精读笔记

2024年10月23日　18:24

文献标题：Mathematical discoveries from programsearch with large language models 用大语言模型进行程序搜索的数学发现
文献类型：学术论文
作者：Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, Alhussein Fawzi
年份：2023
期刊：Nature
影响因子：50.5

引言及重要信息

1、研究背景

> 数学中的许多数学问题是“easy to evaluate”，但是“hard to solve”，因此文章重点讨论如何使用“evalutor”函数评估LLM生成的解决问题的程序，从而获得问题的最佳解决方案。
> LLM的编码能力提高且在代码编程多方面都有所应用，但由于LLM会产生幻觉和错误想法，对于发现开放问题的可验证的新思路，LLM的实现仍旧非常困难。

2、研究的目的和意义

> 研究的目的是生成一个求解程序，使其能在评估函数中获得高分，并最终改进已知的最佳解决方案。
> 展示了LLM在解决复杂科学问题方面的潜力，能够发现新的算法和真正解决问题，而不是在训练数据中检索出来的。

3、文献创新点

> FunSearch搜索的生成解决方案的程序，而不是解决方案本身。
> FunSearch使用的是快速推理模型。以牺牲样本质量为代价，提高搜索效率。（只要LLM训练的语料库足够大，对LLM的选择对实验结果的影响不是决定性的。）
> FunSearch利用“best-shot-prompt”技术，将性能最好的程序反馈给LLM，从而指导LLM生成更好的程序。
> 在LLM对程序进行改进的过程中，固定程序整体的架构，只改进程序中最关键的部分（如，解决cap set问题的程序中的priority函数）。这种技术可以减少 LLM 的搜索空间，并提高搜索效率。
> 使用了岛屿模型，避免陷入局部最优解，维持了种群多样性。
> FunSearch 实现为分布式系统，可以并行运行多个 LLM 和评估器，从而提高搜索效率和可扩展性。

4、前置知识

> cap set问题和bin packing问题
> LLM
> 岛屿模型
> 进化式算法

研究方法

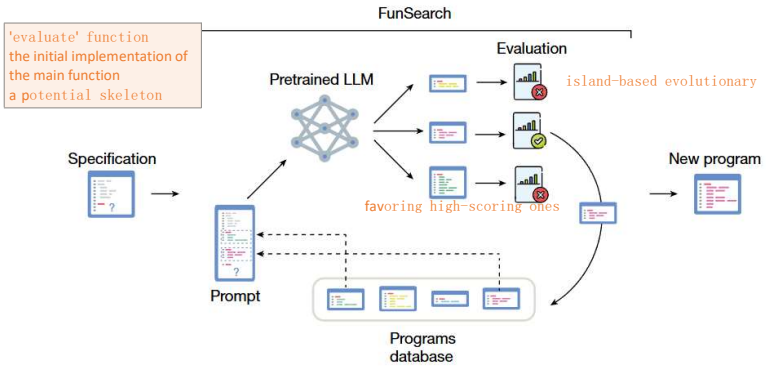


Fig. 1 | Overview of FunSearch

工作原理

⚠ FunSearch将预先训练的大模型（其训练目标是以计算机代码的形式提供创新解决方案）与自动“评估器（Evaluator）”配对，这个评估器就是用来防止幻觉和错误想法出现的。通过在这两个组件之间反复迭代，初始解决方案就会“演变”为新知识。

1、Specification

FunSearch的输入包括某个问题的评估函数以及初始程序（包含样板代码和以程序结构形式存在的关于问题的先前知识）。其中，评估函数用于评估候选解决方案的质量。初始程序可以是简单的算法骨架，也可以是完整的程序。

a

```
"""Finds large cap sets."""
import numpy as np
import utils_capset

# Function to be executed by FunSearch.
def main(n):
  """Runs `solve` on `n`-dimensional cap set and
  ↪  evaluates the output."""
  solution = solve(n)
  return evaluate(solution, n)

def evaluate(candidate_set, n):
  """Returns size of candidate_set if it is a cap
  ↪  set, None otherwise."""
  if utils_capset.is_capset(candidate_set, n):
    return len(candidate_set)
  else:
    return None

def solve(n):
  """Builds a cap set of dimension `n` using
  ↪  `priority` function."""
  # Precompute all priority scores.
  elements = utils_capset.get_all_elements(n)
  scores = [priority(el, n) for el in elements]
  # Sort elements according to the scores.
  elements = elements[np.argsort(scores,
  ↪  kind='stable')[::-1]]

  # Build `capset` greedily, using scores for
  ↪  prioritization.
  capset = []
  for element in elements:
    if utils_capset.can_be_added(element, capset):
      capset.append(element)
  return capset

# Function to be evolved by FunSearch.
def priority(element, n):
  """Returns the priority with which we want to add
  ↪  `element` to the cap set."""
  return 0.0
```

b

```
"""Finds good assignment for online 1d bin
↪  packing."""
import numpy as np
import utils_packing

# Function to be executed by FunSearch.
def main(problem):
  """Runs `solve` on online 1d bin packing instance,
  ↪  and evaluates the output."""
  bins = problem.bins
  # Packs `problem.items` into `bins` online.
  for item in problem.items:
    # Extract bins that have space to fit item.
    valid_bin_indices =
    ↪  utils_packing.get_valid_bin_indices(item,
    ↪  bins)
    best_index = solve(item,
    ↪  bins[valid_bin_indices])
    # Add item to the selected bin.
    bins[valid_bin_indices[best_index]] -= item
  return evaluate(bins, problem)

def evaluate(bins, problem):
  """Returns the negative of the number of bins
  ↪  required to pack items in `problem`."""
  if utils_packing.is_valid_packing(bins, problem):
      return -utils_packing.count_used_bins(bins,
      ↪  problem)
  else:
    return None

def solve(item, bins):
  """Selects the bin with the highest value according
  ↪  to `heuristic`."""
    scores = heuristic(item, bins)
    return np.argmax(scores)

# Function to be evolved by FunSearch.
def heuristic(item, bins):
  """Returns priority with which we want to add
  ↪  `item` to each bin."""
  return -(bins - item)
```

Fig. 2 | Examples of FunSearch specifications for two problem

2、预训练LLM

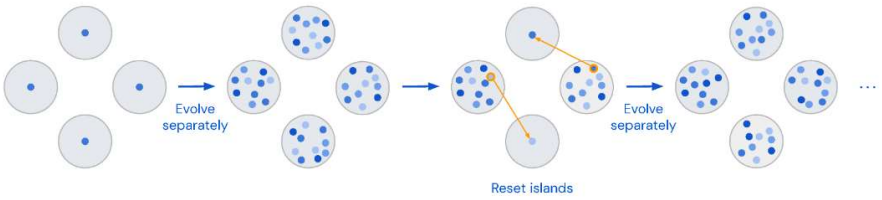
该部分内容是FunSearch的核心，负责根据提示词生成改进的函数。文中使用的Codey是基于PaLM2模型家族构建的LLM，它已经在大量代码语料库上进行了微调，并且可以通过其API公开访问。在实际训练中，文中选择的是快速推理模型（以牺牲文本质量为代价，提高搜索效率）。

3、评估器

将要评估的程序的输出的结果作为evalution函数的输入，对于多个输入使用聚合函数（如，平均值）组合成程序的总体分数。不正确的程序（未在规定的时间和内存限制内执行，或产生无效输出）被丢弃，其余的程序将被发送到程序数据库中。

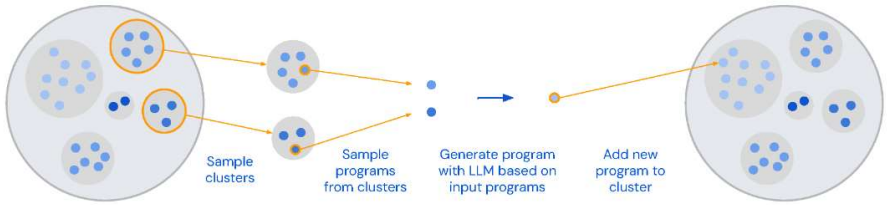
4、程序数据库

计算每个岛屿中表现最好的程序的得分作为岛屿的得分，将得分最低的一半岛屿中的所有程序清空，克隆幸存岛屿中的最高得分个体初始化淘汰岛屿中的新种群。



Extended Data Fig. 2 | Evolutionary method

程序数据库保存正确的程序群体，然后对这些程序进行采样以创建提示词。保留和鼓励程序数据库中的程序多样性至关重要，以启用探索并避免陷入局部最优。为了鼓励多样性，文中采用了岛屿模型，将程序划分为多个子群体，独立进化。



Extended Data Fig. 3 | Program clusters withth islands

当需要创建新的prompt时，系统会从程序数据库中查询并采样k个程序（通常设置为2个，即k=2）。根据这些程序的评分进行排序，从最低分的程序开始，标记为'v0'，其次是'v1'，依此类推。然后将这些程序组合成一个单一的提示。例如，如果程序中要生成的函数名为priority，则提示中会包含priority_v0、priority_v1等，以及一个没有具体实现的priority_vk函数头。这个提示随后被输入到预训练的LLM中，LLM会根据这个提示生成新的程序。

5、提示词

新的提示词是由程序数据库中的“best-shot prompting”生成的，然后被传递给LLM以生成一个新的程序。

6、分布式方法

文中实现了一个分布式系统，该系统具有三种类型的工作器：程序数据库、采样器和评估器，它们异步通信。程序数据库存储和提供程序，采样器使用预训练的LLM生成新函数，评估器评估程序。

研究内容

1、cap set问题

文中使用的研究方法是使用FunSearch搜索一个算法骨架中的“priority”函数，该函数决定每个向量被包含在cap set中的优先级。在n=8维的情况下，FunSearch发现了比之前已知更大的cap set，并且能够生成这些cap set的程序，而不是仅仅发现一个向量集合。此外，FunSearch还发现了新的admissible sets，从而提高了cap set容量的下界，这是20年来在这个问题上的最大进展。

相较于最佳拟合算法，FunSearch启发式函数只有在匹配物品紧急的情况下，才会将物品分配到当前可容纳该物品的最小的packing中。否则物品通常被放在另外一个packing中。这样放置物品会留出更多的空间，避免在packing中留下难以被填满的小空隙。

2、bin packing问题

文中使用的研究方法是将FunSearch用于进化一个启发式函数，该函数为每个箱子分配一个优先级分数，并选择分数最高的箱子来放置物品。FunSearch发现的启发式函数在模拟数据上的表现超过了常用的first fit和best fit启发式函数，并且在不同大小的问题实

例上都有良好的泛化能力。

```python
def heuristic(item: float, bins: np.ndarray) -> np.ndarray:
  """Online bin packing heuristic discovered with FunSearch."""
  score = 1000 * np.ones(bins.shape)
  # Penalize bins with large capacities.
  score -= bins * (bins-item)
  # Extract index of bin with best fit.
  index = np.argmin(bins)
  # Scale score of best fit bin by item size.
  score[index] *= item
  # Penalize best fit bin if fit is not tight.
  score[index] -= (bins[index] - item)**4
  return score
```

Fig. 6 | Example of a short online bin packing heuristic discovered byFunSearch for the OR dataset.

讨论与展望

FunSearch通过结合LLMs的创造力和进化算法的策略，在解决复杂问题和发现新知识方面的潜力。这种方法不仅能够扩展到更大的问题实例，而且能够产生可解释的解决方案，为研究人员提供可操作的见解，并推动科学发现的良性循环。

> LLMs作为创新源泉：FunSearch中的LLM并不依赖于对问题的深入理解，而是作为一个提供多样化、语法正确程序的源泉，这些程序偶尔包含一些有趣的想法。当LLM被限制在算法的关键部分进行操作，并且有一个程序骨架时，它提供的改进建议虽然只是边际上的，但通过与进化算法结合，最终能够在开放问题上发现新知识。
> 程序空间的搜索：FunSearch的有效性关键在于它在程序空间而不是直接在解决方案空间中搜索。这意味着FunSearch寻找的是生成解决方案的程序，而不是解决方案本身。这种方法不仅能够扩展到更大的实例，而且发现的程序通常比原始解决方案更具可解释性。
> Kolmogorov复杂性：FunSearch在某种程度上试图找到具有低Kolmogorov复杂性的解决方案，即用最短的计算机程序来产生给定对象。这与传统搜索方法的归纳偏差非常不同，后者没有这种压缩偏差。
> 对称性和简洁性：FunSearch倾向于发现对称的解决方案，因为这些解决方案更加简洁，需要较少的信息来指定。这种偏好反映了LLMs的自然偏见，即倾向于输出具有类似人类代码特征的代码。
> 问题选择：FunSearch最适合解决那些具有有效评估器、丰富的评分反馈，并且能够提供一个带有待进化部分的骨架的问题。
> 未来展望：随着LLMs的快速发展，预计FunSearch将变得更加有效，能够解决更广泛的问题的。自动定制的算法可能很快就会成为常规做法，并在现实世界的应用中得到部署。

实施细节

1、Distributed System

FunSearch被实现为一个分布式系统，包括三种类型的工作器：程序数据库、采样器和评估器。程序数据库负责存储初始用户提交的程序以及从评估器接收的所有程序。采样器负责执行LLM推断步骤，它们会反复从程序数据库请求提示，并从每个提示生成多个样本。由LLM生成的样本（即程序）被发送到评估器，后者通过在用户指定的输入上执行程序并使用“评估”函数评估输出来对程序进行评分。正确的程序则被发送到程序数据库中存储。FunSearch的每个组件都提供了Python代码和伪代码 。

2、Prompt Building

当请求提示时，程序数据库会抽样k个程序以鼓励LLM合并来自它们的创意（通常设置k=2）。程序根据得分从小到大排序，从版本0开始标记（例如'v0'）。使用这些k个程序构建提示，其中包括将“priority”函数剥离并替换为抽样的程序，然后在提示末尾添加一个没有主体的“priority_v2”函数，LLM将负责完成该函数的主体 。

3、Evolutionary Method and Program Selection

FunSearch的另一个关键特性是用于程序数据库中程序群体进化的方法，以及程序选择的方法：即当请求提示时，程序数据库如何抽样程序。为此，使用了岛屿模型，这是一种并行遗传算法。具体来说，将群体分成m个独立的组或岛屿，每个岛屿都使用用户提供的初始程序进行初始化，并分别进行进化。在构建提示时，首先均匀抽样一个岛屿，然后从该岛屿中抽样k=2个程序。由LLM基于该提示生成的程序后来将被存储在同一个岛屿中。每4小时，我们会丢弃m/2个岛屿中的所有程序，这些岛屿的最佳实例得分最低。然后，这些岛屿中的每一个都被重新播种，使用从存活的m/2个岛屿中随机选择的一个岛屿中得分最高的程序 。

4、Robustness

由于LLM抽样和进化程序中的随机性，重复实验可能会产生不同的结果。对于一些问题，例如通过可接受集问题的帽子集问题和在线箱包装，FunSearch的每次运行都超过了基线，差异仅在于差异的大小。对于其他问题，可能需要多次独立重复实验才能改进以前的最好结果。例如，在n=8维直接构造帽子集的问题尤其具有挑战性，只有140次实验中的4次发现了大小为512的帽子集。

启示与展望

FunSearch的成功展示了将先进的人工智能技术与数学和计算机科学问题结合的潜力，这种跨学科合作模式能够带来突破性的成果。它不仅能够解决传统领域的问题，还能在解决问题时展现出创造性思维，这是AI技术的一个重要进步。FunSearch通过迭代过程和自动化评估系统显著提高了研究效率，减少了人力需求，为未来研究提供了新的方法论。

此外，FunSearch生成的程序不仅解决了问题，还提供了解决方案的可解释性，这对于科学研究和实际应用都至关重要。基于这些优势，FunSearch的方法有望应用于更广泛的科学和工程问题，如优化问题、算法设计和机器学习模型的改进等。文章中提到，为了防止错误和幻觉，采用了"Evaluate"函数来评估LLM生成的结果，这在一定程度上减少了LLM出错的可能性，这是我们在未来的LLM应用中可以借鉴的。

未来，我们可以在FunSearch的基础上探索以下研究方向：

> 利用FunSearch模式解决组合数学领域之外的问题。
> 进一步优化FunSearch算法，提高计算效率，以处理更大规模的问题。
> 探索更有效的人机协作模式，使研究人员能更好地理解和利用AI生成的解决方案，推动科学发现的进程。
> 利用FunSearch等工具开发新的教育和培训方法，帮助学生和研究人员更好地理解复杂科学概念和解决问题的策略。
> 随着AI在科学研究中的作用日益增强，研究其伦理和社会影响，确保技术发展符合人类社会的长远利益。