arXiv:2302.01918v1 [cs.LG] 3 Feb 2023

PyGlove：以代码高效交流机器学习创意

彭代逸 [1] 董轩逸 [1] Esteban Real [1] 卢一峰 [1] Quoc V. Le [1]

摘要

机器学习（ML）的复杂度和规模日益增长，因此多个团队之间需要更高效的协作。例如，当研究团队发明一种新架构（如“残差网络”）时，多个工程团队最好都能采用它。然而，每个团队都要投入精力去研究和理解这项发明，而这种投入无法随着团队或发明数量的增加而有效扩展。本文介绍了对 PyGlove 库的扩展，旨在轻松且可扩展地共享机器学习创意。PyGlove 将创意表示为符号化规则补丁，使研究人员能够写下针对尚未见过的模型的规则。例如，发明者可以写下“添加跳跃连接”的规则。这使团队之间形成网络效应：任何团队都可以立即向所有其他团队发布补丁。这种网络效应让用户能够通过更快地编写更少的代码，迅速克服采用 PyGlove 的成本，并带来随时间推移而扩大的收益。我们介绍了通过符号补丁组织机器学习的新范式，并将其与现有方法进行比较。我们还对一个大型代码库进行了案例研究，结果显示，使用 PyGlove 后，代码行数减少了 80%。

1. 引言

过去几年，机器学习（ML）协作的规模大幅扩大，使代码的高效交换变得更加困难。大学、GitHub 项目，尤其是科技公司，都能连接多位研究人员和工程师。科技公司尤其常被划分为多个独立团队，共享同一个代码库。当这些团队取得新发现时，其他团队需要将其整合到自己的代码中。然而，团队和代码库的专业化可能带来困难。更常见的解决方案是，各团队自行寻找其他团队的发现，再将这些发现整合到自己的机器学习系统中。如果这些发明需要专业知识、十分复杂，或者数量过多，这个过程就会耗费大量时间；

[1]Google Research，Brain 团队。联系邮箱：彭代逸 <daiyip@google.com>。

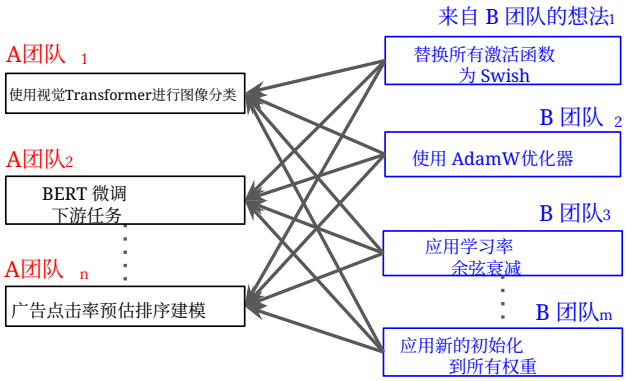

图 *1.* 将 m 新的机器学习创新应用于 n 团队会带来$n \times m$ 代码变更。PyGlove 将机器学习创新表示为可共享的规则，各 m 团队都可以立即应用这些规则。

例如，仅 ICML 会议在 2022年 就收到了 5000 多篇论文）。另一种做法是由发明者亲自将其发现应用到他人的代码库中，但这也会带来其他挑战，例如无法访问代码或缺乏充分的文档。最重要的是，每当两个团队之间发生一次交换，就要付出这些成本。同一项发明最终会被重复实现多次，导致可扩展性难以随发明数量提升。本文介绍 PyGlove，这是我们先前工作的扩展 [15]，旨在简化以代码形式实现的可扩展创意交换。

PyGlove 让一项新发明只需极少的实现工作，就能应用于多个地方。发明者本人可以通过以编程方式广播自己的发现，直接修改其他团队的代码。从总体上看，PyGlove 通过注解和基于规则的补丁来运作。要让代码库兼容 PyGlove，首先必须为其添加格式特殊、轻量级的 Python 注解，以直观的方式描述代码。这些注解是共享代码时使用的共同语言。完成这一步后，就可以通过基于规则的补丁交换代码，这些补丁会说明移植的代码必须如何插入其他位置。例如，设想这样一种情况：“A 团队”维护着一个图像分类器，而“B 团队”独立开发出一个新的卷积层，认为它能改进大多数图像分类器。在

表1。 谷歌对 PyGlove 使用情况的统计表明，补丁对代码复用有着积极影响：实验数量呈线性增长，而补丁规则的数量仅略有增加。

	方面	2020	2021	2022
一个内部基于 PyGlove 的机器学习代码库	已实现的补丁规则累计数量	70+	80+	80+
	通过补丁创建的实验累计数量	10K+	18K+	25K+
	该代码库中的机器学习工程师累计数量	9	12	22
	每个实验的平均补丁规则数量	4.2	3.1	4.7
	每个实验减少的代码行数	≈80%	≈80%	≈80%
全部基于 PyGlove 项目	基于 PyGlove 的项目总数	20+	40+	70+
	内部 PyGlove 用户总数	100+	200+	1200+

PyGlove 方法中，A 团队可以为其（已有的）代码添加注解，例如“这是一个卷积”“这是一个非线性层”等；B 团队则可以为他们的新层添加注解，例如“这些是超参数”（见第 2 节）。A 团队得知这个新层后，可以编写一条单行规则，相当于“用 B 团队的层替换我的所有卷积”，如图 2 所示。此外，PyGlove 还带来了一种新颖的做法：B 团队可以自行编写替换规则——这类规则相当于“在任何图像分类器中，都用我们的层替换所有卷积”。这样，任何拥有 PyGlove 注解图像分类器的团队都可以应用 B 团队生成的规则。这种做法为通过机器学习创新成果库开展未来协作提供了更多可能，我们将在第 2.2 节讨论。

我们强调，卷积层交换示例是一个特别简单的例子，仅用于说明目的。更复杂的示例见图3。此外，PyGlove 基于规则的方法并不局限于架构变更的共享，而是适用于机器学习流程的各个方面，例如数据增强、训练算法或元学习。尤其是，随着机器学习硬件不断进步，模型容量也需要反复扩展。为解决这一问题，人们提出了经验规则和理论规则 [19；20； 24]。此类规则可以在组织内部或社区中传播，从而节省宝贵的工程师时间。

由于团队之间的“网络效应”，采用 PyGlove 所需的成本很快就能通过其带来的收益抵消（图1）。采用成本是为代码库添加注解所需的（固定）工作量，除了添加新注解本身，几乎不需要编写代码。由于这些都是标准的 Python 注解，原有代码的大部分都无需改动。另一方面，PyGlove 的收益会在团队每次交流想法时体现，具体如下。没有 PyGlove 时，m 应用于 n团队项目的创新会带来 $m \times n$ 工作；使用 PyGlove 时，每项创新都需要编写一条 PyGlove 规则（m 规则），而每个团队项目只需负责为自己的模型添加 PyGlove 注解（n 模型），因此只需 $m + n$ 工作，因为规则的应用非常简单。在所有这些情况下，我们的基于规则的方法都与当前方法形成鲜明对比；当前方法通常需要多次就地修改，且难以随模型规模或社区从业者数量的增加而扩展。这些比较详见第 4 节，定量的真实案例见表 1。特别是，我们对一个大型代码库进行的案例研究表明，采用 PyGlove 后，代码行数减少了 80%。

PyGlove 能够广泛应用于机器学习代码的各个方面，乃至机器学习之外，其原因在于它采用了有原则的符号编程设计 [13]。在这一范式下，经 PyGlove 注释的 Python 对象会成为可编辑的符号，而 PyGlove 规则则是作用于这些符号的元程序。这些规则本身也是符号，因此具备组合性，类似于 LISP 程序能够自我编辑 [17]。在第 3 节中，我们将介绍符号编程并阐释 PyGlove 的设计。

总而言之，我们提出：

- 一种利用符号补丁，以代码形式高效、可扩展地共享复杂机器学习构想的方法；
- 展示如何在整个机器学习开发流程中运用符号编程；
- 开源的 PyGlove 库[1] 以及本文使用的补充代码[2]。

2. 通过补丁表达机器学习构想

机器学习（ML）工程通常包括对机器学习设置的迭代改进。本文将设置 定义为一组能够生成可用机器学习模型的组件，包括训练数据、数据预处理、模型本身、训练算法以及

[1] https://github.com/google/pyglove [2] https://github.com/google/pyglove/blob/main/docs/notebooks/ml/高效地_交流_机器学习_理念_作为_代码.ipynb

验证过程。无论是用于生产还是实验，我们都可以将每个改进步骤视为：通过应用代表各种想法的概念规则来编辑初始设置，从而得到最终设置。举例来说，简单地讲，如果机器学习从业者想要将神经网络中的所有卷积层替换为可分离卷积层，就必须在网络的源代码中手动查找并替换每个卷积单元。这个手动过程需要理解不同组件及其相互作用。

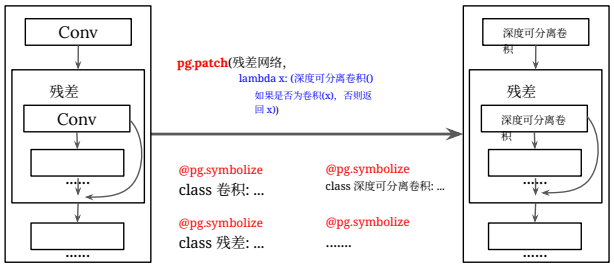

图2。B 团队的新发明深度可分离卷积的开发无需依赖 A团队的代码，但其规则 apply sepconv 可以通过补丁将 A团队的实验升级为新的残差网络。在这段代码中，pg.symbolize 是为常规 Python 类添加可编辑特性的注解，而 pg.patch 则是将规则应用于已注解类对象的操作。

为了自动化这一过程，我们需要能够以编程方式操作机器学习程序。这不仅能实现某个理念，还能将同一理念应用于其他机器学习设置，而无需修改其源代码。为此，我们开发了 PyGlove，它能够表达手动过程中蕴含的规则，并将其封装为可在不同机器学习程序间复用的单元。这不仅能通过直接操作机器学习程序来实现复杂理念，还能通过以编程方式表达规则，即补丁，来捕捉繁琐的日常机器学习实践。例如，前述在A团队的设置中实现 B 团队最新卷积发明的任务，只需应用一行补丁即可通过编程实现，如图2所示。

2.1. 补丁的表达能力

使用符号补丁表达规则并不局限于简单的想法，也适用于更复杂的概念。PyGlove 支持直接操作机器学习配置中的程序部分 [10]，既可以按位置操作，也可以按规则操作。例如，如图3所示，PyGlove 可用于更换数据集、调整神经网络模型的宽度，甚至通过自动机器学习自动执行模型扩展。

符号补丁不仅易于表达想法，还能在不同的机器学习设置之间共享并轻松应用。PyGlove 通过类似 URI 的字符串促进补丁的共享，这种字符串易于人类阅读，并且可以通过全局名称进行唯一标识。例如，对于使用 ImageNet [6] 数据集的机器学习设置，可以将补丁表示为 change dataset?name=imagenet。

2.2. 补丁的组合特性

此外，符号补丁可以组合起来，创造更复杂的想法。例如，近期的大规模自然语言处理算法 LaMDA [21] 可以通过对 Transformer [22]架构应用一组常见补丁来实现，正如我们借助 PyGlove 在图4 中所展示的那样。

组合补丁的能力对于全面探索机器学习想法至关重要。如图5所示，研究团队（团队 B_i）可以通过借助 PyGlove 将最新发现封装为补丁规则，与产品团队（团队 A_i）共享。创建规则后，只需以兼容方式添加注释，就能轻松将其应用于任何产品团队的代码库。这意味着，研究团队的想法可以被许多产品团队采用，而无需每次都进行手动集成；产品团队也能快速尝试多种研究想法。此外，组合多个补丁还能进一步增强网络效应的潜力（见第6节）。

2.3. 补丁有效性的统计分析

谷歌有多个非公开的实际项目将符号补丁应用于日常机器学习实践，并在过去三年中开展了数万次实验。正如首个采用 PyGlove 的项目所展示的那样，符号补丁使大型机器学习代码库能够在不按比例增加开发成本的情况下扩大实验规模；该项目的代码超过 50,000 行。

表1 显示了该项目的使用统计数据：通过逐渐增加补丁规则的数量，机器学习工程师每天启动的不同实验总数呈线性增长。此外，开发成本大幅降低，而且这一影响逐年持续扩大。

3. 机器学习符号编程

在展示符号补丁封装机器学习理念的强大能力与普适性的同时，其核心编程范式——符号编程——在整个机器学习领域也有更广泛的用途——

```
@pg.patcher
def change_数据集(exp, name):
  数据集 = pg.load(name)
  稠密层 = pg.query(
      exp,
      其中=lambda v: is_dense(v))
  out_loc = f'denses[-1].sym_path.units
  返回{
    '数据集': 数据集,
    输出_位置: 数据集.num_类别}
```

```
@pg.patcher
定义缩放_宽度(exp, 缩放):
    定义补丁_函数(k, v, p):
        如果 isinstance(p, (卷积, 全连接层))
            且 k.endswith('通道数'):
            return v * scale
        return v
    返回 补丁_函数
```

```
@pg.patcher
def 搜索_以寻找_宽度(exp, 浮点运算次数):
    def 替换_实验(k, v, p):
        如果 p 为 None:
            返回 AutoML(
                target_浮点运算次数=浮点运算次数, **v.sym_init_arg)
        返回 v
    scale_范围 = [pow(2, x) for x in range(-2, 3)]
    返回 [
        缩放_宽度（缩放=pg.oneof（缩放_范围））,
        替换_实验]
```

图 *3.* 另外三个更复杂的理念通过 PyGlove 修补规则来表达。（左）要更改分类任务的数据集，我们需要同时更改数据集和模型头部的宽度，使其与类别数量一致。具体做法是查询最后一个 Dense 层的位置，并将数据集的新值以及与其位置相关联的 Dense 单元数作为更新返回给实验。（中）要按缩放因子调整卷积层和全连接层的通道数，补丁规则通过一个函数来定义，该函数用于转换对象树中的每个节点，以当前节点的位置、值和父节点作为输入，并修改卷积层和全连接层类的“通道数”成员。（右）这是前一个示例的升级版本，使用自动机器学习根据目标浮点运算次数缩放模型。为此，它会生成一个缩放因子搜索空间，并将其连接到自动机器学习训练器，由后者自动找出最佳选项。

```
LaMDA = pg.patch(注意力_是_
全部_你_需要，  ['移除_编码器',
'扩展_规模', '更改_数据集？ ……',
'应用_门控_GELU', '……'])
```

图 *4.* LaMDA论文 [21] 可通过在 Transformer 论文上应用一系列补丁来获得 [22]。

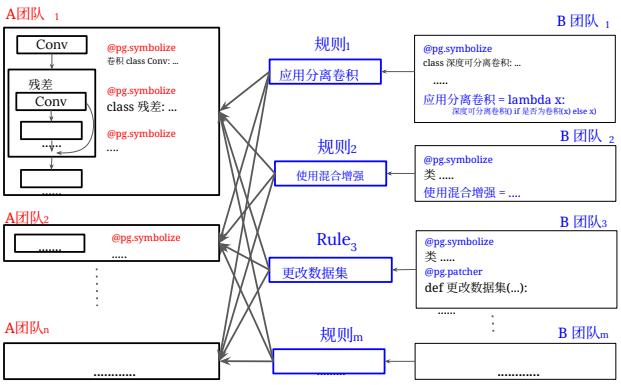

图 *5.* 如果没有 PyGlove，将 m 项更改补丁应用到 n 个团队，需要对代码进行 $n \times m$ 处修改。使用 PyGlove 时，每个 A_i团队都会为自己的代码添加注释，每个 B_j 团队都会编写自己的规则。因此，应用所有更改只需进行 $n+m$ 处修改。

开发过程。在深入探讨其其他应用之前，我们将概述符号编程的含义、它在机器学习中的作用，以及 PyGlove 在这一领域提供的具体功能。

3.1. 为什么符号编程适用于机器学习

符号编程起源于 LISP [17]，，是一种编程范式，在这种范式中，计算机程序可以像处理普通数据一样操作自身的组成部分 [13]。符号编程对机器学习很有用，因为机器学习需要不断基于现有的机器学习设置探索新想法，因此机器学习程序也需要频繁修改。

修改程序可以通过编辑源代码，或在运行时操作程序来实现。然而，由于机器学习需要灵活地探索新想法并开展各种实验，修改源代码并非理想之选。例如，用户通过复制并修改现有设置来创建新设置时，如果原始设置更新了，就很难跟上这些变化。

相比之下，符号编程无需修改源代码，就能在运行时操作程序。这种方法可以动态地向现有机器学习设置中添加新元素，而无需管理源代码变更。此外，它还允许操作预先存在的概念，从而在想法与实现之间建立直接联系。

3.2. PyGlove 的符号编程解决方案

PyGlove 的符号编程解决方案涉及使用符号对象，它们既是符号，也是对象。符号对象既可以作为符号进行操作，也可以像普通对象一样进行评估。符号对象是符号类的实例，可通过`pg.symbolize` 注解创建。图6 展示了

```
@pg.symbolize
类 机器学习实验:

  定义 __init__(self, 数据集, 模型, 损失,
                优化器, 训练轮数, 批次大小):
    self._数据加载器 = DataLoader(
        数据集, 批次大小=batch_size)
    self._模型 = 模型
    self._损失 = 损失
    if TensorFlow.using_tpu_device():
      self._优化器 = CrossShard(optimizer)
    否则:
      self._optimizer = 优化器
    self._steps = epochs * len(dataset) // 批次大小

  定义 训练(self):
      循环 _iter，范围为(self._steps):
        x, y = self._数据加载器.next()
        ....

  def 评估(self):
      ...
```

```
# `exp`：一个符号对象
exp = 机器学习实验(
  数据集=CIFAR(),
  模型=残差网络(num_layers=50),
  损失=交叉熵(),
  优化器=Adam(lr=0.1),
  训练轮数 = 100,
  批次大小 = 256,
)

# `exp` 可以通过符号方式进行修改。
pg.patch(exp,
  '更改数据集?name=ImageNet')

# `exp` 可像常规对象一样进行评估
# 对象。
exp。训练()
```

图 *6.* 仅用一行代码创建符号类（左），并对其对象进行操作（右）

符号类机器学习实验及其对象 exp 的定义。

符号对象具有若干关键特征，包括保留其构造信息（类型、构造函数参数），这些信息定义了它的符号表示。这种表示在源代码形式和运行时形式中完全一致，因此可以直接通过符号表示来操作概念。 [10]此外，符号对象也是一个在运行时存在的程序实体，具有数据成员和方法，其行为与普通对象类似（图 7）。兼具符号性与对象性，使符号对象既能受益于面向对象编程，也能受益于符号编程。前者支持模块化和可扩展的软件设计，后者则通过操作提供元编程的能力。图 8 展示了可应用于符号对象的操作。

3.3. 符号对象如何服务于机器学习

考察符号对象在机器学习中的作用，我们就能看到它们带来的好处。首先，轻松更改符号对象值的能力消除了对独立超参数容器的需求，使机器学习组件的开发和使用更加精简且一致。此外，借助符号操作，只需少量代码即可高效地进行机器学习实验。图 9 展示了机器学习开发过程中使用的符号操作，这些操作基于图 机器学习实验类，该类如图 8所示。

通过获取实验对象并对其进行评估（例如，调用其 9(1) 方法），即可复现实验（图训练 ）。由于符号对象可以序列化，因此可以从文件或实验仓库中加载机器学习配置。如第2节所述，迭代实验（图9(3)）包括

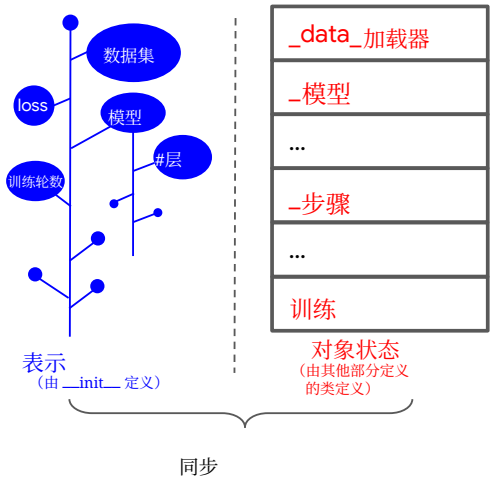

图 *7.* 符号对象的表示由其构造函数签名定义，而其对象状态则由执行逻辑决定。当符号对象的表示发生变化时，其对象状态会重置，因此它们始终保持一致。

使用现有组件或新组件更新其设置。需要新组件时，可以像创建常规组件那样创建，只需额外应用 `pg.symbolize` 装饰器（图9(2)）。虽然可以手动迭代实验，但这样做可能很耗时，尤其是在进行大量修改时。符号占位通过允许将待调优的值替换为搜索空间规格，简化了自动调优（图9(4)）。这样便能在调优算法的控制下，从更大的空间中生成特定实验。此过程适用于超参数和神经网络架构，也适用于损失函数、优化器、数据增强策略等其他实验组件。最后，当实验准备好进行部署时（图9(5)），可以将其序列化到文件中，或保存到实验仓库中。不仅可以保存整个实验，也可以保存其中的各个部分。此外，还可以将新组件和补丁规则作为标准机器学习模块添加。

4. 对 PyGlove 方法的分析

PyGlove 的机器学习方法核心在于封装从一次实验推进到下一次实验所需的工程工作。该方法基于这样一种认识：全新的机器学习构建模块十分稀缺，但它们的组合与交互却层出不穷。PyGlove 的解决方案通过灵活的接口，让思想得以轻松表达、重用和共享。此外，PyGlove 将符号化可编程性融入面向对象范式，使用户能够通过规则在运行时操纵机器学习实验对象，从而封装



图 *8.* 常见的符号操作。符号对象可以轻松地进行验证、检查、比较、克隆、序列化、修改、占位和生成。

整个机器学习过程，而不仅仅是其输出。

4.1. 与现有解决方案的比较

机器学习以表示为核心的特性，促使人们开发出一些解决方案，使其能够使用类数据实体配置机器学习程序，从而在实验过程中进行修改而无需更改源代码。当前的解决方案 [16；9； 23]主要采用配置系统，其中机器学习组件及其配置是彼此独立的实体。另一方面，PyGlove 将机器学习实验视为可变符号对象，将表示与定义合二为一，如图10所示。

人们通常会直接修改机器学习设置，而不是依据配置使用规则进行修改。这是因为基于规则的转换往往需要访问运行时信息；当实验表示与程序定义彼此分离时，这些信息很难获取。例如，尝试更换数据集时（图 3-1），可能很难仅从配置中获取必要的信息（例如类别数量）。此外，只有能够广泛应用时，规则才具有成本效益。表示与定义分离会导致同一概念存在不同的模式，使规则难以广泛应用。然而，在 PyGlove 中，程序定义本身就是一种表示，因此实验表示更加统一，也更容易通过规则处理。

实验表示与其程序定义相分离，不仅会增加编写和使用规则的难度，也会给机器学习的开发带来不便。这是因为两者需要不时进行同步。这种分离会造成三个常见问题：

- 当组件定义更新时，开发者需要修改该组件的表示，反之亦然。这包括但不限于文档、验证规则以及创建子组件的逻辑。
- 在处理实验表示时，需要访问组件定义的自动化工具无法正常运行，例如自动补全和代码导航。
- 调试正在运行的实验时，很难将错误追溯到实验表示，尤其是在处理同一组件类型的复杂组合时。例如，确定是哪个卷积层导致了输入形状不匹配。

4.2. 注意事项与局限性

PyGlove 使用可变编程范式，因此也有其自身的一系列注意事项与局限性。人们普遍认为，可变程序有三大缺点：（1）难以安全地编写，因为对象可能发生变化，因此必须格外谨慎，确保变更期间程序状态保持一致。（2）难以推理，因为下游代码可以更新先前创建的对象，仅通过查看代码更难理解正在发生什么。

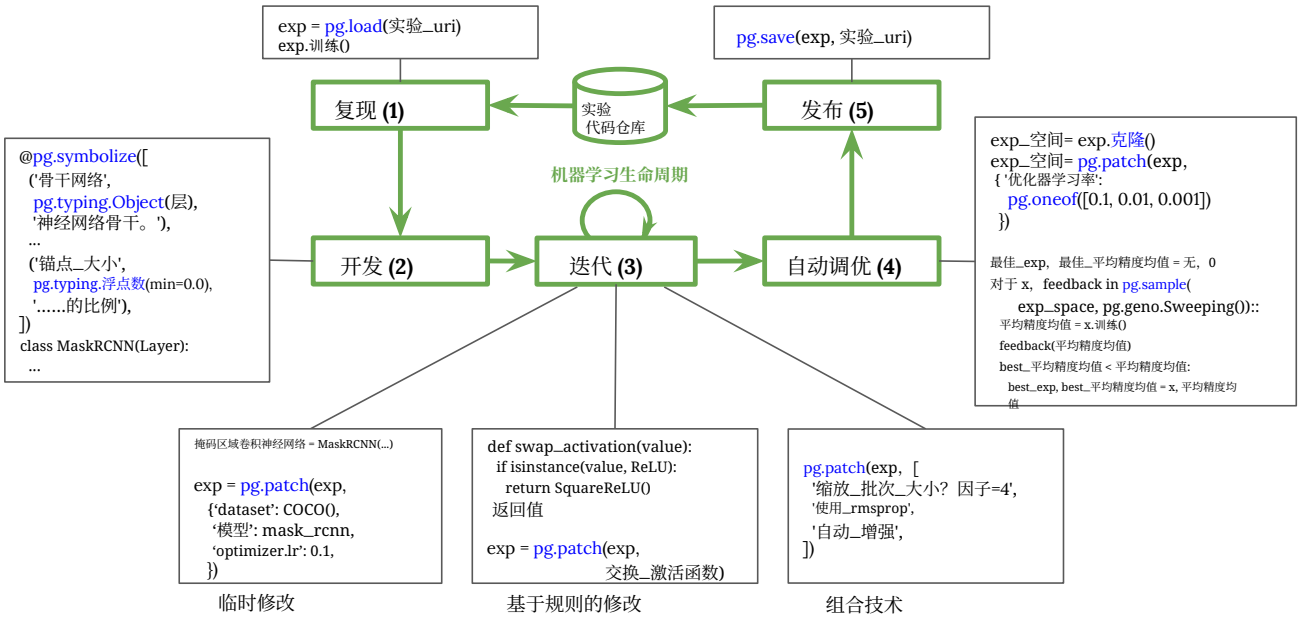

图 *9.* PyGlove 服务于整个机器学习生命周期。PyGlove 简化了从研究想法诞生到部署过程中涉及的工程工作，同时让整个实验或其中的部分可供复用。

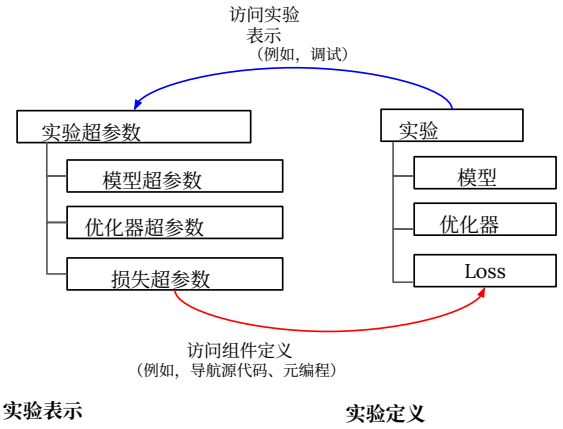

图 *10.* 在非符号化机器学习程序中，实验表示（左）与实验定义（右）的分离。实验表示通常以配置的形式编写，而实验定义则以机器学习组件的源代码形式表达。建造者模式 [8] 通常用于根据实验表示创建机器学习组件实例。

（3）性能较差，因为可变对象需要额外指令来记账参数，以处理变更，因此创建符号对象比创建不可变对象更慢。此外，由于需要大量锁定，使用可变对象进行多线程编程也颇具挑战。虽然这些批评通常是成立的，但我们认为它们对机器学习领域的影响并不显著。机器学习语言提供了一种编译——

化步骤将设置的构建与训练分开（例如 TensorFlow 中的“计算图构建”或 JAX 中的“即时编译”步骤）。在构建阶段变更计算图的能力提供了很大的威力和灵活性，但编译完成后，就不再需要进一步变更。我们在现实世界研究和产品应用中使用 PyGlove 的经验也证实了这一点。最后，与机器学习训练的典型开销相比，符号对象操作带来的额外运行时开销微不足道。

5. 相关工作

PyGlove 是众多现有机器学习库中独具特色的软件库之一 [1; 4; 14; 3; 11; 5; 23; 9; 16]。它并非为机器学习程序添加特定功能，而是通过应用规则，让实验能够从其他实验衍生而来，以此为机器学习引入了一种新的编程范式。这种方法让机器学习理念更易于表达，并支持规则的重用、共享与组合，从而在各种理念之间形成网络效应。PyGlove 的机器学习工程流程与编译器中的变换流水线类似 [2; 12],变换流水线以程序的中间表示（IR）为输入，并在其上依次应用变换。PyGlove 中的 IR 是对象的符号表示，而变换则是补丁规则。不过，PyGlove 与编译器之间存在一些关键差异：（1）在 PyGlove 中，规则应用于程序本身；而在编译器中，变换应用于其他

程序的 IR。（2）PyGlove 中 IR 与执行的统一，使程序的修改与执行能够结合起来，从而实现更动态的运行时行为。

PyGlove 的解决方案属于符号编程的一种形式 [13],但它与其他符号编程解决方案不同，例如 LISP 中的方案 [17], 因为它是在对象层级而非代码定义层级进行操作。这使其功能更强大，因为同一代码定义可以拥有多个可供操作的实例。此外，由于操作针对的是对象的绑定参数，而不是通常自足的底层代码实现，因此更加安全。具体而言，PyGlove 的符号范式是符号编程与面向对象编程（OOP）的结合 [18]。它允许将符号对象作为符号进行操作，并作为对象进行评估，从而兼具两种编程范式的优势。前者允许对程序进行操作，后者则遵循面向对象编程的最佳实践，通过封装和继承开发可复用组件。

在机器学习领域，与符号编程最相关的工作是 [15], , 它将自动机器学习重新概念化为基于算法的符号操作过程，可视为 PyGlove 的一种专门应用。为了拓展其在更广泛机器学习任务中的用途，本文对符号对象范式进行了形式化，强调了封装规则的重要性，并突出了这些规则之间的网络效应。此外，本文还将符号编程的应用拓展到整个机器学习开发流程。因此，本文大幅拓宽了先前工作的范围。

6. 讨论

当所有实验都使用相同的符号表示时，符号补丁是一种共享机器学习理念的强大工具，而在共享代码库中通常就是如此。本文聚焦于单代码库场景，例如工业界就存在这种情况。然而，这种场景并不适用于大多数学术界，也不适用于整个机器学习社区，因为不同团队往往完全独立地构建各自的代码库。扩展到多代码库场景会带来一些问题。尤其是，参与的团队需要就高级接口达成一致。这需要社区协作，并采用共享的软件设计最佳实践，超出了本文的范围。尽管如此，我们推测 PyGlove 提供了新工具，有助于在不同代码库之间建立共享的高级机器学习接口：（1）PyGlove 通过代码注释而非直接编辑来发挥作用，因此可以自然地逐步构建接口。可以为代码库添加注释，使其仅支持特定的目标规则补丁（例如，另一团队刚刚发布的补丁）。（2）PyGlove 消除了对常见“配置对象”范式的需求（见第 4 节）。配置对象引发了关于其格式和约定的争论，阻碍了代码库的趋同。另一方面，符号化机器学习对象可以直接编辑，无需单独配置。（3）符号化方法具有组合性，从而支持多层级接口。例如，接口可以轻松地同时呈现整个图像分类器、其中的每一层，以及每层的输入和输出。（4）发布有用的补丁可以为代码库添加注释提供强大动力，进而可能鼓励更多补丁，形成正向强化。因此，我们希望，除了在单个代码库内共享代码之外，PyGlove 未来还能对跨机器学习代码库的协作产生积极影响。

7. 结论

由于无法以可扩展的方式将概念性规则应用于不同的机器学习设置，机器学习的发展受到阻碍。即使论文的代码和实验定义已开源，从论文及其参考代码库中获取机器学习配置也并非易事，且难以重用，因为这需要手动解析并在其他实验中复现。本文将 PyGlove [15] 的符号能力从自动机器学习拓展到机器学习，以解决这一问题。我们提出将机器学习配置 A 演进到配置 B 的过程封装为一系列补丁规则。这些规则可以在不同机器学习设置之间重用，从而在实验和规则之间形成网络效应。除了补丁之外，我们还展示了符号编程如何有效地服务于机器学习的整个生命周期。我们还将 PyGlove 的解决方案与现有方案进行了比较。通过高度依赖 PyGlove 补丁能力的现实世界研究项目 [7; 25]，我们看到了 PyGlove 改变机器学习程序开发、组织和共享方式的潜力。我们已将 PyGlove 开源，并期待机器学习社区对其进行广泛测试。

参考文献

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Is- ard, M. 等。TensorFlow：面向大规模机器学习的系统。载于 *USENIX* 操作系统设计与实现研讨会（*OSDI*），第 265–283 页，2016 年。

[2] Bacon, D. F., Graham, S. L., 和 Sharp, O. J. 面向高性能计算的编译器转换

。*ACM Computing Surveys (CSUR)*，26(4):345–420，1994 年。

[3] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. Theano：一种 CPU 和 GPU 数学表达式编译器。载于*Python* 科学计算会议论文集（*SciPy*），第 1–7 页，2010 年。

[4] Bradbury, J., Frostig, R.,Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX：Python+NumPy 程序的可组合变换，2018 年。网址：http://github.com/google/jax。

[5] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. MXnet：一种面向异构分布式系统的灵活、高效机器学习库。*arXiv* 预印本 *arXiv:1512.01274*，2015 年。

[6] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., 和 Fei-Fei, L. ImageNet：大规模分层图像数据库。载于 *IEEE* 计算机视觉与模式识别会议论文集（*CVPR*），第 248–255 页，2009 年。[7] Dong, X., Tan, M., Yu, A. W., Peng, D., Gabrys, B., 和 Le, Q. V. AutoHAS：高效的超参数与架构搜索。载于 *ICLR* 神经架构搜索第二届研讨会，2021 年。[8] Gamma, E., Helm, R., Johnson, R., 和 Vlissides, J. 设计模式：面向对象设计的抽象与重用。载于*ECOOP'93*——面向对象编程：第七届欧洲会议，德国凯撒斯劳滕，*1993* 年 *7* 月 *26–30* 日，论文集 *7*，第 406–431 页。Springer，1993 年。[9] Hongkun Yu, Chen Chen, X. D. Y. L. A. R. L. H. P. J. F. Y. F. L. J. K. 和 Li, J. TensorFlow 模型园地。https://github.com/tensorflow/models，2020年。

[10] Hutchins, E. L., Hollan, J. D., 和 Norman, D. A. 直接操控界面。人机交互，1(4):311–338，1985年。

[11] Jia, Y.、Shelhamer, E.、Donahue, J.、Karayev, S.、Long, J.、Girshick, R.、Guadarrama, S. 和 Darrell, T. Caffe：用于快速特征嵌入的卷积架构。载于*ACM*多媒体国际会议（*ACM MM*）论文集，第 675–678 页，2014年。

[12] Lattner, C. 和 Adve, V. LLVM：用于持续程序分析与转换的编译框架。载于代码生成与优化国际研讨会，*2004*年。*CGO 2004*。，第 75–86 页。IEEE，2004年。

[13] McCarthy, J. 符号表达式的递归函数及其机器计算。*ACM* 通讯，3(4):184–195，1960年。

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., 等。PyTorch：一种采用命令式风格的高性能深度学习库。载于神经信息处理系统国际会议论文集（*NeurIPS*），第 32 卷，2019年。

[15] Peng, D., Dong, X., Real, E., Tan, M., Lu, Y., Bender, G., Liu, H., Kraft, A., Liang, C., 和 Le, Q. PyGlove：用于自动化机器学习的符号编程。载于神经信息处理系统国际会议论文集（*NeurIPS*），2020年。

[16] Shen, J., Nguyen, P., Wu, Y., Chen, Z., Chen, M. X., Jia, Y., Kannan, A., Sainath, T., Cao, Y., Chiu, C.- C., 等。Lingvo：用于序列到序列建模的模块化且可扩展的框架。*arXiv* 预印本 *arXiv:1902.08295*，2019年。[17] Steele, G. *Common LISP*：这门语言。爱思唯尔，1990年。[18] Stefik, M. 和 Bobrow, D. G. 面向对象编程：主题与变化。《人工智能杂志》，6(4)：40–40，1985年。[19] Sussman, G. J. 构建稳健系统：一篇随笔。*Citeseer*，113：1324，2007年。[20] Tan, M. 和 Le, Q. EfficientNet：重新思考卷积神经网络的模型扩展。发表于机器学习国际会议（*ICML*），第6105–6114页。PMLR，2019年。[21] Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., 等。LaMDA：面向对话应用的语言模型。*arXiv* 预印本 *arXiv:2201.08239*，2022年。[22] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., 和 Polosukhin, I. 注意力就是你所需的一切。发表于神经信息处理系统国际会议论文集（*NeurIPS*），第30卷，2017年。[23] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., 和 Girshick, R. Detectron2。
h
t
t
p
s://github.com/facebookresearch/detectron2，2019年。

[24] Yang, G., Hu, E., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., 和 Gao, J. 通过零样本超参数迁移调整大型神经网络。发表于神经信息处理系统国际会议论文集（*NeurIPS*），第34卷，2021年。

[25] Zhou, Y., Dong, X., Meng, T., Tan, M., Akin, B., Peng, D., Yazdanbakhsh, A., Huang, D., Narayanaswami, R. 和 Laudon, J. 迈向神经网络与加速器的协同设计。载于机器学习与系统会议论文集（*MLSys*），2022年。