政府许可

本稿件由阿贡国家实验室（“Argonne”）的运营方 UChicago Argonne, LLC 创建。阿贡国家实验室是美国能源部科学办公室下属的实验室，根据合同编号 DE-AC02-06CH11357 运营。美国政府保留一项已付费、非独占、不可撤销的全球许可，并授予代表其行事的其他方该许可，以便复制本文、制作衍生作品、向公众分发副本，以及由政府或代表政府公开演示和展示本文。美国能源部将根据能源部公共访问计划，向公众提供这些联邦资助研究成果。http://energy.gov/downloads/doe-public-access-plan

arXiv:2603.05696v1 [cs.CE] 5 Mar 2026

基于
进化式大语言模型推理的叠层成像自主算法发现

尹向宇,*, † 杜明,† 邓俊静,† 杨智,‡ 韩一默,‡ 和蒋毅†

†美国伊利诺伊州莱蒙特阿贡国家实验室先进光子源

‡美国德克萨斯州休斯敦莱斯大学材料科学与纳米工程系

电子邮件：xyin@anl.gov

摘要

叠层成像是一种计算成像技术，广泛用于高分辨率材料表征，但要获得高质量重建结果，往往需要使用正则化函数，而这些函数大多仍由人工设计。我们推出了 PTYCHI-EVOLVE，这是一个自主框架，利用大语言模型（LLM）发现并进化新型正则化算法。该框架将由大语言模型驱动的代码生成与进化机制相结合，包括语义引导交叉和变异。在三个具有挑战性的数据集（X射线集成电路、脱铁铁蛋白低剂量电子显微镜成像，以及存在串扰伪影的多层切片成像）上进行的实验表明，发现的正则化器优于传统重建方法，结构相似性指数最高提升 $+0.26$，峰值信噪比最高提升 $+8.3$ dB。此外， PTYCHI-EVOLVE 会记录算法谱系和进化元数据，从而支持对发现的正则化器开展可解释、可复现的分析。

引言

自主科学发现，即人工智能（AI）系统独立提出假设、设计实验并解读结果，长期以来一直令研究人员着迷。大语言模型（LLM）的最新进展使这一愿景更加接近现实：现有系统已展现出全自动计算研究流程[1]，而大语言模型引导的进化搜索也发现了数学和计算机科学领域的新算法。[2,3] 这些工作的一个核心见解是，通过进化描述如何构建解决方案的代码，在程序空间中进行搜索，能够带来可解释的发现，这些发现可以得到验证、推广并应用于实践。

叠层成像为此类自主算法发现提供了良好的研究领域。这种计算成像技术通过重建重叠的衍射图样，获得高分辨率图像，其分辨率超越了传统光学的极限。[4–6] 这项技术已成为纳米尺度材料表征不可或缺的工具，应用范围涵盖电池电极、催化纳米粒子、半导体器件和生物样品。然而，叠层成像重建本质上是一个病态且大多欠定的逆问题，其质量很大程度上取决于正则化，即用于编码关于预期解的先验知识和限制条件的数学约束。[7] 尽管数十年来的研究已产生从经典全变分（TV）到现代深度学习先验等多种方法，有效正则化策略的设计仍主要依赖人工完成，要求从业者在相互竞争的方法中进行选择，并针对每项新的成像挑战调整超参数。虽然近期研究已通过贝叶斯优化[8] 和大语言模型辅助的自动化工作流[9] 实现超参数选择自动化，但这些方法是在固定的算法结构中运作，而非发现新的正则化策略。

在本研究中，我们提出了 Ptychi-Evolve，这是一个利用大语言模型推理来发现并进化用于 Pty-Chi 软件包中实现的叠层重建引擎的正则化算法的自主框架。[10] 不同于在固定结构内运行的既有工作， Ptychi-Evolve 探索的是算法本身的空间，生成可执行的 Python 函数来实现新颖的正则化策略，

并通过结合由大语言模型驱动的代码生成与进化优化机制（包括语义引导交叉和变异），PTYCHI-EVOLVE 发现了性能显著优于传统方法的正则化器，同时保留了可解释的算法谱系和进化元数据。

本研究的贡献有四个方面。首先，我们提出了首个用于计算成像正则化算法自主发现的框架，将范围从参数优化扩展到结构化算法搜索。其次，我们引入了语义引导的进化操作，其中交叉和变异由大语言模型对算法功能的理解所指导，而非随机重组。第三，我们开发了灵活的评估流程，支持真实值指标、人类专家反馈和视觉语言模型评估，从而能够在多样化的实验场景中部署。第四，我们在三个具有挑战性的案例研究中展示了显著改进，对进化历史的分析揭示了关于有效正则化策略的可解释科学见解。

相关工作

用于科学发现的大语言模型

大语言模型在科学发现中的应用已成为一个快速发展的研究前沿。例如，来自Sakana AI的AI科学家系统代表了迈向完全自动化研究的最早尝试之一，展示了能够生成新颖研究构想、编写必要代码、执行计算实验并产出完整科学论文的端到端流程。[1]

在算法发现这一特定领域中，FunSearch确立了利用大语言模型引导的进化，在程序空间中进行搜索的范式。[2] 其关键洞见在于，相比原始解，描述如何构造解的程序更适合通过进化加以改进，且由此获得的发现具有内在的可解释性——

。AlphaE-volve通过采用Gemini 2.0模型集成，并将规模扩展至数百行代码的程序，进一步拓展了这一方法。[3] 该系统取得了卓越成果，包括改进基础算法、优化谷歌数据中心的调度，以及加速张量处理单元的算术电路。启发式算法进化（EoH）框架提出了同时进化自然语言“思路”（代表算法构想）和实现这些构想的可执行代码这一概念。[11] 这种双重表示使大语言模型既能从高层次推理算法概念，又能生成具体且可评估的实现。类似地，LLaMEA框架也利用大语言模型自动生成和改进元启发式算法，其性能超越了协方差矩阵自适应进化策略和差分进化等最先进的方法。[12]

除了算法发现，大语言模型还在材料科学、化学等自然科学领域展现出诸多应用。关于自动化、助手、智能体和加速发现，我们推荐读者参阅一篇全面综述，其中记录了34个实例。[13]在显微成像领域，AILA 框架使大语言模型智能体能够在用户设施中自动开展显微成像实验，[14] 而 PEAR 框架[9] 则支持自动化的显微成像数据分析工作流。这些工作共同表明，大语言模型智能体能够有效参与科学情境理解与分析，不过仍面临重大挑战。

叠层成像重建算法与伪影

相干衍射成像技术（如叠层成像）会采集衍射图样，并依靠一种称为相位恢复的逆过程，重建出人类可理解的测量样品图像。相位检索算法包括叠层成像迭代引擎（PIE）系列、[15–17] 最小二乘最大似然（LSQML）、[18]以及差分映射（DM）。[19] 实现这些算法的软件包可用于各种编程框架，并提供丰富的功能，以提高重建质量。[20–24] Pty-Chi[10] 是一个较新的叠层成像数据处理库，采用——

PyTorch[25] 进行数值计算、自动微分、GPU 加速和多进程处理。Pty-Chi 还采用模块化设计，支持通过即插即用替换来修改和改进特定功能，从而推动重建工作流程不断进化。

叠层成像功能强大，但模型失配和实验缺陷常会导致伪影。[26] 例如，由栅格扫描几何引起的周期性伪影（也称为网格病理）尤为明显，特别是在扫描步长相对于照明函数较大时。[27] 这些伪影源于周期性重复的扫描网格、欠采样重叠区域中的混叠以及系统性的探针变化。虽然更密集的采样可以抑制此类伪影，但会增加采集时间和辐射剂量，因此需要算法解决方案。对于厚样品，多切片叠层成像能够重建深度分辨透射函数，但会受到切片间串扰的影响，即特征会泄漏到其他切片中。[28–31] 对于对剂量敏感的生物样品或稀有样品，通常需要采用低剂量采集来减轻辐射损伤。在低剂量条件下，受散粒噪声限制的衍射图样以及稀疏扫描中常见的较低重叠度会削弱重建约束，导致图像颗粒感更强、分辨率更低；与此同时，探测器读出噪声、背景偏移和未建模的静态强度还会进一步造成重建偏差并引入额外伪影。[32–34]

叠层成像中的正则化

正则化是解决成像中不适定逆问题的基础。例如，Rudin、Osher 和 Fatemi 提出的全变分（TV）泛函通过惩罚图像梯度的 L_1 范数，促成保边解。[35] 这种方法及其扩展形式，包括全广义变分（TGV），[36] 已广泛应用于医学成像、显微成像和遥感。各向异性正则化方法根据局部图像结构调整惩罚强度。结构张量分析可计算局部方向场，从而引导沿边缘而非横跨边缘进行选择性平滑

对于具有规则结构的图像，例如集成电路中的金属线条，方向感知正则化能够在遵循其底层几何结构的同时，提升重建质量。

深度学习革命改变了逆问题的正则化。深度图像先验表明，未经训练的卷积网络通过架构归纳偏置提供隐式正则化。[38] 这一结果表明，深度网络的结构本身更倾向于自然图像而非噪声，从而无需任何训练数据即可实现重建。即插即用方法将学习型去噪器整合到经典优化框架中，其理论基础通过与近端算子和交替方向乘子法的联系得以建立。[39,40] 该方法在去模糊、超分辨率和计算机断层成像重建方面取得了最先进的结果。[41]

具体就叠层成像而言，正则化选择会显著影响重建质量。经典方法对恢复的物体施加全变分或稀疏性约束。[42] 这些正则化策略的设计在很大程度上是人工的，需要专家知识来针对特定成像条件选择并调优方法。近期工作探索了学习型方法，包括深度图像先验和深度生成先验。[31,43–45] Du等人[46] 开发了一种即插即用框架，将基于物理的重建与文本引导的扩散模型相结合，使用户能够通过自然语言提示指定要去除的伪影，同时通过交替方向乘子法保持数据保真度。这些学习型方法并非人工设计，但不可解释，且性能在很大程度上不确定。

方法：Ptychi-Evolve框架

Ptychi-Evolve 是一个通过大语言模型引导的进化搜索来发现正则化算法的自主框架。该系统由四个相互关联的组件组成：用于算法生成与分析的大语言模型引擎、用于评估重建质量的多模态评估流水线，以及用于改进

各代中算法的进化式优化机制，以及用于上下文管理、可解释性与检查点管理的历史管理系统。图1概述了完整架构。

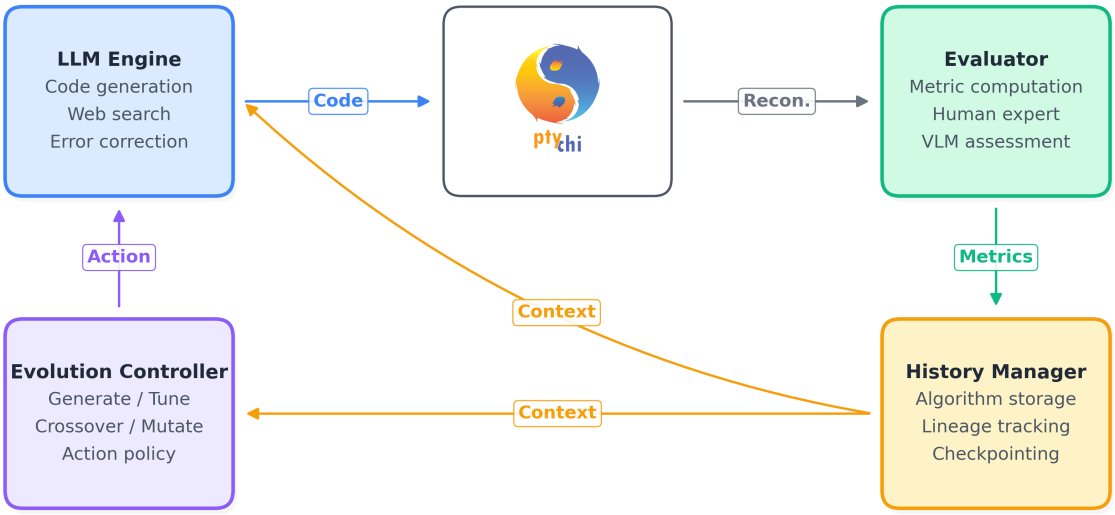


图1： PTYCHI-EVOLVE 的系统架构，展示了发现循环。大语言模型引擎生成正则化器代码，由 Pty-Chi 重建库执行。评估器评估重建质量，并将指标记录到历史管理器中。历史记录中的上下文为演化控制器（用于选择父代算法和操作）以及大语言模型引擎（用于代码生成）提供信息。演化控制器触发相应操作（生成、调优、交叉或变异），提示大语言模型生成新的正则化项变体。

问题表述

叠层成像重建旨在根据衍射强度测量恢复复值物体函数 $\psi(\mathbf{r})$。在标准表述中，相干探针 $P(\mathbf{r})$ 在一系列重叠位置 $\{\mathbf{r}_j\}$ 照射物体，产生传播至探测器的出射波；探测器仅记录强度（而非相位）。前向模型

通过以下关系将物体与测量结果联系起来：

$$I_j(\mathbf{q}) = |\mathcal{F}\left[P(\mathbf{r} - \mathbf{r}_j) \cdot \psi(\mathbf{r})\right]|^2 + \epsilon \tag{1}$$

其中 $\mathcal{F}$ 表示傅里叶变换，$\mathbf{q}$表示探测器坐标， ϵ表示噪声。对于多切片叠层成像，物体表示为一组 S 透射函数 $\{\psi_s\}_{s=1}^{S}$，出射波通过顺序传播和乘法运算计算得到。

重建算法通过迭代最小化数据保真项，同时结合正则化来应对病态性：

$$\hat{\psi} = \arg\min_{\psi} \sum_{j} \mathcal{L}(I_j, \psi) + \lambda R(\psi) \tag{2}$$

此处， $\mathcal{L}$ 用于衡量预测强度与观测强度之间的一致性（通常基于振幅或高斯似然）， $R($$\psi$)通过正则化编码先验知识，而 λ控制正则化强度。我们的目标是自动发现有效的R 及相关优化策略。

大语言模型驱动的算法生成

PTYCHI-EVOLVE 的核心是一个大语言模型引擎，用于生成可执行的 Python 函数形式的正则化算法。每个生成的函数都是 Pty-Chi 的 `Object`类的方法，并遵循预先定义的标准化接口：`regularize llm(self)`作用于存储在 `self.data`中的复值对象张量，并通过 `self.set data()`写入正则化后的结果。该接口支持与 Pty-Chi 库的重建流程无缝集成，其中正则化函数会在每个训练轮次调用。

生成过程始于提示构建。每个提示都包含多个部分，为大语言模型提供上下文。实验背景描述了

成像模态、样品特性和预期挑战；例如，可能会指出集成电路样品因周期性结构而出现网格伪影，或多层切片数据集需要处理不同深度层之间的串扰。算法历史概述了此前尝试过的方法及其性能指标，使大语言模型能够从成功和失败的经验中学习。启用网络搜索后，提示会纳入通过查询正则化技术、特定成像模态和已知挑战而检索到的相关科学文献。最后，技术要求会说明代码约束，包括函数签名、可用的导入项（PyTorch、NumPy、SciPy）以及接口约定。

大语言模型根据这些提示生成候选算法。生成的代码会经过多遍提取，以识别 Python 函数定义，并处理不同模型响应中输出格式的差异。随后，对提取出的代码进行安全验证，以检测潜在的危险操作，包括系统调用、文件输入输出、网络访问以及导入受限模块。在我们的实现中，验证结合了基于大语言模型的安全审计和受限执行环境；该环境会将生成的正则化器代码可用的内置对象和导入项列入白名单。

当算法在评估过程中因语法错误、运行时异常或输出无效而失败时，PTYCHI-EVOLVE会尝试自动校正。大语言模型会收到失败的算法以及错误信息和堆栈跟踪，随后生成针对具体故障进行修正的版本。此校正机制能够解决常见问题，包括张量形状不匹配、数值不稳定性和 API 误用。每个算法最多允许尝试校正两次，之后便会标记为失败，以平衡恢复可能性与计算成本。

多模态评估流水线

必须对发现的算法进行评估，以指导进化搜索过程。鉴于不同实验背景的评估要求各不相同， PTYCHI-

EVOLVE支持三种互补的评估模式。

当存在参考图像时，可进行真实值评估；参考图像可以来自模拟，也可以来自成本高昂的高质量重建结果（旨在寻找能够降低计算成本或对算法参数更具鲁棒性的正则化器）。系统会计算标准图像质量指标，包括结构相似性指数（SSIM）、峰值信噪比（PSNR）、均方根误差（RMSE）和平均绝对误差（MAE）。对于多切片重建结果，系统会分别计算每一层的指标，再通过可配置的策略进行汇总，包括均值、中位数、最小值或基于百分位数的组合。逐层分析有助于识别在所有深度切片上表现良好的正则化器，以及在某些层上牺牲质量的正则化器。

人类专家评估使系统能够应用于没有真实值的新样本。领域专家查看重建图像，并在零到一的连续量表上评估质量，同时提供定性反馈，描述观察到的伪影、分辨率和对比度。专家还可以为后续算法开发提出方向，并将这些建议纳入代提示中。尽管人类评估比自动化指标更费时费力，但它能够捕捉重建质量中无法通过简单数值指标体现的方面。

视觉语言模型（VLM）评估提供了一种折中方案，既能减轻专家的负担，又能实现超越真实值指标的评估。在少样本模式下，VLM 从与质量分数配对的示例图像中学习，逐渐形成关于特定成像模态下何为优质重建结果的隐式模型。在描述模式下，则由自然语言标准引导评估，例如要求重建结果呈现锐利边缘、均匀背景，且不含振铃伪影。这两种模式都会生成结构化评估结果，并与进化搜索过程兼容。感兴趣的读者可参阅我们之前的工作[47]，了解我们如何开发用于叠层成像任务的视觉语言模型。

无论采用哪种评估模式，每种算法都通过动态修补 Pty-Chi 库中的 `PlanarObject.regularize_llm` 方法来执行。启用发现的正则化器后进行重建，并按照实验配置确定的固定迭代次数运行。执行环境将可用导入项限制为经过筛选的一组（PyTorch、NumPy、SciPy 及其子模块），以确保可复现性并防止意外副作用。

进化式优化

PTYCHI-EVOLVE 依次采取动作，在算法设计空间中导航。可用的动作有三种：生成、调优和进化。由大语言模型驱动并受规则约束的动作策略，会根据算法群体的当前状态决定采取哪种动作。

生成会结合实验背景和以往尝试的历史记录，从头创建新的算法。大语言模型借助其对正则化技术的了解，综合文献中的概念，并针对当前具体的成像挑战加以调整，从而提出新颖的方法。在发现的预热阶段，生成是主要动作；它会先建立一个多样化的正则化器初始集合，然后才开始进化或调优。

调优通过调整超参数来优化表现出色的算法，同时保留其算法结构。对于一个成功的算法，大语言模型会分析其组成部分，并提出修改后的参数值，例如正则化权重、迭代次数或学习率。当已有优秀算法但仍可能通过细粒度优化获益时，就会触发调优；这通常表现为算法性能良好，但尚未达到饱和。

进化是 PTYCHI-EVOLVE中最具特色的机制，包含两种语义引导的操作。交叉选取两个成功的父代算法，并生成一个融合其互补优势的后代。与传统遗传算法中的重组不同，大语言模型能够理解每个父代算法的作用，并有意地合并它们的技术。例如，给定一个采用有效全变分

表述、另一个则采用用于串扰抑制的梯度相关惩罚时，交叉可能会产生一个在统一优化框架内纳入这两种方法的后代。变异以单个父代为对象并引入变化，例如添加新组件、移除冗余操作或修改数学表述，同时保留促成该父代成功的核心功能。

动作策略在探索新的算法思路和利用成功方法之间进行权衡。在预热阶段（通常为最初的五到十次迭代），策略仅专注于生成，以建立种群多样性。当已有足够多的成功算法后，策略便会根据可配置的标准开始选择调优和进化动作。当存在优秀级别的算法，且距上次调优动作已过去指定次数的迭代时，就会触发调优。当种群中有足够多的成功算法、能够进行有意义的交叉时，就会触发进化，同样受基于迭代次数的节流机制约束。如果策略选择了无法执行的动作（例如成功算法太少时选择进化），则会回退到生成。

历史管理和可解释性

经过评估的算法会连同全面的元数据一并存储，以便进行事后分析和解读。每条记录都包含完整的源代码、代数编号、动作类型（生成、调优或进化）、进化算法的父代标识、评估指标，以及（如有）由大语言模型生成的分析。该分析会提取各算法采用的技术（例如全变分正则化、梯度相关惩罚或自适应加权），指出关键参数及其取值，并根据观察到的性能提出潜在的改进建议。

算法根据可配置的指标阈值划分为不同的性能层级。使用结构相似性指数进行真实值评估时，典型配置可能将结构相似性指数达到 ≥ 0.90 的算法评为优秀，达到 ≥ 0.80 的评为良好，达到 ≥ 0.60 的评为中等，低于 0.60 的

评为较差。这些分类有助于指导动作选择（调优以优秀算法为目标，进化则要求算法取得成功）和历史压缩。

为限制内存使用，同时保留有效生成与进化所需的信息，系统会定期压缩历史记录。无论处于哪一代，压缩策略都会保留表现最佳的算法，确保最优秀的发现始终可用。所有优秀和良好的算法都会被保留，因为它们代表了证明发现工作值得付出的成功成果。对于中等和较差的算法，策略会保留具有代表性的样本：每个等级中表现最佳的算法，以及近期生成、能够为后续各代提供背景信息的算法。这种选择性保留既维持了种群多样性，也避免了数据无节制增长。

系统会定期以及在收到中断信号（SIGINT、SIGTERM）时对发现状态进行检查点保存，从而支持恢复长时间运行的实验。检查点会保存当前存储的历史记录、优化统计数据，以及恢复发现过程所需的缓存网页搜索结果。重新启动后， Ptychi-Evolve会加载最新的检查点，并从中断处继续发现。

Ptychi-Evolve 维护的完整历史记录支持开展黑箱优化无法实现的可解释性分析。通过追溯算法谱系，可以了解哪些父代技术促成了成功的后代。技术时间线可以揭示特定创新何时出现并在种群中传播。失败分析则能找出常见问题，包括数值不稳定性、过度平滑和内存错误，这些都是发现过程逐渐学会避免的。此类分析能够带来科学见解，帮助理解正则化为何对特定成像挑战有效。

实验

我们在涵盖不同重建挑战的三个具有挑战性的叠层成像数据集上评估了 Ptychi-Evolve 。这些实验展示了该框架能够

在多种条件下发现有效的正则化器，同时深入了解所发现的算法及演化动力学。

实验设置

评估涵盖三个数据集，分别用于代表不同的成像模态和挑战。X射线集成电路（IC）数据集由对半导体集成电路样品进行的硬X射线同步辐射测量构成。该数据集面临多项挑战：初始探针估计不准确，扫描步长相对于探针尺寸较大（导致混叠），而集成电路中的周期性金属线结构会在朴素重建中产生明显的网格伪影。综合这些因素，正则化对于实现高质量重建结果尤为重要。

X射线多层切片数据集是一个模拟的双层堆叠结构，旨在评估对深度相关重建的处理能力。第1层包含低对比度、较柔和的特征，而第2层包含高对比度、锐利的边缘。主要挑战是层间干扰（串扰），即一个深度的特征错误地出现在相邻深度的重建结果中。该数据集用于测试此框架能否找到既能确保切片独立性、又能保留各层特征的正则化器。

电子阿铁蛋白数据集由针对脱铁铁蛋白的低剂量电子叠层成像模拟构成。主要挑战是，为避免生物样品受到辐射损伤而必须限制剂量，这会导致严重噪声和空间分辨率降低。该数据集用于评估此框架能否找到既能抑制噪声、又能保留精细结构细节的正则化器。

对于所有数据集，我们将重建结果与 Pty-Chi 库中实现的未进行正则化的重建进行比较。该基线采用最小二乘最大似然算法（LSQML）[18] 进行标准叠层成像重建，除算法本身固有的约束外，不使用显式先验。我们报告相对于该基线的改进，以分离所发现正则化器的贡献。

实验使用 OpenAI 的 O3 推理模型进行代码生成、分析和进化操作。实验启用了网络搜索以进行上下文检索，使大语言模型能够依据科学文献开展生成。在发现过程中，每次重建评估都会运行固定次数的迭代，具体次数取决于数据集的特点：脱铁铁蛋白为 1000 次，多切片为 1000 次，而更具挑战性的 IC 数据集为 3000 次。我们在图 3 中报告所发现算法的收敛性分析。

主要结果

表 1 总结了全部三个数据集上的发现过程和最终性能。PTYCHI-EVOLVE 成功发现了在所有数据集上均显著优于基线方法的正则化器，提升幅度为 $+0.12$ 至 $+0.26$ 个结构相似性指数。

表 1：发现统计数据与性能。Best 报告了固定发现时限下的发现-评估结构相似性指数（IC：3000 次迭代；脱铁铁蛋白和多切片：1000 次迭代）。Δ结构相似性指数和 Δ 峰值信噪比报告了在图 3 的代表性比较迭代次数下，最佳发现正则化器相对于基线的提升（IC：3000 次迭代；脱铁铁蛋白：1000 次；多切片：1000 次），并使用与图 2 相同的评估参考。对于多切片，Δ 数值根据各层的均值计算得出。

数据集	算法	成功	Best	Δ 结构相似性指数	Δ 峰值信噪比	持续时间
X射线 IC	100	83%	0.785	+0.26	+8.3 dB	16.5 小时
脱铁铁蛋白	147	97%	0.836	+0.12	+3.2 分贝	29.5 小时
多切片	97	94%	0.871	+0.18	+8.0 dB	10.5 小时

在发现评估时限（3000 次迭代）内，最佳发现的正则化器使 X射线 IC 数据集的绝对提升幅度最大，SSIM 达到 $+0.26$，PSNR 比基线高出 $+8.3$ dB。这一显著提升反映了朴素重建结果中伪影的严重程度，也体现了所发现的陷波滤波方法在抑制网格伪影方面的有效性。83% 的算法成功率低于其他数据集，表明这一重建问题颇具挑战性，导致更多尝试失败；不过，该框架通过其校正机制成功应对了这些失败。

铁蛋白数据集显示出较为有限的改进（$+0.12$ 结构相似性指数，$+3.2$ dB 峰值信噪比）

但取得了最高的成功率（97%）。相对较小的增益很可能反映了低剂量下散粒噪声所施加的根本限制，因为没有任何正则化器能够恢复从未被捕获的信息。尽管如此，发现的正则化器在重建质量上提供了有意义的改进，可能影响下游分析。

多层切片数据集的绝对重建质量最高（结构相似性指数为 0.871），各层的平均提升为 $+0.18$ 个结构相似性指数（第 1 层：$+0.17$；第 2 层：$+0.20$）。94% 的成功率表明，为这一问题生成的算法更有可能产生有效的重建结果，这可能是因为定义明确的串扰难题更容易找到清晰的算法解决方案。发现过程中找到了 4 种优秀级和 30 种良好级算法，说明这一问题存在多种有效方法。

图 2 对比展示了不同数据集的重建质量。X射线 IC 图像显示，基线结果存在严重的周期性伪影，而发现的陷波滤波正则化器显著减少了这些伪影。对于脱铁铁蛋白，改善较为细微，但可从噪声减少和蛋白质特征变得更清晰中看出。两个多切片层的基线结果都呈现出典型的串扰伪影（层间特征泄漏产生的对角条纹）；第 1 层的结构更复杂，而第 2 层的对角特征更清晰。发现的梯度相关惩罚和格拉姆正交性技术有效抑制了两个层中的这些伪影。

图 3 展示了各数据集的收敛动态。X射线 IC 的结果表明，在 3000 次迭代时，正则化重建有显著提升，结构相似性指数为 0.79，而基线为 0.52。迭代次数增加后，基线逐渐赶上，这是因为增加计算量在一定程度上弥补了缺少显式正则化的不足；这也表明，对于类似的 IC 数据集，使用这一发现的正则化器可以节省时间和计算资源。

多切片收敛曲线揭示了一个有趣的现象：在迭代初期（大约前 200–250 次），正则化重建的表现可能不如基线。这种初期性能下降是因为正则化器使重建结果偏离

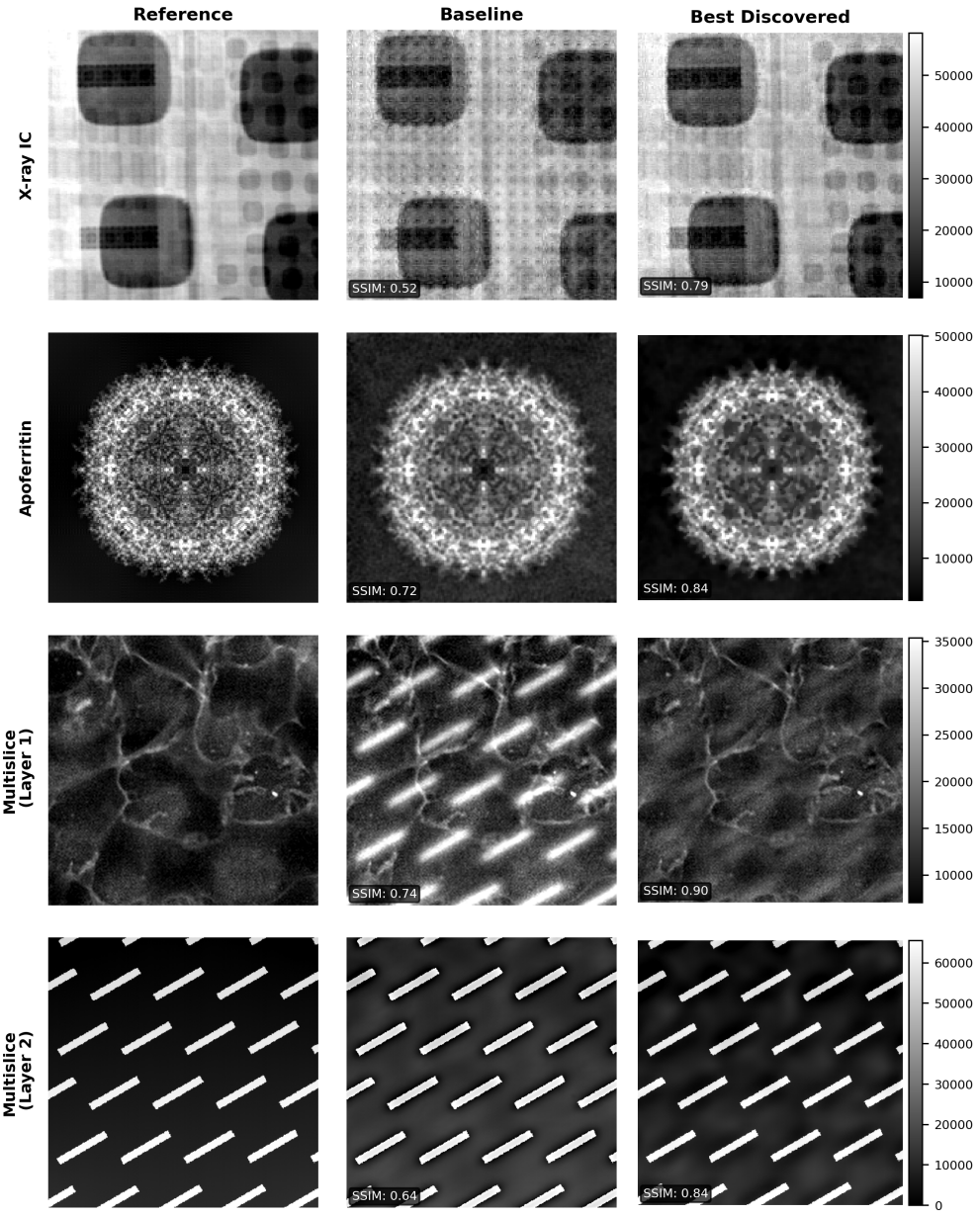
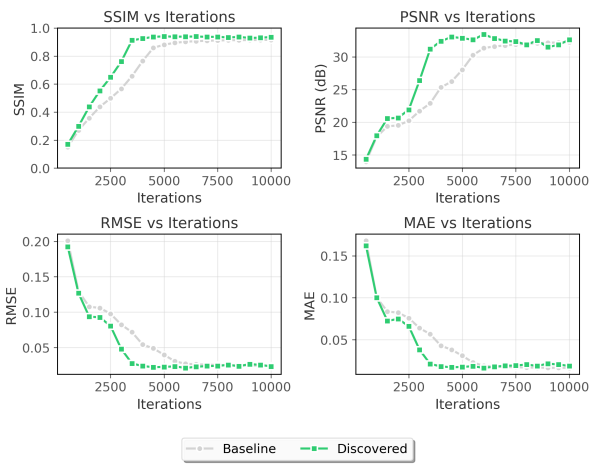
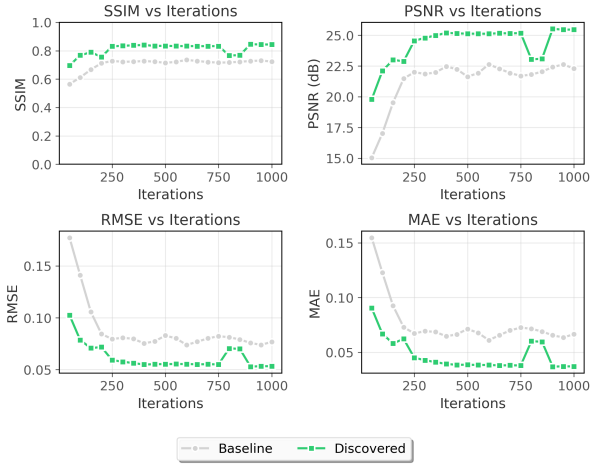


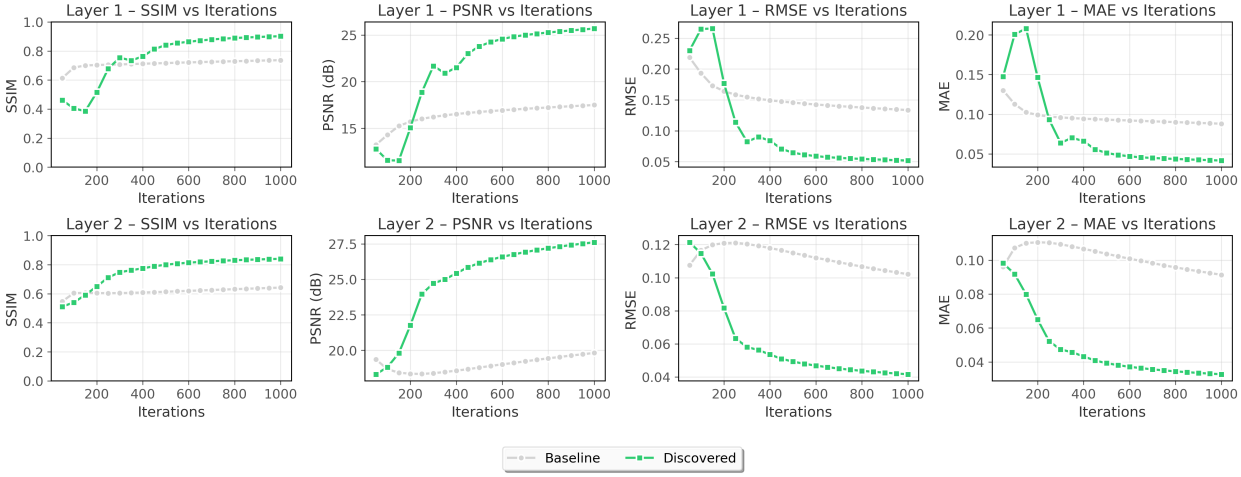
图 2：代表性迭代下重建相位图像的视觉比较。各列分别显示参考图像、基线重建以及发现的最佳正则化器。对于多切片和脱铁铁蛋白，参考为模拟的真实相位；对于X射线 IC，在缺乏真实值的情况下，参考为用作评估参考的长迭代（即10,000次）重建。叠加在基线/最佳结果上的结构相似性指数值是相对于相应参考计算的，且每一行内的所有图像共用统一色标（由右侧色条显示）。X射线 IC显示于3000次迭代；脱铁铁蛋白和多切片显示于1000次迭代。



(a) X射线 IC



(b) 脱铁铁蛋白



(c) 多切片

图 3：收敛性比较，展示基线（灰色）和发现的最佳正则化器（绿色）的 SSIM、峰值信噪比、RMSE 和 MAE 随重建迭代次数的变化；计算时使用与图 2 相同的评估参考。IC 的迭代次数扫描至 10,000 次；脱铁铁蛋白和多切片的迭代次数扫描至 1,000 次。多切片结果按层展示（上排：第 1 层；下排：第 2 层）。

在数据一致解中，结构约束特性发挥积极作用之前，.....约经过 250–300 次迭代后，正则化重建的表现便超过基线，并在此后持续保持优势。这一观察具有实际意义：用户在部署发现的正则化器时，应确保进行足够次数的迭代。

发现的算法分析

对表现最佳的算法进行考察，揭示了数据集特定创新以及一些共同模式。表 2 总结了针对每个数据集发现的关键技术。

表 2：按数据集划分的关键发现技术

数据集	技术	说明
X射线 IC	自适应陷波滤波	基于FFT的网格伪影抑制
X射线 IC	结构张量全变分	边缘感知各向异性平滑
X射线 IC	Barzilai-Borwein	自适应步长加速
脱铁铁蛋白	方差门控	噪声下保持特征
脱铁铁蛋白	软 Huber	噪声自适应收缩
脱铁铁蛋白	Perona-Malik	各向异性扩散
多切片	梯度相关惩罚	惩罚共享梯度
多切片	格拉姆正交性	通过相关性实现切片独立性
多切片	对比度自适应	逐切片正则化强度
多切片	频谱掩膜	逐切片巴特沃斯滤波器

若干独立发现与计算成像领域的成熟技术相呼应，表明大语言模型能够有效检索并应用相关领域知识。Huang等已展示了用于网格伪影抑制的陷波滤波方法，[48] 用于保证切片独立性的格拉姆正交性与立体重建中采用的多视图几何约束相呼应，Perona-Malik 扩散是一种经典的保边滤波器，[49] 而 Huber惩罚则为全变分提供了一种平滑近似，避免了梯度为零时的不可微问题。[50]

然而，大语言模型会针对每项挑战，以新颖的方式组合这些已知技术，生成整体设计超越任何单一技术所能实现的算法。每种表现最佳的算法都融合了多个互补组件，

这些组件之间经过精心编排、相互协作，体现了这类算法

这种综合方案仅靠人工设计或参数调优很难实现。例如，在多切片案例研究中发现的最佳正则化器，在单个正则化步骤中整合了四个梯度分量：(1) 采用反射边界条件的各向同性三维 Charbonnier-TV，其中 TV 惩罚不仅沿空间维度延伸，还沿深度（切片间）方向延伸；(2) 多切片梯度排斥，对多个切片同时呈现强梯度的位置施加惩罚，从而抑制串扰；(3) 格拉姆正交性，通过最小化逐切片 Gram 矩阵的非对角元素，强制层之间保持统计独立；(4) 互补的 Butterworth 频谱掩膜，其中低对比度切片接收高通滤波器，高对比度切片接收低通滤波器，且滤波强度随训练轮次进行指数退火。一个显著创新是对比度自适应 TV 加权：它将倒数对比度图与逻辑对比度图（两者均经过百分位数裁剪）融合起来，自动为低对比度切片分配更强的正则化，为高对比度切片分配更弱的正则化。四个梯度贡献经过聚合后，先通过两阶段鲁棒限幅（先进行尺度不变中位数限幅，再进行逐切片 ℓ_2 限幅），然后应用于对象。正则化器不采用固定步长，而是使用 Adam 更新规则[51] 应用这一组合梯度，并通过持久化的一阶和二阶矩估计，在各个重建轮次之间持续累积状态，同时采用三阶段学习率计划（预热、指数衰减和余弦退火）。每次更新后，逐切片 ℓ_2 归一化可防止正则化改变任何切片的整体能量。在正则化步骤中使用有状态优化器是一种非常规做法：传统正则化器通常在每个训练轮次执行一次闭式运算（例如软阈值处理、单步近端 TV）。这一设计模式并非框架预先规定，而是大语言模型自主发现的一种新颖策略，使正则化器能够在早期训练轮次执行更大、更有效的更新，并随着重建推进而平稳收敛。

表现最佳的 X 射线 IC 算法进一步深化了内部优化的理念。它

将复值对象分解为振幅和相位，然后在每个重建轮次中分别通过专用的迭代子循环求解：振幅迭代 25 次，相位迭代 15 次，联合复数全变分迭代 2 次。振幅路径计算结构张量以估计局部边缘方向，并应用各向异性 Huber-TV，对垂直于检测到的边缘的梯度 $4\times$ 施加比平行于边缘的梯度更强的惩罚，同时结合类 TGV 二阶拉普拉斯项以减少阶梯效应。Huber 阈值每经过四次内部迭代便衰减一次（这是一种延续法，使正则化从平滑逐步过渡到锐利）。相位路径应用振幅加权的全变分，在低信号区域进行更强的平滑，并将结果折返至$[-\pi,\pi)$。联合复数全变分步骤之后，基于 FFT 的陷波滤波会抑制周期性网格伪影。该算法无需预先知道伪影的周期性，而是自动检测功率谱中的最强峰值（排除直流分量），在这些位置构建对称高斯陷波对，并在傅里叶空间中抑制它们。在振幅子循环中，步长使用 Barzilai-Borwein（BB）方法自适应选择；[52] 该方法根据连续的梯度差近似局部曲率（$\tau_{\mathrm{BB}} = \mathbf{s}\cdot\mathbf{y}/\mathbf{y}\cdot\mathbf{y}$）；只有满足 Armijo 型能量下降条件时才会接受候选步长，从而在保证目标函数下降的同时实现拟牛顿加速。振幅的正值性通过带有数据自适应陡峭度参数的 softplus 变换来保证。在正则化步骤中使用迭代求解器，实际上使每个训练轮次的正则化都变成一个近端算子，用于近似求解正则化子问题。这种策略与多切片中的 Adam 方法一样，是自主发现的，而非预先指定的。

表现最佳的脱铁铁蛋白算法采用了不同的策略：它不使用优化循环，而是采用顺序信号处理流水线，分别对振幅进行去噪并校正相位。振幅处理首先进行噪声自适应 Soft-Huber 收缩，将数值拉向单位振幅；收缩阈值由全局方差的 sigmoid 函数控制，因此噪声越大，去噪效果越强；而在信号本身较为干净时，去噪则较弱。

这一过程在噪声较大的条件下会加强去噪，而在信号已经较为干净时则会减轻去噪。接下来，使用两个半径（r=1和 r=2）依次进行引导滤波；每次滤波都根据局部方差，按比例将滤波结果与输入混合，从而在平坦区域抑制噪声的同时保留边缘。当全局噪声超过阈值时，还会启用可选的高斯融合步骤。相位处理包括质量引导的 Itoh 解缠（先按行、再按列），随后通过均值减除去除直流偏置或残余斜坡。然后，对解缠后的相位进行五步 Perona-Malik 扩散[49]，使用传导系数$c = 1/(1 + |\nabla u|^2/\lambda^2)$，在扩散噪声的同时保留边缘。最后，通过较弱的复数 L_2 锚定（$\alpha = 0.1$），将去噪后的振幅和校正后的相位与原始估计混合，形成正则化后的物体，以避免过度正则化并保持对数据的忠实度。这种流水线方法虽然不涉及内部优化器，却体现了另一种形式的算法复杂性：多个去噪和相位校正阶段经过审慎排序并相互配合，且每个阶段都依据数据统计特征进行调整。

纵观表现最好的三种算法，可以发现若干共同模式。有些技术不受数据集影响，始终如一地发挥作用：边缘感知 TV 变体（Huber、Charbonnier、各向异性），既能保留清晰特征，又能平滑噪声；随着迭代进行而逐步降低正则化强度的自适应参数调度（多切片中的谱退火、IC 中的 Huber 延续、脱铁铁蛋白中的方差门控阈值）；以及包括裁剪、归一化和自适应步长在内的稳健梯度处理。这些模式揭示了有效正则化的一般原则，可迁移至不同成像模态。还有两种结构模式值得注意。第一，IC 和脱铁铁蛋白算法都独立采用了极分解策略：它们分别将复值对象分解为振幅和相位，对每个分量施加针对性的正则化，然后再将其重新组合。这种分离使正则化器能够利用领域特定先验（例如振幅正性、相位连续性），而若直接对复值表示进行处理，则难以施加这些先验。第二，两种截然不同的正则化架构

并存：IC 和多切片算法在正则化步骤中嵌入内部优化循环（迭代子循环或有状态 Adam 更新），而脱铁铁蛋白算法则采用顺序信号处理流水线。两种方法都有效，这表明最佳正则化架构取决于问题结构。

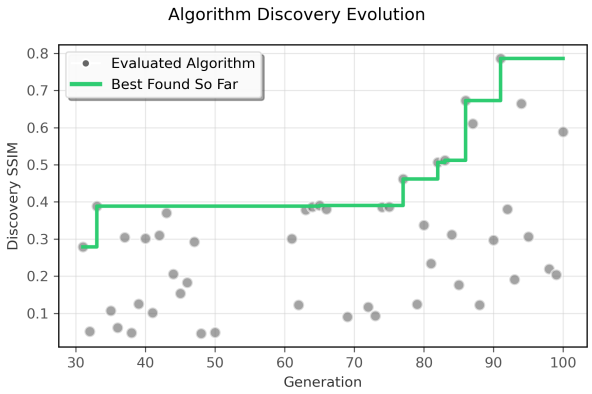
数据集特定创新进一步展现了大语言模型对问题结构进行推理的能力。针对 IC 网格伪影的自动陷波滤波直接从功率谱中检测伪影频率，而非要求用户指定，这体现了该框架发现自校准解的能力。用于多切片的梯度相关惩罚和格拉姆正交性从互补角度解决切片歧义：梯度排除在局部发挥作用（惩罚共享边缘），而格拉姆正交性在全局发挥作用（强制统计独立性）；通过交叉将二者结合，效果优于单独使用任一方法。针对铁蛋白数据集的噪声自适应收缩和方差门控滤波，则专门针对低剂量电子成像中信号相关噪声的特性。

演化动力学

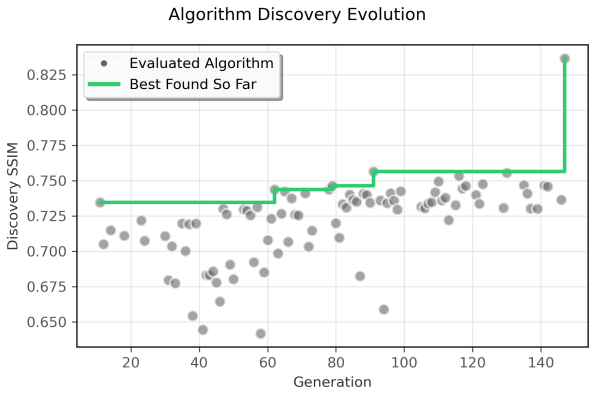
不同数据集的发现过程呈现出不同的动力学特征，反映了问题结构以及进化操作有效性的差异。对于多切片和铁蛋白数据集，交叉操作产生了大多数表现最佳的算法。这一模式表明，这些问题存在组合式解，不同的算法组件可以分别应对挑战中彼此独立的方面。例如，最佳的多切片正则化器由两个成功的父代交叉产生，结合了一个父代的全变分表述与另一个父代的切片独立性约束。大语言模型能够理解每个父代的作用，而非盲目地重新组合代码片段，因此得以实现有意义的综合。

X射线 IC 数据集呈现出一种截然不同的模式：排名前五的算法均为全新生成，而非通过交叉或变异进化而来。我们假设

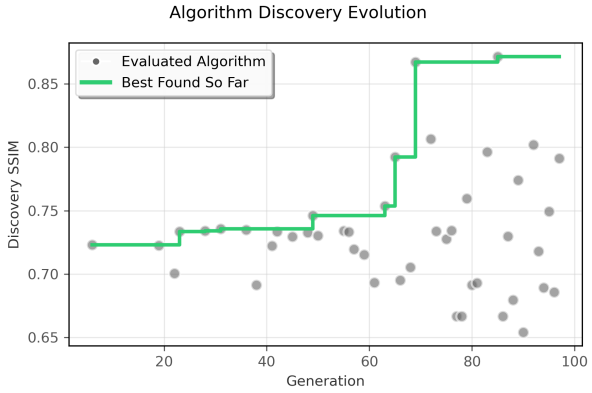
网格伪影这一独特挑战需要新颖的解决方案（具体而言，即频域中的陷波滤波），而这些方案在初始种群中并不存在，也无法通过重组现有方法获得。在描述周期性结构的实验背景启发下进行的全新生成，对于该问题被证明比进化式优化更为有效。



(a) X射线 IC



(b) 脱铁铁蛋白



(c) 多切片

图4：合并历史中发现轨迹随代数编号的变化。灰色点表示已成功评估的算法；绿色曲线表示迄今为止找到的最佳发现评估结构相似性指数（在固定发现时限下计算：IC 为 3000 次迭代；脱铁铁蛋白和多切片为 1000）。

图4展示了发现过程的轨迹。随着历史记录逐渐积累成功的算法，并为后续的生成与进化提供参考，性能通常会随着代数增加而提升。对于多切片，在第 69 代左右出现了明显的转折点：交叉首次将梯度相关惩罚与格拉姆正交性结合起来，使结构相似性指数大幅跃升，并

在此后几代中趋于稳定。对于 X射线 IC，性能提升则更为渐进，因为框架在收敛到陷波滤波策略之前，会先探索多种不同的方法。

技术时间线分析（图 5）揭示了算法创新的出现与传播。在多层切片数据集上，梯度相关惩罚较早出现（第 6 代），格拉姆正交性则较晚出现（第 28 代）；通过交叉将两者结合后，到第 85 代便产生了最佳算法。这一模式表明，进化过程能够发现互补技术并有效地将它们结合起来。

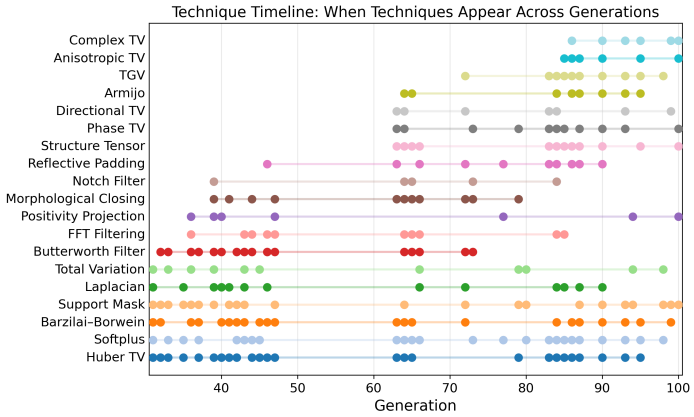
局限性与未来方向

当前工作的若干局限性为未来研究指明了方向。发现过程的计算成本仍然很高，每个数据集需要 10–30 小时，这是因为重建评估耗时较长（每种算法约 10 分钟），且大语言模型操作存在 API 延迟。并行执行评估并探索更高效的重建方法（例如早停），有望大幅加快发现过程。

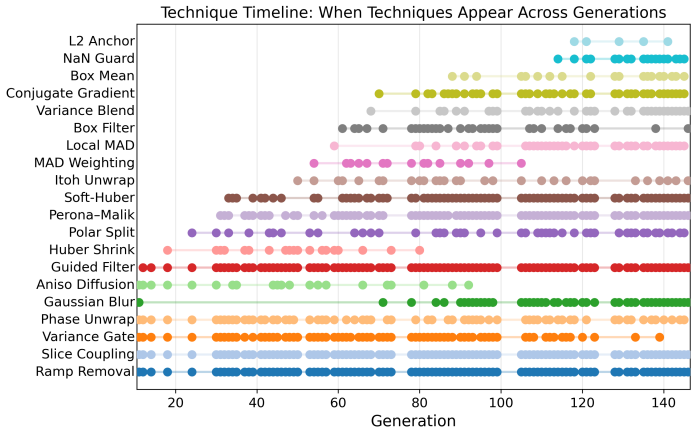
三个案例研究的评估主要依赖于针对参考图像计算的真实值指标。虽然我们支持人工评估和视觉语言模型评估模式，但实验采用的是模拟样本或特性明确的样本，这些样本具有真实值，可用于客观评估和比较。下一步，我们计划展示如何在没有参考图像的新样本上部署 Ptychi-Evolve，并进行人工评估。

发现的算法针对特定数据集和成像条件进行了优化，这表明其直接迁移能力有限：为一个样本发现的正则化器，对于特征尺度或伪影特征不同的另一个样本，可能并非最优选择。目前，每遇到新的实验条件都需要重新进行发现；不过，对于重复成像相似样本（例如 IC）或重建质量至关重要的高价值测量，这一成本或许可以接受。

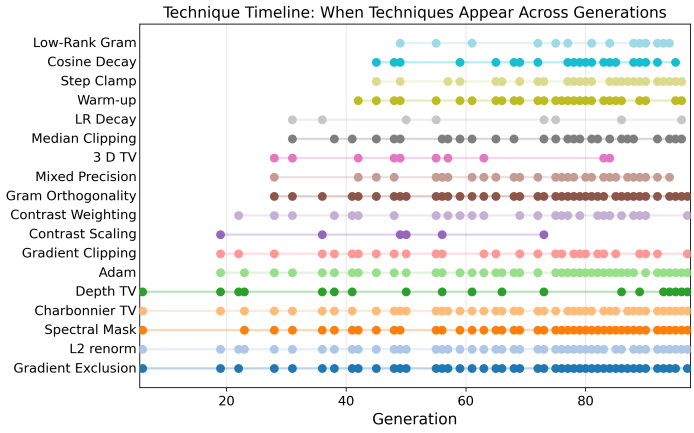
搜索空间受限于必须与 Pty-Chi 的 API 对接这一要求，



(a) X射线 IC



(b) 脱铁铁蛋白



(c) 多切片

图 5：基于合并历史的技术采用时间线，展示成功评估的算法包含各项技术的代数。线条横跨技术首次出现至最后一次出现的时间，标记表示某项技术在该代中至少一个算法中出现（所示技术出现了 ≥ 5 次；技术名称经过规范化，以合并拼写变体）。

规定函数签名、可用状态以及可导入的软件包。更通用的框架若允许修改迭代结构本身，而不仅仅是正则化函数，就有望发现超越正则化范畴的全新重建算法。

未来的研究方向包括多目标发现，同时优化重建质量和计算效率，从而支持在时间敏感型应用中部署。迁移学习方法可以利用针对相关问题发现的算法，在新数据集上启动发现过程。还可以将学习先验融入其中，例如深度图像先验[38]或隐式神经表示；这些方法与传统算法具有相似的迭代工作流程，但用神经网络表示物体，因此能够将所发现正则化策略的可解释性与深度学习的表征能力结合起来。

结论

我们提出了 Ptychi-Evolve，这是一个自主框架，可通过大语言模型引导的进化搜索，发现用于叠层成像重建的正则化算法。该框架将大语言模型在代码生成、科学推理和迭代改进方面的能力，与系统化评估和进化机制相结合， Ptychi-Evolve 能够发现性能优于传统方法的算法。

本研究证明，大语言模型能够有效探索正则化算法空间，并生成有效且可执行的代码，以实现受科学领域知识启发的新策略。进化操作能够促成组合式发现，将互补技术结合起来；大语言模型的语义理解能力则使其能够进行有意义的综合，而非随机重组。发现的算法具有可解释性，能够带来关于不同成像挑战所适用的有效正则化策略的科学见解。对进化历史的分析揭示了技术涌现与传播的模式，这些模式刻画了发现过程。

本研究为科学计算领域的人工智能驱动的算法发现开辟了一个新方向。该方法可从叠层成像推广至其他成像逆问题，例如磁共振成像、计算机断层扫描和地震成像；在这些领域，手工设计的正则化都是自动化与优化的瓶颈。更广泛地说，将大语言模型引导的搜索与系统评估相结合，为探索那些规模庞大、结构复杂、难以人工探索的算法设计空间提供了一种强有力的范式。

最后， PTYCHI-EVOLVE 推进了自主科学发现的愿景，表明大语言模型不仅能够协助人类研究人员，还能独立发现算法创新，从而增强我们对纳米尺度世界的成像能力。随着大语言模型能力持续提升、计算资源日益可及，我们预计自主算法发现将成为科学工具箱中日益重要的工具。

致谢

本材料所依据的工作得到阿贡国家实验室实验室定向研究与开发（LDRD）资助（资助编号 2023-0049）的支持，该资助由美国能源部科学办公室主任根据合同编号 DE-AC02-06CH11357 提供。本研究使用了先进光子源的资源；先进光子源是美国能源部（DOE）科学办公室用户设施，由阿贡国家实验室根据合同编号 DE-AC02-06CH11357 为美国能源部科学办公室运行。

参考文献

(1) Lu, C.; Lu, C.; Lange, R. T.; Foerster, J.; Clune, J.; Ha, D. AI科学家：迈向全自动开放式科学发现。*arXiv* 预印本 *arXiv:2408.06292* **2024年**,

(2) Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; 等人通过程序搜索和大语言模型取得的数学发现。《自然》**2024年**, *625*, 468–475.

(3) Novikov, A.; V˜u, N.; Eisenberger, M.; Dupont, E.; Huang, P.-S.; Wagner, A. Z.; Shi- robokov, S.; Kozlovskii, B.; Ruiz, F. J.; Mehrabian, A.; 等 AlphaEvolve：用于科学和算法发现的编码智能体。*arXiv* 预印本 *arXiv:2506.13131* **2025年**,

(4) Thibault, P.; Dierolf, M.; Menzel, A.; Bunk, O.; David, C.; Pfeiffer, F. 高分辨率扫描X射线衍射显微术。《科学》 **2008年**, *321*, 379–382.

(5) Pfeiffer, F. X射线叠层成像。《自然·光子学》 **2018年**, *12*, 9–17.

(6) Chen, Z.; Jiang, Y.; Shao, Y.-T.; Holtz, M. E.; Odstrˇcil, M.; Guizar-Sicairos, M.; Hanke, I.; Ganschow,S.; Schlom, D. G.; Muller, D. A. 电子叠层成像实现了由晶格振动设定的原子分辨率极限。《科学》 **2021年**, *372*, 826–831.

(7) Marchesini, S. 邀请文章：用于相位恢复的迭代投影算法统一评估。《科学仪器评论》**2007年**, *78*.

(8) Cao, M. C.; Chen, Z.; Jiang, Y.; Han, Y. 通过贝叶斯优化为电子叠层成像自动选择参数。《科学报告》**2022年**,*12*, 12284.

(9) Yin, X.; Shi, C.; Han, Y.; Jiang, Y. PEAR：一个由多个大语言模型智能体支持的稳健而灵活的叠层成像自动化框架。*arXiv* 预印本 *arXiv:2410.09034* **2024年**,

(10) Du, M.; Ruth, H.; Henke, S.; Jiang, Y.; Nikitin, V.; Tripathi, A.; Deng, J.; Klug, J.; Myint, P.; Zhou, T.; Schwarz, N.; Cherukara, M.; Sandy, A.; Vogt, S. Pty-Chi：一个基于 PyTorch 的现代叠层成像数据分析软件包。*arXiv [physics.optics]***2025年**,

(11) Liu, F.; Tong, X.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; Zhang, Q. 启发式算法演化：利用大语言模型实现高效自动算法设计。*arXiv*预印本 *arXiv:2401.02051* **2024年**,

(12) van Stein, N.; B¨ack, T. LLaMEA：一种用于自动生成元启发式算法的大语言模型进化算法。*IEEE Transactions on Evolutionary Computation* **2024年**,

(13) Zimmermann, Y.; Bazgir, A.; Al-Feghali, A.; Ansari, M.; Bocarsly, J.; Brinson, L. C.; Chiang, Y.; Circi, D.; Chiu, M.-H.; Daelman, N.; 等人 《材料科学与化学领域的34 个大语言模型应用案例：迈向自动化、助手、智能体与加速科学发现》。*arXiv* 预印本 *arXiv:2505.03049* **2025年**,

(14) Prince, M. H.; Chan, H.; Vriza, A.; Zhou, T.; Sastry, V. K.; Luo, Y.; Dearing, M. T.; Harder, R. J.; Vasudevan, R. K.; Cherukara, M. J. 科学设施中检索和工具增强型大语言模型的应用机遇。*npj Computational Materials***2024年**,*10*, 251.

(15) Rodenburg, J. M.; Faulkner, H. M. L. 一种用于移动照明的相位检索算法。应用物理快报 **2004年**,*85*, 4795–4797.

(16) Maiden, A. M.; Rodenburg, J. M. 一种改进的用于衍射成像的叠层相位恢复算法。《超显微学》**2009年**,*109*, 1256–1262.

(17) Maiden, A.; Johnson, D.; Li, P. 对叠层迭代引擎的进一步改进。*Optica* **2017**, *4*, 710–736.

(18) Odstrˇcil, M.; Menzel, A.; Guizar-Sicairos, M. 用于广义最大似然叠层成像的迭代最小二乘求解器。光学快报 **2018年**, *26*, 3108–3123.

(19) Elser, V. 通过迭代投影进行相位恢复。美国光学学会期刊**2003**, *20*, 40。

(20) Wakonig, K.; Stadler, H.-C.; Odstrˇcil, M.; Tsai, E. H. R.; Diaz, A.; Holler, M.; Usov, I.; Raabe, J.; Menzel, A.; Guizar-Sicairos, M. PtychoShelves：一个灵活的高级框架，用于高性能分析叠层成像数据。应用晶体学**2020**，*53*。

(21) Nashed, Y. S. G.; Vine, D. J.; Peterka, T.; Deng, J.; Ross, R.; Jacobsen, C. 并行叠层成像重建。光学快报 **2021年**，*22*，32082–32016。

(22) Yue, K.; Deng, J.; Jiang, Y.; Nashed, Y.; Vine, D.; Vogt, S. Ptychopy：用于叠层成像数据分析的图形处理器框架。X射线纳米成像：仪器与方法 V。2021年。

(23) Enders, B.; Thibault, P. 用于叠层成像重建的计算框架。英国皇家学会会刊 *A*：数学、物理与工程科学**2016**，*472*，20160640。

(24) Favre-Nicolin, V.; Girard, G.; Leake, S.; Carnis, J.; Chushkin, Y.; Kieffer, J.; Paleo, P.; Richard, M.-I. PyNX：基于算子的相干X射线成像高性能计算工具包。应用晶体学 **2020**，*53*，1404–1413。

(25) Paszke, A. 等：《PyTorch：一种命令式风格、高性能的深度学习库》。*arXiv* **2019**,

(26) Guizar-Sicairos, M.; Thibault, P.《叠层成像：相位问题的一种解决方案》。今日物理 **2021年**，*74*，42–48。

(27) Liu, S.; Xu, Z.; Xing, Z.; Zhang, X.; Li, R.; Qin, Z.; Wang, Y.; Tai, R. X射线叠层成像中的周期性伪影生成与抑制。《光子学》。2023年；第532页。

(28) Maiden, A. M.; Humphry, M. J.; Rodenburg, J. M. 采用多切片方法进行三维叠层成像透射显微术。美国光学学会杂志 *A* **2012**，*29*，1606–1614。

(29) Suzuki, A.; Furutaku, S.; Shimomura, K.; Yamauchi, K.; Kohmura, Y.; Ishikawa, T.; Takahashi, Y. 对延展厚物体进行高分辨率多切片X射线叠层成像。*Physical Review Letters* **2014**, *112*, 053903。

(30) O'Leary, C. M.; Sha, H.; Zhang, J.; Su, C.; Kahn, S.; Jiang, H.; Zettl, A.; Ciston, J.; Miao, J. 通过多切片叠层成像揭示埋藏异质界面的三维结构。*Physical Review Applied* **2024年**, *22*, 014016。

(31) Du, M.; Huang, X.; Jacobsen, C. 使用改进的双重深度图像先验抑制多切片叠层成像中的串扰。*Journal of Synchrotron Radiation* **2021年**, *28*, 1137–1145。

(32) Seifert, J.; Shao, Y.; Van Dam, R.; Bouchet, D.; Van Leeuwen, T.; Mosk, A. P. 泊松–高斯噪声统计条件下叠层成像的最大似然估计。*Optics Letters* **2023**, *48*, 6027–6030。

(33) Leidl, M. L.; Diederichs, B.; Sachse, C.; M¨uller-Caspary, K. 使用 Wirtinger流研究损失函数和电子剂量对二维材料叠层成像的影响。*Micron***2024年**, *185*, 103688。

(34) Wu, L.; Yoo, S.; Chu, Y. S.; Huang, X.; Robinson, I. K. 用于叠层成像重建的剂量高效自动微分。*Optica* **2024年**, *11*, 821–830。

(35) Rudin, L. I.; Osher, S.; Fatemi, E. 基于非线性全变分的噪声去除算法。***Physica D***: 非线性现象 **1992**, *60*, 259–268。

(36) Bredies, K.; Kunisch, K.; Pock, T. 全广义变分。*SIAM Journal on Imag-ing Sciences* **2010**, *3*, 492–526。

(37) Weickert, J.; 等人图像处理中的各向异性扩散; Teubner Stuttgart, 1998; 第 1 卷。

(38) Ulyanov, D.; Vedaldi, A.; Lempitsky, V. 深度图像先验。《IEEE 计算机视觉与模式识别会议论文集》。2018年；第 9446–9454 页。

(39) Venkatakrishnan, S. V.; Bouman, C. A.; Wohlberg, B. 用于基于模型的重建的即插即用先验。2013 IEEE 全球信号与信息处理大会。2013年；第945–948页。

(40) Zhang, K.; Li, Y.; Zuo, W.; Zhang, L.; Van Gool, L.; Timofte, R. 基于深度去噪器先验的即插即用图像复原。*IEEE*模式分析与机器智能汇刊 **2021年**，*44*，6360–6376。

(41) Kamilov, U. S.; Bouman, C. A.; Buzzard, G. T.; Wohlberg, B. 用于在计算成像中整合物理模型与学习模型的即插即用方法：理论、算法与应用。*IEEE*信号处理杂志 **2023**，*40*，85–97。

(42) Maiden, A. M.; Rodenburg, J. M. 一种改进的用于衍射成像的叠层摄影相位恢复算法。《超显微学》 **2009**，*109*，1256–1262。

(43) Aslan, S.; Liu, Z.; Nikitin, V.; Bicer, T.; Leyffer, S.; G¨ursoy, D. 基于深度生成先验的联合叠层断层成像。《机器学习：科学与技术》**2021年**，*2*，045017。

(44) Barutcu, S.; G¨ursoy, D.; Katsaggelos, A. K. 使用深度图像和生成先验的压缩拼接成像。*arXiv* 预印本 *arXiv:2205.02397* **2022年**，

(45) Cam, R. M.; Deng, J.; Kettimuthu, R.; Cherukara, M. J.; Bicer, T. 通过结合物理引导的基于分数的扩散模型，从有限数据中进行拼接图像重建。2025年 IEEE 第35届国际信号处理机器学习研讨会（MLSP）。2025年；第1–6页。

(46) Du, M.; Rose, V.; Deng, J.; Singh, D.; Chen, S.; Cherukara, M. 保真度保持

利用基础文本到图像模型增强叠层成像。*Optica* **2026**，*13*，45–56。

(47) Umeike, R.; Getty, N.; Xiangyu, Y.; Jiang, Y. 面向低数据环境下X射线叠层成像的通用基础模型适配。*arXiv*预印本 *arXiv:2511.02503***2025年**,

¨(48) Huang, X.; Yan, H.; Ge, M.; Ozt¨urk, H.; Nazaretski, E.; Robinson, I. K.; Chu, Y. S. 将叠层成像与实时扫描探针显微镜相结合时的伪影抑制。应用物理快报**2017**, *111*, 023103.

(49) Perona, P.; Malik, J. 使用各向异性扩散进行尺度空间与边缘检测。*IEEE*模式分析与机器智能汇刊 **2002**, *12*, 629–639.

(50) Huber,P. J.统计学的突破：方法论与分布；Springer，1992；第492–518页。

(51) Kingma, D. P.; Ba, J. Adam：一种随机优化方法。国际学习表征会议（ICLR）。2015。

(52) Barzilai, J.; Borwein, J. M. 两点步长梯度法。*IMA*数值分析杂志 **1988**,*8*, 141–148。

最佳已发现正则化算法

本附录展示了针对三个数据集中的每一个，由 PTYCHI-EVOLVE 发现的最佳正则化算法的完整源代码。每种算法都是一种可直接替换的`regularize llm(self)` 方法，作用于 `self.data` （形状为 $[S, H, W]$、表示物体切片的复值张量），并通过`self.set data()` 将正则化后的结果写回。

多层切片 X 射线（算法 5efac846，结构相似性指数 0.871）

代码清单 1：针对多层切片 X 射线数据集发现的最佳正则化器。结合了各向同性三维 Charbonnier 全变分、多切片梯度排斥、格拉姆正交性和互补频谱掩蔽，并通过有状态复数 Adam 优化器进行更新，且

一个三阶段学习率计划。

```
def regularize_llm(self):
    """Hybrid regulariser combining four gradient components
    with a stateful complex Adam optimizer."""
    import math, torch

    # ---- Shorthands and bookkeeping ----
    x = self.data                               # (S, H, W), complex
    dev, dt = x.device, x.dtype
    S, H, W = x.shape
    eps = 1e-8

    # Track the iteration count across calls
    it = getattr(self, "_iter_cnt", 0) + 1
    self._iter_cnt = it

    # Initialize Adam optimizer state on first call
    if not all(hasattr(self, k) for k in ("_adam_m", "_adam_v", "_adam_t")):
        self._adam_m = torch.zeros_like(x)        # first moment
        self._adam_v = torch.zeros_like(x.real)   # second moment (real)
        self._adam_t = 0
    self._adam_t += 1

    # Mixed-precision helper: use fp16 scratch buffers on CUDA
    def _mp(t):
        if (t.is_floating_point() and t.dtype == torch.float32
                and torch.cuda.is_available()):
            return t.to(torch.float16)
        return t

    # ---- Hyper-parameters ----
    LR_BASE   = 2.4e-2              # base learning rate
    LR_WARMUP = 5                   # linear warm-up iterations
    LR_DECAY  = 0.990               # exponential decay factor
    COS_START = 120                 # cosine tail begins here
    b1, b2    = 0.9, 0.999          # Adam momentum parameters
    STEP_CLAMP = 0.08               # per-element step size clamp

    LAM_TV0   = 3.9e-3              # TV strength
    TV_K      = 1.2                 # logistic steepness for TV weight
    LAM_DEPTH = 6.8e-3              # depth-axis TV weight
    D_CHAR    = 3.3e-2              # Charbonnier smoothing parameter
    LAM_EXCL  = 1.9e-3              # gradient exclusion strength
    LAM_CORR  = 2.9e-3              # Gram orthogonality strength
    LAM_SPEC0 = 1.4e-3              # spectral mask initial strength
    F_CUT     = 0.18                # Butterworth radial cutoff

    # ---- Finite differences (reflective boundary) ----
    dx = x - torch.cat([x[..., :, 1:], x[..., :, -1:]], dim=-1)
    dy = x - torch.cat([x[..., 1:, :], x[..., -1:, :]], dim=-2)
    if S > 1:
        dz = torch.zeros_like(x)
        dz[:-1] = x[:-1] - x[1:]    # reflective BC at last slice
    else:
        dz = torch.zeros_like(x)

    # ---- Adaptive contrast weight for TV ----
    # Blend reciprocal and logistic weighting maps,
    # clipped at the 5th and 95th percentiles
    g_xy = (dx.abs() + dy.abs()).mean(dim=(-2, -1), keepdim=True) + eps
```

```
    flat_c = g_xy.view(-1)
    p5,  p95 = torch.quantile(
        flat_c, torch.tensor([0.05, 0.95], device=dev))
    g_clamp = g_xy.clamp(p5, p95)
    mu,  sig = g_clamp.mean(),  g_clamp.std().clamp(min=eps)
    w_rcp = (g_clamp.mean() / g_clamp).clamp(0.5, 2.0)
    w_log = torch.sigmoid(TV_K * (mu - g_clamp) / sig)
    tv_wgt = LAM_TV0 * 0.5 * (w_rcp + w_log)

    # ==== Component 1: Isotropic 3-D Charbonnier TV ====
    dz_mod = LAM_DEPTH * dz
    denom = torch.sqrt(
        dx.abs()**2 + dy.abs()**2 + dz_mod.abs()**2 + D_CHAR**2)
    w_iso = tv_wgt / denom
    tvx,  tvy,  tvz = _mp(w_iso * dx),  _mp(w_iso * dy),  _mp(w_iso * dz_mod)
    # Compute divergence (scatter-add)
    tv_grad = torch.zeros_like(x)
    tv_grad[..., :, :-1] += tvx[..., :, :-1]
    tv_grad[..., :,  1:] -= tvx[..., :, :-1]
    tv_grad[..., :-1, :] += tvy[..., :-1, :]
    tv_grad[...,  1:, :] -= tvy[..., :-1, :]
    if S > 1:
        tv_grad[:-1] += tvz[:-1]
        tv_grad[1:]  -= tvz[:-1]

    # ==== Component 2: Multi-slice gradient exclusion ====
    # Penalize co-located edges across slices (DDIP-style)
    excl_grad = torch.zeros_like(x)
    if S > 1 and LAM_EXCL > 0:
        g_mag = torch.sqrt(dx.real**2 + dy.real**2 + eps)
        share = g_mag.sum(dim=0, keepdim=True) - g_mag
        w_ex = _mp(LAM_EXCL * share / (g_mag + D_CHAR))dx_ex,
         dy_ex = w_ex * dx, w_ex * dy
        excl_grad[..., :, :-1] += dx_ex[..., :, :-1]
        excl_grad[..., :,  1:] -= dx_ex[..., :, :-1]
        excl_grad[..., :-1, :] += dy_ex[..., :-1, :]
        excl_grad[...,  1:, :] -= dy_ex[..., :-1, :]


    # ==== Component 3: Slice Gram-orthogonality ====
    # Minimize off-diagonal elements of the slice Gram matrix
    # to encourage independent slice content
    flat = x.view(S, -1)
    if S > 32:        # low-rank random projection for many slices
    k = min(32, max(8, S // 4))if not (hasattr(self, "_rand_proj")
    and self._rand_proj.shape == (k, S)):
    self._rand_proj = torch.randn(k, S, device=dev,
     dtype=flat.real.dtype)Y = self._rand_proj @ flatGk = flat @ Y.T


        frob = torch.linalg.norm(Gk) + eps
        corr_grad = (2.0 * LAM_CORR / frob) * (Gk @ Y)
    else:             # full Gram matrix path
        G = flat.conj() @ flat.T
        G.fill_diagonal_(0.0)
        frob = torch.linalg.norm(G) + eps
        corr_grad = (2.0 * LAM_CORR / frob) * (G @ flat)
    corr_grad = corr_grad.view_as(x)

    # ==== Component 4: Complementary spectral masking ====
    # 6th-order Butterworth filter; low-contrast slices get
    # high-pass,  others get low-pass. Strength is annealed.
    lam_spec = LAM_SPEC0 * math.exp(-max(0, it - 5) / 20.0)
    if lam_spec < 1e-6:
        spec_grad = torch.zeros_like(x)
    else:
        cache = getattr(self, "_freq_cache", None)
        if not (isinstance(cache, dict)
                and cache.get("shape") == (H, W)):
            fy = torch.fft.fftfreq(H, device=dev,
                                   dtype=x.real.dtype).view(-1, 1)
            fx = torch.fft.fftfreq(W, device=dev,
```

```
                                    dtype=x.real.dtype).view(1, -1)
            fr = torch.sqrt(fx**2 + fy**2) + eps
            low  = 1.0 / (1.0 + (fr / F_CUT)**6)
            high = 1.0 - low
            self._freq_cache = {
                "shape": (H, W), "low": low, "high": high}
        low_m = self._freq_cache["low"]
        high_m = self._freq_cache["high"]
        is_low = (g_clamp < g_clamp.mean()).to(x.real.dtype)
        mask = is_low * high_m + (1.0 - is_low) * low_m
        F = torch.fft.fft2(x)
        spec_grad = lam_spec * torch.fft.ifft2(mask * F)

    # ---- Aggregate gradient with robust clamping ----
    grad = tv_grad + excl_grad + corr_grad + spec_grad
    grad = torch.nan_to_num(grad)
    # Stage 1: scale-invariant median clamp
    med = torch.median(grad.abs()) + eps
    grad = grad * torch.clamp(
        4.0 * med / (grad.abs() + eps), max=1.0)
    # Stage 2: per-slice L2 norm clamp
    gn = torch.linalg.norm(
        grad.view(S, -1), dim=1, keepdim=True) + eps
    thr = 4.0 * torch.median(gn)
    grad = (grad.view(S, -1)
            * torch.clamp(thr / gn, max=1.0)).view_as(x)

    # ---- Complex Adam update ----
    m = self._adam_m = b1 * self._adam_m + (1 - b1) * grad
    v = self._adam_v = (b2 * self._adam_v
        + (1 - b2) * (grad.real**2 + grad.imag**2))
    m_hat = m / (1 - b1 ** self._adam_t)
    v_hat = v / (1 - b2 ** self._adam_t)
    # Three-phase LR: warm-up -> exponential decay -> cosine tail
    if it <= LR_WARMUP:
        lr = LR_BASE * it / LR_WARMUP
    else:
        lr = LR_BASE * (LR_DECAY ** (it - LR_WARMUP))
        if it > COS_START:
            cos_fac = 0.5 * (1 + math.cos(
                math.pi * (it - COS_START) / COS_START))
            lr *= max(cos_fac, 0.0)
    step = lr * m_hat / (torch.sqrt(v_hat) + eps)
    # Per-element step clamp for stability
    step = (torch.clamp(step.real, -STEP_CLAMP, STEP_CLAMP)
            + 1j * torch.clamp(step.imag, -STEP_CLAMP, STEP_CLAMP))

    x_new = x - step

    # ---- Per-slice L2 renormalization ----
    old_n = torch.linalg.norm(
        x.view(S, -1), dim=1, keepdim=True).clamp(min=eps)
    new_n = torch.linalg.norm(
        x_new.view(S, -1), dim=1, keepdim=True).clamp(min=eps)
    x_new = (x_new.view(S, -1) * (old_n / new_n)).view_as(x)
    x_new = torch.nan_to_num(x_new)

    self.set_data(x_new)
```

X射线集成电路（算法0165719f，结构相似性指数 0.785）

清单 2：针对 X射线集成电路数据集发现的最佳正则化器。其特点是包含一个内部迭代优化循环，其中振幅去噪子问题迭代 25 次， phase 去噪子问题迭代 15 次；采用结构张量引导的各向异性 Huber 全变分、带 Armijo接受准则的 Barzilai–Borwein 步长，以及对周期性伪影进行自动陷波滤波。

```
def regularize_llm(self):
    """Multi-scale anisotropic DTGV-Huber regulariser
    with adaptive notch filtering."""
    import torch
    import torch.nn.functional as F

    # ---- Helper functions ----
    def _grad(u):
        """Forward finite differences along x and y."""
        gy, gx = torch.gradient(u, dim=(-2, -1))
        return gx, gy

    def _div(px, py):
        """Discrete divergence (adjoint of _grad)."""
        dpx_dx = torch.gradient(px, dim=(-2, -1))[1]
        dpy_dy = torch.gradient(py, dim=(-2, -1))[0]
        return dpx_dx + dpy_dy

    def _lap(u):
        """Discrete Laplacian."""
        return _div(*_grad(u))

    def _orientation_field(a, kernel):
        """Estimate local orientation from structure tensor."""
        gx, gy = _grad(a)
        conv = lambda t: F.conv2d(
            t.unsqueeze(1), kernel, padding=1).squeeze(1)
        J11, J22, J12 = conv(gx*gx), conv(gy*gy), conv(gx*gy)
        theta = 0.5 * torch.atan2(
            2.0 * J12, J11 - J22 + 1e-12)
        return torch.cos(theta), torch.sin(theta)

    def _build_notch_mask(a, k=8, sigma_frac=0.01):
        """Detect periodic artifacts from the power spectrum
        and build a multiplicative notch-rejection mask."""
        Hn, Wn = a.shape[-2], a.shape[-1]
        Famp = torch.fft.fftshift(
            torch.fft.fftn(a, dim=(-2, -1)), dim=(-2, -1))
        power = Famp.real**2 + Famp.imag**2
        # Zero out DC neighborhood
        power[Hn//2-1:Hn//2+2, Wn//2-1:Wn//2+2] = 0
        vals, idx = torch.topk(power.flatten(), k=k)
        sigma = sigma_frac * min(Hn, Wn)
        yy, xx = idx // Wn, idx % Wn
        yv, xv = torch.meshgrid(
            torch.arange(Hn, device=a.device),
            torch.arange(Wn, device=a.device), indexing='ij')
        H = torch.ones_like(power)
        thresh = power.mean() * 10
        for y0, x0, v in zip(yy, xx, vals):
            if v < thresh:
                continue
            d2 = (xv - x0)**2 + (yv - y0)**2
            g = torch.exp(-0.5 * d2 / sigma**2)
            H *= (1.0 - g)
            # Symmetric notch partner
            g2 = torch.roll(torch.roll(
                g, shifts=Hn//2, dims=0), shifts=Wn//2, dims=1)
            H *= (1.0 - g2)
        return H

    # ---- Setup ----
```

```
    z = self.data
    single = (z.ndim == 2)
    if single:
        z = z.unsqueeze(0)
    device = z.device
    c_dtype, r_dtype = z.dtype, torch.float32
     , H, W = z.shape

    # Polar decomposition: separate amplitude and phase
    amp = torch.abs(z).to(r_dtype)
    phi = torch.angle(z).to(r_dtype)
    support = getattr(self, 'support_mask', None)
    if support is not None:
        if support.ndim == 2:
            support = support.unsqueeze(0)
        support = support.to(device=device, dtype=r_dtype)

    # ---- Hyper-parameters ----
    lam_tv   = 3e-3        # amplitude TV strength
    lam_tgv2 = 1e-3        # second-order TGV strength
    lam_phi  = 8e-4        # phase TV strength
    lam_cplx = 5e-4        # complex TV strength
    lam_fft  = 2e-3        # notch filter strength
    n_amp_iter  = 25       # amplitude sub-iterations
    n_phi_iter  = 15       # phase sub-iterations
    n_cplx_iter = 2        # complex TV iterations
    tau_amp0 = 0.24        # initial amplitude step size
    tau_phi0 = 0.24        # initial phase step size
    huber_eps0 = 1e-3      # Huber threshold (annealed)
    anisotropy = 4.0       # directional anisotropy ratio
    orient_every = 5       # re-estimate orientation interval
    tol = 5e-4             # early stopping tolerance
    device_eps = torch.tensor(1e-12, dtype=r_dtype, device=device)

    # Gaussian smoothing kernel for structure tensor
    k = torch.tensor([[1,2,1],[2,4,2],[1,2,1]],
    dtype=r_dtype,  device=device).view(1,1,3,3) / 16.0


    # Build notch mask from amplitude power spectrum
    notch_H = _build_notch_mask(amp[0], k=8).to(r_dtype)

    # ==== Amplitude denoising (25 iterations) ====
    # Anisotropic Huber-TV guided by structure tensor,
    # with Barzilai-Borwein step size and Armijo acceptance
    huber_eps = torch.tensor(huber_eps0, dtype=r_dtype, device=device)
    tau_amp = torch.tensor(tau_amp0, dtype=r_dtype, device=device)
    amp_prev = upd_prev = None
    vx, vy = _orientation_field(amp, k)

    for it in range(n_amp_iter):
        # Re-estimate local orientation periodically
        if it % orient_every == 0:
            vx, vy = _orientation_field(amp, k)

        gx, gy  = _grad(amp)
        g_norm = torch.sqrt(gx*gx + gy*gy + device_eps)

        # Decompose gradient into parallel and perpendicular
        # components relative to the local edge direction
        g_par  = vx*gx + vy*gy
        g_perp = -vy*gx + vx*gy
        alpha = torch.abs(g_perp) / (
            torch.abs(g_par) + torch.abs(g_perp) + device_eps)
        coeff = 1.0 + (anisotropy - 1.0) * alpha

        # Huber-weighted anisotropic TV gradient
        px = gx / (coeff * g_norm + huber_eps)
        py = gy / (coeff * g_norm + huber_eps)
        upd = lam_tv * _div(px, py) - lam_tgv2 * _lap(amp)
        amp_new = amp + tau_amp * upd

```

```
        # Barzilai-Borwein step size with Armijo acceptance
        if amp_prev is not None: s = (amp_new - amp_prev).flatten()
        y = (upd - upd_prev).flatten(); sy = torch.dot(s, y)
        yy = torch.dot(y, y) + device_eps
        tau_bb = torch.clamp(sy / yy, 1e-4, 0.25)
        # Accept BB step only if it reduces energy
        E_old = (lam_tv * g_norm.mean()
        + lam_tgv2 * _lap(amp).abs().mean())
        amp_try = amp + tau_bb * upd; gx_t, gy_t = _grad(amp_try)
        E_new = (lam_tv * torch.sqrt(
        gx_t**2 + gy_t**2 + device_eps).mean()
        + lam_tgv2 * _lap(amp_try).abs().mean()); if E_new < E_old:
        tau_amp = tau_bb

        amp_prev, upd_prev = amp, upd; amp = amp_new
        if support is not None: amp = amp * support
        # Anneal Huber threshold every 4 iterations
        if (it + 1) % 4 == 0: huber_eps *= 0.9
        # Early stopping on relative update magnitude
        rel = (torch.mean(torch.abs(upd))
        / (torch.mean(torch.abs(amp)) + device_eps))
        if rel < tol: break


    amp = torch.clamp(amp, min=0.0)

    # ==== Phase denoising (15 iterations) ====
    # Amplitude-weighted Huber-TV on the phase channel
    tau_phi = tau_phi0
    huber_eps = huber_eps0
    w_phi = lam_phi / (amp + 1e-6)  # weight inversely by amplitude
    for it in range(n_phi_iter): phx, phy = _grad(phi)
    ph_norm = torch.sqrt(phx**2 + phy**2 + device_eps)
    qx = w_phi * phx / (ph_norm + huber_eps)
    qy = w_phi * phy / (ph_norm + huber_eps)
    phi_new = phi + tau_phi * _div(qx, qy)
    # Wrap phase to [-pi, pi]
    phi = (phi_new + torch.pi) % (2.0*torch.pi) - torch.pi
    if (it + 1) % 4 == 0: huber_eps *= 0.9
    rel = (torch.mean(torch.abs(phi - phi_new))
    / (torch.mean(torch.abs(phi)) + device_eps)); if rel < tol: break


    # ==== Joint complex TV (2 iterations) ====
    z = torch.polar(amp, phi).to(c_dtype)
    tau_c = 0.24
    for _ in range(n_cplx_iter): r, i = z.real.to(r_dtype),
     z.imag.to(r_dtype); rx, ry = _grad(r); ix, iy = _grad(i)
    rn = torch.sqrt(rx**2 + ry**2 + device_eps)
    in_ = torch.sqrt(ix**2 + iy**2 + device_eps)
    r = r + tau_c * lam_cplx * _div(rx/(rn+device_eps),
     ry/(rn+device_eps)); i = i + tau_c * lam_cplx * _div(
    ix/(in_+device_eps), iy/(in_+device_eps))
    z = r.to(c_dtype) + 1j * i.to(c_dtype)


    # ==== Notch filtering of periodic artifacts ====
    if lam_fft > 0:
        Zf = torch.fft.fftn(z, dim=(-2, -1))
```

```
        Zf_shift = torch.fft.fftshift(Zf, dim=(-2, -1))
        Zf_shift *= (1.0 - lam_fft * notch_H).to(c_dtype)
        z = torch.fft.ifftn(
            torch.fft.ifftshift(Zf_shift, dim=(-2, -1)),
            dim=(-2, -1))

    # Softplus amplitude activation and support masking
    beta = 1.0 / (amp.std(unbiased=False) + 1e-6)
    z_amp = F.softplus(torch.abs(z), beta=beta)
    z = torch.polar(z_amp.to(r_dtype), torch.angle(z)).to(c_dtype)
    if support is not None:
        z = z * support.to(c_dtype)

    if single:
        self.set_data(z[0])
    else:
        self.set_data(z)
```

脱铁铁蛋白（算法 6d2c9f3f，结构相似性指数 0.836）

列表 3：针对铁蛋白数据集发现的最佳正则化器。实现了一个顺序信号处理流水线：噪声自适应软 Huber 收缩、级联引导滤波、Itoh 相位展开、Perona–Malik 各向异性扩散，以及对原始估计的弱 L2 锚定项。

```
def regularize_llm(self):
    """Hybrid regulariser -- crossover of two parental variants.
    Sequential signal-processing pipeline operating in polar
    (amplitude/phase) coordinates."""
    import math
    import torch
    import torch.nn.functional as F

    data0 = self.data                          # [S, H, W], complex
    dev, cdt = data0.device, data0.dtype
    rdt = data0.real.dtype
    eps = 1e-6
    two_pi = 2.0 * math.pi
    S, H, W = data0.shape

    # ---- Utility functions ----
    def _box(x, r):
        """Fast 2-D box mean via average pooling."""
        if r <= 0:
            return x
        return F.avg_pool2d(x, 2*r+1, stride=1, padding=r)

    def _grad(u):
        """Forward differences via circular shift."""
        gx = torch.roll(u, -1, 2) - u
        gy = torch.roll(u, -1, 1) - u
        return torch.stack((gx, gy), dim=0)

    def _div(p):
        """Discrete divergence (adjoint of _grad)."""
        px, py = p[0], p[1]
        return ((px - torch.roll(px, 1, 2))
                + (py - torch.roll(py, 1, 1)))

    def _Lc(u, c):
        """Anisotropic diffusion operator: div(c * grad(u))."""
```

```
        return _div(c * _grad(u))

    # ==== Step 1: Polar decomposition ====
    amp = data0.abs()
    phi = torch.angle(data0)

    # ==== Step 2: Noise-adaptive soft-Huber shrinkage ====
    # Shrink amplitude toward 1.0 with data-driven threshold
    glob_var = amp.var(unbiased=False)
    k_min, k_max = 0.07, 0.20
    k_soft = k_min + (k_max - k_min) * torch.sigmoid(
    (glob_var - 0.02) * 25.0)
    delta = amp - 1.0
    amp = 1.0 + delta * (k_soft / (delta.abs() + k_soft + eps))

    # ==== Step 3: Cascaded guided filtering ====
    # Multi-radius edge-preserving smoothing driven by
    # local variance
    for r in (1, 2):
    mean_r = _box(amp.unsqueeze(1), r).squeeze(1)var_r = _box(
    (amp - mean_r).pow(2).unsqueeze(1), r).squeeze(1)
    w_r = 1.0 / (var_r + eps)
    amp = (mean_r + w_r * amp) / (1.0 + w_r)

    # Optional 3x3 Gaussian smoothing for high-noise regime
    if glob_var > 0.1:gk = torch.tensor(
    [[1,2,1],[2,4,2],[1,2,1]],dtype=rdt,
      device=dev).reshape(1,1,3,3) / 16.0amp = F.conv2d(
    amp.unsqueeze(1), gk, padding=1).squeeze(1)


    # ==== Step 4: 2-D Itoh phase unwrapping ====
    # Sequential row-then-column pass per slice
    phi_u = phi.clone()
    for s in range(S):
        # Unwrap along rows
        d_row = phi_u[s, :, 1:] - phi_u[s, :, :-1]
        d_row = (d_row + math.pi) % two_pi - math.pi
        phi_u[s, :, 1:] = phi_u[s, :, :-1] + d_row
        # Unwrap along columns
        d_col = phi_u[s, 1:, :] - phi_u[s, :-1, :]
        d_col = (d_col + math.pi) % two_pi - math.pi
        phi_u[s, 1:, :] = phi_u[s, :-1, :] + d_col

    # Remove DC/ramp by subtracting slice mean
    phi_u = phi_u - phi_u.mean(dim=(1, 2), keepdim=True)

    # ==== Step 5: Perona-Malik anisotropic diffusion ====
    # Edge-preserving smoothing of the unwrapped phase
    lam_pm = 0.1       # contrast parameter
    for _ in range(5):
        grad_u = _grad(phi_u)
        mag2 = grad_u[0].pow(2) + grad_u[1].pow(2)
        # Conduction coefficient: preserves edges
        c = 1.0 / (1.0 + mag2 / (lam_pm**2) + eps)
        phi_u = phi_u + 0.1 * _Lc(phi_u, c)

    # ==== Step 6: L2 anchor to original estimate ====
    # Blend regularized result with original to prevent
    # over-smoothing (90% regularized, 10% original)
    alpha = 0.1
    data_new = torch.polar(amp, phi_u)
    data_out = alpha * data0 + (1.0 - alpha) * data_new.to(cdt)

    self.set_data(data_out)
```