AlphaEvolve: 面向科学与算法探索的编码代理

亚历山大·诺维科夫*、吴岸*、马文·艾森伯格*、埃米利安·杜邦*、黄柏森*、亚当·索尔特·瓦格纳*、谢尔盖·希罗博科夫*、博里斯拉夫·科兹洛夫斯基*、弗朗西斯科·J·R·鲁伊斯、阿巴斯·梅赫拉比安、M·帕万·库马尔、阿比盖尔·西、斯瓦拉特·乔杜里、乔治·霍兰德、亚历克斯·戴维斯、塞巴斯蒂安·诺沃辛、普什米特·科利和马特杰·巴洛格* Google DeepMind[1]

在本白皮书中，我们介绍了***AlphaEvolve***——一种革命性的编码智能体，它能显著提升最先进大语言模型在攻克开放性科学难题或优化关键计算基础设施等高挑战性任务上的能力。***AlphaEvolve***通过协调大语言模型的自主工作流，直接对代码进行修改以实现算法优化。采用进化策略的***AlphaEvolve***持续接收来自一个或多个评估者的反馈，通过迭代改进算法，有望催生新的科学发现与实践突破。我们通过解决一系列重要计算问题，验证了该方法的广泛适用性。在优化谷歌大规模计算堆栈关键组件时，***AlphaEvolve***为数据中心开发出更高效率的调度算法，在硬件加速器电路设计中实现了功能等效的简化方案，并加速了支撑***AlphaEvolve***运行的大语言模型训练过程。更引人注目的是，***AlphaEvolve***在数学与计算机科学领域发现了一系列可证明正确的新算法，其性能超越现有最优解，极大拓展了自动化发现方法的应用边界（Romera-Paredes等，2023）。其中，***AlphaEvolve***开发的搜索算法找到了仅需48次标量乘法即可完成4×4复值矩阵乘法的运算流程——这是56年来该场景下对Strassen算法的首次改进。我们相信***AlphaEvolve***及同类编码智能体，将在推动科学与计算领域的问题解决方案优化方面产生深远影响。

1. 引言

发现新的高价值知识，如做出新颖的科学发现或开发具有商业价值的算法，通常需要经历一个漫长的构思、探索、对无望假设的反复推敲、实验和验证过程。近期，利用大型语言模型（LLMs）来自动化这一流程的重要环节引起了广泛关注。成功的希望源自近期LLMs令人惊叹的能力[31,73]——它们能通过测试时计算增强自身性能，以及结合语言生成与行动的*agents*技术的兴起[85,111]。这些进步已在一系列成熟基准测试中提升了表现，并加速了假设生成[33]、实验设计[7,42]等探索导向型任务。然而，要让LLM流程完全实现全新科学或实用发现，仍面临挑战。

在本白皮书中，我们介绍了一种名为*AlphaEvolve*的LLM代码超级优化代理，
that 通过结合进化计算和基于LLM的方法来应对这一挑战
ode生成。*AlphaEvolve*专注于广泛的科学与工程领域

[1]See Acknowledgments and Author information section. *Equal contributions.

discovery问题中，发现候选对象可被自动评估。其将候选对象（例如新数学实体或实用启发式方法）表示为算法，并利用一组LLM生成、批判及演化此类算法池。LLM导向的演化过程通过代码执行与自动化评估实现落地。该评估机制使*AlphaEvolve*能够规避基础LLM[43]产生的任何错误建议。

*AlphaEvolve*中的进化过程利用了现代LLM响应反馈的能力，从而能够发现与初始候选池在语法和功能上存在显著差异的候选方案。该过程既适用于以发现新算法为本质目标的问题，也广泛适用于那些目标解本身并非算法、但算法能*describe*如何构建或找到该解的各类问题。对于后一种情况，发现算法仅是工具性目标，但相比直接搜索解决方案，这被证明是一种出人意料的有效策略[80]。

将进化方法与编码大型语言模型（LLMs）相结合的理念，此前已在多种专门场景中得到探索。特别是，*AlphaEvolve*作为*FunSearch*[80]的重大改进版本（见表1），其通过LLM引导的进化来发现启发式规则，旨在构建新颖的数学对象或驱动在线算法的运行。此外，类似方法还被应用于模拟机器人策略发现[55]、符号回归[34, 86]以及组合优化启发式函数合成[61]等任务。与这些系统不同，*AlphaEvolve*利用最先进的（SOTA）LLMs来演化实现跨多函数、多组件的复杂算法的大规模代码片段，从而在规模和通用性上显著超越了前代系统。

FunSearch [80]	*AlphaEvolve*
evolves single function	evolves entire code file
evolves up to 10-20 lines of code	evolves up to hundreds of lines of code
evolves code in Python	evolves any language
needs fast evaluation (≤ 20min on 1 CPU)	can evaluate for hours, in parallel, on accelerators
millions of LLM samples used	thousands of LLM samples suffice
small LLMs used; no benefit from larger	benefits from SOTA LLMs
minimal context (only previous solutions)	rich context and feedback in prompts
optimizes single metric	can simultaneously optimize multiple metrics

表1 | *AlphaEvolve*与我们先前代理的能力与典型行为对比

虽然使用自动化评估指标具有*AlphaEvolve*这一关键优势，但它同时也是一种局限——特别是将需要人工实验的任务排除在我们的研究范围之外。由于数学、计算机科学和系统优化领域的问题通常允许采用自动化评估指标，因此我们在*AlphaEvolve*上的工作主要聚焦于这些领域。具体而言，我们运用*AlphaEvolve*在算法设计和构造性数学中的若干著名开放问题上取得进展，同时也优化了谷歌大规模计算堆栈中关键层的性能。

在数学中，我们研究大量开放性问题，通过根据给定数学定义发现比所有已知构造（对象）具有更优性质的新构造，从而推动问题进展。我们将*AlphaEvolve*应用于超过50个此类问题，其中75%的案例与已知最优构造~相匹配（许多情况下这些构造可能已达最优解）。在~20%的问题中，*AlphaEvolve*超越了现有技术，发现了可证明更优的新构造。这包括对Er d s提出的最小重叠问题集[24]的改进，以及对11维空间接吻数问题[8,30]构造的优化。

最后，我们将*AlphaEvolve*应用于跨越谷歌计算堆栈不同层次的四个工程问题：为谷歌集群管理系统发现调度启发式方法，优化用于训练LLM的矩阵乘法内核，优化TPU内部使用的算术电路，以及优化Transformer中注意力机制的运行时性能。由于这些组件长期反复运行，任何改进都具有极高的价值。

2# 1.0.0 (2021-10-20)### Features* **init:** 初始化项目 ([0c0a0e8](https://github.com/liuweiqiang1

AlphaEvolve ``` 是一种编码代理，它协调一个自主的计算流程，包括对大型语言模型（LLM）的查询，并生成解决用户指定任务的算法。从高层次来看，该协调流程是一种进化算法，逐步开发出能提升任务相关自动评估指标得分的程序。图1展示了*AlphaEvolve*的概览，图2则提供了更详细的视图。```

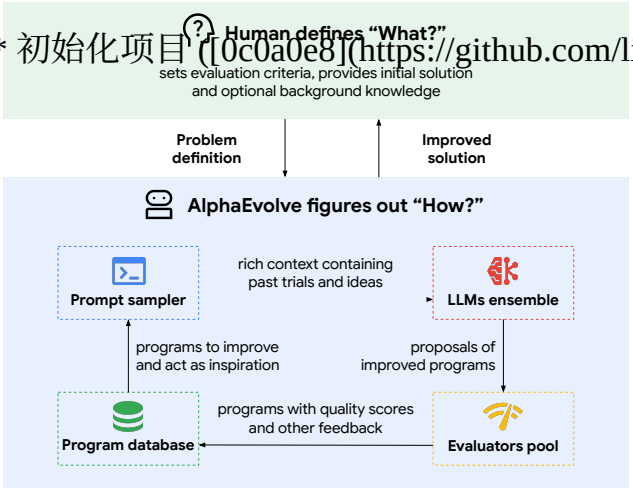
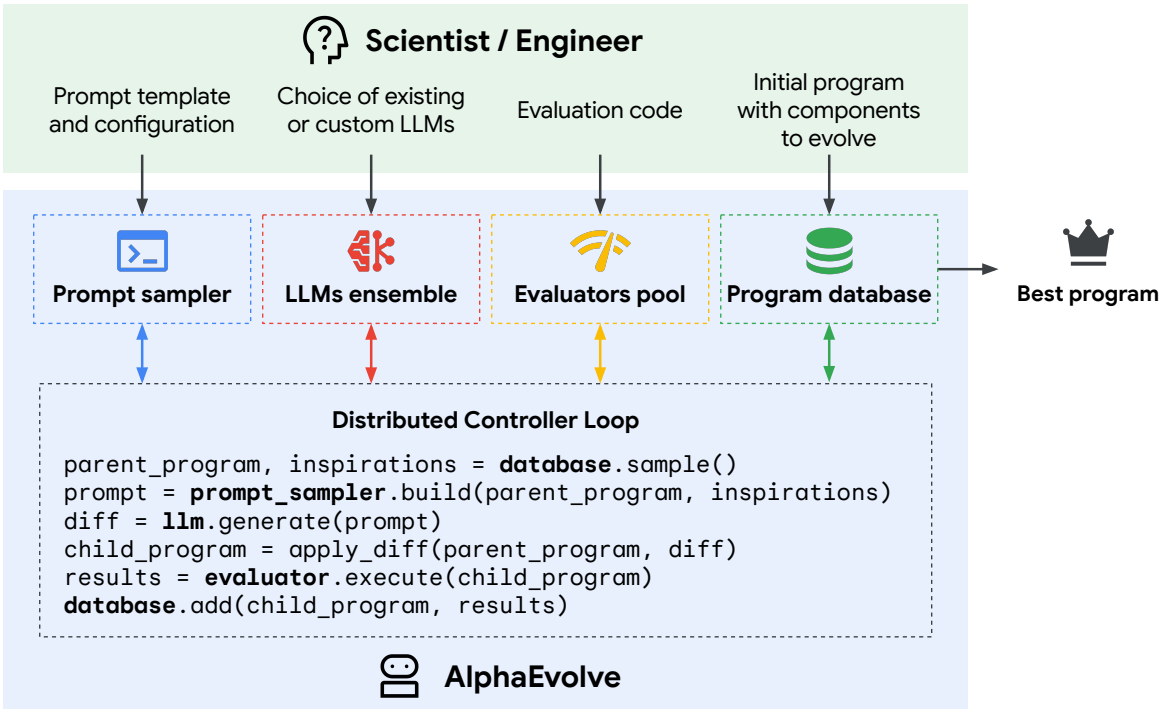


图1 | *AlphaEvolve* 高级概述。

2.1. 任务规范

评估。由于*AlphaEvolve*处理的问题具有机器可评分的解决方案，用户必须提供一种自动评估生成解决方案的机制。该机制采用一个函数h的形式，将解决方案映射到一组标量评估指标上。按照惯例，这些指标需最大化。在我们当前的设置中，h通常被实现为

[2]These discovered algorithms as well as our other new mathematical results can be found at `https://colab.research.google.com/github/google-deepmind/alphaevolve_results/blob/master/mathematical_results.ipynb`.



作为一个Python函数，名为`evaluate`，具有固定的输入/输出签名，返回一个标量字典。

根据应用场景的不同，执行此函数可能仅需单台设备数秒，也可能触发大规模计算。在数学问题中，函数h通常极为简单。例如，当需要寻找满足给定属性的最大可能图时，h会调用演化生成的代码来创建图结构，验证属性是否成立，随后仅返回图的规模作为评分值。更复杂的情况下，函数h可能涉及执行演化搜索算法，或训练并评估机器学习模型。

为了更好地支持代码库中多个组件的协同演进，*AlphaEvolve* 版本提供了一种输入式API机制，允许将代码块标注为由系统自动演进（具体示例见图3a）。该设计通过引入特殊标记（`# EVOLVE-BLOCK-START`、`#` 与 `EVOLVE-BLOCK-END`）作为代码注释，只需极少量修改即可与现有代码库集成，显著降低了技术迁移成本。

在此类进化块中，用户提供的任何代码都将作为初始解决方案，由*AlphaEvolve*进行改进，其余代码则构成一个框架，将进化后的片段整合在一起，以便能够从`evaluate`调用。虽然这一初始实现必须是完整的，但它可以是基础性的——例如，由返回适当类型常量的单行函数组成。

在选择抽象方式上的灵活性。*AlphaEvolve*可以以截然不同的方式应用于同一问题——尤其是当进化程序并非最终输出，而是发现解决方案的手段时。例如，*AlphaEvolve*可以：以原始字符串形式进化解决方案（如经典进化算法）；进化一个明确形式的函数，该函数规定如何从零构建解决方案（如文献[80]采用的方法）；进化定制搜索算法以在固定计算预算内寻找答案；甚至协同进化中间解决方案与搜索算法，使每个搜索算法专门针对特定中间结果进行优化改进。

我们发现，不同层次的抽象适用于不同的问题。例如，我们假设，对于具有高度对称解的问题，进化构造函数更为有利，因为这些函数往往更简洁[80]；而对于非对称解的问题，进化定制化的搜索算法效果更佳。

2.2. 提示采样

由于*AlphaEvolve*利用了最先进的LLM技术，它支持多种类型的自定义，并能提供长上下文作为主要进化提示的一部分。该提示包含从程序数据库中抽样出的多个先前发现的解决方案，以及关于如何对特定解决方案提出修改的系统指令。除了这些关键要素外，用户还可以通过以下方式进一步根据自身需求定制提示。

- *Explicit context*: 关于所解决问题的详细信息，如固定的人工编写指令、方程、代码片段或相关文献（例如pdf文件）。
- *Stochastic formatting*模板占位符，附带人工提供的备选方案以增加多样性，通过单独的配置文件中提供的概率分布进行实例化。公式符号{v*}保持不变。
- *Rendered evaluation results*: 通常这包括一个程序、该程序执行的结果以及由`evaluate`函数分配的分数。
- *Meta prompt evolution*: 由大型语言模型在额外的提示生成步骤中自行提出的指令和上下文，与解决方案程序类似，在单独的数据库中共同演化。

2.3. 创意生成

为了推动进化过程，*AlphaEvolve*利用了最先进大语言模型（SOTA LLMs）的能力，其主要作用是消化关于先前开发解决方案的信息，并提出新的、多样化的改进方案方法。尽管*AlphaEvolve*与模型无关，但在消融实验中我们观察到，随着底层大语言模型的提升，*AlphaEvolve*的表现也持续改善（参见第4节）。

```
# EVOLVE-BLOCK START
"""Image classification experiment in jaxline."""

import jax
...
# EVOLVE-BLOCK-END

...

# EVOLVE-BLOCK-START
class ConvNet(hk.Module):
  def __init__(self, num_classes): ...
  def __call__(self, inputs, is_training): ...


def sweep():
  return hyper.zipit([...])
# EVOLVE-BLOCK-END

...

def evaluate(eval_inputs) -> dict[str, float]:
  ...
  return metrics
```

(

```
The current model uses a simple ResNet architecture with only
three ResNet blocks. We can improve its performance by
increasing the model capacity and adding regularization. This
will allow the model to learn more complex features and
generalize better to unseen data. We also add weight decay to
the optimizer to further regularize the model and prevent
overfitting. AdamW is generally a better choice than Adam,
especially with weight decay.


<<<<<<< SEARCH
    self._block1 = ResNetBlock(num_channels)
    self._block2 = ResNetBlock(num_channels * 2, stride=2)
    self._block3 = ResNetBlock(num_channels * 4, stride=2)
=======
    self._block1 = ResNetBlock(num_channels)
    self._block2 = ResNetBlock(num_channels, stride=1)
    self._block3 = ResNetBlock(num_channels * 2, stride=2)
    self._block4 = ResNetBlock(num_channels * 2, stride=1)
    self._block5 = ResNetBlock(num_channels * 4, stride=2)
    self._block6 = ResNetBlock(num_channels * 4, stride=1)
>>>>>>> REPLACE
<<<<<<< SEARCH
  def optimizer(self, learning_rate):
    return optax.adam(learning_rate)
=======
  def optimizer(self, learning_rate):
    return optax.adamw(learning_rate, weight_decay=1e-4)
>>>>>>> REPLACE
```

(c)

```
Act as an expert software developer. Your task is to iteratively
improve the provided codebase. [...]

- Prior programs

Previously we found that the following programs performed well
on the task at hand:

top_1_acc: 0.796; neg_eval_log_loss: 0.230; average_score: 0.513

"""Image classification experiment in jaxline."""
[...]
class ConvNet(hk.Module):
  """Network."""

  def __init__(self, num_channels=32, num_output_classess=10):
    super().__init__()
    self._conv1 = hk.Conv2D(num_channels, kernel_shape=3)
    self._conv2 = hk.Conv2D(num_channels * 2, kernel_shape=3)
    self._conv3 = hk.Conv2D(num_channels * 4, kernel_shape=3)
    self._logits_module = hk.Linear(num_output_classes)
[...]


- Current program

Here is the current program we are trying to improve (you will
need to propose a modification to it below).

top_1_acc: 0.862; neg_eval_log_loss: 0.387; average_score: 0.624

"""Image classification experiment in jaxline."""
[...]
class ConvNet(hk.Module):
  """Network."""

  def __init__(self, num_channels=32, num_output_classes=10):
    super().__init__()
    self._conv1 = hk.Conv2D(num_channels, kernel_shape=3)
    self._block1 = ResNetBlock(num_channels)
    self._block2 = ResNetBlock(num_channels * 2, stride=2)
    self._block3 = ResNetBlock(num_channels * 4, stride=2)
    self._logits_module = hk.Linear(num_output_classes)
[...]

SEARCH/REPLACE block rules:
[...]

Make sure that the changes you propose are consistent with each
other. For example, if you refer to a new config variable
somewhere, you should also propose a change to add that
variable.

Example:
[...]

Task
Suggest a new idea to improve the code that is inspired by your
expert knowledge of optimization and machine learning.

Describe each change with a SEARCH/REPLACE block.
```

(b)

输出格式。当*AlphaEvolve*要求LLM修改现有代码时，尤其是在较大的代码库中，它要求以特定格式的差异块序列提供修改内容：

```
<<<<<<< SEARCH
  # Original code block to be found and replaced
=======
  # New code block to replace the original
>>>>>>> REPLACE
```

这里，`<<<<<<< SEARCH` 和 `=======` 之间的代码是当前程序版本中需要匹配的精确片段。`=======` 和 `>>>>>>> REPLACE` 之间的代码则是将替换原片段的新内容。这种方式允许对代码的特定部分进行有针对性的更新。

在需要演进的代码非常简短，或完全重写更为
适当的小修改，*AlphaEvolve* 可以配置为指示LLM
哦`将代码块直接完整输出，而非使用差异格式。```

采用的模型。*AlphaEvolve* 采用了大型语言模型的集成方法。具体而言，我们结合使用了Gemini 2.0 Flash和Gemini 2.0 Pro。这种集成策略使我们能够在计算吞吐量与生成解决方案的质量之间取得平衡。Gemini 2.0 Flash凭借其较低的延迟，能够实现更高的候选生成速率，从而在单位时间内探索更多想法。与此同时，能力更强的Gemini 2.0 Pro则偶尔提供更高质量的建议，这些建议能显著推进进化搜索，并可能带来突破性进展。通过这种策略性组合，我们既最大化评估想法的数量，又保留了由更强大模型驱动实现重大改进的潜力，从而优化了整个发现过程。

2.4. 评估

为了追踪*AlphaEvolve*的进展并筛选哪些想法在未来的迭代中传播，由大语言模型提出的每个新解决方案都会自动接受评估。理论上，这一过程等同于简单地对生成的解决方案执行用户提供的评估函数h。实际上，*AlphaEvolve*支持可选机制，以使这一评估更加灵活高效：

- *Evaluation cascade (hypothesis testing)***: 用户可以指定难度递增的测试用例集合，使得新解决方案仅当在早期所有阶段均取得足够有前景的结果时，才会在下一阶段接受评估。这有助于更快地剔除潜力较低的方案。此外，新解决方案在进入主测试用例前会先进行小规模评估，以便尽早过滤存在缺陷的程序。**

- *LLM-generated feedback*在一些应用中，理想的解决方案具有某些特征，这些特征难以在用户提供的评估函数中精确捕捉。

*h*例如，所发现程序的简单性。这些特性可以通过单独的LLM调用来分级，并添加到评分字典中以引导进化，或者在未满足标准时用于丢弃解决方案。

- *Parallelized evaluation**AlphaEvolve*的样本效率使得花费约100个计算小时来评估任何新解决方案成为可能。然而，除非通过并行化单个评估来缩短其实际耗时，否则这会降低新一代解决方案的生成速度，从而限制进化算法连续应用多次变异的能力。在众多应用中，评估过程天然具备高度并行性（例如，从多个随机化初始点同时运行搜索算法），这使*AlphaEvolve*能够通过异步调用评估集群来分配此类工作。

多目标评分。*AlphaEvolve*支持优化多个用户提供的评分，即进化出能在一个或多个评估指标下获得高分的对象。这既具有内在价值，也具有工具价值。虽然在许多应用中，我们确实关心为多个评估指标开发解决方案（或同时在所有指标上表现强劲的单一方案），但我们发现，即使某个特定指标尤为关键，针对多指标进行优化往往也能提升目标单项指标的结果。这可能是因为，在不同评估标准下表现出色的程序通常具有独特的结构或逻辑——通过将这些多样化高性能程序（每个都代表“优秀”的不同定义）的示例融入提供给语言模型的提示中，我们能够激发生成更多样化的候选解决方案，从而增加发现对目标指标极为有效的新颖方法的机会。

2.5. 进化

在其进化过程中，*AlphaEvolve*不断生成越来越多的解决方案，并附有评估结果（分数和程序输出）。这些解决方案存储在一个进化数据库中，其主要目标是在未来世代中优化地重现先前探索过的想法。设计此类数据库的一个关键挑战在于平衡探索与利用，既要持续改进最佳程序，又要保持多样性以鼓励对整个搜索空间的探索。在*AlphaEvolve*中，进化数据库实现了一种算法，该算法灵感来源于MAP精英算法[71]与基于岛屿的种群模型[80, 94]的结合。

2.6. 分布式流水线

AlphaEvolve 其实现为一个异步计算流水线（使用`asyncio` Python库），其中大量计算并发运行，每个计算在其下一步依赖于其他未完成计算的结果时，会阻塞（等待）。具体而言，该异步流水线由控制器、LLM采样器和评估节点组成。整个流水线针对吞吐量（而非单个特定计算的速度）进行优化，以在给定总体计算预算内最大化可提出和评估的创意数量。

$\langle m, n, p \rangle$	best known [reference]	*AlphaEvolve*
$\langle 2, 4, 5 \rangle$	33 [41]	**32**
$\langle 2, 4, 7 \rangle$	46 [90]	**45**
$\langle 2, 4, 8 \rangle$	52 [90]	**51**
$\langle 2, 5, 6 \rangle$	48 [90]	**47**
$\langle 3, 3, 3 \rangle$	23 [50]	23
$\langle 3, 4, 6 \rangle$	56 [47]	**54**
$\langle 3, 4, 7 \rangle$	66 [88]	**63**
$\langle 3, 4, 8 \rangle$	75 [88]	**74**
$\langle 3, 5, 6 \rangle$	70 [47]	**68**
$\langle 3, 5, 7 \rangle$	82 [88]	**80**
$\langle 4, 4, 4 \rangle$	49 [92]	**48**
$\langle 4, 4, 5 \rangle$	62 [46]	**61**
$\langle 4, 4, 7 \rangle$	87 [90]	**85**
$\langle 4, 4, 8 \rangle$	98 [92]	**96**
$\langle 4, 5, 6 \rangle$	93 [47]	**90**
$\langle 5, 5, 5 \rangle$	93 [70]	93

表2 | 计算一个$m \times n$矩阵与一个$n \times p$矩阵乘积所需标量乘法次数的上界；等价于具有参数$\langle m, n, p \rangle$的对应三维张量的秩。除此处展示的示例外，对于所有参数$m, n, p \leq 5$、*AlphaEvolve*，该算法要么匹配要么超越了已知最佳解决方案，并提供了精确算法（完整结果参见附录中的表3）。对于⟨3,4,7⟩、⟨4,4,4⟩以及⟨4,4,8⟩，由*AlphaEvolve*发现的算法采用了复数值乘法，可用于复数或实数值矩阵的精确乘法运算。本表展示的分解可在随附的Google Colab中找到。

3. 结果

3.1. 通过寻找张量分解新算法实现更快的矩阵乘法

从加速机器学习计算到实现逼真的计算机图形学，矩阵乘法作为一项基础运算，支撑着计算机科学领域众多关键算法与应用。自Strassen的开创性研究[92]以来，人们已认识到任何两个矩阵相乘的算法都可表示为给定三维张量分解为秩一张量之和的过程。该分解的秩（项数）直接决定了计算矩阵乘积所需的标量乘法次数。因此，要开发更快的矩阵乘法算法，关键在于寻找特定张量的低秩分解。针对这一难题，研究者尝试了多种方法——从专用交替最小二乘求解器[90]到深度强化学习[25]再到定制搜索算法[46]；然而即便经过数十年努力，即便是两个3×3矩阵相乘这样简单的案例，其可达成的最小秩仍属未知，这充分体现了该问题的复杂性。

从问题描述和一个标准的基于梯度的算法（包括一个初始化器、一个重建损失函数和一个Adam优化器[48]），*AlphaEvolve*是

能够开发出超越现有方法的复杂张量分解算法。为了评估每个进化程序，我们选择一组矩阵乘法目标并运行该算法，使用第2.4节描述的评估级联以多个随机种子初始化。性能通过每个目标上达到的最佳（最低）秩以及达到该秩的种子比例来衡量，为*AlphaEvolve*提供攀爬信号。为确保分解的精确性并避免潜在数值误差，在评估时我们将每个元素四舍五入至最接近的整数或半整数；同时，为鼓励算法生成接近整数的解，我们在LLM提示中以自然语言形式加入这一要求。

如图4所示，*AlphaEvolve*对初始程序进行了重大修改，引入了多项原创性构想以设计出更优算法。虽然表2中的大部分结果（包括⟨4, 4, 4⟩）均源自简单的初始程序，但我们发现针对某些参数，在初始程序中植入自主构想（例如为评估函数添加随机性要素或采用进化算法）能进一步提升性能，这凸显出研究人员与*AlphaEvolve*开展科学协作的可能性。

3.2. 为广泛的开放性数学问题寻找定制搜索算法

数学研究的一个重要前沿在于发现那些在某种度量下具有最优或接近最优特性的对象或{v*}。例子从寻找几何形状的密集排布[28]，到识别满足特定组合或分析约束的函数或集合（如[38, 39, 68, 101]）不等。进展往往依赖于找到一个超越所有已知范例的单一构造，从而为最优值确立新的下界或上界。我们证明{v*}是探索这些问题固有广阔搜索空间的有力工具，能成功应对一系列多样化的开放性数学挑战。

```
@@ -45,9 +45,14 @@
   # EVOLVE-BLOCK-START
   def _get_optimizer(self) -> optax.GradientTransformation:
     """Returns optimizer."""
-    return optax.adam(self.hypers.learning_rate)
+    return optax.adamw(
+        self.hypers.learning_rate, weight_decay=self.hypers.
    weight_decay
+    )

   def _get_init_fn(self) -> jax.nn.initializers.Initializer:
     """Returns initializer function."""
-    return initializers.normal(0.0, self.hypers.init_scale, jnp.
    complex64)
+    # Initialize with a smaller scale to encourage finding low-rank
    solutions.
+    # Increase scale slightly for better exploration.
+    scale = self.hypers.init_scale
+    return initializers.normal(0 + 1j * 0, scale * 0.2, jnp.complex64
    )
```

```
@@ -91,13 +156,86 @@
     """Computes (batched) loss on learned decomposition."""
     # Compute reconstruction loss.
     rec_tensor = self._decomposition_to_tensor(decomposition)  # (B,
    N, M, P)
...
+    # Discretization loss (encourage entries to be multiples of 1/2
    or integer).
+    def dist_to_half_ints(x):
...
+
+    def dist_to_ints(x):
...
+    discretization_loss = 0.0
+    for factor in decomposition:
+      discretization_loss += jnp.mean(dist_to_half_ints(factor))
+      discretization_loss += jnp.mean(dist_to_ints(factor))
+
+    discretization_loss /= (
+        len(decomposition) * 2
+    )  # average across all factors and loss components
+
+    discretization_weight = self._linear_schedule(
+        global_step, start=0.0, end=self.hypers.discretization_weight
+    )
+
+    # Cosine annealing for half-integer loss.
+    cycle_length = self.config.training_steps // 4  # Number of steps
    per cycle
+    cycle_progress = (
+        global_step % cycle_length
+    ) / cycle_length  # Normalized progress within the current cycle
    [0, 1)
+    half_int_multiplier = (1 + jnp.cos(jnp.pi * cycle_progress)) / 2
+    half_int_multiplier = (
+        1 - self.hypers.half_int_start
+    ) * half_int_multiplier + self.hypers.half_int_start
+
+    total_loss = (
+        rec_loss
+        + discretization_weight * discretization_loss *
    half_int_multiplier
+    )
...
```

```
@@ -117,6 +255,18 @@
   return hyper.zipit([
-      hyper.uniform('init_scale', hyper.interval(0.2, 1.5)),
-      hyper.uniform('learning_rate', hyper.interval(0.05, 0.3)),
+      hyper.uniform('init_scale', hyper.interval(0.1, 1.0)),
+      hyper.uniform('learning_rate', hyper.interval(0.01, 0.2)),
+      hyper.uniform('discretization_weight', hyper.interval(0.0, 0.1)
    ),
+      hyper.uniform('hallucination_prob', hyper.interval(0.0, 0.2)),
+      hyper.uniform('hallucination_scale', hyper.interval(0.0, 0.2)),
+      hyper.uniform('noise_std', hyper.interval(0.0, 0.01)),
+      hyper.uniform('target_noise_std', hyper.interval(0.0, 0.01)),
+      hyper.uniform('weight_decay', hyper.interval(0.00001, 0.001)),
+      hyper.uniform('clip_min', hyper.interval(0.0, 0.5)),
+      hyper.uniform('clip_max', hyper.interval(1.0, 3.0)),
+      hyper.uniform('large_value_penalty_weight', hyper.interval(0.0,
     0.01)),
+      # Add noise to the gradient to aid in exploration.
+      hyper.uniform('grad_noise_std', hyper.interval(0.0, 0.001)),
+      hyper.uniform('half_int_start', hyper.interval(0.0, 1.0)),
   ])
 # EVOLVE-BLOCK-END
```

图4 | *AlphaEvolve*为发现更快的矩阵乘法算法所提出的修改方案。左侧展示了完整的差异对比（放大版本见图9a至9c），右侧则突出显示部分摘录内容。在本示例中，*AlphaEvolve*提出了跨多个组件的广泛修改，包括优化器与权重初始化（右上）、损失函数（中右）以及超参数扫描（右下）。这些改动意义重大，在进化过程中需要进行15次变异。

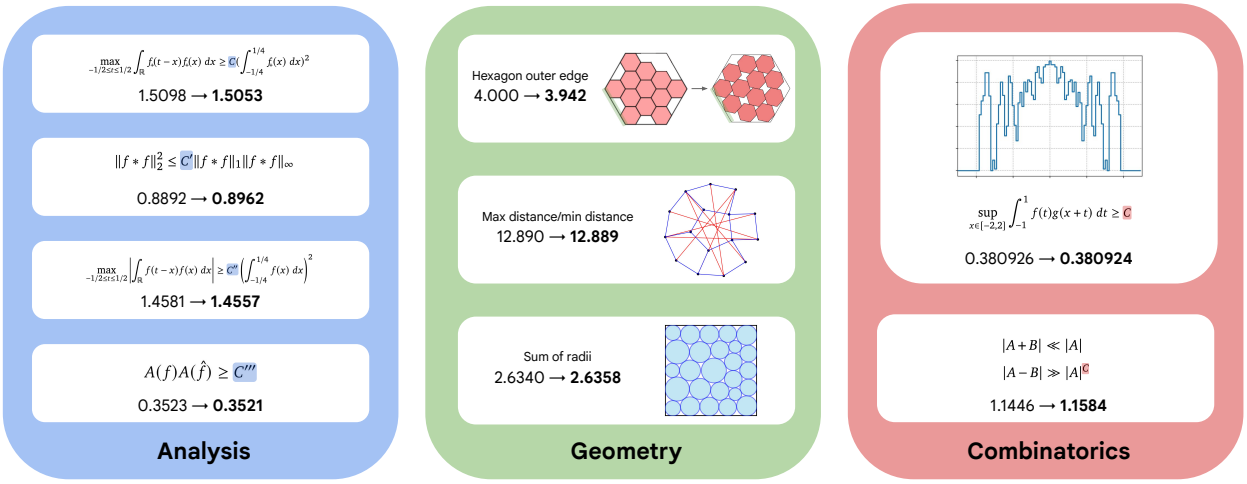


图5 | 通过*AlphaE-volve*发现的突破SOTA的数学构造示例。*AlphaEvolve*的多功能性使我们能够解决分析领域（自相关与不确定性不等式）、几何领域（堆积与最小/最大距离问题）以及组合数学领域（埃尔德什最小重叠问题与有限集合的和差问题）的各类难题。

这里采用的*AlphaEvolve*配置的一个显著优势在于其多功能性和快速应用能力。其核心方法专注于演化启发式搜索程序（下文详述），能够快速部署到各类数学构造问题和猜想上，通常相比传统的定制化方法，需要更少的初始问题特定专家调整。虽然深厚的数学洞察力自然有助于问题表述和搜索空间定义，但*AlphaEvolve*经常展现出通过识别问题领域中的微妙结构，自主发现有效搜索模式和攻击策略的能力。这使得能够高效、大规模地探索众多不同问题。

实现这些发现的关键方法创新在于*AlphaEvolve*能够进化*heuristic search algorithms*，而非直接进化构造本身。对于许多问题，尤其是那些目标函数评估速度快的场景（这在数学领域很常见），我们采用了迭代优化策略。每一代*AlphaEvolve*的任务是进化出一个代表搜索启发式的程序。该程序被赋予固定的时间预算（例如1000秒），并会看到之前最佳启发式找到的最优构造。其目标是利用这一起点和分配的时间，找到更优的构造。因此，进化过程筛选出那些擅长改进已高质量解决方案的启发式。最终构造通常是*AlphaEvolve*发现的一系列不同专业启发式共同作用的结果——早期启发式擅长从随机或简单初始状态实现大幅改进，后期启发式则精于对接近最优配置进行微调。这种多阶段自适应搜索策略的自动化发现难以手动复现，对超越现有技术起到了关键作用。

以下是*AlphaEvolve*带来新突破的一些问题的高层描述
r结果。完整的问题列表和详细信息见附录B。

- 分析

  > 自相关不等式。*AlphaEvolve*能够改进多个自相关不等式的最佳已知界。 > 不确定性原理。*AlphaEvolve*通过优化不确定性原理的构造[32]，为傅里叶分析中出现的问题生成了一种精细配置，从而略微改善了上界。

- > 组合数学与数论

  – 埃尔德什的最小重叠问题。*AlphaEvolve*为最小重叠问题[24]建立了一个新的上界，略微改进了之前的记录[39]。

- >几何与堆叠

  # 包装问题。*AlphaEvolve*在包装问题中取得了多项新成果，例如在形状内包装*N*个点以最大化最小距离与最大距离的比值，以最有效方式将各种多边形包装到其他多边形内，以及涉及避免小面积三角形的点集的海尔布隆问题变体[28]。

完整问题列表见附录B，而由*AlphaEvolve*发现的新构造可在随附的Google Colab中找到。关于这些问题及所用方法的更多示例和细节将在即将发表的论文中提供。这些发现大多源于外部数学家Javier Gomez Serrano和Terence Tao向我们提出的开放性问题，他们还就如何最佳地将这些问题表述为*AlphaEvolve*的输入提供了建议。这凸显了像*AlphaEvolve*这样的AI驱动发现引擎与人类数学专业知识之间协同合作的潜力。

33. 优化谷歌的计算生态系统 m

I除了前面几节介绍的科学应用之外，这里我们还将展示
] ow *AlphaEvolve* 已被用于提升关键任务基础设施的性能
] deliver real-world impact.

3.3.1. Improving data center scheduling

高效地将计算任务调度到机器集群上是一个关键的优化问题，尤其在谷歌数据中心由Borg[99]协调的大规模环境下。这一任务需要根据作业资源需求和机器容量，将作业分配到可用机器上。低效的分配会导致资源滞留：当某台机器因耗尽某类资源（如内存）而无法再接受任务，但其他资源（如CPU）仍有剩余时。提升调度效率可以回收这些滞留资源，从而在同等硬件条件下完成更多作业。

```python
def alpha_evolve_score(required, free):
  cpu_residual = required.cpu / free.cpu
  mem_residual = required.mem / free.mem

  return -1.0 * (cpu_residual + mem_residual +
                 mem_residual / cpu_residual +
                 cpu_residual / mem_residual)
```

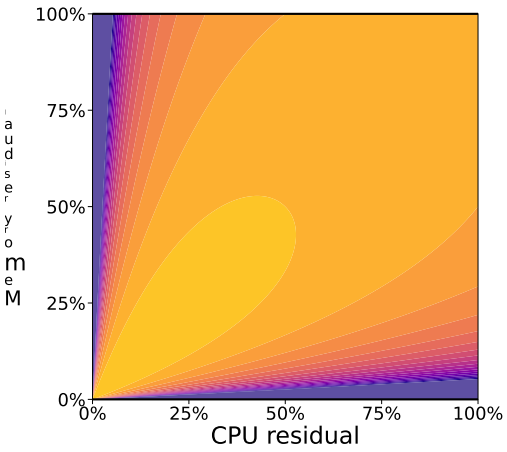


图6 | 左图：由*AlphaEvolve*发现的启发式函数，专为谷歌工作负载与容量定制。右图：启发式评分函数的可视化展示。黄色区域代表高分值，紫色区域则对应低分值。

计算资源的占用。这种恢复对于满足日益增长的计算需求而不成比例地增加资源消耗至关重要。此外，该问题极具挑战性，因为它将典型的工程难题（如可调试性和规模）叠加在经典的装箱难题之上。

我们通过将在线作业调度问题建模为一个包含两个变量的向量装箱问题来应对这一挑战。在此框架下，机器代表具有预设CPU和内存容量的箱子，而到达的作业则是带有特定资源需求的物品。启发式函数以待处理作业的CPU和内存需求，以及目标机器的CPU和内存可用性作为输入，输出该机器的优先级评分。随后，Borg调度器会根据启发式函数确定的优先级评分（兼顾其他目标），将待处理作业分配给得分最高的机器。由于该启发式仅影响已被Borg判定为可用且能运行每个待处理作业的机器排序，因此最终得出的调度决策在构造上本质正确。

*AlphaEvolve*的早期版本用于发现了一种极其简单却高效的启发式函数（如图6所示），该函数由生产环境中现有的启发式函数演化而来。我们利用数据中心的模拟器，基于谷歌整个计算集群历史工作负载与容量的快照，向*AlphaEvolve*提供反馈。通过在一组近期工作负载与容量的未见测试数据集上评估*AlphaEvolve*启发式函数的性能，我们确保了其泛化能力。观察到*AlphaEvolve*的启发式函数表现优于生产环境中的版本后，我们将*AlphaEvolve*的启发式函数全面部署至整个集群。谷歌全集群的部署后实测数据验证了模拟器结果，表明该启发式函数平均持续回收了谷歌全集群0.7%的计算资源——这些资源原本将处于闲置状态。选择*AlphaEvolve*而非深度强化学习方案，是因为其代码解决方案不仅带来更优性能，还在可解释性、可调试性、可预测性及部署便捷性上具有显著优势——这些特性对关键任务系统至关重要。

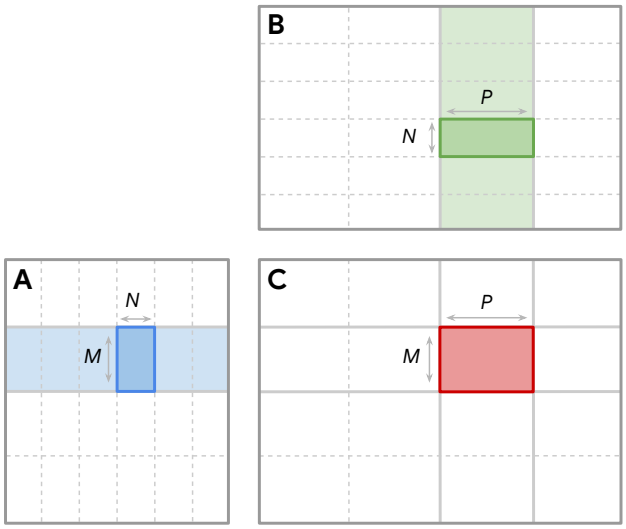


图7 | 矩阵乘积的平铺启发式问题可视化$AB = C$。创建一个能自动为所有输入形状选择正确分块大小（M、N、P）的启发式方法颇具挑战，因为这需要了解矩阵乘法单元的最优形状与内存容量、周边操作的内存需求、融合进内核的额外操作，以及底层编译器的复杂细节等诸多因素。

3.3.2. Enhancing Gemini kernel engineering

训练像Gemini这样的大型模型需要大量计算资源。Gemini基于JAX[9]构建，而Pallas是JAX的扩展，支持编写专为硬件加速器优化执行的高度定制化程序（内核）。因此，高效的Pallas内核对于优化Gemini训练性能至关重要。内核优化的关键环节是调整矩阵乘法运算的分块策略（参见图7），该技术通过将大规模矩阵计算分解为更小的子问题，更好地平衡计算与数据移动，从而加速整体运算。传统上，内核工程师依赖基于搜索的自动调优或人工启发式方法来确定不同输入形状的近优分块配置。基于搜索的调优会中断研究流程，需针对每次输入形状变化重新调整；而手动设计高效分块启发式则因复杂性成为主要工程瓶颈，这要求深入理解内核功能与硬件特性。高性能启发式的核心优势在于其能适应任意输入形状并保持优异性能。因此，为加速新兴硬件的高效内核设计并简化模型开发者的使用流程，我们致力于推动启发式生成过程的自动化。

}我们通过采用*AlphaEvolve*来优化用于训练Gemini的重要矩阵乘法核的平铺启发式方法，以应对这一挑战。目标是最小化该核的实际运行时间。*AlphaEvolve*通过提出候选代码，迭代探索并优化该核的平铺启发式方法，旨在在真实TPU加速器上针对各种输入形状最小化此运行时间。由于*AlphaEvolve*仅优化该核的平铺策略而不改变其功能，因此核的正确性得以保持。

其背后的数学运算。为了构建*AlphaEvolve*的训练和评估数据集，我们自动从内核用户那里收集真实的核输入形状。这些输入形状的一半构成训练集，为进化过程提供优化目标。剩余的输入形状则组成评估集，用于测试所得启发式方法的普遍适用性。

这种自动化方法使*AlphaEvolve*能够发现一种启发式策略，相比现有专家设计的启发式方法，该策略平均能为所有内核带来23%的速度提升，并使Gemini的整体训练时间相应减少1%。此外，*AlphaEvolve*的使用大幅缩短了内核优化时间，从需要数月专注的工程努力减少到仅需数天的自动化实验。这一加速加快了优化内核的部署速度，使内核工程师能够将专业知识投入到更具战略性、更高层次的优化问题上。更进一步，*AlphaEvolve*为自动化手动调优过程并改善Gemini内核使用的工效学提供了途径。由*AlphaEvolve*发现的平铺启发式策略已投入生产应用，直接提升了Gemini的训练效率以及Gemini团队的研究与工程推进速度。此次部署也标志着一个新颖的实例：Gemini借助*AlphaEvolve*的能力，优化了自身的训练流程。

3.3.3. Assisting in hardware circuit design

专用硬件，如谷歌的张量处理单元（TPUs），对于实现大规模运行现代AI系统所需的资源效率至关重要。然而，设计新的计算机芯片是一个复杂且耗时的过程，往往需要数年时间。寄存器传输级（RTL）优化作为该过程中的关键步骤，涉及手动重写硬件描述以提升功耗、性能和面积等指标，这需要由高技能工程师进行数月的迭代工作。

在这项工作中，*AlphaEvolve*面临挑战，需对矩阵乘法单元中一个已高度优化的关键TPU算术电路Verilog实现进行进一步优化。优化目标是在保持组件核心功能的同时，降低面积和功耗。关键在于，最终方案必须通过严格的验证方法，以确保修改后的电路功能正确性不受影响。*AlphaEvolve*成功找到一个简化代码重写方案，剔除了冗余位元，这一改动经TPU设计团队验证确保持正确性。虽然这一具体改进也被下游综合工具独立识别，但*AlphaEvolve*在RTL阶段的贡献展现了其在设计流程早期优化源码RTL的能力。

集成到即将推出的TPU中，这一改进代表了Gemini对TPU算术电路的首次直接贡献，通过*AlphaEvolve*实现，为未来贡献铺平道路。*AlphaEvolve*的一个关键优势在于，它直接以硬件工程师使用的标准语言Verilog传达建议的更改，从而建立信任并简化采用。这一早期探索展示了一种新颖方法，即利用LLM驱动的代码演进协助硬件设计，有望缩短上市时间。

3.3.4. Directly optimizing compiler-generated code

变压器架构[97]被广泛应用于现代神经网络中，从大语言模型到AlphaFold[1]均有采用。其核心计算是注意力机制[4]，而FlashAttention[21]成为最常用的实现方式。在我们的技术栈中，FlashAttention作为加速器内核通过Pallas实现，并由JAX高层代码封装处理输入准备与输出后处理。机器学习编译器（XLA[74]）随后将该实现转化为一系列中间表示（IRs），每个表示都针对特定硬件执行添加更多细节。在这些阶段，优化内存访问编排或计算调度的决策能显著降低特定硬件上的运行时开销。

}我们向*AlphaEvolve*发起挑战，要求其直接优化XLA生成的封装FlashAttention内核的中间表示（IR），包括预处理和后处理代码。我们针对一个在GPU上大规模推理中极具影响力的Transformer模型配置进行优化，目标是最大限度减少该模块的整体执行时间。这项任务尤为艰巨，因为（1）该IR设计初衷是用于调试而非开发者直接编辑；（2）作为编译器生成的代码，其本身已高度优化。为确保优化过程中的数值正确性，*AlphaEvolve*提出的每项修改都需在随机输入数据上对照未修改的基准代码进行验证。最终版本的代码经过人类专家严格确认，确保对所有可能的输入都正确无误。

AlphaEvolve 能够为IR所暴露的两个抽象层次提供有意义的优化。首先，针对目标配置的FlashAttention内核速度提升了32%。其次，*AlphaEvolve*在核输入输出的前后处理环节发现了改进点，使这部分速度提高了15%。这些结果表明*AlphaEvolve*具备优化编译器生成代码的能力，有望将发现的优化方案融入现有编译器以服务于特定用例，或从长远来看，将*AlphaEvolve*整合至编译器工作流本身。

4. 消融实验

Wcarried out ablations on two tasks: finding tensor decompositions for faster matrix 翻译结果：两个任务上
米乘法（第3.1节）和计算接吻数下界（第3.2节），
旨在理解*AlphaEvolve*以下组件的功效。

- evolutionary approach. *AlphaEvolve* 采用进化方法，将先前生成的程序存储于数据库中，并在后续迭代中利用这些程序获取更优程序。为分析进化的重要性，我们考虑一种替代方案：反复向语言模型输入相同的初始程序。此方案被称为“无进化”。

- 提示中的上下文。*AlphaEvolve* 采用具有大上下文窗口的强大语言模型，通过在提示中提供问题特定上下文，其输出可显著改善。为测试上下文的重要性，我们考虑一种替代方法，即不向提示添加任何显式上下文。我们将此方法称为“提示中无上下文”。

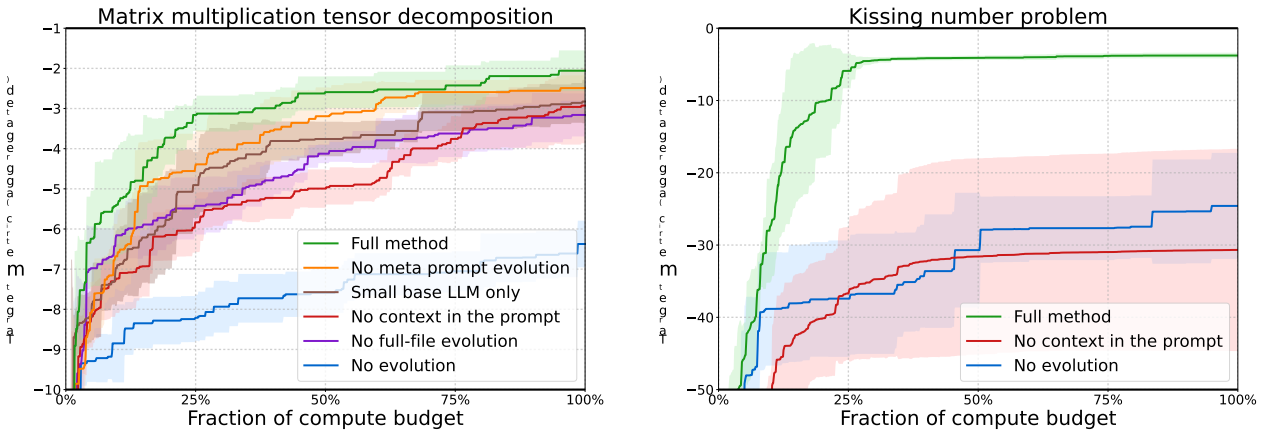


图8 | 左：针对寻找低秩张量分解以实现更快矩阵乘法问题中*AlphaEvolve*的消融研究。右：针对提升接吻数问题中寻找球体堆积方案时*AlphaEvolve*的消融研究。每条曲线展示了在计算预算递增情况下，单个设置在所有考虑目标上的平均性能表现（目标指标数值越高越好）。阴影区域表示基于三个独立*AlphaEvolve*运行结果（采用不同随机种子初始化）计算得出的目标内标准差平均值。

- > 元提示。*AlphaEvolve* 也采用元提示来优化提供给语言模型的输入指令，从而可能超越人类提示者所能达到的性能。为验证元提示的有效性，我们在张量分解任务中禁用了该功能，并将此方法称为“无元提示演化”。

- 全文件进化。与FunSearch等先前方法不同，*AlphaEvolve*能够进化整个代码库，而非仅聚焦于单一函数。为验证全文件进化的重要性，我们在张量分解场景中设计了对照实验：仅进化损失函数。我们将此方法称为"非全文件进化"。

- 强大的语言模型。*AlphaEvolve*依赖于小型与大型语言模型的混合使用，以获得高度多样化的样本。为了理解这一组件的重要性，我们考虑了一种替代方案，即仅使用单一小型基础模型。我们将这种方法称为“仅小型基础LLM”。

图8展示了全包式*AlphaEvolve*方法以及上述各种替代方案的结果。可以看出，每个组成部分都对结果的显著提升起到了重要作用。

5. 相关工作

演化方法。*AlphaEvolve*延续了关于*evolutionary*或*genetic programming*的长期研究传统[52]，其中通过反复使用一组变异和交叉算子来演化程序池[5,49]。特别是，经典演化技术在符号回归应用[64,84]、自动化科学[20]或算法[16]发现以及调度问题[115]中取得了成功。然而，这些方法面临的挑战在于...

方法的局限在于依赖手工编写的演化算子，这些算子不仅设计难度大，还可能无法捕捉领域的重要特性。相比之下，*AlphaEvolve*利用大语言模型实现了这些算子的自动化构建——它借助大语言模型的世界知识来变异程序，无需预先定义一组允许的变异操作。

AlphaEvolve 在最近一系列将大语言模型（LLM）与进化算法相结合的研究中，该工作直接延续了Romera-Paredes等人[80]提出的FunSearch系统——一种数学发现方法。FunSearch随后被应用于下游任务，包括学习贝叶斯优化的获取函数[2]、发现认知模型[13]、计算图间距离[100]以及组合式竞技编程[98]。*AlphaEvolve*在三个关键方面超越了FunSearch及其近期重新实现[23]：首先，FunSearch仅允许进化单个Python函数，而*AlphaEvolve*支持对多种编程语言编写的完整代码库进行进化；其次，FunSearch优化单一目标函数，而*AlphaEvolve*具备多目标优化能力；第三，FunSearch使用的LLM规模较小且仅基于代码训练，而*AlphaEvolve*采用前沿LLM并融合丰富的自然语言上下文与反馈机制。如本文所论证，这些扩展使*AlphaEvolve*能够解决FunSearch无法应对的重要挑战性问题。

other efforts in this category include the approach by Lehman et al. [55], which uses an LLM-guided evolution process to discover programmatic policy for a set of simulated robot; or the approach by Hemberg et al. [40] for code synthesis. similar approach have found use in several scientific and mathematical tasks, including symbolic regression [34, 86], discovering heuristics for combinatorial optimization [61, 112, 114], and synthesizing molecular structure [102]. LLM-guided evolution have also been used to improve AI system by enhancing LLM prompt [26] and searching over neural architecture [14]. *AlphaEvolve* differs from these approach in its scale, flexibility, and general applicability to a broad range of domain.

最近的一些研究通过补充性思路增强了LLM引导进化的基本范式。例如，Surina等人[94]通过强化学习持续微调LLM来补充进化过程；Grayeli团队[34]则采用LLM指导的概念学习步骤增强进化流程，将资源池中高性能程序归纳为自然语言描述。要理解这些理念在{v*}运行规模下的优势，仍需进一步探究。

进化方法在近期的AI联合科学家工作[33]中也得到了应用，该研究旨在通过不同智能体自动化实现科学发现，涵盖假设发现、假设排序及文献综述等任务。AI联合科学家采用*natural language*表示科学假设及其评估标准，而*AlphaEvolve*则专注于演化*code*，并利用程序化评估函数引导进化过程。这一选择使我们能够大幅规避大语言模型的幻觉问题，使得*AlphaEvolve*得以在大量时间步长中持续进行进化。不过从原理上讲，将两种方法结合是可行的，从而形成一种能灵活融合自然语言与程序化表达范式的新方法。

超级优化与算法发现。*AlphaEvolve*可视为一种用于*code superoptimization*的方法，其通过迭代地利用执行反馈改进初始程序。代码超级优化的理念可追溯至20世纪80年代[67]；在预大语言模型时代，该问题的解决途径包括系统枚举[67]、遗传搜索[19]、蒙特卡洛采样[83]以及深度强化学习[66]。此外，在专注于单一问题（如矩阵乘法）的限定场景中，亦存在诸如AlphaTensor等系统，这些系统能够发现可证明正确的算法[25]。

近来，一系列基于大语言模型（LLM）的超优化与算法探索方法逐渐兴起。该领域研究依托于LLM在代码任务中的卓越表现，其最显著的例证莫过于类似AlphaCode[58]这类（模拟）编程竞赛中的成功案例。例如，LLM智能体已被用于优化GPU内核中的特定操作，如注意力机制运算[15]或更广泛的用户自定义运算[54]。另有研究探索利用LLM发现新型进化算法[53]、训练语言模型[56]以及优化仓库级计算机系统[59]。最新研究[105]还提出采用多LLM智能体相互对话的协作模式，以完成数学推导与编程任务。

*AlphaEvolve*的方法利用它来支持进化算法，使我们能够解决重大
">在如第3节所示的情况下，解决一个明显更具挑战性的问题。

人工智能在科学与数学发现中的应用。过去十年间，人工智能系统已被广泛应用于众多科学领域和任务中，从蛋白质结构预测[45]到量子物理[6,81]，再到气候科学[51]。尤为突出的是，近期涌现了大量基于大语言模型（LLM）的方法，这些方法针对多学科的科学问题展开研究，如材料科学[44,69,91,116]、化学[12,62]、生物信息学[65,82]、地球科学[76]以及量子物理[29,75]（相关综述可参见[35,63,78]）。

这些方法中有许多利用大型语言模型（LLMs）来自动化科学发现过程中的多个不同阶段[36, 57, 103, 106, 109]，例如生成和排序假设与创意[37, 87]。在这些方法中，与*AlphaEvolve*尤为相关的是那些采用LLM引导的树搜索算法[11]或LLM引导的进化算法[33, 110, 117]的方法。其他研究则利用LLMs来优化实验规划与设计[7, 10, 42, 72]或实验执行与工作流程[27, 60, 79, 102, 113]。最后，还有研究专注于数据分析阶段[77]。*AlphaEvolve*与大多数这些方法的不同之处在于其采用了程序化的假设表示和评估指标。

{v*}人工智能系统在纯数学领域也推动了进步[22]。在此背景下，FunSearch方法[23, 80]确立了以大型语言模型为引导的进化机制，作为发现数学命题的见证与反例的强大工具——该问题与寻找数学陈述的形式化及非形式化证明[3, 18, 95, 96, 107, 108]具有互补性。{v*}

6. 讨论

AlphaEvolve 展示了将最先进的大型语言模型与进化框架内的自动化评估指标相结合的惊人力量，这种结合不仅能在数十年历史的数学问题上带来新发现，还能为高度优化的计算堆栈提供实际改进。

有趣的是，*AlphaEvolve*通常允许以不同方式处理同一问题：直接寻找解决方案，找到从零构建解决方案的函数，或演化出寻找解决方案的搜索算法。以不同方式应用*AlphaEvolve*会带来不同的偏差（例如，寻找构造性函数可能更有利于发现高度对称的对象[80]），因此适用于不同问题。

AlphaEvolve 也可以被视为一种测试时计算代理，通过其进化过程，显著提升了基础大语言模型的能力（与重复采样等相比）。一方面，这可以看作是一个有力的例证，展示了机器反馈如何能够支持测试时计算扩展，直至实现新科学发现和极具价值的实践优化的领域。另一方面，自然的下一步将是考虑将基础大语言模型经过*AlphaEvolve*增强的性能提炼到下一代基础模型中。这既具有内在价值，也很可能进一步提升后续版本*AlphaEvolve*的性能。

除了蒸馏之外，*AlphaEvolve*能够实际发现提升自身基础设施效率以及其基础大语言模型（未来版本）效率的方法，这一点也令人着迷。目前，收益尚属中等，且用于改进下一代*AlphaEvolve*的反馈周期长达数月。然而，随着这些改进，我们预见建立更多具备稳健评估函数的环境（问题）的价值将得到更广泛认可，进而推动未来产生更多高价值的实际发现。

*AlphaEvolve*的主要局限在于它只能处理那些可以设计出自动化评估器的问题。尽管这在数学和计算科学领域的许多问题中确实可行，但在自然科学等领域，只有部分实验能够被模拟或自动化。虽然*AlphaEvolve*确实允许通过LLM对想法进行评估，但这并非我们优化的场景。不过，近期研究表明这是可行的[33]，一个自然的步骤是将这两种场景结合起来，即在进入可通过代码执行获得机器反馈的实现阶段之前，先由LLM对高层次想法提供反馈。

我们感谢Michael Figurnov对本白皮书的审阅；感谢Alhussein Fawzi、Bernardino Romera-Paredes和Ankit Anand的早期探索与富有洞见的讨论；感谢Stig Petersen和Demis Hassabi的支持与建议；感谢JD Velasquez在管理实际应用方面提供的宝贵意见；同时感谢*AlphaEvolve*的所有早期用户及合作者，他们多样化的使用场景和深刻反馈使其发展成更强大、更通用的工具，适用于广泛的应用领域。我们衷心感谢这些人士对本白皮书所强调应用方向做出的卓越贡献：

陶哲轩、哈维尔·戈麦斯·塞拉诺和乔丹·艾伦伯格为建议具体的开放数学问题并就如何最好地为*AlphaEvolve*表述这些问题提供指导；博格丹·格奥尔基耶夫和约翰内斯·鲍施为将*AlphaEvolve*应用于此类问题所做的贡献。

{v*}Mohammad Amin Bareketain、Patrick Heisel、Chase Hensel、Robert O'Callahan与王鹏明共同主导数据中心调度应用；Federico Piccinini、Sultan Kenjeyev和Andrea Michi作出重大贡献；Kieran Milan、Daniel Mankowitz、Cosmin Paduraru、Calin Cascaval、Tammo Spalink与Natasha Antropova提供专业建议；Aaron Gentleman、Gaurav Dhiman、Parthasarathy Ranganatha及Amin Vahdat负责本项工作的评审。{v*}

Yanislav Donchev 领导了向Gemini内核工程的应用；Richard Tan- burn 做出了重大贡献；Justin Chiu 和Julian Walker 提供了有益的建议；Jean-Baptiste Alayrac、Dmitry Lepikhin、Sebastian Borgeaud、Koray Kavukcuoglu 和Jeff Dean 审阅了本工作。

Timur Sitkov 领导了 TPU 电路设计的应用工作；Georges Rotival 提供了电路评估基础设施；Kirk Sanders、Srikanth Dwarakanath、Indranil Chakraborty、Christopher Clark 对 TPU 设计中的结果进行了验证和确认；Vinod Nair、Sergio Guadarrama、Dimitrios Vytiniotis 和 Daniel Belov 提供了有益的建议；Kerry Takenaka、Jeff Dean、Sridhar Lakshmanamurthy、Parthasarathy Ranganathan 和 Amin Vahdat 审阅了本项工作。

本杰明·切蒂奥伊、谢尔盖·列别杰夫、亚历山大·别利亚耶夫、亨宁·贝克尔、奥列格·希什科夫，``` 感谢Ilia Khasanova在XLA修改方面的帮助以及他们提供的宝贵建议；```
Giorgio Arena、Marco Cornero和Sebastian Bodenstein对本研究进行了审阅。

作者信息

这些作者贡献均等：Alexander Novikov、Ngân Vũ、Marvin Eisenberger、Emilien Dupont、Po-Sen Huang、Adam Zsolt Wagner、Sergey Shirobokov、Borislav Kozlovskii和Matej Balog。

贡献。A.N.和M.B.设计并实现了*AlphaEvolve*的初始版本。M.B.、A.N.、N.V.和P.K.共同制定了项目愿景并界定了问题范围。N.V.和P.-S.H.监督了实际应用工作。E.D.和M.E.在F.J.R.R.和M.B.的协助下，实现了用于迭代*AlphaEvolve*的首个基准问题。A.N.和M.E.开发了*AlphaEvolve*的最终版本，其中S.S.、P.-S.H.做出了贡献，并得到了M.B.、E.D.、A.Z.W.和N.V.的输入。A.N.、S.S.、P.-S.H.和M.E.维护了支撑*AlphaEvolve*的基础设施。M.E.和E.D.在F.J.R.R.的协助下，利用*AlphaEvolve*发现了矩阵乘法的新算法。A.Z.W.在A.M.、M.E.和A.N.的帮助下，致力于解决开放性数学问题的应用。A.N.参与了Borg调度应用的开发。P.-S.H.和N.V.致力于Gemini内核工程的应用。P.-S.H.和A.N.为TPU电路设计应用做出了贡献。B.K.和S.S.研究如何应用*AlphaEvolve*直接优化编译器生成的代码。M.E.执行了消融实验。M.B.、A.N.、M.E.、S.S.和P.-S.H.共同完成了

大多数代码审查工作。M.B.、E.D.、S.C.、N.V.、A.Z.W.、F.J.R.R.、M.E.、A.N.、B.K.、S.S.、A.M.和M.P.K.撰写了论文，并得到了A.S.、P.-S.H和P.K.的意见。N.V.、E.D.、M.E.、S.C.、A.N.和A.Z.W.负责制作图表。F.J.R.R.、A.M.和A.Z.W.整合了配套的Google Colab。S.N.、A.D.和P.K.为这项工作的多个方面提供了建议和支持。M.B.、A.N.、N.V.和G.H.协调了团队工作。P.K.监督并协调了整个研究计划。

通讯作者。Matej Balog、Alexander Novikov 和 Pushmeet Kohli。

参考文献# 1.0.0 (2020-10-20)### Features* **init:** 初始化项目 ([5a0c0e1](https://github.com/liuji

[1] J. Abramson, J. Adler, J. Dunger, R. Evans, T. Green, A. Pritzel, O. Ronneberger, L. Willmore, A. J. Ballard, J. Bambrick等。利用AlphaFold 3精确预测生物分子相互作用结构。*Nature*，630(8016):493–500，2024年。

[2] V. Aglietti, I. Ktena, J. Schrouff, E. Sgouritsa, F. J. R. Ruiz, A. Malek, A. Bellot, 与 S. Chiappa。FunBO：利用FunSearch探索贝叶斯优化的获取函数。收录于 *International Conference on Machine Learning*，2025年。

[3] AlphaProof 与 AlphaGeometry 团队。人工智能达到国际数学奥林匹克竞赛解题银牌标准，2024 年。网址 `https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level`。

[4] D. Bahdanau、K. Cho 和 Y. Bengio。通过联合学习对齐与翻译的神经机器翻译。*arXiv preprint arXiv:1409.0473*，2014年。

[5] W. Banzhaf, P. Nordin, R. E. Keller 与 F. D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*. 人工智能摩根·考夫曼系列丛书, 1998.

[6] J. Bausch, A. W. Senior, F. J. H. Heras, T. Edlich, A. Davies, M. Newman, C. Jones, K. Satzinger, M. Y. Niu, S. Blackwell, G. Holland, D. Kafri, J. Atalaya, C. Gidney, D. Hassabis, S. Boixo, H. Neven, 与 P. Kohli。学习量子处理器的高精度误差解码。*Nature*, 635(8040):834–840, 2024. doi: 10.1038/s41586-024-08148-8.

[7] D. A. 博伊科, R. 麦奈特, B. 克莱恩, 与 G. 戈麦斯. 利用大语言模型实现自主化学研究. *Nature*, 624(7992):570–578, 2023. doi: 10.1038/s41586-023-06792-0.

[# 1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1

[9] J. Bradbury、R. Frostig、P. Hawkins、M. J. Johnson、C. Leary、D. Maclaurin、G. Necula、A. Paszke、J. VanderPlas、S. Wanderman-Milne与张强。JAX：可组合的Python+NumPy程序转换，2018。网址 `http://github.com/jax-ml/jax`。

[10] A. M. 布兰、S. 考克斯、O. 席尔特、C. 巴尔达萨里、A. D. 怀特与P. 肖勒。利用化学工具增强大型语言模型。*Nature Machine Intelligence*，第6卷第5期，第525至525页，2024年。doi：10.1038/s42256-024-00832-8。

[11] A. M. 布兰、T. A. 诺伊科姆、D. P. 阿姆斯特朗、Z. 琼切夫与P. 舒瓦勒。大型语言模型中的化学推理解锁了可控合成规划与反应机理阐明。见 *arXiv preprint arXiv:2503.08537*，2025年。

[12] M. Caldas Ramos, C. J. Collison, 和 A. D. White. 化学领域大语言模型与自主智能体研究综述. *Chemical Science*, 16:2514–2572, 2025. doi: 10.1039/D4SC03921A.

[13] P. S. 卡斯特罗、N. 托马塞夫、A. 阿南德、N. 夏尔马、R. 莫汉塔、A. 德夫、K. 佩尔林、S. 贾因、K. 莱文、N. 埃尔特托、W. 达布尼、A. 诺维科夫、G. C. 特纳、M. K. 埃克斯坦、N. D. 道、K. J. 米勒与K. L. 斯塔肯菲尔德。从人类与动物行为中发现符号认知模型。刊于{v*}，2025年。

[14] A. Chen、D. M. Dohan 与 D. R. So 合著，《EvoPrompting：面向代码级神经架构搜索的语言模型》，发表于 {v*}，2023年。

[15] T. Chen、B. Xu 和 K. Devleker。利用DeepSeek-R1实现GPU内核生成自动化及推理时间优化，2025年。URL `https://developer.nvidia.com/blog/automating-gpu-kernel-generation-with-deepseek-r1-and-inference-time-scaling`。

[16] X. 陈, C. 梁, D. 黄, E. Real, K. 王, H. 范, X. 董, T. 梁, C.-J. 谢, Y. 卢, 和 Q. V. Le. 符号化发现的优化算法。{v*} *in Neural Information Processing Systems*, 2023。

[17] A. Cloninger 与 S. Steinerberger。论自卷积的上确界及其在Sidon集上的应用。*Proceedings of the American Mathematical Society*，145(8):3191–3200，2017年。

[18] K. M. 柯林斯、A. Q. 姜、S. 弗里德、L. 黄、M. 齐尔卡、U. 巴特、T. 卢卡西维茨、Y. 吴、J. B. 特南鲍姆、W. 哈特等。通过交互评估数学语言模型。*Proceedings of the National Academy of Sciences*，121(24):e2318124121，2024。

[19] K. D. 库珀、D. 萨布拉马尼安和L. 托尔松。面向21世纪的自适应优化编译器。*The Journal of Supercomputing*，23:7-22，2002年。

[20] M. Cranmer. 利用PySR和符号回归实现科学领域可解释机器学习。jl. *arXiv preprint arXiv:2305.01582*, 2023.

[21] T. 道、D. 傅、S. 埃蒙、A. 鲁德拉与C. 雷，《FlashAttention：具有IO感知的高效精确注意力机制》，*Advances in neural information processingsystems*卷，第35期，第16344-16359页，2022年。

[22] A. 戴维斯、P. 韦利奇科维奇、L. 布埃辛、S. 布莱克威尔、D. 郑、N. 托马舍夫、R. 坦伯恩、P. 巴塔利亚、C. 布伦德尔、A. 尤哈斯、M. 拉肯比、G. 威廉姆森、D. 哈萨比斯、P. 科利。通过人工智能引导人类直觉推动数学发展。*Nature*，600（7887）：70-72，2021年。doi：10.1038/s41586-021-04086-x。

[23] J. S. 埃伦伯格、C. S. 弗雷泽-塔连特、T. R. 哈维、K. 斯里瓦斯塔瓦与A. V. 萨瑟兰。数学发现的生成建模。*arXiv preprint arXiv:2503.11061*，2025年。

[24] P. 埃尔德什. 数论的一些评注. *Riveon Lematematika*, 9:45–48, 1955.

[26] C. 费尔南多, D. 班纳塞, H. 米哈尔斯基, S. 奥辛德罗, 和 T. 罗克塔舍尔. 提示培育器：通过提示进化的自指式自我改进. *arXiv preprint arXiv:2309.16797*, 2023.

[27] N. 费鲁兹与B. 赫克尔。基于语言模型的可控蛋白质设计。*Nature Machine Intelligence*，4(6):521–532，2022年。

[28] E. Friedman. 埃里克的包装中心。`https://erich-friedman.github.io/packing/`，2025年。访问日期：2025-04-22。

[29] F. Frohnert, X. Gu, M. Krenn, 和 E. van Nieuwenburg。通过动态词嵌入发现量子物理研究中的涌现关联。*Machine Learning: Science and Technology*, 6(1):015029, 2025. doi: 10.1088/2632-2153/adb00a。

[30] M. Ganzhinov. 高度对称线。载于{v

[31] Gemini团队。Gemini 2.5：我们最智能的AI模型，2025年。网址 `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025`。

[32] F. 贡萨尔维斯、D. O. 席尔瓦与S. 施泰因贝格。埃尔米特多项式、环面线性流及根的不确定性原理。*Journal of Mathematical Analysisand Applications*卷，451期（2）：678-711页，2017年。

[33] J. Gottweis、W.-H. Weng、A. Daryin、T. Tu、A. Palepu、P. Sirkovic、A. Myaskovsky、F. Weissenberger、K. Rong、R. Tanno、K. Saab、D. Popovici、J. Blum、F. Zhang、K. Chou、A. Hassidim、B. Gokturk、A. Vahdat、P. Kohli、Y. Matias、A. Carroll、K. Kulkarni、N. Tomasev、Y. Guan、V. Dhillon、E. D. Vaishnav、B. Lee、T. R. D. Costa、J. R. Penadés、G. Peltz、Y. Xu、A. Pawlosky、A. Karthikesalingam与V. Natarajan。迈向AI协科学家。*arXiv preprint arXiv:2502.18864*，2025年。

[35] M. Gridach、J. Nanavati、C. Mack、K

[36] 顾X.与M.克伦。利用知识图谱与大语言模型生成有趣科学构想：基于100位研究团队负责人的评估。收录于*arXiv preprint arXiv:2405.17044*，2024年。

[37] S. Guo, A. H. Sharamatadi, G. Xiong, 和 A. Zhang. 拥抱基础模型推动科学发现. 载于 *Proceedings of the IEEE International Conference on Big Data*, 第1746–1755页, 2024. doi

[38] K. 吉亚尔马蒂、F. 亨内卡特

[39] J. K. 豪格兰德. 最小重叠问题再探. *arXiv preprint arXiv:1609.08000*, 2016.

[40] E. Hemberg, S. Moskal, 与 U.-M. O’Reilly. 利用大型语言模型进化代码. *Genetic Programming and Evolvable Machines*, 25(2):21, 2024. doi: 10.1007/s10710-024-09494-2.

[41] J. E. 霍普克罗夫特与L. R. 克尔。论矩阵乘法中乘法运算次数的最小化问题。*SIAM J. Appl. Math.*第12卷第1期，20-36页，1971年1月。ISSN 0036-1399。doi:10.1137/0120004。

[42] 黄凯、曲阳、H. Cousins、W. A. Johnson、尹东、M. Shah、周栋、R. Altman、王敏、丛乐。CRISPR-GPT：用于基因编辑实验自动化设计的LLM智能体。见 *arXiv preprint arXiv:2404.18021*，2024年。

[43] 黄力、余伟、马威、钟伟、冯哲、王浩、陈强、彭伟、冯晓、秦波等。大型语言模型中的幻觉现象研究：原理、分类、挑战与开放性问题。*ACM Transactions on Information Systems*, 43(2): 1–55, 2025.

[44] S. Jia, C. Zhang, 与 V. Fung. LLMatDesign：基于大语言模型的自主材料发现. 收录于 *arXiv preprint arXiv:2406.13163*, 2024.

[45] J. Jumper、R. Evans、A. Pritzel、T. Green、M. Figurnov、O. Ronneberger、K. Tunyasuvunakool、R. Bates、A. ídek、A. Potapenko、A. Bridgland、C. Meyer、S. A. A. Kohl、A. J. Ballard、A. Cowie、B. Romera-Paredes、S. Nikolov、R. Jain、J. Adler、T. Back、S. Petersen、D. Reiman、E. Clancy、M. Zielinski、M. Steinegger、M. Pacholska、T. Berghammer、S. Bodenstein、D. Silver、O. Vinyals、A. W. Senior、K. Kavukcuoglu、P. Kohli 和 D. Hassabis。使用 AlphaFold 实现高精度蛋白质结构预测。*Nature*，596(7873): 583–589，2021。doi: 10.1038/s41586-021-03819-2。

[46] M. 考尔斯与J. 穆斯鲍尔. 矩阵乘法的翻转图. 载于*Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*, 第381-388页, 2023年.

[47] M. 考尔斯与J. 穆斯鲍尔。若干新型非交换矩阵乘法算法，规模为(*n*, *m*,6)。*ACM Commun. Comput. Algebra*，第58卷第1期：1-11页，2025年1月。ISSN 1932-2232。doi：10.1145/3712020.3712021。

[48] D. P. Kingma 和 J. Ba. Adam: 一种随机优化方法. 见 *International Conference on Learning Representations (ICLR)*, 2015.

[49] J. R. Koza. 遗传编程：通过自然选择实现计算机编程的一种方法。*Statistics and Computing*，4(2):87−112，1994. doi: 10.1007/BF00175355。

[50] J. D. 拉德曼。一种非交换算法，用于以23次乘法计算3×3矩阵的乘积。*Bulletin of the American Mathematical Society*，82(1)：126-128，1976年。

] R. Lam、A. Sanchez-Gonzalez、M. Willson、P. Wirnsberger、M. Fortunato、F. Alet、S. Ravuri、T. Ewalds、Z. Eaton-Rosen、W. Hu、A. Merose、S. Hoyer、G. Holland、O. Vinyals、J. Stott、A. Pritzel、S. Mohamed与P. Battaglia合著。《学习中程全球天气预报技能》*Science*，382(6677):1416−1421，2023年。doi:10.1126/science.adi2336。

[52] W. B. 兰登与 R. 波利。*Foundations of genetic programming*。施普林格科学与商业媒体，2013年。

[53] R. Lange, Y. Tian, 与 Y. Tang. 大型语言模型作为进化策略. 收录于 *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '24 会议论文集, 第579−582页. 美国计算机协会, 2024. doi: 10.1145/3638530.3654238.

[54] R. T. Lange、A. Prasad、Q. Sun、M. Faldor、Y. Tang 和 D. Ha。《AI CUDA 工程师：自主 CUDA 内核的发现、优化与组合》。技术报告，Sakana AI，2025 年 2 月。

[55] J. Lehman、J. Gordon、S. Jain、K. Ndousse、C. Yeh 和 K. O. Stanley。通过大模型实现进化。见*Handbook of evolutionary machine learning*，第331−366页。Springer，2023年。

新加坡：施普林格·自然，2024年。doi：10.1007/978-981-99-3814-8_11。*Evolution Through Large Models*，第331-366页。作者：J. Lehman、J. Gordon、S. Jain、K. Ndousse、C. Yeh与K. O. Stanley。

[58] 李毅、崔大铉、钟俊杰、纳特·库什曼、朱利安·施里特维泽、雷米·勒布朗、汤姆·埃克尔斯、詹姆斯·基林、费尔南多·希梅诺、阿尔瓦罗·德·拉戈、托马斯·休伯特、庞志强、克里斯托弗·德·马森·多托姆、伊戈尔·巴布什金、陈晓、黄培轩、约翰·韦尔布尔、斯坦尼斯拉夫·高瓦尔、阿列克谢·切列帕诺夫、詹姆斯·莫洛伊、丹尼尔·曼科维茨、艾米丽·S·罗布森、普什米特·科利、纳多·德·弗雷塔斯、科伦·卡武克乔卢、奥里奥尔·维尼亚尔斯。《Alpha Code：竞赛级代码生成系统》。*Science*，378(6624):1092−1097，2022年。doi:10.1126/science.abq1158。

[59] H. Lin, M. Maas, M. Roquemore, A. Hasanzadeh, F. Lewis, Y. Simonson, T.-W. Yang, A. Yazdanbakhsh, D. Altinbüken, F. Papa等。ECO：一种面向仓库级计算机的LLM驱动高效代码优化器。*arXiv preprint arXiv:2503.15669*，2025年。

[60] Z. Lin, H. Akin, R. Rao, B. Hie, Z. Zhu, W. Lu, N. Smetanin, R. Verkuil, O. Kabeli, Y. Shmueli, A. dos Santos Costa, M. Fazel-Zarandi, T. Sercu, S. Candido, 及 A. Rives。利用语言模型预测原子级蛋白质结构的进化尺度研究。*Science*, 379(6637):1123−1130, 2023. doi: 10.1126/science.ade2574.

[62] F. 罗、J. 张、Q. 王与C. 杨。利用大型语言模型中的提示工程加速化学研究。*ACS Central Science*，2025年。doi：10.1021/acscentsci.4c01935。

[63] Z. Luo、Z. Yang、Z. Xu、W. Yang 和 X. Du。《LLM4SR：大语言模型在科学研究中的应用综述》。载于*arXiv preprint arXiv:2501.04306*，2025年。

[64] H. Ma、A. Narayanaswamy、P. Riley 和 L. Li。演化符号密度泛函。*Science Advances*，8(36):eabq0279，2022。doi: 10.1126/sciadv.abq0279。

[65] A. Madani、B. Krause、E. R. Greene、S. Subramanian、B. P. Mohr、J. M. Holton、J. L. Olmos、C. Xiong、Z. Z. Sun、R. Socher、J. S. Fraser与N. Naik。大语言模型跨多样蛋白质家族生成功能性序列。*Nature Biotechnology*，41(8):1099−1106，2023年8月。ISSN 1087-0156。doi: 10.1038/s41587-022-01618-2。

[66] D. J. 曼科维茨、A. 米奇、A. 泽尔诺夫、M. 杰尔米、M. 塞尔维、C. 帕杜拉鲁、E. 勒朗、S. 伊克巴尔、J.-B. 莱斯皮奥、A. 艾亨、T. 克佩、K. 米利金、S. 加夫尼、S. 埃尔斯特、J. 布罗希尔、C. 甘布尔、K. 米兰、R. 通、M. 黄、T. 切姆吉尔、M. 巴雷卡坦、Y. 李、A. 曼达内、T. 休伯特、J. 施里特维泽、D. 哈萨比斯、P. 科利、M. 里德米勒、O. 维尼亚尔斯、D. 西尔弗。利用深度强化学习发现更快的排序算法。{v*}，618（7964）：257-263，2023年。doi：10.1038/s41586-023-06004-9。

[67] H. 马萨林. 超级优化器——探微最小程序. 见R. H. 卡茨与M. 弗里曼编, *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), Palo Alto, California, USA, October 5-8, 1987*, 第122-126页. ACM出版社, 1987. doi:10.1145/36206.36194.

[# 1.0.0 (2022-10-10)### Features* **init:** 初始化项目 ([1a0a0c1](https://github.com/liyongning/ly-test/commit/1a0a0c1a0a1a0a0a0a0a0a0a0a0a0a0a0a0a0a0))

[69] S. Miret 和 N. M. A. Krishnan。LLM 准备好迎接现实世界的材料发现了吗？发表于 *arXiv preprint arXiv:2402.05200*，2024年。

[70] J. 莫斯鲍尔与M. 普尔。具有对称性的翻转图及新矩阵乘法方案。*arXiv preprint arXiv:2502.04514*，2025。

[71] J.-B. 穆雷与J. 克伦。通过精英映射照亮搜索空间。*arXiv preprint arXiv:1504.04909*，2015年。

[72] V. Naumov, D. Zagirova, S. Lin, Y. Xie, W. Gou, A. Urban, N. Tikhonova, K. Alawi, M. Durymanov, F. Galkin, S. Chen, D. Sidorenko, M. Korzinkin, M. Scheibye-Knudsen, A. Aspuru-Guzik, E. Izumchenko, D. Gennert, F. W. Pun, M. Zhang, P. Kamya, A. Aliper, F. Ren, 与 A. Zhavoronkov。DORA AI科学家：用于科学探索发现与自动化报告生成的多智能体虚拟研究团队。收录于*bioRxiv preprint:10.1101/2025.03.06.641840*。冷泉港实验室，2025年。doi: 10.1101/2025.03.06.641840。

[73] OpenAI. 推出OpenAI o3和o4-mini，2025年。网址 `https://openai.com/index/introducing-o3-and-o4-mini/`。

[74] OpenXLA。XLA：Python+NumPy程序的可组合转换。URL `https://github.com/openxla/xla`。

[75] H. Pan, N. Mudur, W. Taranto, M. Tikhanovskaya, S. Venugopalan, Y. Bahri, M. P. Brenner, E.-A. Kim. 利用大语言模型进行量子多体物理计算。{v*}，第8卷第1期，第49页，2025年。doi：10.1038/s42005-025-01956-y。

[76] D. Pantiukhin、B. Shapkin、I. Kuznetsov、A. A. Jost 和 N. Koldunov。通过多智能体大语言模型系统加速地球科学发现。载于*arXiv preprint arXiv:2503.05854*，2025年。

[77] Z. Rasheed, M. Waseem, A. Ahmad, K.-K. Kemell, W. Xiaofeng, A. N. Duc, 和 P. Abrahamsson. 大型语言模型能充当数据分析师吗？一种多智能体辅助的定性数据分析方法。*arXiv preprint arXiv:2402.01386*，2024。

[78] 任松、简鹏、任哲、冷超、谢晨与张健。迈向科学智能：基于大语言模型的科学智能体综述。刊于*arXiv preprint arXiv:2503.24047*，2025年。

[79] A. Rive、J. Meier、T. Sercu、S. Goyal、Z. Lin、J. Liu、D. Guo、M. Ott、C. L. Zitnick、J. Ma与R. Fergus。通过将无监督学习扩展至2.5亿个蛋白质序列，生物结构与功能得以涌现。*Proceedings of the National Academy of Sciences*，118(15):e2016239118，2021年。doi:10.1073/pnas.2016239118。

[80] B. 罗梅拉-帕雷德斯、M. 巴雷卡坦、A. 诺维科夫、M. 巴洛格、M. P. 库马尔、E. 杜邦、F. J. R. 鲁伊斯、J. 埃伦伯格、P. 王、O. 法齐、P. 科利、A. 法齐。利用大型语言模型进行程序搜索的数学发现。*Nature*，625(7995): 468–475，2023。doi:10.1038/s41586-023-06924-6。

[81] F. J. R. Ruiz, T. Laakkonen, J. Bausch, M. Balog, M. Barekatain, F. J. H. Heras, A. Novikov, N. Fitzpatrick, B. Romera-Paredes, J. van de Wetering, A. Fawzi, K. Me-ichanetzidis, 及 P. Kohli。利用AlphaTensor优化量子电路。*Nature Machine Intelligence*，7(3):374–385，2025。doi: 10.1038/s42256-025-01001-1。

[83] E. Schkufza、R. Sharma 和 A. Aiken。随机超级优化。见 V. Sarkar 和 R. Bodík 编, *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013, Houston, TX, USA, March 16-20, 2013*，第305–316页。ACM，2013年。doi: 10.1145/2451116.2451150。

[84] M. Schmidt 和 H. Lipson。从实验数据中提炼自由形式的自然定律。*Science*, 324(5923):81–85, 2009. doi: 10.1126/science.1165893.

[85] N. Shinn、F. Cassano、A. Gopinath、K. Narasimhan 和 S. Yao。Reflexion：基于言语强化学习的语言智能体。*Advances in Neural Information Processing Systems*，36:8634–8652，2023年。

[# 1.0.0 (2021-12-10)### Features* **init:** 初始化项目 ([0c0a1e3](https://github.com/liyongning/lytest/commit/0c0a1e3a1a0a0a0a0a0a0a0a0a0a0a0a0a0a0))

[87] C. Si、D. Yang与T. Hashimoto。大型语言模型能否产生新颖的研究构想？一项涵盖100+位自然语言处理研究者的大规模人类研究。发表于*International Conference on Learning Representations*，2025年。

[88] A. 斯米尔诺夫. 矩阵乘法问题的若干双线性算法 $\langle 3, P, Q\rangle$. `https://www.researchgate.net/publication/350897049_Several_Bilinear_Algorithms_for_Matrix_Multiplication_Problems_3_P_Q`, 2021年04月.

[89] A. 斯米尔诺夫. 矩阵乘法的双线性算法 $\langle 4 \times 4 \times 9; 104\rangle$. 布伦特系统的一个不可约无理解？`https://www.researchgate.net/publication/364167198_Bilinear_Algorithm_for_Matrix_Multiplication_4x4x9_104_An_irreducibly_irrational_solution_of_the_Brent_system`, 2022年10月.

[90] A. V. 斯米尔诺夫. 双线性复杂度与矩阵乘法的实用算法. *Computational Mathematics and Mathematical Physics*, 53(12):1781–1795, 2012.

[91] Z. Song, M. Ju, C. Ren, Q. Li, C. Li, Q. Zhou, 和 J. Wang. LLM-Feynman：利用大型语言模型实现通用科学公式与理论发现。见*arXiv preprint arXiv:2503.06512*，2025年。

[92] V. 斯特拉森. 高斯消元法并非最优解. *Numerische mathematik*, 13(4): 354–356, 1969.

[93] A. Surina、A. Mansouri、L. Quaedvlieg、A. Seddas、M. Viazovska、E. Abbe与C. Gulcehre。利用大语言模型进行算法发现：进化搜索与强化学习的结合。发表于*arXiv preprint arXiv:2504.05108*，2025年。

[94] R. Tanese。*Distributed genetic algorithms for function optimization*。密歇根大学，1989年。

[# 1.

[96] T. H. Trinh、Y. Wu、Q. V. Le、H. He 和 T. Luong。无需人类示范解决奥数几何问题。*Nature*，625(7995):476–482，2024年。

[97] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, 和 I. Polosukhin. 注意力就是你需要的一切。载于 *Advances in Neural Information Processing Systems*，2017年。

[98] P. Veli kovi , A. Vitvitskyi, L. Markeeva, B. Ibarz, L. Buesing, M. Balog, 和 A. Novikov。增强人类在组合式竞技编程中的表现。2024。

[99] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, 和 J. Wilkes。谷歌使用Borg进行大规模集群管理。载于*Proceedings of the Tenth European Conference on Computer Systems*，EuroSys '15，美国纽约州纽约市，2015年。计算机协会出版。ISBN 9781450332385。doi: 10.1145/2741948.2741964。

[100] S. Verma、A. Goyal、A. Mathur、A. Anand 和 S. Ranu。GRAIL：基于大语言生成代码的图编辑距离与节点对齐。发表于*International Conference on MachineLearning*，2025年。

[101] C. 维努埃萨·德尔里奥. *Generalized Sidon sets*. 博士论文, 马德里自治大学, 2010年.

[102] 王浩、M. Skreta、C. T. Ser、高伟、孔令杰、F. Strieth-Kalthoff、段晨阳、庄永斌、余越、朱毅、杜宇飞、A. Aspuru-Guzik、K. Neklyudov和张驰。基于大语言模型的化学空间高效进化搜索。载于*InternationalConference on Learning Representations*，2025年。[103] 王强、D. Downey、H. Ji和T. Hope。SciMON：面向新颖性优化的科学灵感生成系统。载于*Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*，第279–299页，泰国曼谷，2024年。计算语言学协会。doi: 10.18653/v1/2024.acl-long.18。

[104] E. P. 怀特. 关于埃尔德什最小重叠问题的一个新界. *Acta Arithmetica*, 208:235–255, 2023.

[# 1.0.0 (2023-01-10)### Features* **init:** 初始化项目 ([0a0a0e1](https://github.com/liuwei8023/liuwei-utils/commit/0a0a0e1b0a0a0a0a0a0a0a0a0a0a0a0a0a0a0))* **utils:** 添加工具函数 ([0a0a0e1](https://github.com/liuwei8023/liuwei-utils/commit/0a0a0e1b0a0a0a0a0a0a0a0a0a0a0a0a0a))

[106] 夏洋、金鹏、谢松、何磊、曹超、罗荣、刘刚、王宇、刘钊、陈永杰、郭志、白宇、邓鹏、闵毅、陆志、郝涵、杨航、李杰、刘畅、张健、朱睿、毕然、吴凯、张伟、高坤、裴强、王琦、刘鑫、李阳、朱晖、陆毅、马明、王哲、谢涛、K. Maziarz、M. Segler、杨志、陈卓、石阳、郑松、吴磊、胡超、戴鹏、刘天翼、刘辉、秦涛。《自然语言模型：破译自然语言以促进科学发现》。收录于*arXiv preprint arXiv:2502.07527v2*，2025年。

[107] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, 与 A. Anandkumar。Leandojo：基于检索增强语言模型的定理证明。*Advances in Neural Information Processing Systems*, 36:21573–21612, 2023。

[108] K. Yang, G. Poesia, J. He, W. Li, K. Lauter, S. Chaudhuri, D. Song. 形式化数学推理：人工智能的新前沿。*arXiv preprint arXiv:2412.16075*，2024年。

[110] Z. Yang, W. Liu, B. Gao, T. Xie, Y. Li, W. Ouyang, S. Poria, E. Cambria, 与 D. Zhou. MOOSE-Chem：大语言模型在化学科学未知假设再发现中的应用。见{v*}，2025。

G. Ye、X. Cai、H. Lai、X. Wang、J. Huang、L. Wang、W. Liu 和 X. Zeng 合著的《DrugAssist：一种用于分子优化的大规模语言模型》发表于*arXiv preprint arXiv:2401.10334*，2023年。

[115] F. Zhang, S. Nguyen, Y. Mei, 和 M. Zhang. *Genetic Programming for Production Scheduling*. 施普林格出版社, 2021年.

[116] 张H、宋Y、侯Z、Miret S与刘B。HoneyComb：一种基于LLM的灵活材料科学智能体系统。载于Y. Al-Onaizan、M. Bansal与陈Y.-N.编，
Findings of the Association for Computational Linguistics: EMNLP 2024，第3369–3382页。计算语言学协会，2024年11月。doi: 10.18653/v1/2024.findings-emnlp.192。

[117] 周毅、刘辉、T.斯里瓦斯塔瓦、梅宏与谭才。基于大语言模型的假设生成。载L.佩莱德-科恩、N.卡尔德隆、S.利萨克与R.赖卡特编，
Proceedings of the 1st Workshop on NLP for Science (NLP4Science)，第117–139页。计算语言学协会，2024年。doi: 10.18653/v1/2024.nlp4science-1.10。

A. 更快的矩阵乘法：完整结果

> 完整结果列表。我们在表3中提供了*AlphaEvolve*获得的最佳排名。总体而言，实验中考虑了54种矩阵乘法规模，这些规模大致代表$\langle m, n, p\rangle$（其中$2 \leq m, n \leq 5$），并对p设置了合理的截断值。由于基础矩阵乘法张量的对称性，三个轴的任意排列均存在等效算法，因此我们仅关注排序后的尺寸$m \leq n \leq p$。

除了两种尺寸外，在所有考虑的尺寸中，*AlphaEvolve*都发现了与已知最佳排名相当或更优的程序。据观察，我们在增大问题规模时遇到了一些困难：当在单个GPU加速器的评估器上，对1000个随机种子运行所发现的程序，尺寸超过⟨5,5,5⟩时，经常会出现内存不足的情况。因此，要将我们的设置扩展到更大的矩阵尺寸，还需要进一步优化。

$\langle m, n, p\rangle$	best known [reference]	*AlphaEvolve*	$\langle m, n, p\rangle$	best known [reference]	*AlphaEvolve*	$\langle m, n, p\rangle$	best known [reference]	*AlphaEvolve*
$\langle 2, 2, 2\rangle$	7 [92]	7	$\langle 2, 3, 6\rangle$	30 [90]	30	$\langle 3, 4, 4\rangle$	38 [90]	38
$\langle 2, 2, 3\rangle$	11 [90]	11	$\langle 2, 3, 7\rangle$	35 [90]	35	$\langle 3, 4, 5\rangle$	47 [25]	47
$\langle 2, 2, 4\rangle$	14 [90]	14	$\langle 2, 3, 8\rangle$	40 [90]	40	$\langle 3, 4, 6\rangle$	56 [47]	**54**
$\langle 2, 2, 5\rangle$	18 [90]	18	$\langle 2, 3, 9\rangle$	45 [90]	45	$\langle 3, 4, 7\rangle$	66 [88]	**63**
$\langle 2, 2, 6\rangle$	21 [90]	21	$\langle 2, 3, 10\rangle$	50 [90]	50	$\langle 3, 4, 8\rangle$	75 [88]	**74**
$\langle 2, 2, 7\rangle$	25 [90]	25	$\langle 2, 4, 4\rangle$	26 [90]	26	$\langle 3, 5, 5\rangle$	58 [88]	58
$\langle 2, 2, 8\rangle$	28 [90]	28	$\langle 2, 4, 5\rangle$	33 [41]	**32**	$\langle 3, 5, 6\rangle$	70 [47]	**68**
$\langle 2, 2, 9\rangle$	32 [90]	32	$\langle 2, 4, 6\rangle$	39 [90]	39	$\langle 3, 5, 7\rangle$	82 [88]	**80**
$\langle 2, 2, 10\rangle$	35 [90]	35	$\langle 2, 4, 7\rangle$	46 [90]	**45**	$\langle 4, 4, 4\rangle$	49 [92]	**48**
$\langle 2, 2, 11\rangle$	39 [90]	39	$\langle 2, 4, 8\rangle$	52 [90]	**51**	$\langle 4, 4, 5\rangle$	62 [46]	**61**
$\langle 2, 2, 12\rangle$	42 [90]	42	$\langle 2, 5, 5\rangle$	40 [90]	40	$\langle 4, 4, 6\rangle$	73 [47]	73
$\langle 2, 2, 13\rangle$	46 [90]	46	$\langle 2, 5, 6\rangle$	48 [90]	**47**	$\langle 4, 4, 7\rangle$	87 [90, 92]	**85**
$\langle 2, 2, 14\rangle$	49 [90]	49	$\langle 3, 3, 3\rangle$	23 [50]	23	$\langle 4, 4, 8\rangle$	98 [92]	**96**
$\langle 2, 2, 15\rangle$	53 [90]	53	$\langle 3, 3, 4\rangle$	29 [90]	29	$\langle 4, 4, 9\rangle$	104 [89]	108
$\langle 2, 2, 16\rangle$	56 [90]	56	$\langle 3, 3, 5\rangle$	36 [90]	36	$\langle 4, 5, 5\rangle$	76 [25]	76
$\langle 2, 3, 3\rangle$	15 [90]	15	$\langle 3, 3, 6\rangle$	40 [90]	40	$\langle 4, 5, 6\rangle$	93 [47]	**90**
$\langle 2, 3, 4\rangle$	20 [90]	20	$\langle 3, 3, 7\rangle$	49 [90]	49	$\langle 5, 5, 5\rangle$	93 [70]	93
$\langle 2, 3, 5\rangle$	25 [90]	25	$\langle 3, 3, 8\rangle$	55 [90]	55	$\langle 6, 6, 6\rangle$	153 [70]	156

表3 | 表2的完整版本，展示了*AlphaEvolve*在所有考虑参数下进行张量分解所获得的最佳排名。在54个目标中，*AlphaEvolve*在38个案例中与现有技术水平持平，在14个案例中超越（绿色标注），并在2个案例中稍逊（红色标注）。在所有案例中，*AlphaEvolve*提供了精确算法，分解中使用了整数或半整数条目。对于⟨3, 4, 7⟩、⟨4, 4, 4⟩以及⟨4, 4, 8⟩，由*AlphaEvolve*发现的算法采用了复数值乘法，可用于精确计算复数或实数矩阵的乘法。本表展示的分解可在随附的Google Colab中找到。

```
@@ -45,9 +45,14 @@
    # EVOLVE-BLOCK-START
    def _get_optimizer(self) -> optax.GradientTransformation:
      """Returns optimizer."""
-     return optax.adam(self.hypers.learning_rate)
+     return optax.adamw(
+         self.hypers.learning_rate, weight_decay=self.hypers.weight_decay
+     )

    def _get_init_fn(self) -> jax.nn.initializers.Initializer:
      """Returns initializer function."""
-     return initializers.normal(0.0, self.hypers.init_scale, jnp.complex64)
+     # Initialize with a smaller scale to encourage finding low-rank solutions.
+     # Increase scale slightly for better exploration.
+     scale = self.hypers.init_scale
+     return initializers.normal(0 + 1j * 0, scale * 0.2, jnp.complex64)

@@ -80,6 +85,66 @@
      # Gradient updates.
      updates, opt_state = self.opt.update(grads, opt_state, decomposition)
      decomposition = optax.apply_updates(decomposition, updates)
+     # Add a small amount of gradient noise to help with exploration
+     rng, g_noise_rng = jax.random.split(rng)
+     decomposition = jax.tree_util.tree_map(
+         lambda x: x
+         + self.hypers.grad_noise_std * jax.random.normal(g_noise_rng, x.shape),
+         decomposition,
+     )
+
+     # Add noise to the decomposition parameters (exploration).
+     _, noise_rng = jax.random.split(rng)
+     noise_std = self._linear_schedule(
+         global_step, start=self.hypers.noise_std, end=0.0
+     )
+     decomposition = jax.tree_util.tree_map(
+         lambda x: x + noise_std * jax.random.normal(noise_rng, x.shape),
+         decomposition,
+     )
+
+     # Cyclical annealing for clipping threshold.
+     cycle_length = 2000  # Number of steps per cycle
+     cycle_progress = (
+         global_step % cycle_length
+     ) / cycle_length  # Normalized progress within the current cycle [0, 1)
+
+     # Map cycle progress to a sinusoidal curve. Ranges from 0 to 1.
+     clip_threshold_multiplier = (1 + jnp.cos(2 * jnp.pi * cycle_progress)) / 2
+
+     clip_threshold = self.hypers.clip_min + clip_threshold_multiplier * (
+         self.hypers.clip_max - self.hypers.clip_min
+     )
+
+     def soft_clip(x, threshold):
+       # Clipping the real and imaginary parts separately.
+       x_re = jnp.real(x)
+       x_im = jnp.imag(x)
+
+       x_re_clipped = jnp.where(
+           x_re > threshold, threshold + (x_re - threshold) * 0.1, x_re
+       )
+       x_re_clipped = jnp.where(
+           x_re_clipped < -threshold,
+           -threshold + (x_re_clipped + threshold) * 0.1,
+           x_re_clipped,
+       )
```

图9.0 (图21(左)的放大版本，给出了发现更快矩阵乘法的程序[0a0a0c1](https://github.com/liuweiqin/weapp-ts/c
\n# 12.20 features, 结合init:发现更快的矩阵乘法程序
将4×4矩阵相乘的算法(*1/3*)。

```
+
+      x_im_clipped = jnp.where(
+          x_im > threshold, threshold + (x_im - threshold) * 0.1, x_im
+      )
+      x_im_clipped = jnp.where(
+          x_im_clipped < -threshold,
+          -threshold + (x_im_clipped + threshold) * 0.1,
+          x_im_clipped,
+      )
+
+      return x_re_clipped + 1j * x_im_clipped
+
+    decomposition = jax.tree_util.tree_map(
+        lambda x: soft_clip(x, clip_threshold), decomposition
+    )
+
      return decomposition, opt_state, loss

    def _loss_fn(
@@ -91,13 +156,86 @@
      """Computes (batched) loss on learned decomposition."""
      # Compute reconstruction loss.
      rec_tensor = self._decomposition_to_tensor(decomposition)  # (B, N, M, P)
+
+    # Add noise to the target tensor (robustness).
+    rng, noise_rng = jax.random.split(rng)
+    target_noise = self.hypers.target_noise_std * jax.random.normal(
+        noise_rng, self.target_tensor.shape
+    )
+    noisy_target_tensor = self.target_tensor + target_noise
+
+    # Hallucination loss (encourages exploration by randomly replacing values)
+    hallucination_prob = self.hypers.hallucination_prob
+    hallucination_scale = self.hypers.hallucination_scale
+
+    def hallucinate(x, hallucination_rng):
+      mask = jax.random.bernoulli(hallucination_rng, p=hallucination_prob)
+      noise = hallucination_scale * jax.random.normal(
+          hallucination_rng, x.shape
+      )
+      return jnp.where(mask, noise, x)
+
+    _, factor_rng = jax.random.split(rng)
+    decomposition = jax.tree_util.tree_map(
+        lambda x: hallucinate(x, jax.random.split(factor_rng)[0]),
+        decomposition,
+    )
+
      # Add a batch dimension to `target_tensor` to ensure correct broadcasting.
      # Define the loss as the L2 reconstruction error.
-    rec_loss = l2_loss_complex(self.target_tensor[None, ...], rec_tensor)
+    rec_loss = l2_loss_complex(noisy_target_tensor[None, ...], rec_tensor)

      # We must return a real-valued loss.
-    return jnp.real(rec_loss)
+
+    # Discretization loss (encourage entries to be multiples of 1/2 or integer).
+    def dist_to_half_ints(x):
+      x_re = jnp.real(x)
+      x_im = jnp.imag(x)
+      return jnp.minimum(
+          jnp.abs(x_re - jnp.round(x_re * 2) / 2),
+          jnp.abs(x_im - jnp.round(x_im * 2) / 2),
+      )
+
```

图10 | (2021-12-20版#feature展示*inv发现初始进度的程序[0a0a0c1](https://github.com/liuweiqin/weapp-ts/c

\n 将4×4矩阵相乘的算法(*2/3*)。

```
+    def dist_to_ints(x):
+      return jnp.abs(x - jnp.round(x))
+
+    discretization_loss = 0.0
+    for factor in decomposition:
+      discretization_loss += jnp.mean(dist_to_half_ints(factor))
+      discretization_loss += jnp.mean(dist_to_ints(factor))
+
+    discretization_loss /= (
+        len(decomposition) * 2
+    )  # average across all factors and loss components
+
+    discretization_weight = self._linear_schedule(
+        global_step, start=0.0, end=self.hypers.discretization_weight
+    )
+
+    # Cosine annealing for half-integer loss.
+    cycle_length = self.config.training_steps // 4  # Number of steps per cycle
+    cycle_progress = (
+        global_step % cycle_length
+    ) / cycle_length  # Normalized progress within the current cycle [0, 1)
+    half_int_multiplier = (1 + jnp.cos(jnp.pi * cycle_progress)) / 2
+    half_int_multiplier = (
+        1 - self.hypers.half_int_start
+    ) * half_int_multiplier + self.hypers.half_int_start
+
+    total_loss = (
+        rec_loss
+        + discretization_weight * discretization_loss * half_int_multiplier
+    )
+
+    # Add penalty for large values (stability).
+    large_value_penalty = 0.0
+    for factor in decomposition:
+      large_value_penalty += jnp.mean(jnp.abs(factor) ** 2)
+    large_value_penalty /= len(decomposition)
+    total_loss += self.hypers.large_value_penalty_weight * large_value_penalty
+
+    return jnp.real(total_loss)
+

 def l2_loss_complex(x: jnp.ndarray, y: jnp.ndarray) -> jnp.ndarray:
   """Elementwise L2 loss for complex numbers."""
@@ -117,6 +255,18 @@
   return hyper.zipit([
-      hyper.uniform('init_scale', hyper.interval(0.2, 1.5)),
-      hyper.uniform('learning_rate', hyper.interval(0.05, 0.3)),
+      hyper.uniform('init_scale', hyper.interval(0.1, 1.0)),
+      hyper.uniform('learning_rate', hyper.interval(0.01, 0.2)),
+      hyper.uniform('discretization_weight', hyper.interval(0.0, 0.1)),
+      hyper.uniform('hallucination_prob', hyper.interval(0.0, 0.2)),
+      hyper.uniform('hallucination_scale', hyper.interval(0.0, 0.2)),
+      hyper.uniform('noise_std', hyper.interval(0.0, 0.01)),
+      hyper.uniform('target_noise_std', hyper.interval(0.0, 0.01)),
+      hyper.uniform('weight_decay', hyper.interval(0.00001, 0.001)),
+      hyper.uniform('clip_min', hyper.interval(0.0, 0.5)),
+      hyper.uniform('clip_max', hyper.interval(1.0, 3.0)),
+      hyper.uniform('large_value_penalty_weight', hyper.interval(0.0, 0.01)),
+      # Add noise to the gradient to aid in exploration.
+      hyper.uniform('grad_noise_std', hyper.interval(0.0, 0.001)),
+      hyper.uniform('half_int_start', hyper.interval(0.0, 1.0)),
   ])
 # EVOLVE-BLOCK-END
```

图9c | 为图4（左）的放大版本，展示了发现4×4矩阵乘法更快算法的程序(*3/3*)。此处hyper是用户提供的用于生成超参数扫描的库。

B. *AlphaEvolve*数学发现的详细内容

'T本节详细介绍了所有构建的数据和验证代码均出现在伴随的Google Colab。

B.1. 第一自相关不等式

对于任意函数$f: \quad \rightarrow \quad$，定义*autoconvolution*的$f$为

$$f * f(t) := \int_{\mathbb{R}} f(t-x)f(x)\,dx.$$

设C_1表示满足以下条件的最大常数

$$\max_{-1/2\le t\le 1/2} f * f(t) \ge C_1 \left(\int_{-1/4}^{1/4} f(x)\,dx\right)^2 \tag{1}$$

原文	翻译文本
此问题出现在加法组合学中，与...相关	

the size of Sidon set. 目前已知

$$1.28 \le C_1 \le 1.5098,$$

> 其中下界由[17]给出，上界则通过阶跃函数构造在[68]中实现。*AlphaEvolve*发现了一个包含600个等距区间的阶跃函数，该函数在$[-1/4, 1/4]$上给出了稍优的上界$C_1 \le 1.5053$。

B.2. 第二自相关不等式

```plaintext
设为满足以下条件的最小常数
```

$$\|f * f\|_2^2 \le C_2 \|f * f\|_1 \|f * f\|_\infty$$

对所有非负的$f: \quad \rightarrow \quad$。已知

$$0.88922 \le C_2 \le 1$$

下界来自于一个阶梯函数的构造[68]。*AlphaEvolve*在$[-1/4, 1/4]$上找到了一个具有50个等间距区间的阶梯函数，给出了一个稍好的下界$0.8962 \le C_2$。

B.3. 第三自相关不等式

设C_3为满足条件的最大常数

$$\max_{-1/2\le t\le 1/2} |f * f(t)| \ge C_3 \left(\int_{-1/4}^{1/4} f(x)\,dx\right)^2$$

对于任意函数$f: \quad \rightarrow \quad$。显然$C_3 \le C_1$，因为我们现在允许$f$取正值和负值。存在一个阶梯函数给出了上界$C_3 \le 1.45810$[101, 第75页]。*AlphaEvolve*发现了一个在$[-1/4,1/4]$上具有400个等距区间的阶梯函数，它给出了一个稍好的上界$C_3 \le 1.4557$。

B.4. 一个不确定性不等式

给定一个函数$f:\quad \rightarrow \quad$，定义其傅里叶变换$\hat{f}(\xi) := \int_{\mathbb{R}} f(x)e^{-2\pi ix\xi}\,dx$ 且

$$A(f) := \inf\{r > 0 : f(x) \geq 0 \text{ for all } |x| \geq r\}.$$

让C_4为满足以下条件的最大常数

$$A(f)A(\hat{f}) \geq C_4$$

对所有偶数f，其中$\max(f(0), \hat{f}(0)) < 0$。已知[32]

$$0.2025 \leq C_4 \leq 0.3523.$$

(论文中给出的上界为0.353，但将其解四舍五入至第四位小数后得到0.3523)。我们采用与文献[32]类似的线性组合方法，但通过*AlphaEvolve*优化了常数项，从而将上界提升至$C_4 \leq 0.3521$。

为了获得C_4的上界，需要构造一个满足条件的特定“测试函数”f，并计算该函数的值$A(f)A(\hat{f})$，从而提供一个上界$C_4 \leq A(f)A(\hat{f})$。按照[32]中的方法，测试函数采用$f(x) = P(x)e^{-\pi x^2}$的形式寻找，其中$P(x)$是一个作为埃尔米特多项式$H_{4k}(x)$线性组合构造的偶多项式。这种形式特别有用，因为$H_n(x)e^{-\pi x^2}$的傅里叶变换是$i^n H_n(\xi)e^{-\pi\xi^2}$。对于一个偶多项式$P(x) = \sum c_{4k}H_{4k}(x)$，$f(x)$的傅里叶变换为$\hat{f}(\xi) = \sum c_{4k}i^{4k}H_{4k}(\xi)e^{-\pi\xi^2} = (\sum c_{4k}H_{4k}(\xi))e^{-\pi\xi^2} = P(\xi)e^{-\pi\xi^2}$。因此，$A(f)$与$P(x)$的最大正根相关，而$A(\hat{f})$与$P(\xi)$的最大正根相关。具体来说，若$P(x) \geq$对大$|x|$为0，则$A(f)$是$P(x)$的最大正根，$A(\hat{f})$是$P(\xi)$的最大正根，这意味着$A(f) = A(\hat{f})$。不等式变为$C_4 \leq (A(f))^2$。

该方法涉及为多项式$P(x) = c_0H_0(x) + c_1H_4(x)+c_2H_8(x)+\ldots$寻找系数$c_0, c_1, c_2, \ldots$，使得$P(x)$满足特定约束条件（与$f(0) < 0, \hat{f}(0) < 0$相关，并在$|x|$较大时保持正值），并最小化$P(x)$的最大正根。在我们的方法中，多项式$P(x)$的构造使得$P(0) = 0$（这一条件在优化过程中用于简化约束），意味着$P(x)$含有$x^2$的因子。因此，$P(x)$的最大正根$r_{\max}$即为$P(x)/x^2$的最大正根。通过此构造得出的$C_4$上界为$r_{\max}^2/(2\pi)$。

由*AlphaEvolve*针对$P(x) = c_0H_0(x) + c_1H_4(x) + c_2H_8(x)$精炼得到的常数为$[c_0, c_1, c_2] \approx [0.32925, -0.01159, -8.9216 \times 10^{-5}]$。利用这些系数构建$P(x)$，通过求解$P(x)/x^2$的最大正根$r_{\max}$（，并计算$r_{\max}^2/(2\pi)$，我们得到了改进后的上界$C_4 \leq 0.3521$。从定性上看，我们的线性组合与文献[32]中所发现的极为相似，这从经验上验证了他们的假设——该构造近乎最优。

B.5. 埃尔德什最小重叠问题

设C_5为满足以下条件的最大常数

$$\sup_{x\in[-2,2]} \int_{-1}^{1} f(t)g(x+t)\,dt \geq C_5$$

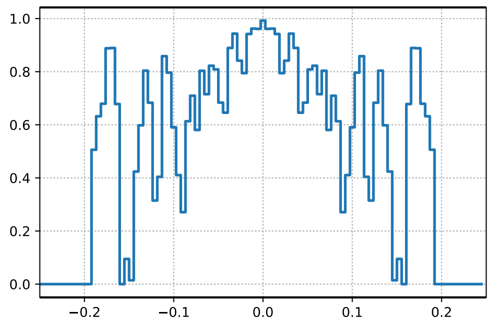


图10 | 由*AlphaEvolve*构建的针对Erd s最小重叠问题的解

在所有非负的f, g情况下：$[-1,1] \to [0,1]$，其中$f + g = 1$在$[-1,1]$上，且$\int f = 1$，此处我们将f, g在$[-1,1]$之外的部分扩展为零。该常数控制着[24]中最小重叠问题解的渐近行为。边界

$$0.379005 \le C_5 \le 0.380927$$

known，其中下界是通过[104]中的凸规划方法获得的。

已知（见[39]）该常数等于所有阶梯函数h上的下确界哦 n [0, 2] 取值为 [0, 1] 且满足 $\int_0^2 h(x)dx = 1$ of

$$\max_k \int h(x)(1 - h(x+k))dx.$$

># 1. What is the difference between a. and a. 2. What is the difference between a. and a. 3. What is the difference between a. and a. 4. What is the difference between a. and a. 5. What is the difference between a. and a. 6. What is the difference between a. and a. 7. What is the difference between a. and a. 8. What is the difference between a. and a. 9. What is the difference between a. and a. 10. What is the difference between a. and a. 11. What is the difference between a. and a. 12. What is the difference between a. and a. 13. What is the difference between a. and a. 14. What is the difference between a. and a. 15. What is the difference between a. and a. 16. What is the difference between a. and a. 17. What is the difference between a. and a. 18. What is the difference between a. and a. 19. What is the difference between a. and a. 20. What is the difference between a. and a. 21. What is the difference between a. and a. 22. What is the difference between a. and a. 23. What is the difference between a. and a. 24. What is the difference between a. and a. 25. What is the difference between a. and a. 26. What is the difference between a. and a. 27. What is the difference between a. and a. 28. What is the difference between a. and a. 29. What is the difference between a. and a. 30. What is the difference between a. and a. 31. What is the difference between a. and a. 32. What is the difference between a. and a. 33. What is the difference between a. and a. 34. What is the difference between a. and a. 35. What is the difference between a. and a. 36. What is the difference between a. and a. 37. What is the difference between a. and a. 38. What is the difference between a. and a. 39. What is the difference between a. and a. 40. What is the difference between a. and a. 41. What is the difference between a. and a. 42. What is the difference between a. and a. 43. What is the difference between a. and a. 44. What is the difference between a. and a. 45. What is the difference between a. and a. 46. What is the difference between a. and a. 47. What is the difference between a. and a. 48. What is the difference between a. and a. 49. What is the difference between a. and a. 50. What is the difference between a. and a. 51. What is the difference between a. and a. 52. What is the difference between a. and a. 53. What is the difference between a. and a. 54. What is the difference between a. and a. 55. What is the difference between a. and a. 56. What is the difference between a. and a. 57. What is the difference between a. and a. 58. What is the difference between a. and a. 59. What is the difference between a. and a. 60. What is the difference between a. and a. 61. What is the difference between a. and a. 62. What is the difference between a. and a. 63. What is the difference between a. and a. 64. What is the difference between a. and a. 65. What is the difference between a. and a. 66. What is the difference between a. and a. 67. What is the difference between a. and a. 68. What is the difference between a. and a. 69. What is the difference between a. and a. 70. What is the difference between a. and a. 71. What is the difference between a. and a. 72. What is the difference between a. and a. 73. What is the difference between a. and a. 74. What is the difference between a. and a. 75. What is the difference between a. and a. 76. What is the difference between a. and a. 77. What is the difference between a. and a. 78. What is the difference between a. and a. 79. What is the difference between a. and a. 80. What is the differe

B.6. 有限集合的和与差

$$1.14465 \le C_6 \le \frac{4}{3}, \tag{2}$$

关于上界，参见[38，推论3]；关于下界，参见[38，定理1]。该采用于下界的主要工具是Gyarmati等人[38]的以下结果：

$$C_6 \ge 1 + \frac{\log \frac{|U-U|}{|U+U|}}{\log(2\max(U)+1)} \tag{3}$$

对于# 1. 题目描述（中等难度）# 2. 解法一：回溯

```
class Solution { List<List<Integer>> resp = new ArrayList<>(); List<Integer> ans = new ArrayList<>(); public List<List<Integer>> combinationSum(int[] candidates, int target) { if(candidates == null || candidates.length == 0){ return resp; } backTracking(candidates,target,0); return resp; } public void backTracking(int[] candidates,int target,int startIndex){ if(target < 0){ return; } if(target == 0){ resp.add(new ArrayList<>(ans)); return; } for(int i=startIndex;i<candidates.length;i++){ ans.add(candidates[i]); backTracking(candidates,target-candidates[i],i); ans.remove(ans.size()-1); } } }
```

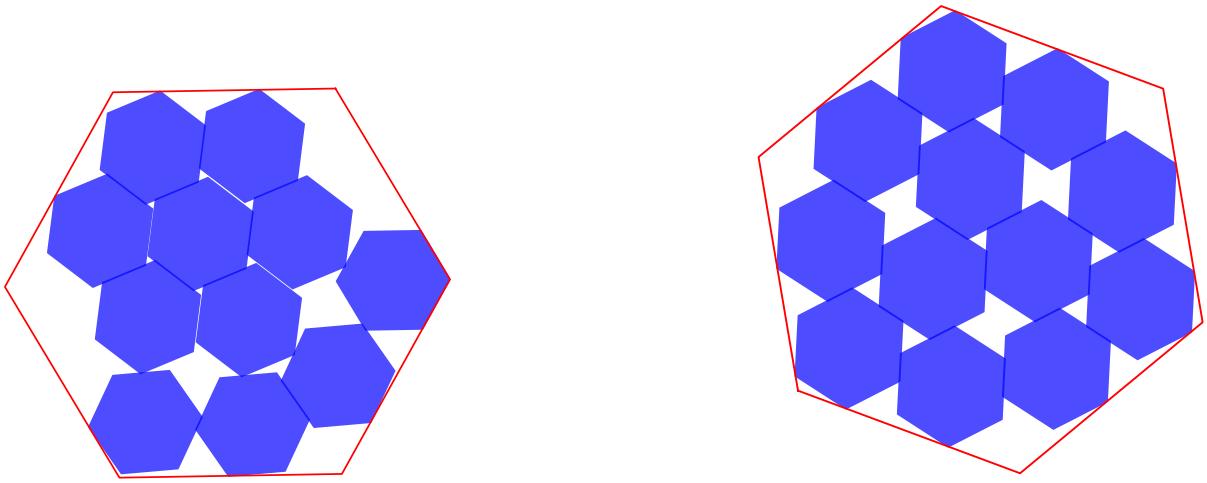


图11 | 由*AlphaEvolve*发现的装箱问题构造。左：将11个单位六边形装入边长为3.931的正六边形内。右：将12个单位六边形装入边长为3.942的正六边形内。

B.7. 在正六边形内规则排列单位正六边形

考虑将边长为单位长度的n个互不相交的正六边形填充到一个更大的正六边形中，以最小化外六边形的边长。对于n =11和n =12，目前已知的最佳构造分别使用边长为3.943和4.0的外六边形[28]。*AlphaEvolve*发现的填充排列将这些边界改进为3.931和3.942。这些排列如图11所示。

B.8. 最小化最大距离与最小距离的比值

对于# 1. 题目描述（中等难度）# 2. 解法一：回溯class Solution { List<List<Integer>> resp = new ArrayList<>(); List<Integer> ans = new ArrayList<>(); public List<List<Integer>> combinationSum(int[] candidates, int target) { if(candidates == null || candidates.length ==0){ return resp; } backTracking(candidates,target,0); return resp; } public void backTracking(int[] candidates,int target,int startIndex){ if(target < 0){ return; } if(target ==0){ resp.add(new ArrayList<>(ans)); return; } for(int i=startIndex;i<candidates.length;i++){ ans.add(candidates[i]); backTracking(candidates,target-candidates[i],i); ans.remove(ans.size()-1); } }}

在二维空间中，*AlphaEvolve*发现了16个点，其比例为≈ $\sqrt{12.889266112}$，优于the best known bound of $\sqrt{12.890}$ [28]. (In this reference, instead of the ratio itself, the 报告了比率的平方，我们采用相同的约定。

在三维空间中，*AlphaEvolve*发现了14个点，其比例为≈ $\sqrt{4.165849767}$，这一发现改进了b已知$\sqrt{4.168}$的最佳上界[28]。

B.9. 三角形海伦堡问题

该问题的目标是在单位面积三角形内或边上找到n个点，使得这些点所形成的最小三角形面积最大化。对于n = 11的情况，现有最佳记录为0.036 [28]，而AlphaEvolve发现了一种构造，其最小面积大于0.0365，如图13（左）所示。

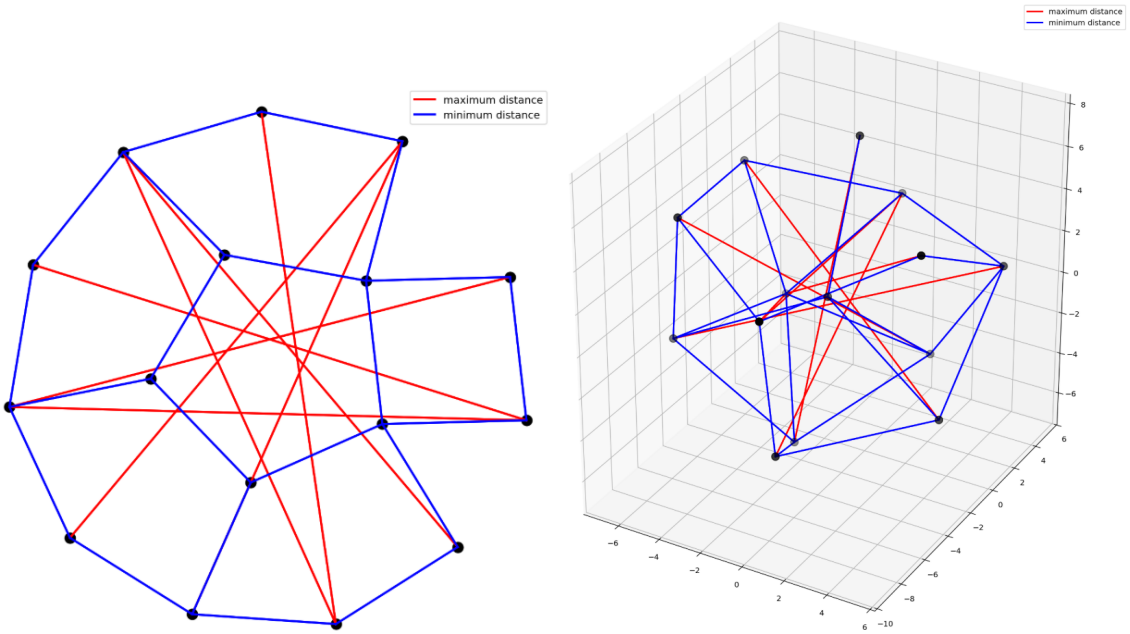


图12 | 左图：二维空间中的16个点，其最大距离与最小距离之比达到≈ $\sqrt{12.889266112}$。右图：三维空间中的14个点，其比例达到≈ $\sqrt{4.165849767}$。这两项构造均改进了已知的最佳界限。

B.10. 凸区域的Heilbronn问题

'这个翻译是正确的。
t 由这些点构成的最小三角形的面积最大化。*AlphaEvolve* 改进
two of the best known bound.

对于 it. Here's the translation of the given text into Chinese, keeping the placeholders as specified:对于 n =
\n图13.0（中）-12-20}## Heu, SOTA 0.027 [28], 而 *AlphaEvolve* (https://github.com/liuweiqin/weapp-ts/c
0.0278（见图13（右））。

B.11. 11维的接吻数

```text　接吻问题探讨的是在给定一个单位球体周围，最多可以排列多少个互不相交且与之相切的单位球体。在*d*维空间中，这个最大数量被称为*d-dimensional kissingnumber [8]*。对于d =11维，已知的最佳下界为592个[30]，而*AlphaEvolve*将其提升至593个。为了证明11维空间中接吻数593的下界，*AlphaEvolve*找到了593个11维整数坐标点，满足以下条件：(a) 这些点的最大范数小于它们之间的最小两两距离；(b) 对于任意两点，其点积的平方根小于两点的范数。设M表示所找到点的最大范数，并将所有点归一化使其范数等于M。由于第二个性质，此归一化不会降低它们的最小两两距离。由此得到一个包含593个点的11维集合，其范数完全相同且最小两两距离大于范数。因此，可将范数缩放至2，并在这些点处放置单位球体中心，从而构建出由593个点构成的有效接吻构型。```
```

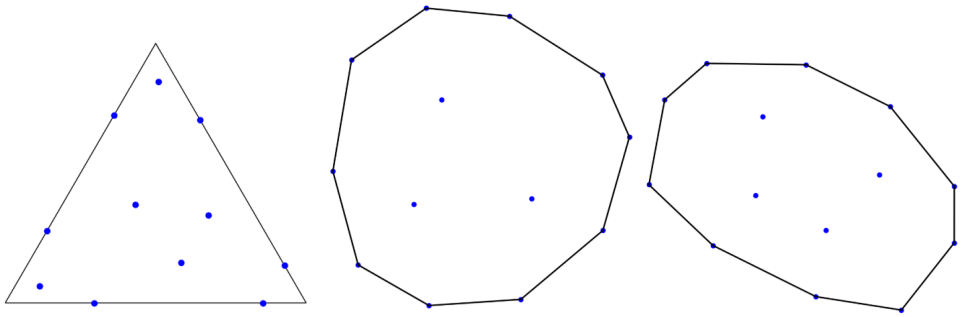


图13 | 通过*AlphaEvolve*改进海伦布朗问题两个变体的已知最佳边界，所发现的新构造。左图：单位面积三角形内11个点，所有构成三角形面积≥0.0365。中图：单位面积凸区域内13个点，所有构成三角形面积≥0.0309。右图：单位凸区域内14个点，最小面积≥0.0278。

B.12. 在单位正方形内填充圆以最大化半径之和

G给定一个正整数n，问题在于将n个互不相交的圆装入一个单位正方形内，使得
to# 1. 题目描述（简单难度）# 2. 解法一：递归class Solution { public boolean isSymmetric(TreeNode roo
在上述文本中，需要翻译的内容是"tate of the art [28]"。根据要求，保持公式标记{v*}不变，并直接输出

对于$n = 26$，SOTA（当前最优）结果为2.634，而AlphaEvolve将其提升至2.635；参见图14（左）。对于$n = 32$，SOTA记录为2.936，AlphaEvolve进一步优化至2.937；参见图14（中）。

G给定一个正整数n，问题在于将n个互不相交的圆内嵌于一个矩形中。
]+1.0.(2021最大化半径之和* *AlphaEvolve*初始发现了一种新构造(https://github.com/liuwei8023/liuwei-uti
|源文本翻译为中文：对于$n = 21$，将最先进水平从2.364 [28]提升至2.3658；见图14（右）。

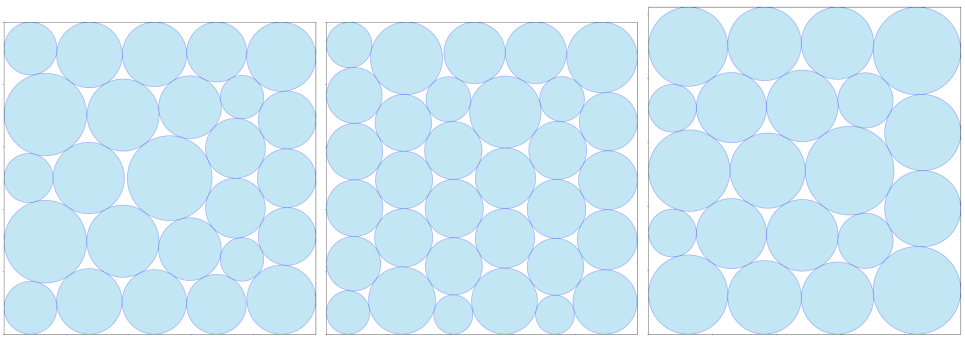


图14 | 通过 *AlphaEvolve* 改善已知的最大化半径和的圆堆叠界限而发现的新构造。左：在单位正方形中的26个圆，半径和为 ≥ 2.635。中：在单位正方形中的32个圆，半径和为 ≥ 2.937。右：在周长为4的矩形中的21个圆，半径和为 ≥ 2.365。