CellRank资源排查：稀疏矩阵、核组合与中间结果保存

完整操作教程 · 参考固定 · 往返恢复验收

先检查邻居图与核的稀疏表示，再分别测核构建和命运估计阶段，保存转移矩阵及细胞顺序。

运行边界：示例代码需按安装版本和真实项目输入复核；未提供真实生物数据，语法或小样本检查不代表科学结果和性能验收。

CellRank内存不足，应先定位是读取AnnData、构建kernel、组合转移矩阵，还是GPCCA等下游估计。细胞数相同，邻居图密度、方法与中间对象也会不同。不要在排错时调用toarray把一个本来稀疏的细胞×细胞矩阵展开。

先保留方法前提与细胞范围

本文示例限定已经建立并核验邻居图、且有可信pseudotime列的AnnData。使用PseudotimeKernel与ConnectivityKernel组合，只是一个明确方法示例；没有拟时序依据的数据不能临时填编号替代。仅相似度核不能凭空提供生物学方向。

阶段	主要观察	注意
输入	细胞数、图键、nnz、类型	AnnData还有表达与其他对象占用
单核构建	实际算法和转移矩阵	不能只看原始文件大小
核组合	同时保留的kernel及矩阵	多个副本可能重叠
命运估计	状态数、求解器与中间数组	稀疏输入不保证所有步骤稀疏

不稠密化地检查并保存转移矩阵

保存为kernel_pilot.py，输入为已验收input.h5ad；kernel-pilot目录必须不存在。示例按当前CellRank kernel API执行，不重新改变邻居数或拟时序方向来让内存下降。参数权重0.8与0.2须有方法理由，不能当所有数据的默认推荐。

```
from pathlib import Path
import anndata as ad
import cellrank as cr
import numpy as np
from scipy import sparse

adata = ad.read_h5ad("input.h5ad")
assert adata.obs_names.is_unique
assert "connectivities" in adata.obsp and "pseudotime" in adata.obs
assert np.isfinite(adata.obs["pseudotime"].to_numpy()).all()
graph = adata.obsp["connectivities"]
assert sparse.issparse(graph), "inspect dense neighbor graph before continuing"
print("cells", adata.n_obs, "graph_nnz", graph.nnz)
out = Path("kernel-pilot")
out.mkdir()
pk = cr.kernels.PseudotimeKernel(adata, time_key="pseudotime")
pk.compute_transition_matrix()
ck = cr.kernels.ConnectivityKernel(adata)
ck.compute_transition_matrix()
kernel = 0.8 * pk + 0.2 * ck
kernel.compute_transition_matrix()
matrix = kernel.transition_matrix
assert sparse.issparse(matrix)
matrix = matrix.tocsr()
assert matrix.shape == (adata.n_obs, adata.n_obs)
assert np.isfinite(matrix.data).all() and (matrix.data >= 0).all()
assert np.allclose(np.asarray(matrix.sum(axis=1)).ravel(), 1)
print("csr_bytes", matrix.data.nbytes + matrix.indices.nbytes + matrix.indptr.nbytes)
sparse.save_npz(out / "transition.npz", matrix)
adata.obs_names.to_series(index=adata.obs_names).to_csv(out / "cells.csv")
(out / "cellrank-version.txt").write_text(cr.__version__ + "\n", encoding="utf-8")
```

csr_bytes只是该CSR的数组载荷，不含AnnData、kernel对象、求解器和峰值副本。组合期间可能同时保留多份矩阵；监控应以完整作业和阶段为单位，而不是把这一个数字当总RAM。保存时还要一并记录输入、邻居图设置、拟时序来源和核权重。

把下游估计单独测量

先确认核正常，再测GPCCA的Schur、宏状态和命运概率等所需步骤。状态数、算法及求解器会改变资源；不能承诺图是稀疏的就不会生成较大的中间数组。命运概率计算通过，也仍需检查终末状态与生物学合理性。

写出npz可保留矩阵，但它不是完整分析检查点。读回时必须核对细胞ID及顺序、方向、方法参数和当前API，再按所选入口恢复；仅有维度相同不够。不要加载来源不明的pickle对象来修复缺失的项目记录。

按错误阶段修正

图键缺失先回到邻居图准备；非有限拟时序或孤立区域先检查方法输入；意外dense先追到转换代码。降低邻居数、删细胞、换kernel或状态数会改变分析条件，另立新运行并重新评估，不能当纯资源优化。

保留失败日志、安装版本与同一输入的阶段测量。必要计算在合理表示下仍有确定RAM缺口，再用代表规模复验候选资源；并发核任务也要考虑矩阵峰值是否重叠，不按压缩h5ad体积乘一个固定倍数选容量。

资源记录按阶段填入开始与结束时间、细胞数、基因数、图中非零元素数及实际内存额度。两次比较应使用同一输入和相同算法条件；若减少细胞或改变核权重，结果定义也变化，需要另做科学核对。缓存矩阵时同时保存细胞顺序及生成参数，不能只保留一个没有来源的方阵文件。

定位内存究竟花在哪个阶段

AnnData文件大小不是运行时内存上限。读取h5ad后，表达矩阵、obs/var、邻居图、降维、kernel对象和临时拷贝可能同时存在。先在单细胞小样本上记录n_obs、n_vars、obsp图的非零元素、CSR/CSC类型，以及进程峰值RSS；每次只运行一个阶段，避免同时构建多个kernel让测量结果无法归因。

CellRank的稀疏转移矩阵仍有索引、数据和行指针开销；当图密度升高时，实际存储会比预期大。矩阵载荷估算只包括CSR的data、indices和indptr数组，不包括Python对象、AnnData副本、求解器和操作系统页缓存。申请资源时应使用代表性输入和完整工作流峰值，而不是仅凭npz大小或平均每细胞字节数简单外推。

要区分构建kernel失败和后续宏状态估计失败。前者检查输入时间键、邻居图、细胞顺序和矩阵结构；后者检查GPCCA状态数、稀疏线性代数依赖和求解器日志。错误栈最末一行常只是内存分配发生位置，前面几步可能已经保留了多个大对象，因此最好给每一阶段加时间戳、打印矩阵形状和稀疏度。

逐项判断稀疏矩阵是否正确

稀疏矩阵结构检查可以验证维度、有限值、非负性和行归一化，但这些是结构条件，不代表转移方向符合生物学。含自环或吸收状态时，边界行为应按所用kernel及版本文档解释。若行和偏离1，先判断是否存在零出度细胞、局部图断裂或某个kernel未完成计算，不要直接给每行除以总和来掩盖异常。

保存npz前记录矩阵的行列细胞ID顺序、时间键来源、邻居图键和kernel权重。读回后可核对行列名清单SHA、矩阵形状、非零数以及选定行的概率分布。只有维度一样并不能证明细胞顺序相同；错位的方阵也可能顺利被下游接受，却把转移关系关联到错误细胞。

组合kernel的权重表达方法选择，不能只当调参降低内存的旋钮。分别比较单核和组合核的生物学含义、矩阵稀疏度与敏感性；任何权重变化都生成新方法版本。避免为了省内存把稀疏矩阵显式转成dense，或通过删除低概率边悄悄更改图；若确需图裁剪，应记录阈值并比较结果变化。

保存可恢复信息并进行资源评估

如果需要跨阶段恢复，除矩阵外还应保存AnnData输入散列、预处理版本、细胞顺序、kernel配置和CellRank依赖版本。某些估计器对象带有内部缓存或依赖，不保证pickle在升级后仍可读。更稳妥的做法是将可验证的输入、稀疏矩阵和参数清单作为显式产物，需要恢复时重建对象，再检查恢复后的矩阵和小规模结果是否一致。

内存评估采用“先单任务、再并发”的顺序。首先对与正式数据相同的输入和流程运行一次；接着在同一资源限制下复测，以排除偶然缓存影响；最后才测试计划中的并发数。多个进程各自读入AnnData或重复计算kernel时，资源总量可能接近每进程峰值之和。并发会提高吞吐，但也可能令每个任务都因内存争用失败。

出现内存错误时，先核实进程被系统OOM杀死还是Python主动抛出MemoryError，再查看实际作业额度、节点可用内存与邻居图密度。若环境只提供较小资源，先用同一输入减少无关对象副本或按官方支持方式分阶段计算；若算法阶段本身确有矩阵规模限制，测试专门稀疏求解器方案并确认结果一致。不能仅凭服务器标称内存推断容器或作业实际可用量。

用小型回归检查防止后续版本悄悄变形

为固定版本建立一个小型基准对象和预期结构检查，例如细胞数、矩阵形状、非零元素数量范围、行和容差及几行代表性细胞的邻居分布。基准对象不含真实敏感数据时，可放在代码包中；若只能使用项目数据，应遵循访问控制并保存清单而非拷贝原始数据。更新CellRank或SciPy后先跑这些检查，再运行完整任务。

结构测试只能发现接口、顺序或稀疏性回归，不能证明命运状态或生物学解释不变。版本升级后还需比较下游终末状态、宏状态和概率分布，重点看细胞标签是否大范围变化。若变化明显，检查默认参数和求解器是否有变化，并保留旧版结果作为对照，不应直接把新结果覆盖到共享项目目录。

共享服务器上的隔离与并行

在共享机器上，单个进程的RSS不等于作业可用内存。容器、调度器或服务单元可能对任务设定更小的硬限制。启动前查看实际cgroup或调度额度，写入作业记录。kernel计算应在独立环境和独立输出目录中运行，避免多个用户覆盖相同缓存。原始AnnData建议只读，临时文件写入专属路径，完成后核实存储策略再清理。

并发估算不要只看CPU线程。两个任务同时构建相似矩阵时，峰值可能完全重合；先用测得的单任务高水位和预留余量估算上限，然后低并发试跑并观察系统OOM记录。若还有其他课题组共享节点，应由作业调度或明确资源配额控制竞争，而不是让每个脚本默认使用全部CPU和内存。对于持续运行的项目，记录各项目的实际峰值比给出固定“推荐配置”更可靠。

按阶段记录一张资源表

推荐资源记录列包括stage、n_obs、n_vars、graph_nnz、kernel类型、矩阵格式、start_time、elapsed_seconds、peak_rss、作业内存限制、退出码和输出散列。kernel构建与GPCCA估计分开记，才知道资源缺口发生在哪里。把同一数据和参数的重复运行放在同一表中，记录是否冷启动及缓存情况，避免把缓存热机的较低时间误当正式基线。

当需要比较资源变化时，先在一份固定输入上测量，再比较同样方法下不同作业额度。若同时改了图构建、核权重和状态数，即使运行更快也不能判断是服务器变大还是问题定义变简单。结果表应包含方法配置散列，让资源数据与科学结果一一对应。

常见错误与快速分流

“key not found”通常是AnnData缺少指定图或时间列，应检查obsp/obs的键名和预处理输出；“expected sparse matrix”则要确认图是否在早期步骤被稠密化。若组合kernel后内存突然上升，检查两个核是否同时驻留以及是否生成临时副本。此时先停止不必要的并发和重复对象，再在隔离目录复测，不要把服务器上其他进程一起结束。

若下游特征或状态数不合理，回看kernel方向及状态设定，而不只是关注资源。若系统记录显示cgroup OOM，实际额度低于进程需求；若是Python报错，应结合异常堆栈看是算法限制、求解器依赖还是输入数据。所有诊断要附带时间、进程

号和日志位置，方便和调度器或管理员的记录对应。

每个项目还应明确哪些中间结果属于可重建缓存、哪些属于必须归档的最终证据。缓存删除后能否按固定版本、固定输入重建，应先抽查一次；若不能重建，就不能仅按“临时文件”处理。存储节省与可复现性需要一起评估。

定期复查稀疏格式和图密度的原因也包括软件升级：同一份数据经过不同预处理或邻居图算法，非零边数会变化。保留建图参数和输入版本，才能解释资源曲线变化。

配套练习包

本PDF对应的ZIP包含本文代码块、主题练习模板或合成数据、README和SHA256校验文件。请从同一教程页面下载练习包；先核对校验值，再在独立目录复现。

参考资料

CellRank：ConnectivityKernel与保存接口 · 来源资料

CellRank：PseudotimeKernel · 来源资料

CellRank：GPCCA估计阶段 · 来源资料

操作建议：先在合成小样本上验证流程，再依据项目参考、标签合同、存储策略与获配资源验收真实数据。