\frac{\partial r}{\partial \theta}$ 结合起来计算 $\frac{\partial y}{\partial \theta}$。Scallop 通过利用溯源半环 [Green 等人 2007]提出的框架来解决这一问题。该框架为一类应用提出了通用的代数结构：这些应用为元组定义注释（即标签），并将注释从关系代数（RA）查询的输入传播到输出。我们的主要贡献之一，是将该框架创新性地扩展到一个支持可微推理的 RA 扩展片段，该片段包含递归、否定和聚合。Scallop 实现了一个可扩展的溯源结构库，其中包括扩展的最大值—最小值半环和 top-k证明半环 [Huang 等人 2021]提出的半环。我们进一步证明，不同的溯源结构能够为梯度计算提供不同的启发式方法，从而有效地根据各个应用的特性定制学习过程。

我们使用 4.5 万行 Rust 代码，为 Scallop 实现了一套全面的开源工具链。它包括编译器、解释器和 PyTorch 绑定，可将 Scallop 程序集成到现有的机器学习流程中。我们通过一组涵盖图像与视频处理、自然语言处理、规划和知识图查询领域的八种神经符号应用，在监督学习、强化学习、规则学习和对比学习等多种学习设置下评估 Scallop。评估结果表明，Scallop 表达能力强，所产生的解决方案准确率与最先进的模型相当，且往往更高。我们还展示了 Scallop 解决方案在运行时间、数据效率、可解释性和泛化能力方面的额外优势。

如果一篇关于编程语言的论述不提及该语言的发展脉络，那就不够完整。TensorLog [Cohen 等人 2017] 和 DeepProbLog（DPL） [Manhaeve 等人 2021]开创了为概率逻辑编程语言（例如 ProbLog [Dries 等人 2015]）引入可微推理的思路。Scallop 最初由 [Huang 等人 2021] 提出，旨在通过使用 Datalog 取代 Prolog，并放宽其精确概率语义，提升 DPL 的可扩展性。我们在 [Huang 等人 2021] 的工作基础上，拓展其表达能力，形式化其语义，开发可定制的溯源框架，并提供实用的工具链。

本文其余部分的组织如下。第 2 节介绍 Scallop。第 3 节描述 Scallop 的符号推理语言。第4 节介绍可微推理框架。第5 节描述我们对 Scallop 的实现。第 6 节在基准测试套件上对 Scallop 进行实证评估。第 7 节调研相关工作，第 8节总结全文。

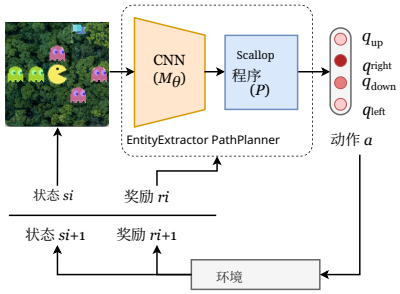
2 示例性概述

我们使用Scallop介绍一个基于强化学习（RL）的规划应用，我们称之为吃豆人迷宫。该应用如图 2a所示，涉及一个智能代理在简化版吃豆人迷宫游戏中执行一系列动作。迷宫是一个隐式 5×5网格，由多个单元格组成。每个单元格要么为空，要么包含一个实体，该实体可以是 角色（吃豆人）、目标 （旗帜），或 敌人 （幽灵）。在每一步中，代理将角色向四个方向之一移动：上、下、右或左侧。当角色到达目标或撞上敌人时，游戏结束。迷宫以原始图像的形式提供给代理，并在每一步更新，因此代理需要处理感官输入、提取相关特征，并从逻辑上规划要采取的路径。此外，游戏的每一局都会随机化角色、目标和敌人的初始位置。

具体而言，如图2b所示，游戏被建模为智能体与环境之间一系列交互。步骤 i 时的游戏状态 $s_i \in S$ 是一张 200×200 彩色图像（$S = \mathbb{R}^{200\times200\times3}$）。智能体向环境提出一个动作 $a_i \in A = \{$上，下，右，左侧$\}$，环境据此生成一张新图像 s_{i+1} 作为下一个状态。环境还会向智能体返回奖励 r_i



(a) 一个游戏会话的三种状态。



(b) 使用 Scallop 的应用程序架构。

```
1  类      PacManAgent ( torch .nn. Module ):
2   def  __初始化__(自身,  网格_维度,  cell_大小 ):
3    # 初始化 ...
4    自身. 提取      _实体   =
5     EntityExtractor ( 网格_维度,      单元格_大小 )
6    自身. 路径_规划器   = ScallopModule (
7     文件 =" 路径_planner .scl"     ,
8     溯源 ="diff-top-k-proofs"          ,  k=1 ,
9     输入_映射 ={" 角色 ":          单元格,
10     " 目标 ":  单元格,  " 敌人 ":    单元格 },
11     输出_映射 ={" 下一步_动作 ":          动作 })
12
13   def  前向 (自身 ,      游戏_状态_图像 ):
14    角色 ,   目标 ,  敌人     =
15     自身。提取_实体 (游戏_状态_图像)
16    下一个_动作     =  自身。路径_规划器 (
17     角色 =角色,      目标 =目标,    敌人 =敌人 )
18    返回 下一个     _动作
```

(c) Python 实现片段。

图 2. 使用 Scallop 实现的规划应用吃豆人迷宫示意图。

：达到目标时为 1，否则为 0。该过程会重复进行，直到游戏结束或达到预设的步数上限。

一种实现我们应用的流行强化学习方法是 Q-学习。其目标是学习一个函数Q： $S \times A \to \mathbb{R}$ 该函数返回在状态 a_i 下执行动作 s_i所获得的预期奖励。[1] 由于游戏状态是图像，我们采用深度 Q-学习 [Mnih 等人 2015], 提出的方法，该方法使用带有已学习参数 Q 的卷积神经网络（CNN）模型来近似该函数。我们应用端到端的深度学习方法，训练模型以预测给定游戏状态下每个动作的 Q-值。该方法通过 50K 个训练回合达到 84.9% 的测试成功率，其中一个回合指从开始到结束的一次完整游戏会话。

相比之下，使用 Scallop 的神经符号解决方案只需 50 个训练回合，测试成功率即可达到 99.4%。Scallop 将智能体的任务分解为独立的神经组件和符号组件，从而实现神经符号范式的这些优势，如图 2b 所示。这些组件各自执行最适合其范式的子任务：神经组件在每一步感知图像中各个单元格的像素，以识别其中的实体；符号组件则推理从角色到目标的无敌人路径，从而确定最优的下一步动作。图 2c 展示了使用流行的 PyTorch 框架实现这一架构的概览。

具体而言，神经组件仍然是卷积神经网络，但现在它每次接收输入图像中单个单元格的像素，并对其中的实体进行分类。神经组件（`EntityExtractor`）的实现是标准的，为简洁起见省略。它在第 14-15 行调用，输入为 游戏_状态_图像，即 $\mathbb{R}^{200\times200\times3}$ 中的一个张量，并返回三个 $\mathbb{R}^{5\times5}$ 张量的实体。例如，角色 是一个 $\mathbb{R}^{5\times5}$ 张量，而角色ij 是角色位于单元格（i, j）中的概率。随后，实体的表示被传递给第16-17行的符号组件，由其推导出每个动作的 Q值。符号组件在第 6-11 行进行配置，

[1]我们省略 Q- 学习算法，因为为了说明本例中神经符号编程的相关方面，并不需要它。

```
1 // 文件路径_planner .scl 2 typeactor (x: i32 , y: i32),goal(x: i32 ,y:i32), enemy (x: i32 , y:i32) 3
4 const UP = 0,DOWN = 1,RIGHT = 2, LEFT = 35 relsafe_cell (x,y) = range (0, 5, x), range (
0, 5, y), notenemy (x, y)6 releadge(x, y,x, yp , UP) = safe_cell (x,y),safe_cell (x, yp), yp == y + 17 //
DOWN、RIGHT 和左侧边的规则被省略..... 8 9rel next_pos (p,q,a) = actor (x, y), edge(x,y,p,q,a)
10rel path(x, y, x, y) = next_pos (x, y, _) 11 rel path(x1 , y1 , x3 , y3) = path(x1 ,y1 ,x2, y2),
edge(x2 , y2 ,x3 , y3 , _)12 rel next_action (a) = next_pos (p,q,a), path(p, q, r, s), goal(r, s)
```

图 3。Scallop 中吃豆人迷宫应用的逻辑程序。

由图中所示的 Scallop 程序构成。3接下来，我们将结合该程序，阐述第1 节概述的 Scallop 的三个关键设计决策。

关系模型。 在 Scallop 中，表示符号的主要数据结构是关系。在我们的示例中，游戏状态可以通过 5×5 网格离散网格单元中出现的实体类型来进行符号描述。因此，我们可以使用表示三种实体类型的二元关系来表示符号组件的输入：角色、目标和敌人。例如，事实`actor(2,3)`表示角色位于单元格 (2,3) 中。同样，由于有四种可能的动作，符号组件的输出由一元关系`next_`动作表示。

神经网络从非结构化输入中提取的符号具有相关概率，例如我们的示例中神经组件生成的 $\mathbb{R}^{5\times5}$ 张量角色（图 2c 第14行）。因此，Scallop 允许将元组与概率相关联，例如 0。96 :: 角色 (2,3)，以表示该角色以 0.96 的概率位于单元格(2,3) 中。更一般地说，Scallop 通过输入输出映射（图2c中的第9-11 行），支持将神经组件中的张量转换为符号组件中的关系，或进行反向转换，从而使两个组件能够无缝交换信息。

声明式语言。 神经符号语言的另一个关键考量是，应该为符号推理提供哪些构造。Scallop 使用一种基于 Datalog 的声明式语言，我们将在第 3 节介绍，并在此通过图 3中的程序加以说明。该程序使用一组逻辑规则实现了示例中的符号组件。遵循 Datalog 语法，这些规则是“如果—那么”规则，从右向左阅读，逗号表示合取。

回想一下，我们希望确定一个动作 a（上、下、右或左），将其指向与角色所在单元格（x，y）相邻的单元格（p，q），并且从（p，q）到目标所在单元格（r，s）之间存在一条无敌人路径。图中展示的九条规则通过逐步构建复杂度不断提高的关系，简洁地计算出这一复杂推理模式；最后一条规则（第 14 行）计算所有此类动作。[2]

最复杂的概念可能是路径关系，它在第 10–11 行通过递归定义。递归使模式能够以简洁的方式定义，使训练后的应用能够泛化到任意大于 5×5 的网格（而纯神经网络版本则不能），并让该模式更适合根据输入—输出示例进行合成。除了递归，Scallop 还支持否定和聚合；这些特性结合起来，使该语言足以在实践中描述常见的高层推理模式。

可微推理。 定义了神经组件和符号组件之后，最后一个主要问题是如何仅通过端到端监督来训练神经组件。

[2]我们省略了展示一个辅助关系，该关系将所有网格单元标记为概率 0.99，作为迈出一步的惩罚。因此，较长的路径会受到更大的惩罚，从而促使符号程序优先选择向目标靠近。

在我们的示例中，每个游戏会话会获得 1或 0的奖励形式的监督，具体取决于智能体的动作序列是否成功地引导角色到达目标且没有撞到任何敌人。这种形式的监督称为算法监督或弱监督，免除了在神经组件和符号组件的接口处标注中间关系的需要，例如 角色、目标 和 敌人关系。然而，这也使得学习这些关系的张量的梯度变得具有挑战性，而这些梯度又是使用梯度下降技术训练神经组件所必需的。

Scallop 的关键洞见是利用逻辑程序的结构来指导梯度计算。此类计算的最佳启发式方法取决于若干应用特征，例如可用数据量、推理模式和学习设置。Scallop 提供了一个便捷接口，供用户从内置启发式方法库中进行选择。此外，由于所有这些启发式方法都遵循逻辑程序的结构，Scallop 将它们统一实现为通用且可扩展的 溯源框架的实例，详见第 4 节。对于我们的示例，图 2c 中的第 8 行指定使用 `diff-top-k-proofs` ，并将 `k=1` 设为启发式方法。这是 Scallop 中的默认设置，适用于许多应用，正如我们在第6节所展示的那样。

3 语言

我们概述 Scallop 的符号推理语言，该语言此前已通过图中所示的程序加以说明。3。附录 A 给出了该语言的形式语法。下面，我们通过推断亲属关系的示例来说明每种关键构造。

3.1 数据类型

Scallop 中的基本数据类型是集合值关系，由静态类型的原始值组成的元组构成。原始数据类型包括不同尺寸的有符号整数和无符号整数（例如`i32`、`usize`）、单精度和双精度浮点数（`f32`、`f64`）、布尔值（`bool`）、字符（`char`）和字符串（`String`）。以下示例声明了两个二元关系，母亲 和父亲：

```
类型 母亲 (c: 字符串 , m: 字符串 ), 父亲(c: 字符串 , f: 字符串)
```

值 关系可以通过单个元组或由常量字面值组成的元组集合来指定 ：

```
关系 母亲 ("Bob", "Christine")// Christine 是 Bob 's 母亲关系 父亲 ={("Alice", "Bob"), ("John", "Bob")}// Bob 是两个孩子的父亲
```

作为简写，原始值可以命名并用作常量变量：

```
常量 父亲 = 0, 母亲 = 1, 祖母 = 2, ..... // 其他关系关系 复合关系(父亲，母亲，祖母) // 父亲 '的母亲是祖母
```

类型声明是可选的，因为 Scallop 支持类型推断。复合关系关系的类型会被推断为（usize，usize，usize），因为无符号整数的默认类型是`usize`。

3.2（Horn）规则

由于 Scallop 的语言基于 Datalog，因此支持类似“如果-那么”的霍恩子句。每条规则由一个头原子和一个体组成，两者由符号 :- 或 =连接。以下代码展示了定义<给祖母关系的两条规则。合取通过规则体中由“,”分隔的原子指定，而析取则通过具有相同头谓词的多条规则指定。头中出现的每个变量也必须出现在体中的某个正原子中（我们将在下文介绍负原子）。

```
rel 祖母 (a, c) :- 父亲 (a, b), 母亲(b, c) // 父亲 's 母亲rel 祖母 (a, c) :- 母亲 (a, b), 母亲(b, c) // 母亲 's 母亲
```

合取和析取也可以使用诸如and、or和implies这样的逻辑连接词来表示。例如，下面的规则等价于将前面两条规则合并。

```
rel grandmother(a, c) = (mother(a, b) or father(a, b)) and mother(b, c)
```

Scallop 支持通过外部函数（FF）创建值。外部函数具有多态性，包括算术运算符，例如 + and -；比较运算符，例如 != and >=；类型转换，例如 [i32] as 字符串；以及内置函数，例如 $hash and $string_concat。它们只对原始值进行操作，而不能处理关系元组或原子。下面的示例展示了如何使用外部函数将字符串连接起来，得到结果 full_name("Alice Lee")。

```
关系 first_name("Alice"), last_name("Lee")关系 full_name($string_concat(x,"␣",y)) =
first_name(x), last_name(y)
```

请注意，FF 可能会因运行时错误而失败，例如除以零和整数溢出；在这种情况下，该单条事实的计算会被省略。在下面的示例中，将 6 除以分母时，对于分母0 不会计算结果，因为这会导致外部函数执行失败：

```
关系 分母 ={0,1, 2}关系 结果（6 / x） =分母(x) // 结果仅包含整数 3 和 6
```

此语义的目的是支持概率扩展（第 3.3），而不是默默抑制运行时错误。在处理浮点数时，包含 非数（非数）的元组也会被丢弃。

递归。一个关系 r 依赖于 s，当原子 s 出现在带有头原子 r的规则体中时。一个递归关系是指直接或传递地依赖于自身的关系。以下规则通过使用复合关系复合现有的亲属关系事实，推导出额外的亲属关系事实。

```
rel kinship(r3,a,c) = kinship(r1,a,b), kinship(r2,b,c), composition(r1,r2,r3)
```

否定。 Scallop 支持通过在规则体中的原子上使用not 运算符实现分层否定。以下示例展示了一条规则，将has_no_子女 关系定义为既不是父亲也不是母亲的人p 。注意，我们需要用正原子人 限定p ，才能使规则格式正确。

```
rel person = {"Alice", "Bob", "Christine"} // can omit () since arity is 1
rel has_no_children(p) = person(p) and not father(_, p) and not mother(_, p)
```

一个关系 r 以否定方式依赖于 s，如果一个被否定的原子 s 出现在某条规则的体中，且该规则的头原子为 r。在上面的示例中，has_no_子女 以否定方式依赖于 父亲。一个关系不能直接或传递地以否定方式依赖于自身，因为Scallop仅支持分层否定。编译器会拒绝以下规则，因为其中的否定不是分层的：

```
关系 某物_是_真 () =非 某物_是_真 () // 编译错误！
```

聚合。 Scallop还支持分层聚合。内置聚合器包括常见的聚合器，例如计数、求和、最大值，以及一阶量词全称量词 和存在量词。除运算符外，聚合还指定绑定变量、用于约束这些变量的聚合体，以及用于赋值结果的结果变量。以下示例中的聚合表示：“变量n被赋值为p的计数，其中p 是一个人”：

```
rel num_people(n) = n := count(p: person(p))
```

在规则中，p 是绑定变量，n 是结果变量。根据聚合器的不同，可能存在多个绑定变量或多个结果变量。此外，Scallop 支持在聚合中使用 where 子句来实现 SQL 风格分组。在下面的示例中，我们计算每个人 p的子女数量，其中 p充当分组变量。

(标签) $t \in T$(假) $\mathbf{0} \in T$(真)$\mathbf{1} \in T$(析取)$\oplus : T \times T \to T$(合取)$\otimes : T \times T \to T$(否定)$\ominus : T \to T$(饱和)$\Subset : T \times T \to$ 布尔

图 4. 溯源的代数接口。

（谓词） p（聚合器）$g ::=$ 计数 | 求和 | 最大值 | 存在量词 | ...（表达式）$e ::= p \mid \gamma_g$（e）（$\mid \pi_\alpha$）（e）（$\mid \sigma_\beta$）（e）$\mid e_1 \cup e_2 \mid e_1 \times e_2 \mid e_1 - e_2$（规则）$r ::= p \leftarrow e$（层）$s ::= \{r_1, \ldots, r_n\}$（程序）$s ::= s_1; \ldots; s_n$

图 5。SclRam 核心片段的抽象语法。

```
关系 父母 (a, b) =父亲 (a, b) 或 母亲(a, b)关系 数字_子女 (p,n) =n: =计数(c: 父母 (c,
p) 其中 p: 人 (p))
```

最后，诸如 全称量词 和存在量词 之类的量词聚合器会返回一个布尔值变量。例如，在下面的聚合中，变量sat 被赋予以下陈述的真值（真或假）：“对于所有的 a 和b，如果 b 是a的父亲，那么a 是b的儿子或女儿”。

```
关系 完整性_约束（sat） =sat: =全称量词(a, b: 父亲(a, b) 蕴含(儿子(b, a) 或 女儿 (b, a)))
```

聚合有几项语法检查。首先，与否定类似，聚合也必须分层——关系不能通过聚合依赖于自身。其次，绑定变量必须由聚合体中的一个正原子限定。

3.3 概率扩展

尽管 Scallop 主要是为神经符号编程而设计的，但其语法也支持概率编程。这在将Scallop代码与神经网络集成之前调试 Scallop代码时尤其有用。设想一位机器学习程序员希望从自然语言句子“Bob带他的女儿Alice 去海滩”中提取结构化关系。该程序员可以仿照神经网络，生成 Alice（A）和Bob（B）之间亲属关系的概率分布，如下所示：

```
rel kinship = {0.95::(FATHER, A, B); 0.01::(MOTHER, A, B); ... }
```

这里，我们列出了 Alice 和 Bob 之间所有可能的亲属关系。对于每一种关系，我们使用语法 [概率]::[元组] 为带有概率的亲属元组添加标签。用于分隔它们的分号“;”表示这些关系互斥——Bob 不可能同时是 Alice 的母亲和父亲。

Scallop 还支持从概率分布中进行采样的运算符。它们与聚合使用相同的表层语法，因此可以按分组变量进行采样。以下规则会确定性地选出给定两人 a 和 b之间概率最高的亲属关系；在此聚合中，二者是隐式分组变量。最终，根据上述概率，只会推导出一个事实：0.95::top_1_kinship(FATHER, A, B)。

```
rel top_1_kinship(r, a, b) = r := top<1>(rp: kinship(rp, a, b))
```

还支持其他类型的采样，包括类别采样（categorical<K>）和均匀采样（uniform<K>），其中静态常量 K 表示试验次数。

最后，也可以为规则添加表示置信度的概率标签。以下规则表示，祖母的女儿是某人的母亲，其置信度为 90%。

```
关系 0.9:: 母亲(a, c) =祖母 (a, b) 且 女儿 (b, c)
```

概率规则是语法糖。它们通过在规则的体中引入一个辅助的零元（即布尔值）事实来实现，该事实以带标签的概率被视为真。

$$
\begin{array}{rlcll}
\text{(常量)} & \mathbb{C} & \ni & c & ::= int \mid bool \mid str \mid \ldots \\
\text{（元组）} & \mathbb{U} & \ni & u & ::= c \mid (u_1, \ldots, u_n) \\
\text{（带标签元组）} & \mathbb{U}_T & \ni & u_t & ::= t :: u \\
\text{（事实）} & \mathbb{F} & \ni & f & ::= p(u) \\
\text{（带标签事实）} & \mathbb{F}_T & \ni & f_t & ::= t :: p(u)
\end{array}
\qquad
\begin{array}{rlcll}
\text{（元组）} & U & \in & \mathcal{U} & \triangleq \mathrm{P}(\mathbb{U}) \\
\text{（带标签元组）} & U_T & \in & \mathcal{U}_T & \triangleq \mathrm{P}(\mathbb{U}_T) \\
\text{（事实）} & F & \in & \mathcal{F} & \triangleq \mathrm{P}(\mathbb{F}) \\
\text{（数据库）} & F_T & \in & \mathcal{F}_T & \triangleq \mathrm{P}(\mathbb{F}_T)
\end{array}
$$

图 6. SclRam 的语义域。

4 推理框架

上一节介绍了 Scallop的表层语言，供程序员用于表达离散推理。然而，该语言还必须支持可微推理，以实现端到端训练。在本节中，我们通过溯源框架正式定义该语言的语义。我们将展示 Scallop如何仅通过定义不同的溯源，统一支持不同的推理模式——离散、概率和可微推理。

我们首先介绍溯源框架的基础知识（第 4.1 节）。然后介绍低层表示 SclRam、其操作语义，以及它与 Scallop 应用其余部分的接口（第 4.2-4.4 节）。接下来介绍微分，以及用于可微推理的三种不同溯源方式（第 4.5 节）。最后讨论实际应用中的考量（第 4.6 节）。

4.1 溯源框架

Scallop 的溯源框架支持在逻辑程序执行过程中，为关系元组附加并传播额外信息。该框架基于溯源半环 [Green 等人 2007]的理论。图4 定义了 Scallop 的溯源代数接口。我们将这些额外信息称为来自标签空间 T的标签 t。存在两个特殊标签，$\mathbf{0}$和$\mathbf{1}$，分别表示无条件假和真 标签。标签通过二元析取 $\oplus$、二元合取 $\otimes$和一元否定 $\ominus$ 操作传播，对应逻辑中的或、与和非。最后，饱和 检查$\bigcirc=$ 可作为不动点迭代的可定制停止机制。

上述所有组件组合形成一个 7-元组（T，$\mathbf{0}$，$\mathbf{1}$，$\oplus$，$\otimes$，$\ominus$，$\bigcirc=$），我们称其为 溯源 T。Scallop 提供了一个溯源内置库，用户只需实现此接口即可添加自定义溯源。

示例*4.1*。 最大值-最小值-概率 (mmp) $\triangleq$ ($[0,1], 0, 1,$最大值， 最小值 $\lambda x.(1-x)$ ==)，是一个内置的概率溯源，其中标签是介于 0和 1 之间的数字，通过最大值和最小值进行传播。这些标签不代表真概率，而只是一种近似值。我们将在本节稍后讨论更丰富的溯源，以进行更准确的概率计算。

溯源必须满足若干性质。首先，5 元组 $(T, \mathbf{0}, \mathbf{1}, \oplus, \otimes)$ 应构成一个半环。也就是说，$\mathbf{0}$ 是加法单位元，且在乘法下具有湮灭作用；$\mathbf{1}$ 是乘法单位元，$\oplus$ 和 $\otimes$ 满足结合律和交换律，并且 $\otimes$ 对 $\oplus$ 满足分配律。为保证不动点的存在（将在第 4.3 节讨论），它还必须是吸收的，即 $t_1 \oplus (t_1 \otimes t_2) = t_1$[Dannert 等人 2021]。此外，我们还需要 $\ominus \mathbf{0} = \mathbf{1}$、$\ominus \mathbf{1} = \mathbf{0}$、$\mathbf{0} \bigcirc\neq \mathbf{1}$、$\mathbf{0} \bigcirc= \mathbf{0}$ 和 $\mathbf{1} \bigcirc= \mathbf{1}$。即使某种溯源违反了某一项性质（例如吸收性），对于不使用受影响特性（例如递归）的应用，它仍然有用；用户也可以只是在希望绕过这些限制时使用它。

4.2 SclRam 中间语言

Scallop 程序会被编译为一种称为SclRam的低层表示。图 5 展示了 SclRam核心片段的抽象语法。表达式类似于扩展关系代数中的查询。它们作用于关系谓词（p），使用用于聚合（γ_g，搭配聚合器 g）、投影（π_α，搭配映射 α）和选择（σ_β，搭配条件 β）的一元操作，以及并集（$\cup$）、乘积（$\times$）和差集（$-$）等二元操作。

SclRam r 中的规则表示为 $p \leftarrow e$，其中谓词 p 是规则头，表达式 e 是规则体。无序的规则集合共同构成层 s，层的序列 s_1；…；s_n构成一个 SclRam程序。同一层中的规则具有不同的头谓词。将层 s 中的头谓词集合记为 P_s，我们还要求程序中的 $P_{s_i} \cap P_{s_j} = \emptyset$ 全称量词 $i \neq j$。表层语言中的分层否定和聚合在 SclRam 中作为语法限制强制执行：在层 s_i的规则中，如果关系谓词 p 被用于聚合（γ）之下或差集（–）的右侧，那么该谓词 p 不能出现在 P_{s_j} 中，如果 $j \geq i$。

接下来，我们定义图 6 中的语义域，后续将用它们来定义 SclRam 的语义。元组 u 可以是常量，也可以是元组序列。事实 $p(u) \in \mathbb{F}$ 是一个以关系谓词 p 命名的元组 u。元组和事实都可以带标签，从而形成带标签元组 $(t :: u)$ 和带标签事实 $(t :: p(u))$。给定一组带标签元组 U_T，如果存在一个 t，使得 $t :: u \in U_T$，我们就称 $U_T \vDash u$。一组带标签事实构成一个数据库 F_T。我们使用括号表示法 $F_T[p]$，表示 F_T 中谓词 p 下的带标签事实集合。

4.3 SclRam 的操作语义

现在，我们在图 7 中介绍 SclRam 核心片段的操作语义。SclRam 程序 s 接收一个 外延数据库 (EDB) F_T 作为输入，并返回一个 内涵数据库 (IDB)$F'_T = [\![s]\!](F_T)$。该语义以底层溯源 T 为条件。我们将其称为 带标签语义，以区别于传统 Datalog 中的 无标签语义 。

基本关系代数。 评估 SclRam 中的表达式会根据图 7顶部定义的规则，生成一组带标签元组。谓词 p 的评估结果是数据库中该谓词下的所有事实。选择会筛选出满足条件的元组 β，而投影则会根据映射 α转换元组。映射函数 α 是部分函数：由于它可能对值应用外部函数，因此可能执行失败。并集 $e_1 \cup e_2$ 中的元组可以来自 e_1 或 e_2。在（笛卡尔）乘积中，每对输入元组会组合在一起，我们使用 $\otimes$ 来计算标签合取。

差与否定。要评估差表达式 $e_1 - e_2$，需根据从 e_1 评估得到的元组 u 是否出现在 e_2的结果中，分为两种情况。如果没有出现，我们只需将该元组及其标签传播到结果中（Diff-1）；否则，我们会得到来自 e_1 的 $t_1 :: u$，以及来自 e_2的 $t_2 :: u$。与无标签语义中从结果中删除元组 u 不同，我们会将标签 $t_1 \otimes (\ominus t_2)$ 与u 一同传播（Diff-2）。这样，在否定过程中信息不会丢失。图 8 比较了不同语义下一个差表达式示例的评估结果。在无标签语义下，元组 (2,3) 会从结果中移除；而在带标签语义下，它仍会保留。

Aggregation. 聚合。聚合器在 SclRam 中是作用于元组 g 集合（未带标签）的离散函数$U \in$ U。它们返回一个聚合元组集合，以适应`argmax` 等可能产生多个结果的聚合器。例如，我们有计数$(U) = \{|U|\}$。然而，在概率域中，离散符号并不足够。给定 n 要进行聚合的带标签元组，每个带标签元组都可以被开启或关闭，从而产生 2^n 个不同的可能世界。每个可能世界都是输入集合 $U_T(|U_T| = n)$的一个划分。将正部分表示为 X_T、负部分表示为 $X_T = U_T - X_T$，则与该可能世界关联的标签是 X_T中的标签与 X_T中的取反标签的合取。在此可能世界上进行聚合，涉及对正部分 X_T中的元组应用聚合器 g。如果枚举所有可能世界，这一过程本质上是指数级的。然而，我们可以针对每个聚合器和每个溯源进行优化，以实现更好的性能。例如，对带最大-最小-概率标签的元组进行计数，可以通过 $O(n \log(n))$算法实现，其速度远快于指数级算法。图9 展示了一个运行示例，以及在最大-最小-概率溯源下对计数表达式的评估。所得计数可以是 0–9，其中每个计数都可以由多个可能世界推导得到。

规则与不动点迭代。 在数据库 F_T 上求值规则 $p \leftarrow e$，涉及对表达式 e 求值，并将结果与 F_T中谓词 p 下已有的事实合并。对 e 求值的结果可能包含带有不同标签的重复元组，这是由于诸如以下的表达式造成的。

Expression semantics

$$\alpha : \mathbb{U} \to \mathbb{U}, \quad \beta : \mathbb{U} \to \text{Bool}, \quad g : \mathcal{U} \to \mathcal{U}, \quad [\![e]\!] : \mathcal{F}_T \to \mathcal{U}_T$$

$$\frac{t :: p(u) \in F_T}{t :: u \in [\![p]\!](F_T)} \text{(Predicate)} \qquad \frac{t :: u \in [\![e]\!](F_T) \qquad \beta(u) = \text{true}}{t :: u \in [\![\sigma_\beta(e)]\!](F_T)} \text{(Select)} \qquad \frac{t :: u \in [\![e]\!](F_T) \qquad u' = \alpha(u)}{t :: u' \in [\![\pi_\alpha(e)]\!](F_T)} \text{(Project)}$$

$$\frac{t :: u \in [\![e_1]\!](F_T) \cup [\![e_2]\!](F_T)}{t :: u \in [\![e_1 \cup e_2]\!](F_T)} \text{(Union)} \qquad \frac{t_1 :: u_1 \in [\![e_1]\!](F_T) \qquad t_2 :: u_2 \in [\![e_2]\!](F_T)}{(t_1 \otimes t_2) :: (u_1, u_2) \in [\![e_1 \times e_2]\!](F_T)} \text{(Product)}$$

$$\frac{t :: u \in [\![e_1]\!](F_T) \qquad [\![e_2]\!](F_T) \not\ni u}{t :: u \in [\![e_1 - e_2]\!](F_T)} \text{(Diff-1)} \qquad \frac{t_1 :: u \in [\![e_1]\!](F_T) \qquad t_2 :: u \in [\![e_2]\!](F_T)}{(t_1 \otimes (\ominus\, t_2)) :: u \in [\![e_1 - e_2]\!](F_T)} \text{(Diff-2)}$$

$$\frac{X_T \subseteq [\![e]\!](F_T) \qquad \{t_i :: u_i\}_{i=1}^n = X_T \qquad \{\bar{t}_j :: \bar{u}_j\}_{j=1}^m = [\![e]\!](F_T) - X_T \qquad u \in g(\{u_i\}_{i=1}^n)}{(\bigotimes_{i=1}^n t_i) \otimes (\bigotimes_{j=1}^m (\ominus\, \bar{t}_j)) :: u \in [\![\gamma_g(e)]\!](F_T)} \text{(Aggregate)}$$

规则语义

$$. : \mathrm{U}_T \to \mathrm{U}_T, \qquad [\![r]\!] : \mathrm{F}_T \to \mathrm{F}_T$$

（规范化）$U_T = \{(\bigoplus_{i=1}^n t_i) :: u \mid t_1 :: u, \ldots, t_n :: u$ 均为 U_T 中具有相同元组 u 的带标签元组$\}$

$$\frac{t^{\text{old}} :: u \in [\![p]\!](F_T) \qquad \langle [\![e]\!](F_T) \rangle \not\ni u}{t^{\text{old}} :: p(u) \in [\![p \leftarrow e]\!](F_T)} \text{(Rule-1)} \qquad \frac{t^{\text{new}} :: u \in \langle [\![e]\!](F_T) \rangle \qquad [\![p]\!](F_T) \not\ni u}{t^{\text{new}} :: p(u) \in [\![p \leftarrow e]\!](F_T)} \text{(Rule-2)}$$

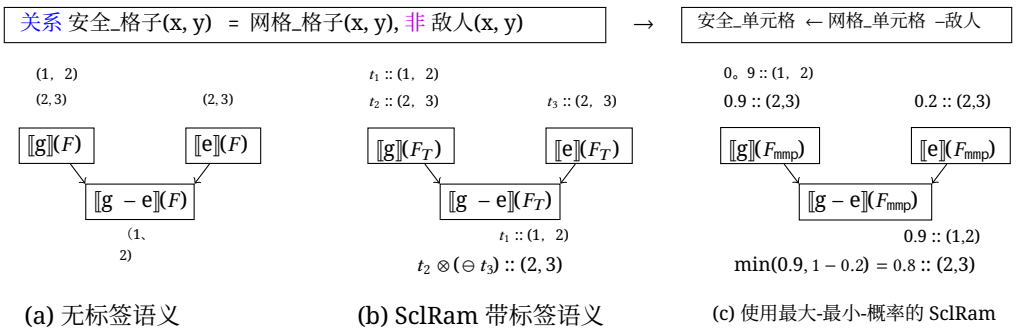
$$\frac{t^{\text{old}} :: u \in [\![p]\!](F_T) \qquad t^{\text{new}} :: u \in \langle [\![e]\!](F_T) \rangle}{(t^{\text{old}} \oplus t^{\text{new}}) :: p(u) \in [\![p \leftarrow e]\!](F_T)} \text{(Rule-3)}$$

层和程序语义

$$\text{lfp}^\circ : (\mathrm{F}_T \to \mathrm{F}_T) \to (\mathrm{F}_T \to \mathrm{F}_T), \qquad [\![s]\!], [\![\bar{s}]\!] : \mathrm{F}_T \to \mathrm{F}_T$$

（饱和） $F_T^{\text{old}} \triangleq F$新$_T$当且仅当 $\forall t^{\text{new}} :: p(u) \in F$新$_T, \exists t^{\text{old}} :: p(u) \in F$旧$_T$使得 $t^{\text{old}} \oplus t^{\text{new}}$（不动点）
$\text{lfp}^\circ(h) = h \circ \cdots \circ h = h^n$ 如果存在量词一个最小的 $n > 0$，使得 $h^n(F_T) \triangleq h^{n+1}(F_T)$（层） $[\![s]\!] = \text{lfp}^\circ(\lambda F_T.(F_T - \bigcup_{p \in P_s} F_T[p]) \cup (\bigcup_{r \in s} [\![r]\!](F_T)))$（程序）$[\![s]\!] = [\![s_n]\!] \circ \cdots \circ [\![s_1]\!]$，其中 $s = s_1; \ldots; s_n$.

图 7. SclRam 核心片段的操作语义。

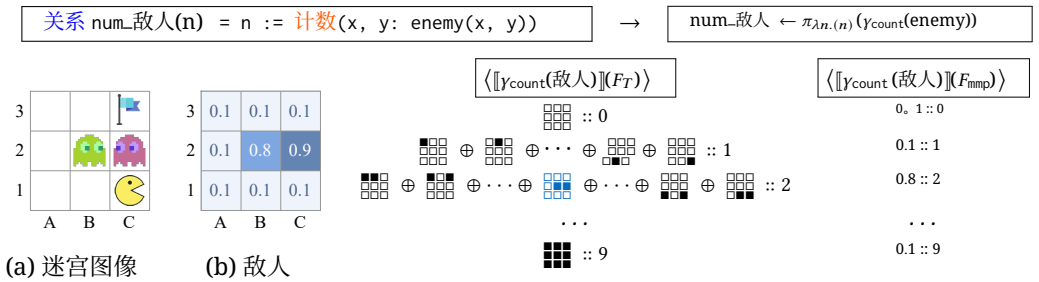


(a) 无标签语义　　(b) SclRam 带标签语义　　(c) 使用最大-最小-概率的 SclRam

图 8. 改编自第2 节的示例规则被编译为带有SclRam 差集的规则。 g 和e 是关系名称的缩写。这些图展示了在不同语义下对表达式g – e 的评估。

并集、投影或聚合。因此，我们对要合并的集合执行规范化，以连接（⊕）不同的标签。接下来，合并新推导出的元组（$\langle [\![e]\!](F_T) \rangle$）与之前推导出的元组（$[\![p]\!](F_T)$）时，有三种情况。如果某个事实只存在于旧集合或新集合中，我们只需将该事实传播到输出中。当一个元组 u 同时出现在旧集合和新集合中时，我们将旧标签与新标签的析取（$t^{\text{old}} \oplus t^{\text{new}}$）传播出去。综合这几种情况，我们得到一组谓词 p 下新标记的事实。

SclRam 中的递归实现方式类似于 Datalog 中的最小不动点迭代 [Abiteboul 等人1995]。迭代按层进行，以确保分层否定和聚合。计算层 s 的单步，意味着计算 s 中的所有规则，并返回更新后的



(a) 迷宫图像　(b) 敌人　(c) 聚合表达式的评估

图9. 在吃豆人迷宫中计数敌人的示例。上方展示了 Scallop 规则以及带聚合的编译后 SclRam 规则。(a) 和 (b) 将迷宫和概率敌人关系的内容可视化。例如，我们有 tB2 :: 敌人 $(B, 2)$，其中 $t\text{B}_2 = 0.8$。在 (c) 中，我们展示了抽象带标签语义下以及采用最大最小概率溯源时的两个归一化（. 在图7中定义）评估结果。每个符号（例如 ）表示一个与我们的竞技场相对应的世界（■：敌人；□：无敌人）。一个世界是 9 个标签的合取，例如 = $t\text{A}_3 \otimes (\ominus t\text{A2}) \otimes \cdots \otimes (\ominus t\text{C1})$。我们标记出正确的世界 ，它给出的答案为 2。

```
rel path(x1,y1,x3,y3) = (edge(x1,y1,x3,y3) 或 path(x1,y1,x2,y2) 且 edge(x2,y2,x3,y3)) 且非 enemy(x3,y3)
```

迭代计数 i	1	2	3	4	5	6	7
$t^{(i)}_{\text{C1-C3在 } F(i)_T}$	–		(相同)	⊕ ⊕ ⋯ ⊕	(相同)	⊕ ⊕ ⋯ ⊕	(相同)
$t^{(i)}_{\text{C1-C3}}$ 在 $F^{(i)}_{\text{mmp}}$	–	0.1	0.1	0.2	0.2	0.9	0.9
$t^{(i)}_{\text{C1-C3}}$ 饱和?	–	假	true	false	true	假	true
$F^{(i)}_{\text{mmp}}$ 已饱和?	假	假	假	假	假	false	true

图 10. 展示不动点迭代的示例，用于检查角色在 C1 处是否能在不遇到敌人的情况下到达 C3（图 9a）。用于推导该结果的 Scallop 规则定义在顶部，我们假设双向边已填充，并由 **1** 标记。令$t_{\text{C1-C3}}$ 为与 path(C,1,C,3) 关联的标签。我们使用类似这样的符号来表示沿图示路径对 enemy 的取反标签的合取，例如 $= (\ominus t_{\text{C2}}) \otimes (\ominus t_{\text{C3}})$。第 2 次迭代是首次$t_{\text{C1-C3}}$ 被推导出的时刻，但路径被敌人阻断。第 6 次迭代中，最佳路径被推导并写入标签中。此后，在 最大-最小-概率 溯源下，标签$t_{\text{C1-C3}}$ 和数据库F_{mmp} 均达到饱和，导致迭代停止。相比之下，Datalog 中的无标签语义将在 4 次迭代后停止，而采用 mmp 的 SclRam 达到饱和的速度较慢，但能够探索更优的推理链。

数据库。注意，我们定义了一个专用的最小不动点算子 **lfp**°，当整个数据库达到 饱和时，迭代便会停止。图 10 展示了一个涉及递归和数据库饱和的评估过程。整个数据库在第 7 次迭代时达到饱和，并找到代表吃豆人到达目标的最优路径的标签。由于存在值创建等特性，SclRam 并不总能保证终止。但可以针对每种溯源分别证明其终止性。例如，很容易证明，如果一个程序在无标签语义下终止，那么它在带有 最大-最小-概率溯源的带标签语义下也会终止。

4.4 外部接口和执行流水线

到目前为止，我们只阐释了 max-min-prob 溯源，其中标签是近似概率。还有其他概率溯源，其标签更为复杂，例如证明树或布尔公式。因此，对于每种溯源 T，我们引入一个输入标签空间 I、一个输出标签空间 O、一个标签函数 τ： $I \to T$，以及一个恢复函数 ρ： $T \to O$。例如，所有概率溯源都共享相同的输入和输出标签空间 $I = O = [0, 1]$，以提供统一接口，而内部标签空间 T则可能不同。我们将这 4 元组 $(I,\ O,\ \tau,\ \rho)$ 称为

溯源	T	$\mathbf{0}$	$\mathbf{1}$	$t_1 \oplus t_2$	$t_1 \otimes t_2$	$\ominus t$	$t_1 \hat{=} t_2$	$\tau(r_i)$	$\rho(t)$
diff-max-min-prob	$\mathbb{D}$	$\hat{0}$	$\hat{1}$	最大值(t_1, t_2)	$\min(t_1, t_2)$	$\hat{1} - t$	$t_1^{\text{fst}} == t_2^{\text{fst}}$	$(r_i, \vec{\mathbf{e}}_i)$	t
差分加乘概率	$\mathbb{D}$	$\hat{0}$	$\hat{1}$	截断$(t_1 + t_2)$	$t_1 \cdot t_2$	$\hat{1} - t$	true	$(r_i, \vec{\mathbf{e}}_i)$	t
diff-top-k-proofs	Φ	$\emptyset$	$\{\emptyset\}$	$t_1 \vee_k t_2$	$t_1 \wedge_k t_2$	$\neg_k t$	$t_1 == t_2$	$\{\{\text{pos}(i)\}\}$	加权模型计数(t, Γ)

图 11. 三种可微溯源的定义。

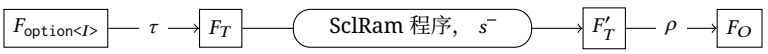
$\hat{a}_i = (a_i, \nabla a_i) \in \mathbb{D}\ \hat{0} = (0, \vec{0})\ \hat{a}_1 + \hat{a}_2 = (a_1 + a_2, \nabla a_1 + \nabla a_2)\ \hat{a}_1 \cdot \hat{a}_2 = (a_1 \cdot a_2, a_2 \cdot \nabla a_1 + a_1 \cdot \nabla a_2)$ 最小值$(\hat{a}_1, \hat{a}_2) = \hat{a}_i$，其中 $i = \text{argmin}_i(a_i)$ 最大值 $(\hat{a}_1, \hat{a}_2) = \hat{a}_i$，其中 $i = \text{argmax}_i(a_i)\ \hat{1} = (1, \vec{0}) - \hat{a}_1 = (-a_1, -\nabla a_1)$ 截断$(\hat{a}_1) = (\text{截断}_0^1(a_1)\ \nabla a_1)$

Fig. 12. 对偶数 $\mathbb{D} \triangleq [0,1] \times \mathbb{R}^n$ 上的运算，其中 n 是输入概率的数量 s.

（变量）$i \in 1 \ldots n$ $\varphi_1 \vee_k \varphi_2 = \text{top}_k(\varphi_1 \cup \varphi_2)$（文字）$\nu ::= \text{pos}(i) \mid \text{neg}(i)$（证明）$\eta ::= \{\nu_1, \nu_2, \ldots\}$（公式） $\Phi \ni \varphi ::= \{\eta_1, \eta_2, \ldots\}\ \varphi_1 \wedge_k \varphi_2 = \text{top}_k(\{\ \eta \mid (\eta_1, \eta_2) \in \varphi_1 \times \varphi_2, \eta = \eta_1 \cup \eta_2, \eta \text{ 无冲突}\})$ $\neg_k \varphi = \text{top}_k(\text{cnf2dnf}(\{\{\neg\nu \mid \nu \in \eta\} \mid \eta \in \varphi\}))\ \Gamma(i) = (r_i, \vec{\mathbf{e}}_i)$

图 13. diff-top-k-proofs 溯源所用的定义。

外部 溯源 T 的接口。整个执行流程如下图所示 ：



在 Scallop 应用的上下文中，外延数据库以 $F_{\text{option<I>}}$ 的形式提供。在标注阶段，τ 会应用于每个输入标签，以获得 F_T，随后 SclRam 程序会在 F_T 上运行。为方便起见，并非所有输入事实都需要标注——未标注的输入事实在 F_T 中会直接与标签 $\mathbf{1}$ 关联。在恢复阶段，ρ 会应用以获得 F_O，即整个流程返回的内涵数据库。Scallop 允许用户指定一组输出关系，且 ρ 只会应用于这些关系下的标签，以避免冗余计算。

4.5 基于溯源的可微推理

现在，我们阐明溯源如何也支持可微推理。设外延数据库中的所有概率构成向量 $\vec{r} \in \mathbb{R}^n$，而由此得到的内涵数据库中的概率构成向量 $\vec{y} \in \mathbb{R}^m$。微分涉及求出输出概率 $\vec{y}$ 以及导数 $\nabla\vec{y} = \frac{\partial \vec{y}}{\partial \vec{r}} \in \mathbb{R}^{m \times n}$。

在 Scallop 中，可以使用可微溯源（DP）来获取这些结果。DP 具有相同的外部接口——设输入标签空间 $I = [0,1]$ 和输出标签空间 O 均为对偶数$\mathbb{D}$ 的空间（图12）。现在，每个输入标签 $r_i \in [0,1]$ 都是一个概率，而每个输出标签$\hat{y}_j = (y_j, \nabla y_j)$则封装了输出概率$y_j$及其相对于输入的导数 ∇y_j。由此，我们可以通过分别堆叠 $\vec{y}$和 $\nabla\vec{y}$，得到预期的输出y_j和 ∇y_j。

Scallop 提供了 8 种可配置的内置 DP，它们在运行效率、推理粒度和性能方面各有实证优势。本节将详细介绍其中 3 种简单而灵活的 DP，其定义如图所示。11。在接下来的讨论中，我们用 r_i表示 i的第 $\vec{r}$ 个元素，其中 i称为一个 变量 （ID）。向量 $\vec{\mathbf{e}}_i \in \mathbb{R}^n$是标准基向量，除第 i 个分量外，其余分量均为 0。

4.5.1 `diff-max-min-prob` *(*`dmmp`*).* 这种溯源是 `mmp` 的可微版本。从输入标签中获取 r_i时，我们通过附加 $\vec{\mathbf{e}}_i$ 作为其导数，将其转换为对偶数。请注意，在整个执行过程中，导数最多始终只有一个非零项，且具体为 1 或 -1。饱和检查仅基于概率部分的相等性，因此导数不会影响终止。它的所有操作都可以通过时间复杂度为 $O(1)$ 的算法实现，因此运行时间效率极高。

4.5.2 `diff-add-mult-prob` *(*`damp`*).* 这种溯源与 `dmmp` 具有相同的内部标签空间、标签函数和恢复函数。顾名思义，它的析取和合取操作分别是对偶数的 + 和 ·。执行析取时，我们会截断对偶数执行 + 后得到的实部，同时保持导数不变。`damp` 的饱和函数被设计为始终返回真，以避免不终止。但这一选择使其不太适用于复杂递归程序。`damp` 中操作的时间复杂度为 $O(n)$，虽慢于 `dmmp`，但在实践中仍然非常高效。

4.5.3 `diff-top-k-proofs` *(*`dtkp`*).* 这种溯源扩展了*top-k* 证明半环，最初由 [Huang 等人提出。 2021] 此外，它还支持否定和聚合。如图11 和13所示，`dtkp` 的标签是 $\varphi \in \Phi$ 析取范式（DNF）中的布尔公式。DNF 中的每个合取子句称为一个证明 η。一个公式最多可以包含 k 个证明，其中 k是可调节的超参数。在整个执行过程中，布尔公式通过 $\vee_k$、$\wedge_k$ 和 $\neg_k$进行传播，它们在 DNF 公式上的作用类似于或、与和非。这些计算结束时，会应用 top_k，仅保留证明概率最高的 k 个证明：

$$\Pr(\eta) = \prod_{v \in \eta} \Pr(v), \quad \Pr(\text{pos}(i)) = r_i, \quad \Pr(\text{neg}(i)) = 1 - r_i. \tag{1}$$

在 $\wedge_k$期间合并两个证明时，可能会出现文字冲突，例如 pos(i) 和 neg(i)；此时我们会删除整个证明。要对 φ执行否定 $\neg_k$，我们首先对所有文字取反，从而得到与 $\neg\varphi$等价的合取范式（CNF）。然后执行 cnf2dnf 操作（包括冲突检查），将其转换回析取范式（DNF）。为了从 DNF 公式 φ_j中获得输出对偶数 $\hat{y}_j$，即第 j 个输出元组的标签，我们采用了 [Manhaeve 等人提出的可微加权模型计数（WMC）过程 2021]。WMC 根据各个变量的权重计算布尔公式 φ_j的权重。具体而言，$\hat{y}_j = \text{WMC}(\varphi_j, \Gamma)$，其中 $\Gamma(i) = (r_i, \bar{e}_i)$ 是从变量到其对偶数的映射。请注意，WMC 是 #P 完全的，也是使用这种溯源方式时运行时间的主要开销来源。可调节的 k 使用户能够在运行时间和推理粒度之间进行权衡。详细的运行时间分析见附录 B。

4.6 实践考量

最后，我们讨论有关 SclRam 扩展和溯源选择的一些实践问题。

附加功能。 我们仅介绍了ScLRam的核心片段 的语法和语义。ScLRam 还支持以下实用功能：1）采样操作及支持这些操作的溯源扩展，2）分组聚合，3）基于标签的提前移除及其在溯源中的扩展，以及 4）`dtkp`中的互斥标签。我们在附录B中对这些功能进行了形式化描述。

溯源选择。 一个自然的问题是，如何为给定的 Scallop 应用选择可微溯源。根据我们在第6节中的实证评估，`dtkp` 通常表现最佳，而设置 $k = 3$ 通常是兼顾运行效率和学习性能的良好选择。这表明用户可以从`dtkp`这一默认选项开始。总体而言，溯源选择过程与机器学习中常见的超参数调优过程类似。

5 实现

我们使用 4.5 万行 Rust 代码实现了 Scallop 的核心系统。各个模块的代码行数如表 1 所示。在编译器中，表层语言与 front-IR 和 back-IR 之间存在两级中间表示，直至 ScLRam 语言。在 front-IR 中，我们执行类型推断等分析，并进行语法糖去除等转换。在 back-IR 中，我们生成查询计划并应用优化。运行时间系统直接运行于 ScLRam 之上，并基于针对标签语义特化的半朴素求值机制

模块	LoC
编译器	19K
运行时间	16K
解释器	2K
scallopy	4K

表1。核心 Scallop 模块的代码行数。



图 14. 基准任务的可视化。每个任务名称旁标明了训练数据集的大小和输出域。由于吃豆人迷宫已在第 2 节中展示，因此此处省略。

语义。对于解释执行和编译执行的 SclRam 程序，分别提供动态 和静态 运行时间系统。目前，所有计算仅在 CPU 上进行，但可以针对机器学习按批次并行化。

Scallop 可通过多种接口使用，例如解释器、编译器和交互式终端。它还提供了语言绑定，例如用于 Python 的 `scallopy` 以及用于 WebAssembly 和 JavaScript 的 scallop-wasm。借助 `scallopy`，Scallop 可以无缝集成到 PyTorch 等机器学习框架中，其中 `scallopy` 模块与其他 PyTorch 模块一样使用。在 `scallopy` 中指定 `jit` 后，Scallop 程序可以通过即时 (JIT) 编译为 Rust，构建成二进制文件，并动态加载到 Python 环境中。

Scallop 提供 18 种内置溯源（4 种用于离散推理，6 种用于概率推理，8 种用于可微推理）。加权模型计数算法使用 句子决策图 (SDD)[Darwiche 2011] 通过朴素的自底向上编译实现。我们允许每种溯源为聚合和采样等操作提供自己的实现。这赋予了充分的灵活性，并支持对复杂操作进行优化。我们的溯源框架还与`scallopy`集成，使得能够在 Python 中快速创建和测试新的溯源。

6 评估

我们在包含八个神经符号应用的基准测试套件上评估 Scallop 语言和框架。我们的评估旨在回答以下研究问题：

RQ1 Scallop 在解决多样化神经符号任务方面有多强的表达能力？**RQ2** 在准确率方面，Scallop 的解决方案与最先进水平的基线方法相比如何？**RQ3** Scallop 的可微推理模块是否具有较高的运行时间效率？**RQ4** Scallop 是否能有效提升泛化能力、可解释性和数据效率？**RQ5** Scallop 解决方案有哪些失效模式？我们该如何缓解？

在以下各节中，我们首先介绍基准任务以及为每项任务所选择的基线方法（第 6.1 节）。然后，我们在第 **RQ1** 至**RQ5** 节中分别回答 6.2 至第 6.6 节中的问题。所有与Scallop和运行时间相关的实验均在一台配备两颗20核英特尔至强 CPU、四块 GeForce RTX 2080 Ti GPU 和 768 GB 内存的机器上进行。

6.1 基准测试与基线方法

我们在图中概述了基准测试。14这些基准测试涵盖了涉及感知和推理的广泛任务。输入数据的模态包括图像和视频，以及

自然语言文本和知识库（KB）。图中还展示了训练数据集的规模。接下来，我们将详细介绍这些基准任务及其对应的基线方法。

MNIST-R：一个合成*MNIST*测试套件。该基准用于测试Scallop的各种功能，例如否定和聚合。每项任务都以一张或多张来自MNIST数据集的手写数字图像作为输入[Lecun等人1998]并对图中描绘的数字执行简单算术（sum2、sum3、sum4）、比较（小于）、否定（非3或4）或计数（count-3、count-3-or-4）。对于count-3和count-3-or-4，我们对一组8张图像中的数字进行计数。在此测试套件中，我们使用基于卷积神经网络的模型DeepProbLog（DPL）[Manhaeve等人2021],以及我们之前的工作[Huang等人2021]（先前工作）作为基线方法。

HWF：手写公式解析与评估。HWF，由[李等人2020],提出，涉及手写公式的解析和评估。公式以图像序列的形式提供，其中每个图像表示一个数字（0-9）或一个算术符号（+、−、×、÷）。公式根据语法构成良好形式，且不会除以零。公式的大小范围为1至7，并作为输入的一部分给出。目标是评估公式并获得一个有理数作为结果。我们从[李等人2020]的基线方法NGS-*m*-BS、NGS-RL和NGS-MAPO中进行选择，它们是专门为此任务设计的神经符号方法。

Pathfinder：具有长程依赖的图像分类。在这项来自[Tay等人2020],的任务中，输入是一幅图像，其中包含两个可能由弯曲的虚线连接的点。目标是判断这两个点是否相连。该任务包含两个子任务，即Path和Path-X，其中Path包含32×32幅图像，而Path-X包含128×128幅图像。我们选取标准卷积神经网络和基于Transformer的模型作为基线方法，同时还选取最先进水平的神经网络模型S4[Gu等人。2021], S4*[Gu等人2022],和SGConv[Li等人2022]。

吃豆人迷宫：玩吃豆人迷宫游戏。如第2节所述，该任务测试智能体识别图像中的实体并规划吃豆人到达目标的路径的能力。RL环境提供游戏状态图像作为输入，智能体必须规划在每一步采取的最优动作{上，下，左侧，右}。由于环境在每次会话中都是随机化的，因此不存在“训练数据集”。我们选取基于卷积神经网络的深度Q网络（DQN）作为基线。与其他任务不同，我们使用“成功率”指标进行评估，即在1000次游戏会话中，统计吃豆人在规定时间预算内到达目标的次数。

CLUTRR：从自然语言语境中进行亲属关系推理。在这项来自[Sinha等人2019],的任务中，输入包含一段关于一组人物的自然语言（NL）文本。文本中的每个句子都暗示着亲属关系。目标是推断给定一对人物之间的关系。文本中没有明确陈述目标关系，必须通过推理链加以推导。我们的基线模型包括RoBERTa[Liu等人2019],双向长短期记忆网络[Graves等人2013], GPT-3-FT（微调）、GPT-3-ZS（零样本）和GPT-3-FS（5-shot）[Brown等人2020]。在另一种设置CLUTRR-G中，提供的不是自然语言文本，而是与该文本对应的结构化亲属关系图，这使其成为一个知识图谱推理问题。对于CLUTRR-G，我们选用GAT[Veličković等人2017]和CTP[Minervini等人2020]作为基线方法。

[{"id":0,"output":"***Mugen***：视频-文本对齐与检索。Mugen[Hayes等人2022]基于一款名为CoinRun的游戏[Cobbe等人2019]。在视频-文本对齐任务中，输入包含一段时长为3.2秒的游戏视频片段，以及一小段描述视频中发生事件的自然语言文字。目标是计算一个相似度分数，用于表示二者的“对齐”程度。随后有两个任务：视频到文本检索（VTR）和文本到视频检索（TVR）。在TVR中，输入是一段文本和一组包含16个视频的视频集合，目标是检索出与文本最匹配的视频。在VTR中，目标是从视频中检索文本。我们将我们的方法与SDSC[Hayes等人2022]。"}]

CLEVR：组合语言与基础视觉推理*[Johnson等人2017]*。在这项视觉问答（VQA）任务中，输入包含一张几何物体的渲染图像，以及一个询问物体计数、属性和关系的自然语言问题。目标是

Task	输入	神经网络	接口关系	Scallop 程序	特性 R	N	A		LoC
MNIST-R	图像	CNN	digit(*id, digit*)	算术、比较、否定。 tion，以及计数。		✔	✔		2†
HWF	图像	CNN	符号(*id*, 符号) 长度(长度)	解析并计算公式 针对识别出的符号。	✔				39
Pathfinder	图像	CNN	点(*id*) 虚线(来自_*id*，至_*id*)	检查这些点是否 用虚线连接。	✔				4
吃豆人-Maze	图像	CNN	角色(*x, y*) 敌人(*x, y*) 目标(*x, y*)	通过以下方式规划最优动作 寻找一条无敌人路径 从角色到目标。	✔	✔	✔		31
CLUTRR (-G)	NL	RoBERTa	亲属关系(关系*a,sub*、*obj*)	推断所查询的关系 通过递归地应用所学到的 组合规则。	✔	✔	✔		8
	查询*	–	问题(子项, 对象)						
	Rule	–	复合关系(r_1, r_2, r_3)						
Mugen	视频	S3D	动作(帧，动作，模)	检查以下内容中指定的事件 自然语言文本是否与动作 视频中识别出的动作相匹配。	✔	✔	✔		46
	NL	DistilBERT	表达式(表达式_标识，动作) mod(*expr_id, mod*)						
CLEVR	图像	Fast R-CNN	对象_属性(对象_标识符，属性，值) obj_关系(关系，o_1, o_2)	解释 CLEVR-DSL 程序（提取自 问题）在场景图上 （提取自图像）。	✔	✔	✔		51
	NL	双向长短期记忆网络	filter_expr(*e, ce, attr, val*) 计数_expr(*e, ce*), ...						
VQAR	图像	Fast R-CNN	obj_name(对象_标识符, 名称) obj_attr(对象_标识, 值) obj_rela(*rela*, o_1, o_2)	评估场景上的查询 图（从图像中提取） 借助常识 知识库 (KB)。	✔				42
	KB*	–	是_a(*name1,name2*)，.....						

表 2. 各任务中 Scallop 解决方案的特征。未学习的结构化输入以 *表示。所用神经模型包括用于自然语言 (NL) 的 RoBERTa [Liu 等人 2019], DistilBERT [Sanh 等人 2019], 以及 BiLSTM [Graves 等人 2013]；用于图像的 CNN 和 FastRCNN [Girshick 2015]；以及用于视频的 S3D [Xie 等人2018]。我们展示了各解决方案所用 Scallop 的三个关键特性：(R)递归、(N)否定和 (A)聚合。 †：对于 MNIST-R，每个子任务的代码行数 (LoC) 均为 2。

根据图像回答问题。我们选取 NS-VQA 作为基线方法 [Yi 等人 2018]和 NSCL[Mao 等人 2019], 作为专门为此任务设计的神经符号方法 。

VQAR：结合常识推理的视觉问答。与 CLEVR 一样，该任务也涉及视觉问答，但有三个显著差异：它包含来自 GQA 数据集的真实图像[Hudson 和 Manning 2019]；查询采用程序化形式，要求检索图像中的物体；此外还有一个以常识知识库（KB）形式提供的额外输入[Gao 等人 2019]其中包含诸如（长颈鹿，是一种，动物）这样的三元组，用于常识推理。该任务的基线方法包括 NMNs [Andreas 等人 2016]和 LXMERT [Tan 和 Bansal 2019]。

6.2 RQ1：我们的解决方案与表达能力

为回答 **RQ1**，我们展示了针对基准任务（表 2）的 Scallop解决方案。对于每项任务，我们明确指定充当神经组件与符号组件之间桥梁的接口关系。神经模块处理感知输入，并将其输出映射为接口关系中的（概率）事实。随后，我们的 Scallop 程序以这些事实作为输入，执行所描述的推理，从而生成最终输出。如 特征列所示，我们的解决方案使用了 Scallop 提供的所有核心特征。

附录中提供了每个任务的完整 Scallop 程序C。这些程序都很简洁，表 2最后一列的代码行数对此进行了说明。我们重点介绍 HWF、Mugen 和 CLEVR 三个任务，以展示Scallop 的表达能力。对于 HWF，Scallop程序由一个公式解析器组成。它能够根据用于简单算术表达式的上下文无关文法，解析概率输入符号。对于 Mugen，Scallop 程序是一个 时序

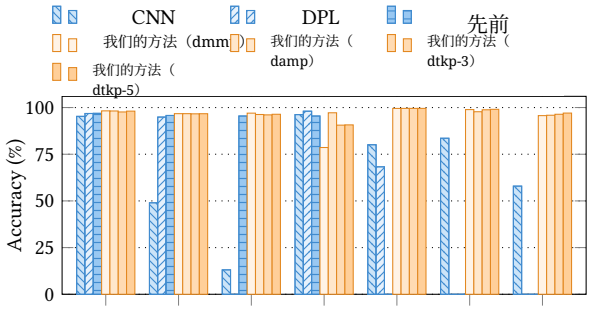


图 15. MNIST-R 套件准确率比较

方法	Scallop			DQN
	dmmp	damp	dtkp-1	
成功率	8.80%	7.84%	**99.40%**	84.90%
#剧集	50	50	**50**	50K

表 3。吃豆人迷宫性能。

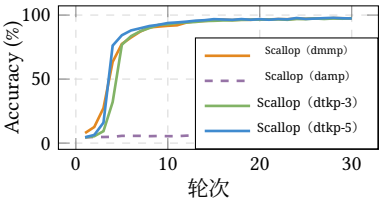


图 16。 HWF 学习曲线。

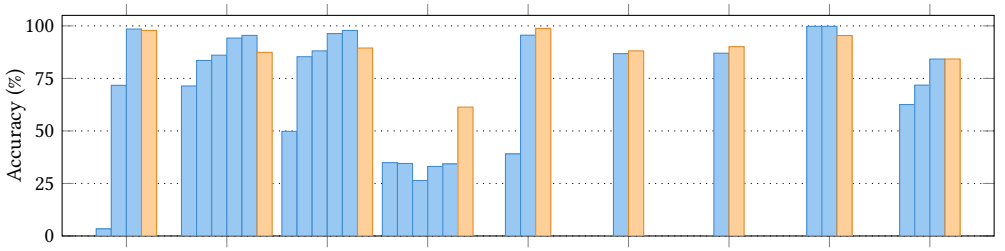


图 17。总体基准准确率比较。对于每项任务，均标出了我们方案表现最佳的溯源结构。在所展示的任务中，`dtkp` 在 6 个任务上表现最佳，`damp` 在 2 个任务上表现最佳，而`dmmp` 在 1 个任务上表现最佳。

规范检查器，其中规范从自然语言文本中提取，用于匹配从视频中摘录的顺序事件。对于 CLEVR，Scallop 程序是 CLEVR-DSL 的解释器。CLEVR-DSL 是在 CLEVR数据集中引入的一种领域特定函数式语言 [Johnson 等人 2017]。

我们注意到，我们之前的工作 [Huang 等人 2021], 仅支持正Datalog，因此无法表达 8 项任务中的 5 项，因为这些任务需要否定和聚合，如“N”和“A”列所示。此外，HWF 需要浮点支持，而我们之前的工作也缺少这一支持。

除了多种多样的感知数据和推理模式外，Scallop 程序还应用于各种学习设置。如第2节所示，吃豆人迷宫的程序用于在线表示学习 设置。对于 CLUTRR，我们编写完整性约束（类似于第3.2节所示的约束），以推导语义损失，用于约束语言模型输出。对于 CLUTRR-G，我们将可学习的权重附加到复合关系 事实（例如复合关系(FATHER, MOTHER, 祖母)）上，从而能够从数据中学习此类事实，这类似于 ILP 中的规则学习 。对于 Mugen，我们的程序在对比学习设置下进行训练，因为它需要最大化对齐的视频文本对之间的相似度分数，同时最小化未对齐的视频文本对之间的相似度分数。

6.3 RQ2：性能和准确率

为回答 **RQ2**，我们从两个方面评估我们方法的性能和准确率：1）与现有基线方法相比，我们解决方案的最佳性能；以及 2）具有不同溯源结构（`dmmp`、`damp`、`dtkp` 以及不同 k）时，我们解决方案的性能。

我们首先在所有基准任务上将我们的解决方案与选定的基线方法进行比较，如图 15、表 3以及图 17所示。首先，我们重点介绍两个应用：吃豆人迷宫和 CLUTRR，它们从我们的解决方案中受益最多。对于吃豆人迷宫，与DQN 相比，我们在训练回合方面获得了1,000× 的加速，并达到了近乎完美的 99.4% 成功率。

请注意，我们的解决方案对环境动态（即游戏规则）进行了编码，而这些动态在DQN 模型中不可用且难以纳入。对于 CLUTRR，我们相较于基线方法取得了 25%的提升，其中包括 GPT-3-FT——在 CLUTRR 数据集上微调的最先进水平的大语言模型。接下来，对于HWF 和CLEVR 等任务，我们的解决方案取得了相当的性能，甚至相比于专为各项任务设计的神经符号基线方法 NGS-m-BS、NSCL 和 NS-VQA 也是如此。在路径和Path-X 上，我们的解决方案相较于底层模型卷积神经网络取得了 4% 的准确率提升，甚至优于精心设计的基于 Transformer 的模型 S4。

Scallop 解决方案在每项任务上的性能取决于所选的溯源结构。从表 3 和图 15–17可以看出，尽管 `dtkp`通常表现最佳，但本文介绍的每种溯源方法都有其用武之地，例如，`dtkp` 适用于吃豆人迷宫和 VQAR，`damp` 适用于小于（MNIST-R）和 Mugen，而 `dmmp` 适用于 HWF 和 CLEVR。请注意，在正Datalog下，Scallop 的`dtkp` 与 [Huang 等人 2021],的方法相同，因此我们能够取得相近的性能。总之，支持配置溯源有助于针对不同应用调整我们的方法。

6.4 RQ3：运行效率

我们评估了采用不同溯源结构的 Scallop 解决方案的运行效率，并将其与基线神经符号方法进行比较。如表 4所示，在 MNIST-R 任务上，Scallop 的速度显著快于 DeepProbLog（DPL）。DPL 是一种基于 Prolog、使用精确概率推理的概率编程系统。例如，在 sum4 任务上，DPL 需要 40 天才能完成仅 4K 个训练样本的处理，这表明它在实际应用中慢得难以接受。相比之下，Scallop 解决方案只需几分钟即可完成一个训练轮次（15K 个样本），且不会牺牲测试准确率（见图 15）。对于 HWF，即使与手工设计的专用 NGS-m-BS 方法相比，Scallop 也能实现相当的运行效率。

比较不同的溯源结构时，我们看到，对于 `dtkp`，增加 k会导致运行时间显著膨胀。这是意料之中的，因为增加 k 会产生更大的布尔值公式标签，使 WMC 过程的速度呈指数级下降。在实践中，我们发现，对于 `dtkp` ， $k = 3$ 是在运行效率与推理粒度之间的良好平衡点。事实上，`dtkp` 概括了 DPL，因为可以将一个极大的 $k \geq 2^n$ （n 是输入事实的总数）设为极大值，以进行精确的概率推理。

6.5 RQ4：泛化能力、可解释性和数据效率

现在，除了准确率和运行时间之外，我们还考虑机器学习模型的其他重要理想特性，例如在未见输入上的泛化能力、输出的可解释性，以及训练过程的数据效率。为简洁起见，在每种情况下我们都聚焦于一个基准任务。

我们评估了 Scallop在CLUTRR 任务上的泛化能力。CLUTRR 中的每个数据点都标注有一个参数 k，表示推断目标亲属关系所需的推理链长度。为了测试不同解决方案的系统性泛化能力，我们使用具有$k \in \{2、3\}$ 的数据点对其进行训练，并使用具有 $k \in \{2、.....、10\}$的数据点进行测试。如图 18所示，神经网络基线方法在更复杂的未见实例上的准确率急剧下降，而 Scallop 解决方案的准确率下降得更为缓慢，表明其具有更好的泛化能力。

接下来，我们展示 Scallop 在 Mugen 任务中的可解释性。尽管该任务的目标是判断视频文本对是否匹配，但我们方法中的感知模型会提取可解释符号，即受控角色在特定帧上的动作。图 19 显示，预测的（动作，mod）对与视频中的事件完全吻合。因此，我们的解决方案不仅能判断视频文本对是否匹配，还能说明原因。

最后，我们在 HWF 任务上评估 Scallop 的数据效率，使用较少的训练数据（表5）。当使用较少数据进行训练时，NGS-MAPO 等方法的表现会显著下降，而 Scallop 的测试准确率则缓慢下降，其数据效率与最先进水平相当

Task	Scallop				基线
	dmmp	damp	dtkp-3	dtkp-10	
sum2	**34**	88	72	185	21,430（DPL）
sum3	**34**	**119**	71	1,430	30,898 (DPL)
sum4	**34**	154	77	4,329	超时（DPL）
小于	35	**42**	34	43	2,540 (DPL)
非3或4	37	**33**	**33**	**34**	3,218（DPL）
HWF	89	107	**120**	8,435	79 (NGS-*m*-BS)
CLEVR	**1,964**	1,618	2,325	超时	–

表 4. 所选基准任务上的运行效率比较。所示数字为每个 epoch 的平均训练时间（秒）。以粗体标出我们准确率最高的变体。

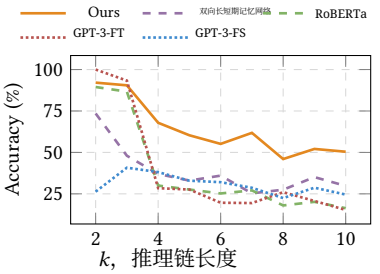


图 18. CLUTRR 数据集上的系统性泛化能力。



(攀爬,向上)　(收集,硬币)　(杀死,脸)

图 19. Mugen 数据集中示例视频片段最可能的预测（动作, mod）对。

%Train	Scallop	NGS		
	dtkp-5	RL	MAPO	*m*-BS
100%	97.85	3.4	71.7	98.5
50%	95.7	3.6	9.5	95.7
25%	92.95	3.5	5.1	93.3

表5. 仅使用部分数据进行训练时，各种方法在 HWF 上的测试准确率。数字以百分比（%）表示。

神经符号 NGS-*m*-BS方法。吃豆人迷宫任务还展示了Scallop 的数据效率，因为与DQN 相比，它只需少得多的训练回合，同时取得了高得多的成功率。

6.6 RQ5：失败模式分析

与纯神经模型相比，Scallop解决方案提供了更高的透明度，使程序员能够有效地进行调试。通过手动检查接口关系，我们观察到，错误的主要来源在于神经组件的不准确预测。例如，用于CLUTRR的 RoBERTa 模型只能正确提取 84.69% 的亲属关系。这里有两个潜在原因——要么神经组件的能力不足，要么我们的解决方案没有提供充分的监督来训练它。前者可以通过采用更好的神经网络架构或更多数据来解决。后者可以通过不同方式缓解，例如调整所选的溯源，或将完整性约束（在第 3.2 节中讨论）纳入训练和/或推理过程。例如，在吃豆人迷宫中，一条“竞技场中不应有超过一个 目标”的约束有助于提高鲁棒性。

7 相关工作

我们综述四个彼此不同但又相互交叠的领域中的相关工作：溯源推理、Datalog 与逻辑编程、概率编程与可微编程，以及神经符号方法。

关系代数与溯源。 关系代数在数据库领域得到了广泛研究 [Abite- boul 等人 1995], 并将其扩展到递归 [Jachiet 等人 2020]在 Datalog 中。溯源半环框架 [Green 等人 2007] 最初是为正关系代数（RA$^+$）提出的，后来扩展到差集 [Amsterdamer 等人 2011] 和不动点 [Dannert 等人 2021]。该框架已应用于 Datalog 的各种变体 [Khamis 等人 2021]并支持程序合成等应用 [Si 等人 2019]。Scallop 采用扩展的溯源半环框架，并展示了其在可配置可微推理中的应用。

Datalog 是一种声明式逻辑编程语言，其中规则是逻辑公式。尽管其表达能力不如 Prolog，但它在编程语言和

数据库领域的研究者广泛研究如何扩展语言并进行优化 [Khamis 等人 2021]。人们构建了多种基于 Datalog 的系统，用于程序分析 [Scholz 等人 2016] 和企业数据库 [Aref 等人 2015]。Scallop 也是一种基于 Datalog 的系统，它扩展了 [Huang 等人 2021]，加入了否定和聚合等众多语言构造。

概率编程 是一种供程序员对分布进行建模并执行概率采样与推理的范式。此类系统包括 ProbLog [Dries 等人 2015], Pyro[Bingham 等人 2018], Turing [Ge 等人 2018], 和 PPL [van de Meent 等人 2018]。与现代机器学习系统集成后，这类系统非常适合统计建模和构建生成模型。Scallop 也支持 ProbLog 风格的精确概率推理，这是我们溯源框架的一种实现方式。不过，目前尚不支持高级统计采样和生成式建模，这些将留待未来工作。

可微编程 (DP) 系统旨在支持编写可微分的代码。DP 的常见做法包括符号微分和自动微分（auto-diff）[Baydin 等人 2015], 由此产生了 PyTorch 等流行的机器学习框架 [Paszke 等人 2019],TensorFlow [Abadi 等人 2015], 和 JAX [Bradbury 等人 2018]。然而，这些系统大多是为实值函数设计的。Scallop 也是一种可微编程系统，但侧重于使用离散、逻辑和关系符号进行编程。

神经符号方法应运而生，旨在将符号推理融入现有的学习系统中。它们的成功已在诸多应用中得到证明 [Chen 等人，2020；Li 等人，2020；Mao etal. 2019；Minervini et al.2020；Wang et al.2019；Xuet al. 2022；Yi et al. 2018]。与 [Li 等人，2020；Mao et al.2019；Yiet al. 2018], 类似，我们主要关注涉及感知后进行符号推理的解决方案。融入符号知识的其他方式包括语义损失 [Xuet al. 2018, 2022], 程序合成 [Chen et al. 2021a；Shah et al.2020], 以及调用大语言模型（LLMs） [Cheng etal. 2022；Zelikman etal. 2023]。大多数上述神经符号方法都构建了自己的领域特定语言或专门的推理组件，其中许多都可以使用Scallop 进行编程，并且是我们的溯源框架的实例 [Chen et al. 2020；Mao et al.2019；Xu et al. 2018, 2022]。TensorLog[Cohen et al. 2017], DPL [Manhaeve et al. 2021], 以及 [Huangetal. 2021] 可以被视为具有不同表达能力的统一神经符号框架。Scallop受DPL 启发，并且进一步提供了一种更具可扩展性、更可定制且更易于使用的语言和框架。

8 结论

我们介绍了 Scallop，这是一种用于整合深度学习与逻辑推理的神经符号编程语言。我们提出了一种声明式语言，以及一个基于溯源计算的推理框架。通过将该框架应用于多种机器学习任务，我们证明了它具有实用性。具体而言，我们的实验表明，Scallop 解决方案与许多现有基线方法相比表现相当，甚至更胜一筹。

未来，我们计划从三个方面扩展 Scallop：1）支持更多机器学习范式，例如生成式建模、开放域推理、上下文学习和对抗学习。2）进一步提升 Scallop 语言和框架的易用性、效率与表达能力。我们计划提供与 TensorFlow 和 JAX 等其他机器学习框架的绑定，利用 GPU 等硬件加速计算，并支持代数数据类型等构造。3）将 Scallop 应用于现实世界和安全关键领域。例如，我们计划将其与 CARLA 自动驾驶模拟器集成 [Dosovitskiy 等人 2017] ，以便为自动驾驶系统指定软时间约束。我们还计划将 Scallop 应用于医疗领域，利用电子健康记录（EHR）数据实现可解释的疾病诊断。

致谢

我们感谢 Neelay Velingker、Hanjun Dai、Hanlin Zhang 和 Sernam Lin 对演示和实验提出的宝贵意见。我们感谢匿名审稿人和 artifact 审稿人提供的有益反馈。本研究得到了美国国防高级研究计划局（# FA8750-19-2-0201）、美国国家科学基金会（#2107429 和 #1836936）以及美国海军研究办公室（# N00014-18-1-2021）的资助。

工件可用性

复现本文实验所需的所有软件均可在此处获取：[Li 等人。 2023]此外，Scallop 的最新源代码及其文档可在以下网址获取：https://github.com/scallop-lang/scallop。

参考文献

Martín Abadi、Ashish Agarwal、Paul Barham、Eugene Brevdo、Zhifeng Chen、Craig Citro、Greg S. Corrado、Andy Davis、Jeffrey Dean、Matthieu Devin 等。2015。TensorFlow：异构分布式系统上的大规模机器学习。arXiv：1603.04467 Serge Abiteboul、Richard Hull 和 Victor Vianu。1995。数据库基础：逻辑层。Addison-Wesley Longman Publishing Co., Inc. Yael Amsterdamer、Daniel Deutch 和 Val Tannen。2011。论溯源在含差集查询中的局限性。（2011）。arXiv：1105.2255 Jacob Andreas、Marcus Rohrbach、Trevor Darrell 和 Dan Klein。2016。神经模块网络。发表于*IEEE*计算机视觉与模式识别会议（*CVPR*）。Molham Aref、Balder ten Cate、Todd J. Green、Benny Kimelfeld、Dan Olteanu、Emir Pasalic、Todd L. Veldhuizen 和 Geoffrey Washburn。2015。LogicBlox 系统的设计与实现。发表于*ACM*数据管理国际会议（*SIGMOD*）。Atilim Gunes Baydin、Barak A. Pearlmutter 和 Alexey Andreyevich Radul。2015。机器学习中的自动微分：综述。（2015）。arXiv：1502.05767 Eli Bingham、Jonathan P. Chen、Martin Jankowiak、Fritz Obermeyer、Neeraj Pradhan、Theofanis Karaletsos、Rohit Singh、Paul Szerlip、Paul Horsfall 和 Noah D. Goodman。2018。Pyro：深度通用概率编程。《机器学习研究杂志》（2018）。James Bradbury、Roy Frostig、Peter Hawkins、Matthew James Johnson、Chris Leary、Dougal Maclaurin、George Necula、Adam Paszke、Jake VanderPlas、Skye Wanderman-Milne 和 Qiao Zhang。2018。*JAX*：*Python* 的可组合变换+*NumPy* 程序。https://github.com/google/jax。Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。2020。语言模型是少样本学习者。发表于神经信息处理系统大会（*NeurIPS*）。Swarat Chaudhuri、Kevin Ellis、Oleksandr Polozov、Rishabh Singh、Armando Solar-Lezama、Yisong Yue 等。2021。神经符号编程。《编程语言基础与趋势》7，3（2021）。Mark Chen、Jerry Tworek、Heewoo Jun、Qiming Yuan、Henrique Ponde de Oliveira Pinto、Jared Kaplan、Harri Edwards、Yuri Burda、Nicholas Joseph、Greg Brockman 等。2021b。评估基于代码训练的大语言模型。（2021）。arXiv：2107.03374Qiaochu Chen、Aaron Lamoreaux、Xinyu Wang、Greg Durrett、Osbert Bastani 和 Isil Dillig。2021a。通过神经符号程序合成实现网页问答。发表于*ACM*编程语言设计与实现国际会议（*PLDI*）。Xinyun Chen、Chen Liang、Adams Wei Yu、Denny Zhou、Dawn Song 和 Quoc V. Le。2020。神经符号阅读器：面向阅读理解的分布式表征与符号表征的可扩展集成。发表于国际学习表征会议（*ICLR*）。Zhoujun Cheng、Tianbao Xie、Peng Shi、Chengzu Li、Rahul Nadkarni、Yushi Hu、Caiming Xiong、Dragomir Radev、Mari Ostendorf、Luke Zettlemoyer 等。2022。在符号语言中绑定语言模型。（2022）。arXiv：2210.02875 Karl Cobbe、Oleg Klimov、Chris Hesse、Taehoon Kim 和 John Schulman。2019。量化强化学习中的泛化能力。发表于国际机器学习会议（*ICML*）。PMLR，1282–1289。William W. Cohen、Fan Yang 和 Kathryn Rivard Mazaitis。2017。TensorLog：深度学习遇上概率数据库。arXiv：1707.05390Andrew Cropper、Sebastijan Dumancic、Richard Evans 和 Stephen H. Muggleton。2022。归纳逻辑编程三十年。机器学习 111，1（2022）。

Scallop：一种用于神经符号编程的语言

Katrin M. Dannert、Erich Grädel、Matthias Naaf 和 Val Tannen。2021。定点逻辑的半环溯源。发表于计算机科学逻辑会议（CSL）。Adnan Darwiche。2011。SDD：命题知识库的一种新型规范表示。发表于国际人工智能联合会议（IJCAI）。Alexey Dosovitskiy、German Ros、Felipe Codevilla、Antonio Lopez 和 Vladlen Koltun。2017。CARLA：一个开放的城市驾驶模拟器。发表于机器人学习会议（CoRL）。Anton Dries、Angelika Kimmig、Wannes Meert、Joris Renkens、Guy Van den Broeck、Jonas Vlasselaer 和 Luc De Raedt。2015。ProbLog2：概率逻辑编程。发表于欧洲机器学习与数据库知识发现会议（ECML PKDD）。Difei Gao、Ruiping Wang、Shiguang Shan 和 Xilin Chen。2019。从两张图到 N 个问题：一个用于视觉与常识组合推理的视觉问答数据集。（2019）。arXiv:1908.02962 Hong Ge、Kai Xu 和 Zoubin Ghahramani。2018。Turing：一种灵活的概率推断语言。发表于人工智能与统计国际会议（AISTATS）。Ross B. Girshick。2015。Fast R-CNN。（2015）。arXiv:1504.08083 Alex Graves、Abdel-rahman Mohamed 和 Geoffrey Hinton。2013。使用深度循环神经网络进行语音识别。发表于IEEE声学、语音与信号处理国际会议（ICASSP）。Todd J. Green、Grigoris Karvounarakis 和 Val Tannen。2007。溯源半环。发表于ACM数据库系统原理研讨会（PODS）。Albert Gu、Karan Goel 和 Christopher Ré。2021。利用结构化状态空间高效建模长序列。（2021）。arXiv:2111.00396 Albert Gu、Isys Johnson、Aman Timalsina、Atri Rudra 和 Christopher Ré。2022。如何训练你的 HiPPO：具有广义正交基投影的状态空间模型。（2022）。arXiv:2206.12037 Sumit Gulwani、Oleksandr Polozov、Rishabh Singh 等。2017。程序合成。《编程语言基础与趋势》4，1-2（2017）。Thomas Hayes、Songyang Zhang、Xi Yin、Guan Pang、Sasha Sheng、Harry Yang、Songwei Ge、Qiyuan Hu 和 Devi Parikh。2022。MUGEN：用于视频、音频和文本多模态理解与生成的试验平台。arXiv:2204.08058 Jiani Huang、Ziyang Li、Binghong Chen、Karan Samel、Mayur Naik、Le Song 和 Xujie Si。2021。Scallop：从概率演绎数据库到可扩展的可微推理。发表于神经信息处理系统大会（NeurIPS）。Drew A Hudson 和 Christopher D Manning。2019。GQA：一个用于真实场景视觉推理和组合式问答的新数据集。发表于IEEE计算机视觉与模式识别会议（CVPR）。Louis Jachiet、Pierre Genevès、Nils Gesbert 和 Nabil Layaida。2020。递归关系查询的优化：图查询中的应用。发表于ACM数据管理国际会议（SIGMOD）。Justin Johnson、Bharath Hariharan、Laurens Van Der Maaten、Li Fei-Fei、C Lawrence Zitnick 和 Ross Girshick。2017。CLEVR：用于组合语言与基础视觉推理的诊断数据集。发表于IEEE计算机视觉与模式识别会议（CVPR）论文集。2901–2910。Justin Johnson、Ranjay Krishna、Michael Stark、Li-Jia Li、David Shamma、Michael Bernstein 和 Li Fei-Fei。2015。使用场景图进行图像检索。发表于IEEE计算机视觉与模式识别会议（CVPR）。Mahmoud Abo Khamis、Hung Q. Ngo、Reinhard Pichler、Dan Suciu 和 Yisu Remy Wang。2021。Datalog 在（前）半环上的收敛。arXiv:2105.14435 Y. Lecun、L. Bottou、Y. Bengio 和 P. Haffner。1998。应用于文档识别的基于梯度的学习。《IEEE学报》86，11（1998）。Aitor Lewkowycz、Anders Andreassen、David Dohan、Ethan Dyer、Henryk Michalewski、Vinay Ramasesh、Ambrose Slone、Cem Anil、Imanol Schlag、Theo Gutman-Solo 等。2022。使用语言模型解决定量推理问题。（2022）。Qing Li、Siyuan Huang、Yining Hong、Yixin Chen、Ying Nian Wu 和 Song-Chun Zhu。2020。通过整合神经感知、语法解析和符号推理实现闭环神经符号学习。发表于国际机器学习会议（ICML）。Yuhong Li、Tianle Cai、Yi Zhang、Deming Chen 和 Debadeepta Dey。2022。是什么让卷积模型擅长长序列建模？（2022）。arXiv:2210.09298 Ziyang Li、Jiani Huang 和 Mayur Naik。2023。Scallop：一种神经符号编程语言。https://doi.org/10. 5281/zenodo.7709690 Yinhan Liu、Myle Ott、Naman Goyal、Jingfei Du、Mandar Joshi、Danqi Chen、Omer Levy、Mike Lewis、Luke Zettlemoyer 和 Veselin Stoyanov。2019。RoBERTa：一种稳健优化的 BERT 预训练方法。（2019）。arXiv:1907.11692 Robin Manhaeve、Sebastijan Dumancic、Angelika Kimmig、Thomas Demeester 和 Luc De Raedt。2021。DeepProbLog 中的神经概率逻辑编程。《人工智能》298（2021）。

Jiayuan Mao、Chuang Gan、Pushmeet Kohli、Joshua B Tenenbaum 和 Jiajun Wu。2019。《神经符号概念学习器：从自然监督中解读场景、词语和句子》。（2019）。arXiv:1904.12584 Pasquale Minervini、Sebastian Riedel、Pontus Stenetorp、Edward Grefenstette 和 Tim Rocktäschel。2020。《端到端可微证明中的推理策略学习》。载于国际机器学习会议（ICML）。Volodymyr Mnih、Koray Kavukcuoglu、David Silver、Andrei A Rusu、Joel Veness、Marc G Bellemare、Alex Graves、Martin Riedmiller、Andreas K Fidjeland、Georg Ostrovski 等。2015。《通过深度强化学习实现人类水平的控制》。Nature 518, 7540（2015）。Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。2019。《PyTorch：一种命令式风格、高性能的深度学习库》。载于神经信息处理系统大会（NeurIPS）。Pranav Rajpurkar、Jian Zhang、Konstantin Lopyrev 和 Percy Liang。2016。《SQuAD：用于文本机器理解的 100,000+ 个问题》。载于自然语言处理经验方法大会（EMNLP）。Victor Sanh、Lysandre Debut、Julien Chaumond 和 Thomas Wolf。2019。《DistilBERT，BERT 的蒸馏版本：更小、更快、更便宜、更轻量》。（2019）。arXiv:1910.01108 Bernhard Scholz、Herbert Jordan、Pavle Subotić 和 Till Westmann。2016。《关于 Datalog 中快速的大规模程序分析》。载于编译器构造国际会议（CC）。Ameesh Shah、Eric Zhan、Jennifer Sun、Abhinav Verma、Yisong Yue 和 Swarat Chaudhuri。2020。《利用可采纳的神经启发式方法学习可微程序》。载于神经信息处理系统大会（NeurIPS）。Xujie Si、Mukund Raghothaman、Kihong Heo 和 Mayur Naik。2019。《使用数值松弛合成 Datalog 程序》。载于国际人工智能联合会议（IJCAI）。Koustuv Sinha、Shagun Sodhani、Jin Dong、Joelle Pineau 和 William L. Hamilton。2019。《CLUTRR：文本归纳推理诊断基准》。（2019）。arXiv:1908.06177 Hao Tan 和 Mohit Bansal。2019。《LXMERT：从 Transformer 学习跨模态编码器表示》。（2019）。arXiv:1908.07490 Yi Tay、Mostafa Dehghani、Samira Abnar、Yikang Shen、Dara Bahri、Philip Pham、Jinfeng Rao、Liu Yang、Sebastian Ruder 和 Donald Metzler。2020。《长程竞技场：高效 Transformer 的基准测试》。（2020）。arXiv:2011.04006 Jan-Willem van de Meent、Brooks Paige、Hongseok Yang 和 Frank Wood。2018。《概率程序设计入门》。arXiv:1809.10756 Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。2017。《注意力就是一切》。载于神经信息处理系统大会（NeurIPS）。Petar Veličković、Guillem Cucurull、Arantxa Casanova、Adriana Romero、Pietro Lio 和 Yoshua Bengio。2017。《图注意力网络》。（2017）。arXiv:1710.10903 Po-Wei Wang、Priya L. Donti、Bryan Wilder 和 Zico Kolter。2019。《SATNet：使用可微满足性求解器连接深度学习与逻辑推理》。载于国际机器学习会议（ICML）。Saining Xie、Chen Sun、Jonathan Huang、Zhuowen Tu 和 Kevin Murphy。2018。《重新思考时空特征学习：视频分类中的速度—准确率权衡》。载于欧洲计算机视觉会议（ECCV）。Jingyi Xu、Zilu Zhang、Tal Friedman、Yitao Liang 和 Guy Van den Broeck。2018。《结合符号知识的深度学习语义损失函数》。载于国际机器学习会议（ICML）。Ziwei Xu、Yogesh S Rawat、Yongkang Wong、Mohan Kankanhalli 和 Mubarak Shah。2022。《别把麦片倒进咖啡里：用于时间动作分割的可微时序逻辑》。载于神经信息处理系统大会（NeurIPS）。Kexin Yi、Jiajun Wu、Chuang Gan、Antonio Torralba、Pushmeet Kohli 和 Josh Tenenbaum。2018。《神经符号视觉问答：将推理与视觉和语言理解解耦》。载于神经信息处理系统大会（NeurIPS）。Eric Zelikman、Qian Huang、Gabriel Poesia、Noah D. Goodman 和 Nick Haber。2023。《Parsel：用于语言模型算法推理的（解）复合框架》。arXiv:2212.10561

$item$:= @$attrName$($attrArg$*)def def: = $importDef$ | $typeDef$ | $constDef$ | $relaDef$ | $queryDef$ $importDef$:= import $file$ $typeDef$:= type $typeName$ = $aliasName$ | type $typeName$ <: $superTypeName$ | type $relationName$([$name$:]$type$*) $constDef$:= const $constName$[: $type$] = $constant$ $relaDef$:= 关系 [tag ::] $relaName$($expr$*) | 关系 $relaName$ = {$taggedTuple$*} | 关系 [tag ::] $atom$:- $formula$ $taggedTuple$:= [tag::] ($constant$*)$atom$:= $relaName$($expr$*) $formula$:= $formula$ 且 $formula$ | $formula$ 或 $formula$ | $formula$ 蕴含 $formula$ | $atom$ | ¬$atom$ | $constraint$ | $reduce$ $reduce$:= var* = $aggregator$(var*: $formula$ [其中 var*: $formula$]) | var* = $sampler$(var*: $formula$ [其中 var*: $formula$]) $constraint$:= $expr$ $type$:= u8 | u16 | u32 | u64 | usize | i8 | i16 | i32 | i64 | isize | f32 | f64 | 布尔型 | 字符 | 字符串 $expr$: = $expr$ $binOp$ $expr$ | $unaryOp$ $expr$ | $expr$ 作为 $type$ | 如果 $expr$ 那么 $expr$ 否则 $expr$ | $$$func$($expr$*) | $variable$ | $constant$ $binOp$:= + | - | * | / | % | && | || | == | != | < | <= | > | >= $unaryOp$:= ! | - $aggregator$: = 最小值 | 最大值 | argmin<var*> | argmax<var*> | 计数 | 求和 | 乘积 | 存在量词 | 全称量词 $sampler$:= 前<K> | 分类分布<K> | 均匀分布<K>

图 20. 语言语法（var： $variable$ 和 f： $formula$）。

（标签）t ∈ T（假）$\mathbf{0}$ ∈ T（真）$\mathbf{1}$ ∈ T（析取）$\oplus$: $T \times T \to T$（合取）$\otimes$: $T \times T \to T$（否定）$\ominus$: $T \to T$（饱和）$\Subset$: $T \times T \to$ 布尔值（提前移除）丢弃: $T \to$ 布尔值（权重）权重: $T \to \mathbb{R}$

图 21. 溯源结构的完整接口。

SCALLOP 语言语法

在本附录中，我们给出了图20中 Scallop 完整语言的抽象语法。

B 推理框架

在本附录中，我们介绍对溯源框架的扩展，并说明 fullSclRam 提供的其他功能。

B.1 溯源扩展

我们在图中展示了完整的溯源接口，其中包括两项额外功能：提前移除和用于采样的权重。21对于提前移除，我们引入了一个函数 *discard*，它根据标签返回一个布尔值。当返回真时，与该标签关联的事实将从计算中提前移除。对于采样，我们允许每个溯源实现一个函数 *weight*，该函数返回一个位于 $\mathbb{R}$ 中的实数，表示该标签的权重。稍后介绍完整的SclRam语法和语义时，我们将用到这些功能。

（谓词） p（聚合器）$g ::=$ 计数 | 求和 | 乘积 | 存在量词 |最大值 |最小值 |argmax | argmin（采样器）$\mu ::=$ top<K> | 类别<K> |均匀<K>^（表达式）$e ::= \emptyset \mid p \mid \mathbb{1}(e) \mid \mathbb{0}(e) \mid \gamma_g(e) \mid \hat{\gamma}_g(e_1, e_2) \mid \psi_\mu(e) \mid \psi_\mu(e_1, e_2) \mid \pi_\alpha(e) \mid \sigma_\beta(e) \mid e_1 \cup e_2 \mid e_1 \times e_2 \mid e_1 \bowtie e_2 \mid e_1 \cap e_2 \mid e_1 - e_2$（规则）$r ::= p \leftarrow e$（层）$s ::= \{r_1, \ldots, r_n\}$（程序）$s ::= s_1; \ldots; s_n$

图 22. SclRam 的完整抽象语法。

B.2 SclRam 的抽象语法

我们重新审视 SclRam 的抽象语法，并添加以下扩展：

(1) 空集（$\emptyset$）、单次覆盖（$\mathbb{1}(e)$）和零次覆盖（$\mathbb{0}(e)$），(2) 交集（$\cap$）、自然连接（$\bowtie$）和反连接（$\triangleright$），(3) 分组聚合（$\hat{\gamma}_g(e_1, e_2)$），其中 e_1 是用于查找分组的表达式，e_2 是要聚合的主表达式，并根据计算 e_1 得到的分组进行条件筛选，(4) 采样（$\psi_\mu(e)$），其中 μ 表示一个采样器，^ (5) 分组采样（$\psi_\mu(e_1, e_2)$）；它是一种采样，与聚合类似，其中 e_1 是用于查找分组的表达式。

备注。 在带标签语义下，One-overwrite $\mathbb{1}(e)$ 对定义魔法集转换尤其有用。覆盖 **1** 会使魔法集谓词中的元组成为纯需求事实，用于优化，而不受某些给定标签的条件约束。

B.3 SclRam 的操作语义

我们在图中展示了完整语义。23 和图中。24，并加入了分组、采样和提前移除。请注意，采样器 $\mu : \mathcal{U}_{\mathbb{R}} \to \mathcal{U}$ 接收带有权重标签的输入事实，并根据权重进行采样。提前移除发生在每次规则更新前的规范化阶段。

B.4 溯源结构

我们在以下各节中为每个所介绍的溯源结构提供详细的运行时间分析。此外，我们讨论扩展的溯源 `diff-top-k-proofs-me` 其可以处理互斥事实。

B.4.1 diff-max-min-prob。

运行时间分析。 我们在表6中展示了最大-最小-概率溯源结构中每个运算符的时间复杂度。接下来，我们在图1中介绍优化后的计数算法，该算法对一组最大-最小-概率 带标签元组执行计数。请注意，为简单起见，我们只展示了 `mmp` 的算法。但它可以轻松扩展到 `dmmp`。很容易证明，其运行时间复杂度为 $O(n \log(n))$。

B.4.2 差分加乘概率。

运行时间分析。 我们在表中展示差分加乘概率溯源结构的时间复杂度 7。我们将输入中的变量总数记为 n。由于我们是在对偶数上进行计算，因此时间复杂度由维度为 n的向量的梯度计算决定。

Expression semantics $\alpha : \mathbb{U} \to \mathbb{U}, \quad \beta : \mathbb{U} \to \text{Bool}, \quad g : \mathcal{U} \to \mathcal{U}, \quad \mu : \mathcal{U}_{\mathbb{R}} \to \mathcal{U}, \quad [\![e]\!] : \mathcal{F}_T \to \mathcal{U}_T$

$$\frac{}{[\![\varnothing]\!](F_T) = \emptyset}\ (\textsc{Empty Set}) \qquad \frac{t :: p(u) \in F_T}{t :: u \in [\![p]\!](F_T)}\ (\textsc{Predicate})$$

$$\frac{t :: u \in [\![e]\!](F_T)}{\mathbf{0} :: u \in [\![\emptyset(e)]\!](F_T)}\ (\mathbf{0}\text{-}\textsc{Overwrite}) \qquad \frac{t :: u \in [\![e]\!](F_T)}{\mathbf{1} :: u \in [\![\mathbb{1}(e)]\!](F_T)}\ (\mathbf{1}\text{-}\textsc{Overwrite})$$

$$\frac{t :: u \in [\![e]\!](F_T) \qquad \beta(u) = \text{true}}{t :: u \in [\![\sigma_\beta(e)]\!](F_T)}\ (\textsc{Select}) \qquad \frac{t :: u \in [\![e]\!](F_T) \qquad u' = \alpha(u)}{t :: u' \in [\![\pi_\alpha(e)]\!](F_T)}\ (\textsc{Project})$$

$$\frac{t :: u \in [\![e_1]\!](F_T) \cup [\![e_2]\!](F_T)}{t :: u \in [\![e_1 \cup e_2]\!](F_T)}\ (\textsc{Union}) \qquad \frac{t_1 :: u_1 \in [\![e_1]\!](F_T) \qquad t_2 :: u_2 \in [\![e_2]\!](F_T)}{(t_1 \otimes t_2) :: (u_1, u_2) \in [\![e_1 \times e_2]\!](F_T)}\ (\textsc{Product})$$

$$\frac{t_1 :: u \in [\![e_1]\!](F_T) \qquad t_2 :: u \in [\![e_2]\!](F_T)}{(t_1 \otimes t_2) :: u \in [\![e_1 \cap e_2]\!](F_T)}\ (\textsc{Intersect})$$

$$\frac{t_1 :: (u, u_1) \in [\![e_1]\!](F_T) \qquad t_2 :: (u, u_2) \in [\![e_2]\!](F_T)}{(t_1 \otimes t_2) :: (u, u_1, u_2) \in [\![e_1 \bowtie e_2]\!](F_T)}\ (\textsc{Natural-Join})$$

$$\frac{t :: u \in [\![e_1]\!](F_T) \qquad [\![e_2]\!](F_T) \not\vdash u}{t :: u \in [\![e_1 - e_2]\!](F_T)}\ (\textsc{Diff-1}) \qquad \frac{t_1 :: u \in [\![e_1]\!](F_T) \qquad t_2 :: u \in [\![e_2]\!](F_T)}{(t_1 \otimes (\ominus\, t_2)) :: u \in [\![e_1 - e_2]\!](F_T)}\ (\textsc{Diff-2})$$

$$\frac{t :: (u_1, u_2) \in [\![e_1]\!](F_T) \qquad [\![e_2]\!](F_T) \not\vdash u_1}{t :: (u_1, u_2) \in [\![e_1 \triangleright e_2]\!](F_T)}\ (\textsc{AntiJoin-1})$$

$$\frac{t_1 :: (u_1, u_2) \in [\![e_1]\!](F_T) \qquad t_2 :: u_1 \in [\![e_2]\!](F_T)}{(t_1 \otimes (\ominus\, t_2)) :: (u_1, u_2) \in [\![e_1 \triangleright e_2]\!](F_T)}\ (\textsc{AntiJoin-2})$$

$$\frac{X_T \subseteq [\![e]\!](F_T) \qquad \{t_i :: u_i\}_{i=1}^n = X_T \qquad \{\overline{t}_j :: \overline{u}_j\}_{j=1}^m = [\![e]\!](F_T) - X_T \qquad u \in g(\{u_i\}_{i=1}^n)}{(\bigotimes_{i=1}^n t_i) \otimes (\bigotimes_{j=1}^m (\ominus\, \overline{t}_j)) :: u \in [\![\gamma_g(e)]\!](F_T)}\ (\textsc{Aggregate})$$

$$\frac{\begin{array}{c} t' :: (u_g, u'_g) \in [\![e_1]\!](F_T) \qquad X_T \subseteq [\![e_2]\!](F_T) \\ \{t_i :: (u_g, u_i)\}_{i=1}^n = X_T \qquad \{\overline{t}_j :: (u_g, \overline{u}_j)\}_{j=1}^m = [\![e_2]\!](F_T) - X_T \qquad u \in g(\{u_i\}_{i=1}^n) \end{array}}{t' \otimes (\bigotimes_{i=1}^n t_i) \otimes (\bigotimes_{j=1}^m (\ominus\, \overline{t}_j)) :: (u_g, u'_g, u) \in [\![\hat{\gamma}_g(e_1, e_2)]\!](F_T)}\ (\textsc{Group-By Aggregate})$$

$$\frac{\{t_i :: u_i\}_{i=1}^n = [\![e]\!](F_T) \qquad u_j \in \mu(\{\text{weight}(t_i) :: u_i\}_{i=1}^n)}{t_j :: u_j \in [\![\psi_\mu(e)]\!](F_T)}\ (\textsc{Sample})$$

$$\frac{t' :: (u_g, u'_g) \in [\![e_1]\!](F_T) \qquad \{t_i :: (u_g, u_i)\}_{i=1}^n \subseteq [\![e]\!](F_T) \qquad u_j \in \mu(\{\text{weight}(t_i) :: u_i\}_{i=1}^n)}{(t' \otimes t_j) :: (u_g, u'_g, u_j) \in [\![\hat{\psi}_\mu(e_1, e_2)]\!](F_T)}\ (\textsc{Group-By Sample})$$

图 23. SclRam 的操作语义。

算法 1： 对最大-最小概率标记元组进行计数

数据： $U_{\text{mmp}} = \{t_1 :: u_1, t_2 :: u_2, \ldots, t_n :: u_n\}$: $\mathbb{U}_{\text{mmp}}$，带标签元组集合
结果： U'_{mmp}: $\mathcal{U}_{\text{mmp}}$

/* 按标签从小到大对所有正元组排序。O(nlog(n)) */
$\mathbf{t}^{\text{pos}} = sorted([t_i \mid i = 1 \ldots n])$;
$\mathbf{t}^{\text{neg}} = [1 - t^{\text{pos}}_{n-i+1} \mid i = 1 \ldots n]$;
/* 遍历正标签和负标签之间的所有可能划分。O(n) */
$U'_{\text{mmp}} = \{t^{\text{neg}}_n :: 0, t^{\text{pos}}_1 :: n\}$;
对于 $i = 1 \ldots (n-1)$，**执行**
    将最小值相加$(t^{\text{pos}}_{i+1}, t^{\text{neg}}_i) :: (n-i)$至 U'_{mmp};
返回 U'_{mmp}

规则语义

$$\boxed{.: \mathrm{U}_T \to \mathrm{U}_T, \qquad [\![r]\!]\colon \mathrm{F}_T \to \mathrm{F}_T}$$

$$\frac{t_1 :: u, \ldots, t_n :: u \text{ are all tagged-tuples in } U_T \qquad t = \bigoplus_{i=1}^{n} t_i \qquad \mathrm{discard}(t) = \mathrm{false}}{t :: u \in \langle U_T \rangle} \text{ (NORMALIZE)}$$

$$\frac{t^{\mathsf{old}} :: u \in [\![p]\!](F_T) \qquad \langle [\![e]\!](F_T) \rangle \not\vdash u}{t^{\mathsf{old}} :: p(u) \in [\![p \leftarrow e]\!](F_T)} \text{ (RULE-1)} \qquad \frac{t^{\mathsf{new}} :: u \in \langle [\![e]\!](F_T) \rangle \qquad [\![p]\!](F_T) \not\vdash u}{t^{\mathsf{new}} :: p(u) \in [\![p \leftarrow e]\!](F_T)} \text{ (RULE-2)}$$

$$\frac{t^{\mathsf{old}} :: u \in [\![p]\!](F_T) \qquad t^{\mathsf{new}} :: u \in \langle [\![e]\!](F_T) \rangle}{(t^{\mathsf{old}} \oplus t^{\mathsf{new}}) :: p(u) \in [\![p \leftarrow e]\!](F_T)} \text{ (RULE-3)}$$

层与程序语义

$$\boxed{\mathrm{lfp}^\circ : (\mathrm{F}_T \to \mathrm{F}_T) \to (\mathrm{F}_T \to \mathrm{F}_T), \qquad [\![s]\!], [\![\bar{s}]\!] : \mathrm{F}_T \to \mathrm{F}_T}$$

（饱和） $F_T^{\mathsf{old}} \triangleq F\mathrm{new}_T$当且仅当 $\forall t^{\mathsf{new}} ::\ p(u) \in F\mathrm{new}_T$， $\exists t^{\mathsf{old}} ::\ p(u) \in F\mathrm{old}_T$，使得 $t^{\mathsf{old}} = t^{\mathsf{new}}$（不动点）$\mathrm{lfp}^\circ(h) = h \circ \cdots \circ h = h^n$如果存在最小的 $n > 0$，使得 $h^n(F_T) \triangleq h^{n+1}(F_T)$（层） $\mathrm{lfp}^\circ(\lambda F_T.(F_T - \bigcup_{p \in P_s} F_T[p]) \cup (\bigcup_{r \in s} [\![r]\!](F_T)))$（程序）$[\![s]\!] = [\![s_n]\!] \circ \cdots \circ [\![s_1]\!]$，其中 $s = s_1; \ldots; s_n$.

图 24. SclRam 的操作语义（续）

操作	算法	时间复杂度
$t_1 \oplus t_2$	最大值(t_1, t_2)	$O(1)$
$t_1 \otimes t_2$	$\min(t_1, t_2)$	$O(1)$
$\ominus t$	$\hat{1} - t$	$O(1)$
$t_1 \overset{?}{=} t_2$	$t_1 == t_2$	$O(1)$
$\rho(t)$	t	$O(1)$

表6. `diff-max-min-prob` 溯源结构运行时间分析

操作	算法	时间复杂度
$t_1 \oplus t_2$	$clamp(t_1 + t_2)$	$O(n)$
$t_1 \otimes t_2$	$t_1 \cdot t_2$	$O(n)$
$\ominus t$	$\hat{1} - t$	$O(n)$
$t_1 \overset{?}{=} t_2$	true	$O(1)$
$\rho(t)$	t	$O(1)$

表7. 差分加乘概率 溯源结构运行时间分析

操作	算法	时间复杂度
$t_1 \oplus t_2$	$\mathrm{top}_k(t_1 \cup t_2)$	$O(nk)$
$t_1 \otimes t_2$	$\mathrm{top}_k(\{\ \eta \mid (\eta_1, \eta_2)\ \in t_1 \times t_2, \eta = \eta_1 \cup \eta_2, \eta$无冲突 $\})$	$O(n^2k^2)$
$\ominus t$	$\mathrm{top}_k(\mathrm{cnf2dnf}(\{\{\neg \nu \mid \nu \in \eta\}\ \mid \eta \in t\}$	$O(2^n)$
$t_1 \overset{?}{=} t_2$	$t_1 == t_2$	$O(nk)$
$\rho(t)$	加权模型计数(t, Γ)	$O(2^n)$

表 8。`diff-top-k-proofs` 溯源结构运行时间分析。我们假设每个标签的项数为 $O(n)$，子句数为 $O(k)$。请注意，我们展示的是 cnf2dnf 和加权模型计数算法的朴素实现的时间复杂度。

B.4.3 diff-top-k-proofs。

运行时间分析。 我们在表格 8 中展示了 diff-top-k-proofs 溯源结构的时间复杂度。请注意，我们展示的是 cnf2dnf 函数的朴素实现以及加权模型计数过程的运行时间分析。优化留待未来工作。

B.4.4 diff-top-k-proofs-me.

支持互斥事实。我们保留原有的`diff-top-k-proofs`，并扩展它以支持互斥标签（因此得名`-me`）。大多数操作保持不变，我们首先为其提供一个新的输入标签空间：

$$I_{\texttt{dtkp-me}} = [0, 1] \times \texttt{Option<}\mathbb{N}\texttt{>}.$$

概率保持不变，第二部分是一个可选标识符，用于标识某个互斥关系。不提供该标识符意味着该事实不属于任何互斥关系。如果提供了，我们将这样的标签记作

$$t_i = (r_i, \mathsf{Some}(\mathbf{me}_i))$$

示例 *B.1.* 考虑 sum2 任务，我们有两个分布，定义如下：

```
rel digit = {0.90::(A, 0); 0.01::(A, 1); ...; 0.02::(A, 9)}
rel digit = {0.01::(B, 0); 0.91::(B, 1); ...; 0.01::(B, 9)}
```

那么，输入事实为：

$$[\![\texttt{digit}]\!](F_{\texttt{Option<I>}}) = \left\{ \begin{array}{lll} (0.90, \mathsf{Some}(1)) & :: & (A, 0), \\ (0.01, \mathsf{Some}(1)) & :: & (A, 1), \\ & \vdots & \\ (0.02, \mathsf{Some}(1)) & :: & (A, 9), \\ \hline (0.01, \mathsf{Some}(2)) & :: & (B, 0), \\ (0.91, \mathsf{Some}(2)) & :: & (B, 1), \\ & \vdots & \\ (0.01, \mathsf{Some}(2)) & :: & (B, 9) \end{array} \right\}.$$

有两个**互斥**，分别是 1 和 2，这表示存在两组互斥。digit A 的前 10 个事实属于互斥组 1，接下来的 10 个 digit B 事实属于互斥组 2。

实际执行时，两个具有相同**互斥**（$\textbf{互斥}_i = \textbf{互斥}_j$）的事实 f_i 和 f_j 不能同时出现在同一个证明中。这一限制通过dtkp的 $\wedge_k$ 和 $\neg_k$ 操作期间的冲突检查来强制执行。如果证明违反了互斥假设，就会被视为无效证明并丢弃。例如，执行以下规则

```
rel not_possible() = digit(A, x), digit(A, y), x != y
```

由于这一机制，不会产生有效输出，也就是说，与not_possible()关联的内部标签将为 $\emptyset$，并会在计算过程中被提前移除。

C 评估

C.1 MNIST-R

实验设置。MNIST数据集总共[Lecun等人。 1998]的数据集包含6万个手写数字。根据每项任务所使用的数字数量，我们进一步将MNIST数据集划分为下游任务数据点。例如，对于sum-2，我们有3万个训练对；对于sum-3，我们有2万个训练对；对于sum-4，我们有1.5万个训练对。由于计数任务消耗8 张图像作为一个数据点，我们有7.5万个计数任务训练对。测试集和训练集的划分遵循原始MNIST任务设置。我们的神经模块由一个2层卷积神经网络和一个2层多层感知机构成，隐藏层大小为1024。我们在训练数据集上训练Scallop模块10 个周期，学习率设置为0。001，批量大小为64，并使用二元交叉熵损失闭合训练循环。

Scallop 代码。图中展示了 MNIST-R 基准测试所用的 Scallop 代码。25。

C.2 手写公式

```
类型 数字_1(u32), 数字_2(u32)关系 求和_2(a + b) =
数字_1(a), 数字_2(b)
```

(a) 用于 MNIST sum-2 的 Scallop代码。

```
类型数字_1(u32), 数字_2(u32), 数字_3(u32)关系求和_3(a + b + c) =数
字_1(a), 数字_2(b), 数字_3(c)
```

(b) 用于 MNIST sum-3 的 Scallop代码。 类型 数字_1(u32), 数字_2(u32), 数字_3(u32), 数字_4(u32) 关系 求和_4(a + b + c + d) =数字_1(a), 数字_2(b), 数字_3(c), 数字_4(d) (c) 用于 MNIST sum-4 的 Scallop代码。 类型 数字_1(u32), 数字_2(u32) 关系 小于_（a <b） =数字_1(a), 数字_2(b) (d) 用于 MNIST 小于 的 Scallop代码。 类型 数字(u32) 关系 非_3_或_4() = 非 数字 (3) 且 非 数字 (4) (e) 用于 MNIST 非3或4 的 Scallop代码。 类型 数字(数字_id : u32 , 数字_value : u32) 关系 计数_3(x) :- x =计数 (o: 数字 (o, 3))

(f) MNIST 计数-3 的 Scallop代码。

```
类型 数字(数字_标识符： u32， 数字_值： u32)关系 计数_3_或_4(x) =x =计数
(o: 数字 (o, 3) 或 数字 (o, 4))
```

(g) MNIST 计数-3-或-4 的 Scallop代码。

图 25. MNIST-R 的 Scallop代码。

实验设置。 HWF [Li 等人。 2020] 数据集接收一个图像序列，其中每个图像可以是数字或算术符号，目标是计算公式的结果。我们对基于MNIST 的任务采用相同的基于 CNN 的符号识别网络，唯一的区别是该模型现在执行 14 类分类。然后，我们生成一个包含两个关系的概率数据库：1) 长度(usize)，一个存储公式长度的非概率关系；以及2) 符号(id: usize, σ:字符串)，一个将索引 ID 映射到符号的概率关系 $\sigma \in \Sigma$。对于推理组件，我们在 Scallop 中编写了一个公式解析器（如26所示）。此外，Scallop还用于根据解析后的抽象语法树（AST）执行计算，以获得最终的有理数输出。训练循环在末尾通过二元交叉熵损失闭合，形成一个端到端训练流程，使梯度能够从输出一路反向传播到符号识别网络。为了在枚举每个可能的解析树时裁剪庞大的输出域，我们仅采样一部分符号预测并将其发送至Scallop。训练数据集包含10K 个图像-答案对。我们将批量大小设为 16，学习率设为 0.0001，训练轮次设为 100。为了提高学习效率，我们采用一种采样策略，仅保留每个图像分类任务中7 个最可能的分类结果，并将其发送至 Scallop。

```
1 type符号 ( index :usize , 符号:字符串 ) 2 type 长度(n: usize) 3 4 关系 digit = {"0","1","2","
3","4", "5", "6", "7", "8", "9"} 5 6 type term( value: 32 位浮点数, begin : usize , end: usize )7 关系
term(x as 32 位浮点数 , b, b + 1) =符号 (b,x) and digit (x) 8 9type mult_div ( value: 32 位浮点
数 ,begin :usize , end:usize)10 关系 mult_div (x, b,r) =term(x, b, r) 11关系 mult_div(x * y, b,
e) =mult_div (x, b, m) and 符号 (m,"*") and 12term(y, m + 1,e)13关系mult_div (x/ y, b, e) =
mult_div (x, b, m) and 符号 (m,"/") and 14 term(y, m + 1, e) 1516 type add_minus ( value:
32 位浮点数 ,begin :usize , end: usize )17关系 add_minus (x, b, r) =mult_div(x, b,r) 18 关系
add_minus (x +y, b, e) =add_minus (x, b, m)and符号 (m,"+") and19mult_div (y, m + 1, e)
20 关系 add_minus (x- y, b, e) =add_minus (x, b,m)and 符号(m, "-") and21mult_div (y, m + 1,
e) 2223 type result (value : 32 位浮点数 ) 24 关系 result(y) =add_minus (y, 0, l) and 长度 (l)
25 26 查询 result
```

图 26. HWF 的 Scallop代码。

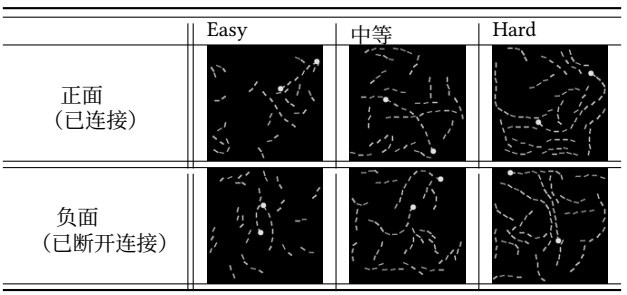


图 27. 每个难度等级的正例和反例。

C.3 Pathfinder

问题定义。 Pathfinder 是一项最初旨在测试神经网络长程推理能力的任务。它被纳入名为长程竞技场（LRA）的基准测试中 [Tay 等人 2020] 用以评估神经网络，或更具体地说，Transformer 的长程推理能力 [Vaswani 等人 2017]。在这项任务中，我们会获得一些图像：背景为深色，其中有两个白色点，二者之间可能由虚线连接。模型需要对这两个点是否连通作出二元预测。该数据集有多个版本：一个仅包含 32×32 图像（Path），另一个包含 128×128 （Path-X）。这两个版本

```
// 神经网络的输入 类型 dash(i8, i8)   类型 dot(i8)   // 连通性检查
rel path(x, y) = dash(x, y) 或 path(x, z) 且 dash(z, y)
rel connected() = dot(x), dot(y), path(x, y), x != y
```

图 28. Pathfinder 的 Scallop 代码。

分为简单、普通和困难 3 种难度，具体取决于连接两个点的虚线长度——虚线越长，难度越大。表格27 展示了这 3 种难度，每种难度各包含 Path-X 中一个正样本和一个负样本。

架构。 若要将符号化推向极致，可以用单个符号表示每一小段虚线和每一个点。然而，这样一来，感知网络（可能是卷积神经网络）就需要精确地分割图像，以提取所有虚线和点作为实体。同时，在处理单个连通关系时，推理模块也会承担额外负担。因此，我们采用一种简单的基于网格的连通图：每个节点表示一个概念上的“点”，每条边表示一段概念上的“虚线”。我们让神经网络接收图像，并输出一个特征向量，其维度等于点的数量加上边的数量。随后，将该向量中的每个 logit 视为相应“点”或“虚线”存在的概率。我们为此任务使用的 Scallop 程序（如图28所示）只是一个传递闭包，用于检查是否存在两个由虚线连接的不同点。

实验设置。 PATH 和 PATH-X 均有 60 万个训练数据点，分别是尺寸为 32×32 和 128×128 像素的图像，y 标签表示图像上两个点之间是否存在一条连通路径。我们通过一个 4 层卷积神经网络识别“虚线”和“点”关系，并通过一个 2 层多层感知机分类器识别局部连通性。我们将学习率设为 0.0001，批量大小设为 64，训练轮数设为 100。

C.4 吃豆人迷宫

吃豆人迷宫是一种基于强化学习的规划应用，旨在引导角色抵达旗帜，同时避免碰到任何敌人。我们在图 29 中展示了用于此任务的 Scallop 代码。除第 2 节所示代码外，我们还添加了额外的“完整性约束违反”项，这些项需要在学习过程中最小化。

实验设置。 我们将迷宫设为 5×5 矩阵，并将竞技场中存在的敌人数量上限设为 5。在训练过程中，我们将批量大小设为 24，经验回放缓冲区大小设为 3000，每个回合的动作数上限设为 30。我们每 10 轮更新一次目标网络，将探索率设为 0.9，衰减系数设为 0.98，学习率设为 0.0001。

C.5 CLUTRR

借助 CLUTRR [Sinha 等人 2019] 我们进入了亲属关系推理这一 NLP 领域。CLUTRR 数据集中的每个数据点都由一段自然语言文本、一个查询及其答案组成。文本讲述一个家庭的日常活动；查询是由两个名字组成的元组，表示我们希望预测其关系的目标人物对；答案则属于 20 种基本关系之一，例如“母亲”“叔叔”和“儿媳”。该数据集还进一步分为

```
1 // 来自神经网络的输入 2 type grid_node (x: usize , y: usize ) // 每个网格单元为 0.99 3 type actor (
x: usize , y: usize ) 4 type goal(x: usize , y: usize ) 5 type enemy (x: usize , y: usize ) 6 7 // 基本连通性 8
const UP = 0, RIGHT = 1, DOWN = 2, LEFT = 39 rel safe_node (x, y) = grid_node (x, y), not
enemy (x, y) 10 rel edge(x, y, x, yp , UP) = safe_node (x, y), safe_node (x, yp), yp == y + 111 rel
edge(x, y, xp , y, RIGHT ) = safe_node (x, y), safe_node (xp , y), xp == x + 112 rel edge(x, y, x, yp ,
DOWN) = safe_node (x, y), safe_node (x, yp), yp == y - 1 13 rel edge(x, y, xp , y, LEFT) = safe_node
(x, y), safe_node (xp , y), xp == x - 1 14 15 // 获取下一个位置 16 rel next_pos (xp , yp , a) = actor (x, y),
edge(x, y, xp , yp , a) 17 18 // 连通路径；将以路径上没有敌人为条件 19 rel path(x, y, x, y) = next_pos (x,
y, _) 20 rel path(x1 , y1 , x3 , y3) = path(x1 , y1 , x2 , y2), edge(x2 , y2 , x3 , y3 , _) 21 22 // 获取下一个动
作 23 rel next_action (a) = next_pos (x, y, a), goal(gx , gy), path(x, y, gx , gy) 24 25 // 约束违反 26 type
too_many_goal (), too_many_actor (), too_many_enemy () 27 rel too_many_goal () = n := count (x, y:
goal(x, y)), n > 128 rel too_many_actor () = n := count (x, y: actor (x, y)), n > 129 rel too_many_
enemy () = n := count (x, y: enemy (x, y)), n > 530 rel violation () = too_many_goal () or too_
many_enemy () or too_many_actor ()
```

图 29. 吃豆人迷宫的 Scallop代码。

```
1 // 定义属于关系类型的亲属关系 2 type Relation =usize 3 4 // 神经网络的输入 5 type
question (sub: String , obj: String ) 6 type kinship (rela: Relation , sub: String , obj:
String ) 7 type composition (Relation , Relation , Relation ) 8 9 // 用于推导最终答案的规则
10 rel kinship (r3 , x, z) = composition (r1 , r2 , r3), kinship (r1 , x, y), 11 kinship (r2 , y,
z), x != z 12 rel answer (r) = question (s, o), kinship (r, s, o)
```

图 30. CLUTRR 的 Scallop代码。

根据事实数量划分为 10 个难度级别（k）推导出最终结果所需。请注意，亲属关系复合关系的知识库（例如“父亲的父亲是祖父”）并未包含在数据集中。因此，我们设计了三个实验，以缓解知识库缺失的问题：a) 使用手动指定的规则进行学习，b) 规则学习，以及 c) 不使用规则进行学习。我们在图中展示了 Scallop 代码。30。

使用人工指定的规则进行学习。 解决缺少所需数据库问题的一个直接策略是手动添加数据库。我们创建了一个外部知识库，其中包含 92 个亲属关系复合三元组和 3 条规则。为了定义如何连接两个已知关系以推导出第三个关系，我们设计了一个高阶谓词 复合关系。例如，“父亲的母亲是祖母”这条规则编码为 复合关系(FATHER, MOTHER, GRAND MOTHER)。将知识库接入推理流程的完整 Scallop 程序如图 30 所示。给定知识库、上下文和问答对后，我们便可以在新数据集上进行学习。由于上下文长度各不相同，我们首先将上下文拆分为多个窗口，以确保感知模型在整个数据集中处理长度相近的上下文。然后，采用 RoBERTa 等大型语言模型 [Liu 等人 2019], 从这些窗口化文本中提取关系。从上下文中提取的亲属关系会存入`kinship(rela, sub, obj)`。例如，`kinship(SON, "Alice", "Bob")` 表示 Bob 是 Alice 的儿子。另一方面，查询则存储在关系 `question(sub, obj)`中。接下来，我们将所有上下文窗口预测出的关系与知识库和查询合并到一个概率数据库中，进行逻辑推理。最后，我们使用二元交叉熵损失比较概率查询结果与真实标注。

规则学习。手动指定 92 条复合关系规则可能会很耗时。使用 Scallop，可以更智能地完成这项工作。由于有 20 种基本亲属关系，因此我们有一个包含 $20^3 = 8$K 种可能复合关系事实的有限空间。我们将所有 8K 条复合关系事实都视为概率性的、需要学习的内容，因此将它们存储在大小为 $20 \times 20 \times 20$ 的张量中。我们可以随机初始化这个张量，但将最大权重设得较低（即 0.1 ）。在训练过程中，我们允许梯度反向传播到这个复合关系张量，使这些权重也能得到学习。在这种情况下，用户无需手动指定任何复合关系事实，所有内容都会即时学习。请注意，8K 条复合关系事实可能会拖慢训练速度。因此，我们使用多项式采样选取 150 条复合关系事实进行训练。测试时，我们只需选取排名前 150 的事实用于推理。

不依赖规则进行学习。 我们还可以联合提取实体关系，并在没有人工指定规则的情况下学习组合规则。这是一项更具挑战性的任务，需要找出组合推断出的关系并得出所需答案的方法，而不依赖中间监督。

实验设置。 我们有 10K 个训练数据点，即 passage、查询和答案组成的三元组。我们使用 RoBERTa 作为骨干语言模型。此外，我们训练了一个基于 2 层多层感知机的关系提取器，它接收三个组成部分的嵌入、passage 的全局嵌入，以及查询主体和客体的嵌入，并返回 21 类分类结果。我们将批量大小设为 32，学习率设为 0。00001，并对模型训练 100 个 epoch。

C.6 Mugen

实验设置。 我们从 Mugen 中抽取 1K 个[Hayes 等人提出的 2022] 视频与文本对作为训练数据集，并另取 1K 个用于测试。我们的神经模块由 S3D 图像嵌入和

```
1 // 神经网络的输入 2 type action (usize , String ) 3 type expr (usize , String ) 4 type expr_start (
usize ) 5 type expr_end ( usize ) 6 type action_start ( usize ) 7 type action_end ( usize ) 8 9 type
match_single (usize , usize , usize ) 10 type match_sub (usize , usize , usize , usize ) 11 12 // 检查文本规
范的某个子部分是否与 13 // 视频内容匹配 14 rel match_single (tid , vid , vid + 1) = expr (tid , a),
action (vid , a) 15 rel match_sub (tid , tid , vid_start , vid_end ) =16 match_single (tid , vid_start ,
vid_end ) 17 rel match_sub (tid , tid , vid_start , vid_end ) =18 match_sub (tid , tid , vid_start , vid_
mid ), 19 match_single (tid , vid_mid , vid_end ) 20 rel match_sub ( tid_start , tid_end , vid_start , vid_
end ) =21 match_sub ( tid_start , tid_end - 1, vid_start , vid_mid ), 22 match_single (tid_end , vid_
mid , vid_end ) 23 24 // 检查整个文本规范是否与视频内容匹配 25 rel match () = expr_start ( tid_start ),
26 expr_end ( tid_end ), action_start ( vid_start ), action_end ( vid_end ), 27 match_sub ( tid_start , ti
d_end , vid_start , vid_end ) 28 29 // 当过多相同的文本表达式连续出现时， 30 // 会发生完整性违规 31
rel too_many_consecutive_expr () = expr (tid , a), 32 expr (tid + 1, a), expr (tid + 2, a), expr (tid
+ 3, a)
```

图31。Mugen 的 Scallop代码。

distillBert文本嵌入。然后，我们将嵌入输入一个两层多层感知机，其隐藏层大小为256。我们在训练数据集上对Scallop模块训练了1000轮，学习率设为0.0001，批次大小为3，并使用二元交叉熵损失完成训练循环。Scallop代码如图所示。31

C.7 CLEVR

CLEVR [Johnson 等人 2017] 代表 **组**合式 **语**言与 **基**础 **视**觉 **推**理，是一个用于测试模型执行 视觉问答（VQA）能力的合成数据集。该数据集的每个数据点都包含一幅图像，以及一个关于该图像的问答对。图像由包含3 种形状、8 种颜色、2种材质和2种尺寸的字典随机生成，程序化查询也会随机生成。给定一幅包含立方体和球体等简单物体的渲染图像，我们旨在回答关于该图像的问题。

架构。 对象特征向量由一个 4 层卷积神经网络提取。随后，特征向量被传入四个 2 层多层感知机分类器和另一个 3 层多层感知机分类器。

1 //是/否问题2rel eval_yn (e,x > y) = greater_than_expr (e, a,b), 3 eval_num (a, x),eval_num(b,y)4 rel eval_yn(e,x < y) = less_than_expr (e, a,b),5 eval_num (a, x),eval_num (b,y) 6rel eval_yn (e,x == y) = equal_expr (e, a,b),7 eval_num (a, x),eval_num (b,y) 8rel eval_yn (e,x == y) = equal_color_expr (e, a, b), 9eval_query (a, x), eval_query (b, y)10rel eval_yn (e, x == y) = equal_material_expr(e, a, b),11eval_query (a, x), eval_query (b, y)12rel eval_yn (e,x == y) = equal_shape_expr(e, a, b),13eval_query (a, x), eval_query (b, y)14rel eval_yn (e, x == y) = equal_size_expr(e, a, b),15eval_query (a, x), eval_query(b, y)16releval_yn (e,b) = b := exists (o:eval_objs (f, o) 17 where e: exists_expr (e,f))18 19// 计算对象数量20 rel eval_num (e,n) = n:= count (o: eval_objs (f, o)where e: count_expr (e, f)) 21 22//对象筛选 23 releval_objs (e, o) = scene_expr (e), obj(o) 24 rel eval_objs (e, o) = unique_expr (e, f), eval_objs (f, o) 25 rel eval_objs(e, o) = filter_size_expr (e, f, s),eval_objs(f, o), size(o, s) 26 rel eval_objs (e, o) = filter_color_expr (e, f, c), eval_objs (f, o), color (o, c) 27 rel eval_objs(e,o) = filter_material_expr (e, f, m), eval_objs (f,o),material (o, m) 28 rel eval_objs (e, o) = filter_shape_expr (e, f, s), eval_objs (f, o), shape (o, s) 29 rel eval_objs (e,o) = and_expr (e, f1 , f2), eval_objs(f1 , o), eval_objs(f2, o) 30 rel eval_objs (e,o) = or_expr (e,f1 ,f2) and (eval_objs (f1, o) or eval_objs (f2, o)) 31 32//相同属性 33 releval_objs (e, o) = same_size_expr (e, f), 34 eval_objs (f,p), size(p, c), size(o,c), o != p 35rel eval_objs (e, o) = same_color_expr (e, f), 36eval_objs (f, p),color(p, c), color(o,c), o != p 37rel eval_objs(e, o) = same_material_expr (e, f), 38 eval_objs (f,p), material (p,m), material(o,m), o != p 39rel eval_objs (e, o) = same_shape_expr (e, f), 40eval_objs (f, p),shape(p,s), shape (o, s), o != p41 42 // 关系 43rel relate(" 右侧 ",p, o) = relate("左侧", o, p) 44rel relate (" 前方 ", p, o) = relate(" 后方 ", o, p) 45rel eval_objs(e,o) = relate_expr(e, f, r), 46eval_objs (f,p),relate(r,p, o), o != p 47 48 // 评估查询49 rel eval_query (e, s) = query_size_expr (e, f), eval_objs(f,o), size(o,s)50 rel eval_query (e, c) = query_color_expr (e, f),eval_objs (f, o), color(o,c)51 rel eval_query (e, m) = query_material_expr (e, f), eval_objs (f,o), material(o,m)52 rel eval_query (e, s) = query_shape_expr (e, f),eval_objs (f, o), shape (o, s)

```
54  // 最终结果55 关系 结果(y 作为字符串)  = 根_表达式 (e)，求值_yn (e,
y)56 关系 结果(y 作为字符串)  = 根_表达式 (e)，求值_数字 (e, y)57 关系 结果
(y)  = 根_表达式 (e)，求值_查询 (e, y)
```

图 32。用于 CLEVR 的 Scallop代码。

分别获取属性和空间关系。学习率设置为 0。0001，批量大小为 32，潜在维度为 1024。我们在图 32 中展示了我们的 Scallop代码。