arXiv:2301.10022v2 [cs.LG] 6 May 2024

作为非线性偏微分方程无网格求解器的 Koopman 神经算子*

Wei Xi ong,† Xiaomeng Huang,† Ziyang Zhang,‡ Ruixuan Deng,§ Pei Sun,¶ 和 Yang Tian**

各种偏微分方程（PDE）缺乏解析解，由此催生了一系列用于数值求解的计算技术。尽管在开发神经算子（一类基于神经网络的偏微分方程求解器）方面取得了许多最新进展，但在学习非线性偏微分方程族的长期行为时，这些求解器的准确性和可解释性会降低。本文提出一种新的神经算子——Koopman 神经算子（KNO），以应对这些挑战。我们的方法同样旨在学习 Banach 空间之间的无限维映射，以充当目标偏微分方程族的解算子；与现有模型不同，我们将方程求解过程表述为一个非线性动力系统。通过近似 Koopman 算子——一个支配动力系统所有可能观测的无限维算子——使其作用于该动力系统的流映射，我们便能够通过求解简单的线性预测问题，等价地学习非线性偏微分方程族的解。我们在五个具有代表性的偏微分方程（例如纳维–斯托克斯方程和瑞利–贝纳德对流）以及三个真实动力系统（例如全球水汽分布模式和西边界流）上，验证了 KNO 在网格无关的长期预测和零样本预测方面的表现。在这些实验中，与以往最先进的模型相比，KNO 展现出显著优势，表明其有望支持科学与工程领域的多种应用（例如偏微分方程求解、湍流建模和降水预报）。

I. 引言

A. 偏微分方程求解器至关重要

在科学与工程中，偏微分方程（PDE）是描述各种问题（例如流体力学、量子力学和土木工程问题）的基础工具 [1]。然而，尽管偏微分方程的求解已取得重大进展 [2], ，许多重要的偏微分方程（如纳维–斯托克斯方程）仍然缺乏解析解 [3]。因此，在实际应用中求解复杂偏微分方程的迫切需求，催生了多种用于计算近似偏微分方程解的技术 [4]。

给定 $\Phi = \Phi\left(D; \mathbb{R}^{d_\phi}\right)$（输入的巴拿赫空间），以及$\Gamma = \Gamma\left(D; \mathbb{R}^{d_\gamma}\right)$（解的巴拿赫空间），它们定义在有界开集 $D \subset \mathbb{R}^d$ 上，这些偏微分方程求解器预期能够近似一个解算子 $\mathcal{Q}$，该算子关联 Φ 与 Γ，适用于典型的时变偏微分方程族

$$\partial_t \gamma\left(x_t\right) = \left(\mathcal{L}_\phi \gamma\right)\left(x_t\right) + \eta\left(x_t\right),\ x_t \in D \times T, \tag{1}$$

$$\gamma\left(x_t\right) = \gamma_B,\ x_t \in \partial D \times T, \tag{2}$$

$$\gamma\left(x_0\right) = \gamma_I,\ x_0 \in D \times \{0\}. \tag{3}$$

在式 (1-3) 中，γ_B 和 γ_I 表示边界条件和初始条件，集合 $T = [0, \infty)$ 是时间域，微分算子 $\mathcal{L}_\phi$ 根据 ϕ 确定，固定函数 $\eta(\cdot)$ 属于由 $\mathcal{L}_\phi$ 确定的适当函数空间，而 $\gamma(\cdot)$ 是偏微分方程族的解。这一表述导出了如下解算子 $\mathcal{Q}$：$(\phi,\ \gamma_B,\ \gamma_I) \mapsto \gamma$；当边界条件和初始条件为常数时，该算子可简化为 $\mathcal{Q}$：$\phi \mapsto \gamma$。为方便起见，在后续推导中，我们始终考虑固定的边界条件和初始条件。在实际应用中，研究人员会考虑由 θ 参数化的参数化对应形式 $\mathcal{Q}_\theta \simeq \mathcal{Q}$，以定义优化问题 [5]。

B. 现有的偏微分方程求解器种类繁多

迄今为止，人们已开发出多种偏微分方程求解器，通常可将其归类如下：

(1) 经典数值求解器。最早且最常见的偏微分方程求解器，例如有限元 [6] 和有限差分 [7] 方法，会根据特定的网格设计对空间进行离散化，以求解偏微分方程。这些方法依赖于网格粒度：准确度倾向于使用精细网格，而效率则更适合粗糙网格 [8]。因此，当目标偏微分方程较为复杂时，它们不可避免地要在准确度和效率之间进行权衡 [9]。

* 通信请联系 X.R.D.、P.S. 和 Y.T.。
† 清华大学地球系统科学系，中国北京 100084。
‡ 华为技术有限公司中央研究院马尔可夫实验室，2012 实验室，中国。
§ruiton@umich.edu；密歇根大学统计系，美国密歇根州安娜堡，MI 48104。
¶peisun@cityu.edu.mo；澳门城市大学健康科学学院，澳门，999078，中国。** tyanyang04@gmail.com；澳门城市大学健康科学学院，澳门，999078，中国；同时任职于澳门城市大学数据科学学院，澳门，999078，中国。

(2) **基于神经网络的求解器**。为革新偏微分方程求解的计算技术，研究人员提出了三类基于神经网络的求解器，以快速近似或增强经典方法 [10, 11]:

(2a) **网格依赖的有限维算子**。 第一类求解器在将定义域 D 和 T 离散化为 x 和 y 个网格后，将解算子近似为有限维欧氏空间之间的参数化神经网络，即 $\mathcal{Q}_\theta$: $\mathbb{R}^x \times \mathbb{R}^y \times \Theta \to \mathbb{R}^x \times \mathbb{R}^y$ [12–14]。这些求解器依赖于网格，并且需要针对 n 的不同取值进行微调，因此泛化能力有限 [5]。

(2b) **神经有限元方法**。 第二类求解器直接将方程解 $\gamma(\cdot)$ 参数化为神经网络，由此等价地得到 $\mathcal{Q}_\theta$: $D \times T \times \Theta \to \mathbb{R}$[10, 15–18]。尽管这类求解器不依赖网格且精度较高，但它们只能学习偏微分方程的某个特定实例，而非整个方程族 [5]。因此，与经典数值方法类似，每当实例发生变化时，这些求解器都需要重新设计网络并进行训练。同时，这些求解器大多不适用于底层偏微分方程仍然未知的情况（有限元网络[18], 是一个例外，它支持基于真实数据进行时空预测）。

(2c) **神经算子**。 第三类求解器通过神经网络学习与网格相关的无限维解算子，即 $\mathcal{Q}_\theta$: $\Phi \times \Theta \to \Gamma$ [5, 9, 19–23]。这些求解器通过以适用于不同离散化的方式学习网络参数，克服了对网格的依赖。

由于这些求解器直接学习解算子，因此只需针对目标偏微分方程族训练一次。要生成该偏微分方程族不同实例的方程解 $\gamma(\cdot)$，只需对网络进行一次前向传播，计算上十分高效 [5, 9]。虽然神经算子最初因核积分算子的计算成本高昂，在竞争力上不如其他基于神经网络的求解器，但最新方法——傅里叶神经算子 [9], ——通过快速傅里叶变换解决了这一限制。

与经典数值求解器相比，基于神经网络的求解器，尤其是神经算子，在处理偏微分方程较为复杂的科学与工程问题时效率更高 [5, 9]。因此，我们的研究主要聚焦于神经算子的设计。

C. 现有偏微分方程求解器面临的挑战

尽管神经算子在理论基础（例如，参考文献 [22]）、近似器设计（例如，参考文献 [9, 23]）和应用（例如，参考文献 [24]）方面已取得显著进展，但现有的神经算子求解器仍面临诸多挑战，其中一个关键问题是现有模型学习复杂偏微分方程长期动力学的能力有限。

为理解这一挑战，我们来考虑一个与方程（1-3） [5] 相关的格林函数， $\mathcal{J}_\phi$: $(D \times T) \times (D \times T) \to \mathbb{R}$

$$\gamma\left(x_{t+\varepsilon}\right)=\int_{D\times\{t\}} \mathcal{J}_\phi\left(x_t, y_t\right) \eta\left(y_t\right) \mathrm{d}y_t, \ \forall\ x_t \in D \times\{t\}. \tag{4}$$

神经算子 [5] 的初始思路是将该格林函数参数化为一个核积分算子（即下文中与 κ_θ 相关的积分项），并定义一种神经网络的迭代更新策略

$$\widehat{\gamma}\left(x_{t+\varepsilon}\right)=\sigma\left(W \widehat{\gamma}\left(x_t\right)+\int_{D\times\{t\}} \kappa_\theta\left(x_t, y_t, \phi\left(x_t\right), \phi\left(y_t\right)\right) \widehat{\gamma}\left(y_t\right) \mathrm{d}y_t\right), \ \forall\ x_t \in D \times\{t\}, \tag{5}$$

其中， $\varepsilon \in (0, \infty)$表示时间差，符号 $\widehat{\gamma}$: $D \times T \to \mathbb{R}^{d_{\widehat{\gamma}}}$ 表示通过特定嵌入生成的方程解 γ 的神经网络表示，映射 σ: $\mathbb{R} \to \mathbb{R}$ 表示逐元素激活函数，符号 W: $\mathbb{R}^{d_{\widehat{\gamma}}} \to \mathbb{R}^{d_{\widehat{\gamma}}}$ 表示线性变换，而 κ_θ: $\mathbb{R}^{2(d+d_\phi)} \to \mathbb{R}^{d_{\widehat{\gamma}}}$ 是由 θ [5]参数化的神经网络。使用众所周知的傅里叶神经算子可以提高这种迭代更新的计算效率

[9], 即，式（5）可改写为 $\widehat{\gamma}(x_{t+\varepsilon}) = \sigma\left[W\widehat{\gamma}(x_t) + \mathcal{F}^{-1}(\mathcal{F}(\kappa_\theta) \cdot \mathcal{F}(\widehat{\gamma}(x_t)))\right]$ 对于任意 $x_t \in D \times\{t\}$ （符号 $\mathcal{F}(\cdot)$表示傅里叶变换，在实际应用中可通过快速傅里叶变换实现）。

乍一看，式（5）中的迭代更新类似于动力系统视角: 我们研究一个无限维非线性动力系统 $\gamma_t = \gamma(D \times \{t\})$ 的演化映射 ζ: $\mathbb{R}^{d_\gamma} \times T \to \mathbb{R}^{d_\gamma}$ （记号 $\gamma(D \times \{t\})$ 表示函数 γ 作用于所有

集合中的元素 $D \times \{t\}$）

$$\gamma_{t+\varepsilon} = \gamma_t + \int_t^{t+\varepsilon} \zeta\left(\gamma_\tau, \tau\right) \mathrm{d}\tau, \ \forall t \in T. \tag{6}$$

在这个动力系统中，每个快照 γ_t 都是定义域 D 上时刻 t分布的一个场，即方程 (1-3) 的时变方程解。场 γ_t 的时间演化由无限维演化映射 ζ 表征： $\mathbb{R}^{d_\gamma} \times T \to \mathbb{R}^{d_\gamma}$

$$\partial_t \gamma_t = \zeta\left(\gamma_t, t\right), \ \forall \gamma_t \in \mathbb{R}^{d_\gamma} \times T, \tag{7}$$

这正是动力系统理论[25, 26]的核心。当定义域 D 表示一个有界空间范围时，方程 (5) 可被视为神经科学 [27], 、地球物理学 [28], 、流体力学 [29], 和工程学 [30]中各种现象的一般时空动力学模型。

对于大多数非线性偏微分方程族，所关注的演化映射 $\zeta(\cdot, \ \cdot)$ 甚至比方程解 $\gamma(\cdot)$ 本身更加复杂。按照现代动力系统理论 [31], 的术语，方程 (6) 通常会导出流映射 θ 的余循环性质。

$$\theta_t^{t'} = \theta_{t+\tau}^{t'} \circ \theta_t^{t+\tau}, \ \forall t \leq t + \tau \leq t' \in T \tag{8}$$

其中，记号 $\circ$ 表示映射的复合。该余循环性质定义了目标偏微分方程族的方程解 $\gamma(\cdot)$ 如何在相邻时间区间内演化。随着式 (6) 中的时间差 ε增大，挑战往往随之出现，因为 $\gamma(\cdot)$ 的动力系统可能是非自治的（即，与 $\zeta(\cdot, \cdot)$ 相关联的流映射 θ 依赖于时间），也可能是自治的（即，流映射 θ 不依赖于时间，因此 $\partial_t \zeta(\cdot, t) \equiv 0$），具体取决于不同的偏微分方程族 [25, 26]。如果非线性映射 $\zeta(\cdot, \cdot)$ 依赖于时间，那么对偏微分方程长期行为（即，$\varepsilon \to \infty$）进行准确建模将越来越具有挑战性，因为方程解 $\gamma(\cdot)$ 的演化规则会随时间变化。为了保持准确度，现有模型需要扩大模型规模或增加复杂度，因此不可避免地面临准确度与效率之间的权衡。当然，模型也可以将策略限制为只预测短期动力学（即，$\varepsilon = 1$）。然而，偏微分方程的长期动力学学习是众多重要应用的基石，例如天气预报 [32],疫情防控 [33], 经济分析 [34], 以及地球建模 [35]。因此，克服偏微分方程长期行为学习方面的局限，对于优化偏微分方程求解器十分必要，这也是本研究的核心目标。

D. 我们的框架与贡献

本文尝试通过提出一种名为 Koopman 神经算子（KNO）的新型神经算子，克服预测偏微分方程长期动力学的挑战。我们的框架和贡献总结如下：

(1) 偏微分方程族作为方程解的非线性动力系统的长期行为。除了学习整个目标偏微分方程族的解算子外，我们还同时形式化了式（6）所描述的方程解非线性动力系统。这一表征有助于利用动力系统理论优化式（5）中的神经算子迭代更新策略。

（2）**通过 Koopman 算子实现非线性动力学的等效线性预测**。为捕捉复杂的长期动力学，我们的模型经过设计，用于学习 Koopman 算子——一种作用于动力系统所有观测的无限维线性算子，使其作用于方程解动力系统的演化映射 $\zeta(\cdot, \ \cdot)$。通过这种方式，我们可以将原始任务转换为一个等效但更简单的线性预测问题。

（3）**在零样本预测和长期预测中平衡准确度与效率**。我们将 KNO 与现有的先进模型（例如傅里叶神经算子 [9] 及其他竞争模型）进行比较，包括在零样本预测中（即在未训练过的离散化粒度或未训练过的预测区间上进行测试）以及在五种具有代表性的偏微分方程（例如纳维–斯托克斯方程和 Rayleigh-B´énard 对流）和三个真实动力系统（例如全球水汽分布模式和西边界流）上的长期预测实验。在这些实验中，KNO 被证明是一种具有竞争力的方法，可用于实现偏微分方程求解和动力系统建模。

开源代码发布于 https: // github.com/Koopman-Laboratory/KoopmanLab。关于这个工具箱，也可参阅文献 [36] 。

II. KOOPMAN 神经算子：理论与计算

A. 时变 Koopman 算子 r

方程（6-8）中的非线性且可能为非自治的动力系统，使长期预测成为一项艰巨挑战。在实践中，研究者总是希望处理线性动力系统（即，$\partial_t \gamma_t = \mathcal{A} \gamma_t$ 其中 $\mathcal{A}$ 是一个线性算子），其动力学足够简单，且能够被准确建模。因此，一个自然的问题是：我们能否将方程（6-8）中的原始系统转换为线性系统，从而在误差合理的前提下简化学习任务?

根据现代动力系统理论，可以通过构造 Koopman 算子 $\mathcal{K}$（一种无限维线性算子，用于描述动力系统的所有可能观测），并使其作用于流映射 $\zeta(\cdot, \ \cdot)$, 从而

在适当的空间 [31] 中将原始动力系统线性化。举例来说，作为简单说明，我们以系统 γ_t 为自治系统（即，$\theta_t^{t'}$ 可简化为 $\theta^{t'}$）的情况为例。以时间差 ε为参数的一族 Koopman 算子 $\mathcal{K}^\varepsilon$：$\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)\to\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)$，基于一组观测函数（或称测量函数）$\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)=\{\mathbf{g}|\mathbf{g}$ 定义如下：$\mathbb{R}^{d_\gamma}\times T\to\mathbb{C}\}$ [31]

$$\mathcal{K}^\varepsilon\mathbf{g}\left(\gamma_t\right)=\mathbf{g}\left(\theta^\varepsilon\left(\gamma_t\right)\right)=\mathbf{g}\left(\gamma_{t+\varepsilon}\right),\ \forall\gamma_t\in\mathbb{R}^{d_\gamma}\times T. \quad (9)$$

给定由 $\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)$ 定义的适当空间，我们可以通过式（9）将 γ_t 的动力学线性化。这一思路已在流体动力学 [37]，机器人学 [38]，等离子体物理学 [39],和神经科学 [40] 中取得显著成效。

不同于现有的基于机器学习的 Koopman 算子模型——它们要么仅适用于自治动力系统（例如式（9）所描述的情况） [41–44] ，要么需要先验了解非自治动力系统中 Koopman 算子的特征值谱（例如实特征值和复特征值的数量） [45], ——我们的框架关注一种更一般的情况，即考虑一种适用于非自治和自治动力系统的时变 Koopman 算子 [46]

$$\mathcal{K}_t^{t+\varepsilon}\mathbf{g}\left(\gamma_t\right)=\mathbf{g}\left(\theta_t^{t+\varepsilon}\left(\gamma_t\right)\right)=\mathbf{g}\left(\gamma_{t+\varepsilon}\right),\ \forall t\le t+\varepsilon\in T. \quad (10)$$

如式（10）所示，该 Koopman 算子支配着由 $\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)$ 定义的空间中 $\mathbf{g}(\gamma_t)$ 的时变线性演化流

$$\partial_t\mathbf{g}\left(\gamma_t\right)=\lim_{\varepsilon\to0}\frac{\mathcal{K}_t^{t+\varepsilon}\mathbf{g}\left(\gamma_t\right)-\mathbf{g}\left(\gamma_t\right)}{\varepsilon}. \quad (11)$$

在数学中，式（9）定义的 Koopman 算子的伴随算子是动力系统的 Perron-Frobenius 算子 [47] ，而相关 Lie 算子（详情见 A）的伴随算子是哈密顿动力学的 Liouville 算子 [48, 49]。这些性质将我们的方法与统计物理学和量子力学中关于动力系统线性表示的著名理论联系起来 [50]。

B. 偏微分方程求解中的时变 Koopman 算子

为了解 Koopman 算子实现的$\mathbf{g}(\gamma_t)$线性化与偏微分方程求解之间的关系，我们可以考虑由方程（1-3）描述的可积版本的 Lax 对（$\mathcal{M},\mathcal{N}$）偏微分方程 [51]

$$\mathcal{M}=\mathsf{D}_x^n+\alpha\gamma\left(x_t\right)I,\ \alpha\in\mathbb{C}, \quad (12)$$
$$\mathcal{M}\psi\left(x_t\right)=\lambda\psi\left(x_t\right),\ \lambda\in\mathbb{C}, \quad (13)$$
$$\partial_t\psi\left(x_t\right)=\mathcal{N}\psi\left(x_t\right), \quad (14)$$

其中，D_x^n 是 n阶全导数算子，I是恒等算子。我们注意到，式（13）实际上定义了与算子$\mathcal{M}$在时刻 t相关的一个特征值问题。对式（13）求时间导数，可得线性算子 $\mathcal{M}$ 和 $\mathcal{N}$之间的关系

$$\left(\partial_t\mathcal{M}+\mathcal{M}\mathcal{N}-\mathcal{N}\mathcal{M}\right)\psi\left(x_t\right)=\partial_t\lambda\psi\left(x_t\right). \quad (15)$$

该关系意味着

$$\partial_t\mathcal{M}+[\mathcal{M},\mathcal{N}]=0, \quad (16)$$

其中，$[\mathcal{M},\mathcal{N}]=\mathcal{M}\mathcal{N}-\mathcal{N}\mathcal{M}$ 表示算子 [52]的对易子。根据式（12–16），结合式（11），我们可以确定算子 $\mathcal{N}$ 与时变 Koopman 算子 $\mathcal{K}_t^{t+\varepsilon}$之间的关系

$$\psi\left(D\times\{t\}\right)=\mathbf{g}\left(\gamma_t\right)\Rightarrow\mathcal{N}=\lim_{t+\varepsilon\to t}\frac{\mathcal{K}_t^{t+\varepsilon}\mathbf{g}\left(\gamma_t\right)-\mathbf{g}\left(\gamma_t\right)}{\varepsilon}. \quad (17)$$

需要注意的是，当方程解 γ_t 的演化是自治的时，式 (17) 仍然成立（即可以直接在式 (17) 中考虑一个时不变的 Koopman 算子 $\mathcal{K}$ ）。

因此，通过 Koopman 算子对$\mathbf{g}(\gamma_t)$进行线性化，与 Lax 对密切相关，并且可以从可积偏微分方程 [51]的逆散射变换角度加以理解。这种关系已在数学和物理学领域得到全面研究 [53–56], ，从而确保了在偏微分方程求解过程中使用 Koopman 算子理论的有效性。

C. Koopman 算子近似的计算思路

在形式化定义时变 Koopman 算子并确认其与偏微分方程求解的关系之后，我们接下来的任务是探索一种可用于计算表示该 Koopman 算子的思路。受 Hankel-DMD [57], sHankel-DMD [58], 和 HAVOK [59] 算法的启发，我们考虑由单位时间步长定义的可观测量的 Krylov 序列 [60] $\varepsilon\in[0,\infty]$

$$\mathcal{R}_n=\left[\mathbf{g}\left(\gamma_0\right),\mathbf{g}\left(\gamma_\varepsilon\right),\mathbf{g}\left(\gamma_{2\varepsilon}\right),\ldots,\mathbf{g}\left(\gamma_{n\varepsilon}\right)\right], \quad (18)$$
$$=\left[\mathbf{g}\left(\gamma_0\right),\mathcal{K}_0^\varepsilon\mathbf{g}\left(\gamma_0\right),\mathcal{K}_\varepsilon^{2\varepsilon}\mathcal{K}_0^\varepsilon\mathbf{g}\left(\gamma_0\right),\ldots,\mathcal{K}_{(n-1)\varepsilon}^{n\varepsilon}\mathcal{K}_{(n-2)\varepsilon}^{(n-1)\varepsilon}\cdots\mathcal{K}_0^\varepsilon\mathbf{g}\left(\gamma_0\right)\right], \quad (19)$$

这在用于计算矩阵特征值 [60]的 Krylov 子空间方法中也能见到。通过系统的 Hankel 矩阵表示，可以高效地采样此类序列

$$\mathcal{H}_{m\times n}=\begin{bmatrix}\mathbf{g}\left(\gamma_0\right) & \mathbf{g}\left(\gamma_\varepsilon\right) & \cdots & \mathbf{g}\left(\gamma_{n\varepsilon}\right)\\ \vdots & \vdots & \vdots & \vdots\\ \mathbf{g}\left(\gamma_{(m-1)\varepsilon}\right) & \mathbf{g}\left(\gamma_{m\varepsilon}\right) & \cdots & \mathbf{g}\left(\gamma_{(m+n-1)\varepsilon}\right)\end{bmatrix}, \tag{20}$$

其延迟嵌入的维度为 $m\in\mathbb{N}^+$。 $\mathcal{H}_{m\times n}$ 的列近似表示 Krylov 子空间中的函数。我们考虑该 Krylov 子空间，是因为它有可能张成 Koopman 算子的一个不变子空间 $\mathbb{K}\subset\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)$。

$$\mathbb{K}=\operatorname{span}\left(\mathcal{R}_n\right)\simeq\operatorname{span}\left(\mathcal{H}_{(m,n)}\right) \tag{21}$$

若 $n\geq\dim\left(\mathbb{K}\right)-1$（记号 $\dim\left(\cdot\right)$ 表示维数）。为了解这些可能性，我们考虑将原始 Koopman 算子 Galerkin 投影到 $\mathbb{K}$，记为 $\widehat{\mathcal{K}}_t^{t+\varepsilon}$: $\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)\to\mathbb{K}$对任意 $t\in T$。对于任意函数 $\mathbf{h}\left(\cdot\right)\in\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)$，我们有

$$\langle\widehat{\mathcal{K}}_t^{t+\varepsilon}\mathbf{h}\left(\gamma_t\right),\mathbf{g}\left(\gamma_{i\varepsilon}\right)\rangle=\langle\mathcal{K}_t^{t+\varepsilon}\mathbf{h}\left(\gamma_t\right),\mathbf{g}\left(\gamma_{i\varepsilon}\right)\rangle,\ \forall i=0,\ldots,m, \tag{22}$$

其中 $\langle\cdot,\cdot\rangle$ 是内积。根据文献 [61,62], , 限制在 $\mathbb{K}$ 上的 Koopman 算子可近似原始 Koopman 算子

$$\lim_{m\to\infty}\int_{\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)}\|\widehat{\mathcal{K}}_t^{t+\varepsilon}\mathbf{h}\left(\gamma_t\right)-\mathcal{K}_t^{t+\varepsilon}\mathbf{h}\left(\gamma_t\right)\|_F\mathrm{d}\mu=0,\ \forall\mathbf{h}\left(\cdot\right)\in\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right) \tag{23}$$

若原始 Koopman 算子有界，且 $\mathbb{K}$恰好是其不变子空间（或简单地说 $n\to\infty$）。记号 μ表示 $\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)$上的一个测度， $\|\cdot\|_F$是 Frobenius 范数。因此，我们期望通过一个受限算子来近似原始 Koopman 算子，使得

$$\mathcal{H}_{m\times n}\left(k+1\right)=\widehat{\mathcal{K}}_{k\varepsilon}^{(k+1)\varepsilon}\mathcal{H}_{m\times n}\left(k\right),\ \forall k=1,\ldots,n, \tag{24}$$

其中， $\mathcal{H}_{m\times n}(k)$表示 $\mathcal{H}_{m\times n}$的第 k列。

一个难点在于，上述所有推导（即 Krylov 子空间方法、Hankel 矩阵表示和 Galerkin 投影）最初都是针对自治系统 [31, 57–59] 提出的。尽管我们可以像式（18- 24）所示，为时变 Koopman 算子构造这些数学表达式，但在计算实现中，这种构造不可避免地需要进行在线优化，因为所研究的 Koopman 算子可能随时间变化。昂贵的在线优化不利于求解偏微分方程，这促使我们考虑其他方案。正如文献 [63], 所建议的，我们可以假设 Koopman 算子具有常数形式 $\widehat{\mathcal{K}}_t^{t+\varepsilon}$，但前提是 ε 足够小（即时间区间足够短，使得在此期间 $\widehat{\mathcal{K}}_t^{t+\varepsilon}$ 的变化可以忽略不计）。基于这一假设，我们可以进一步考虑一个条件，使得$\mathbf{g}(\gamma_t)$ 的动力系统近似遍历（即，方程解的观测 $\mathbf{g}(\gamma_t)$ 最终会遍历 $\mathbb{R}^{d_\gamma}$ 中所有可能的状态 $t\to\infty$，因此$\mathbf{g}(\gamma_t)$ 在某一特定状态上所花费的时间比例等于该状态出现的概率） [57, 64]。在这一条件下，随着时间趋于无穷，时间平均将近似于可观测量的真实期望。基于这一条件，我们可以定义一个由时间差 ε 控制的 Koopman 算子期望

$$\overline{\mathcal{K}}_\varepsilon=\lim_{t\to\infty}\frac{1}{t}\int_{[0,t)}\mathbf{g}\left(\gamma_\tau\right)^{-1}\mathbf{g}\left(\gamma_{\tau+\varepsilon}\right)\mathrm{d}\tau\simeq\operatorname*{argmin}_{P\in\mathbb{R}^{d_\gamma+1}}\sum_{k=1}^{n}\|\mathcal{H}_{m\times n}\left(k+1\right)-P\mathcal{H}_{m\times n}\left(k\right)\|_F. \tag{25}$$

给定一个固定的 ε，公式（25）中的 Koopman 算子 $\mathcal{K}_\varepsilon$:$\mathcal{G}\left(\mathbb{R}^{d_\gamma}\times T\right)\to\mathbb{K}$ 可以理解为 $\widehat{\mathcal{K}}_t^{t+\varepsilon}$在不同 t下的时间平均。$\mathcal{K}_\varepsilon$ 的表示仅需进行离线优化，在求解偏微分方程时具有计算优势。

因此，需要对 Koopman 算子进行离线优化，以支持较高的时间分辨率（即较小的 ε）。与此同时，我们需要找到观测函数 $\mathbf{g}(\cdot)$ 的合适设计，使其在合理误差范围内满足遍历性条件。尽

管这两项要求使解析推导或算法方法极具挑战性（例如，参见 Ref.[63] ，其中采用离线低秩分解与在线降阶建模相结合的框架来处理第一项要求），但我们认为，基于神经网络的方法有望在实践中满足这些要求。

D. Koopman 神经算子的神经网络架构

上述想法的关键在于，如何定义有效的神经网络架构，以实现基于 Koopman 算子的偏微分方程求解流程。下面，我们介绍一种可行的架构设计——Koopman 神经算子（KNO），其开源 PyTorch 工具箱已在文献 [65] 中发布。一般而言，KNO 架构由以下部分组成：

(1) **第1部分：观测**。 给定方程 (1–3) 中偏微分方程的输入 $\phi_t = \phi(D \times \{t\})$，我们首先通过编码器将其转换为空间 $\mathcal{G}\left(\mathbb{R}^{d_{\widehat{\gamma}_t}} \times T\right)$ 中的$\mathbf{g}(\widehat{\gamma}_t)$。该编码器是一个带双曲正切（·）激活函数的单一非线性层，用于表示观测函数$\mathbf{g}(\cdot)$。示意图请参见图 1。

(2) **第 2 部分：傅里叶变换**。我们应用傅里叶变换，将 $\mathbf{g}(\widehat{\gamma}_t)$ 映射为$\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t) = \mathcal{F} \circ \mathbf{g}(\widehat{\gamma}_t)$，并在傅里叶空间中参数化网络的后续部分。与文献[9], 类似，$\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$ 通过快速傅里叶变换计算，其中傅里叶级数在 ω 处截断，该值为频率模态的最大数量。一方面，如文献 [9], 所述，这一过程便于计算式（4）所示的迭代更新策略。另一方面，截断傅里叶变换对我们的模型至关重要，因为它将系统划分为两部分。第一部分对应截断后剩余的低频模态，第二部分则由被截断的高频模态组成。尽管没有理论保证，但在实践中，低频模态通常比波动较大的高频模态更稳定，除非系统完全是随机的。换言之，与高频模态相比，低频模态更有可能在误差可接受的情况下满足遍历性条件。因此，通过滤除高频模态，并在**第 3 部分**中仅针对低频模态学习 Koopman 算子，我们可以凭经验降低数据驱动的 Koopman 算子近似的难度。同时，我们在模型中设计了一个额外部分（**第5部分**），用于提取$\mathbf{g}(\widehat{\gamma}_t)$ 的高频模态，以补充高频波动的丢失信息。因此，我们的模型也能够捕捉真实系统中那些遍历性较差、难以使用 Koopman 算子预测的波动部分。详情请参见图 1。

(3) **第 3 部分：Hankel 表示与离线 Koopman 算子**。对于每个 $t \in \varepsilon\mathbb{N}^+$，给定 $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$，我们设定延迟嵌入维度 $m \in \mathbb{N}$，以定义 $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$ 的 Hankel 矩阵 $\mathcal{H}_{m\times n}$（注意，n 等于可访问样本的数量）。为确保 Hankel 矩阵张成的空间能够成功逼近目标 Koopman 算子的不变子空间，我们训练一个 $o \times o$ 线性层来学习 Koopman 算子 $\mathcal{K}_\varepsilon{}^r$ 的 r 次幂：$\mathcal{G}\left(\mathbb{R}^{d_{\widehat{\gamma}}} \times T\right) \to \widehat{\mathbb{K}}$，遵循公式 (24-25)。基于学得的算子 $\mathcal{K}_\varepsilon{}^r$，我们可以将最新可观测量 $\mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n-1)\varepsilon}\right)$ 的未来状态预测为 $\mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) = \left[\mathcal{K}_\varepsilon{}^r \widehat{\mathcal{H}}_{m\times n}(n)\right]^{\intercal}(m)$，其中符号 T 表示矩阵转置。这里我们选择学习 $\mathcal{K}_\varepsilon$ 的 r 次幂，因为灵活的 $r \in \mathbb{N}$ 值让我们有机会调整时间分辨率。具体而言，给定数据集预先确定的 先验时间分辨率 ε，我们可以通过设置预测长度 r 来控制学得的 Koopman 算子的内部时间分辨率。基于这一设定，要预测 $\mathbf{g}_{\mathcal{F}}(\widehat{\gamma}_t)$ 在持续时间为 ε 的时间段内的演化，需要 Koopman 算子在傅里叶空间中迭代 r 次（即每次迭代仅对应一段较短的 ε/r）。换言之，即使采样数据的时间分辨率 ε 可能不够高，我们仍可将 r 设为非 1 值，以提高 Koopman 算子实际作用的时间分辨率。这一过程在实践中很有优势，因为时间依赖的 Koopman 算子在较短时间段内的变化更可忽略，从而降低离线优化的难度。请参见图 1，了解**第 3 部分**的示意图。

(4) **第4部分：傅里叶逆变换**。 对于**第3部分**中的每个预测状态$\mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$，我们通过傅里叶逆变换将其从傅里叶空间变换到 $\mathcal{G}\left(\mathbb{R}^{d_{\widehat{\gamma}}} \times T\right)$，即$\mathbf{g}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) = \mathcal{F}^{-1} \circ \mathbf{g}_{\mathcal{F}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$。请注意，系统中的高频波动已在**第2部分** 中被滤除，无法通过傅里叶逆变换直接恢复（丢失信息的补充由**第5部分**实现）。有关**第4部分**的示例，请参见图 1。

(5) **第5部分：高频信息补充**。根据对特征图进行的傅里叶分析，卷积层可以放大高频分量 [66]。因此，我们在**第1部分**的输出上训练卷积网络 $\mathcal{C}$，以提取其高频信息，记为$\mathbf{g}_{\mathcal{C}}(\widehat{\gamma}_t)$，作为**第2至第4部分**的补充。同时，

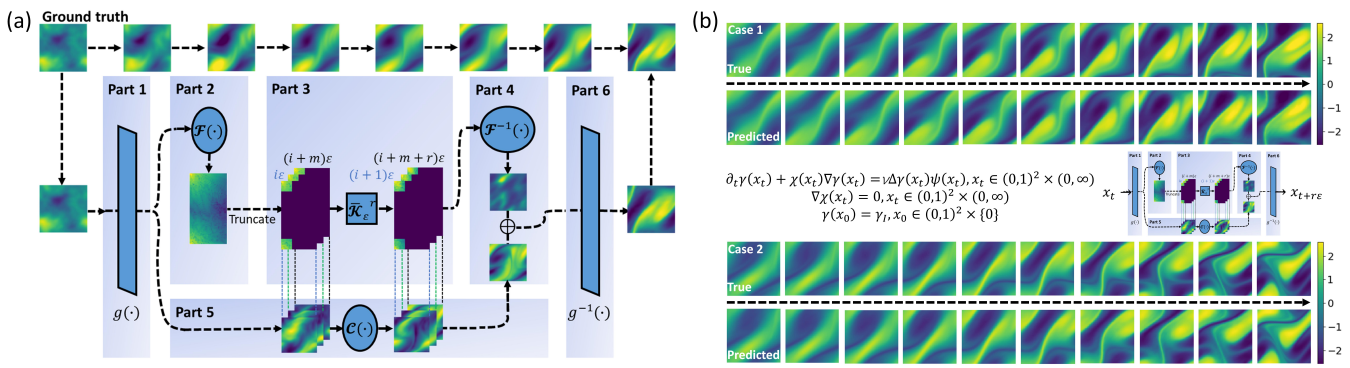

图 1. KNO 神经网络架构的概念示意图。请注意，每个部分的布局都略作调整，以呈现更清晰的版本。

卷积网络还实现了与**第2至第4部分**并行的独立前向预测，即$\left[\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(i+1)\varepsilon}\right), \ldots, \mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(i+m+1)\varepsilon}\right)\right]^{\mathsf{T}} = \mathcal{C}\left[\mathbf{g}\left(\widehat{\gamma}_{i\varepsilon}\right) \ldots, \mathbf{g}\left(\widehat{\gamma}_{(i+m)\varepsilon}\right)\right]^{\mathsf{T}}$，适用于任意 $i = 1, \ldots, n$。在发布的 KNO 工具箱 [65], 中，$\mathcal{C}$ 的一种基础实现是卷积层，这种实现简单实用。如果在复杂偏微分方程中前向预测变得困难，也可以考虑其他有效的卷积网络设计。例如，我们可以采用简单的三段式网络设计来实现 $\mathcal{C}$。$\mathcal{C}$ 的第一段和第三段是用于数据重塑的卷积层，第二段是 $\mathcal{C}$ Inception 模块（即 GoogLeNet 的基本组件） [67], ，其效率已在实践中得到广泛验证。工具箱中也提供了这种设计。图 1 展示了该设计的说明

第**第5部分**的……红黑大战

(6) 第6部分：逆观测。 给定由**第2至第4部分**和**第5部分**独立预测的最新可观测量的两个未来状态，$\mathbf{g}(\widehat{\gamma}_{(m+n)\varepsilon})$ 和$\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$，我们通过$\mathbf{g}_{\mathcal{U}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) = (1-\lambda)\mathbf{g}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right) + \lambda\mathbf{g}_{\mathcal{C}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$将它们统一，其中 $\lambda \in [0,1]$ 控制低频和高频信息的相对权重。我们训练一个非线性解码器（一个采用双曲正切（·）激活函数的单层非线性层），以表示观测函数$\mathbf{g}^{-1}(\cdot) \simeq \mathbf{g}_{\mathcal{U}}^{-1}(\cdot)$的逆，并将 $\mathbf{g}_{\mathcal{U}}\left(\widehat{\gamma}_{(m+n)\varepsilon}\right)$ 转换为 $\widehat{\gamma}_{(m+n)\varepsilon}$，即空间 $\mathbb{R}^{d_{\mathcal{Y}}}$中方程解的目标状态。详情请参见图 1。

基于**第 1–6 部分**，我们提出了一种不同于式 (4) 的新迭代更新策略。对于任意 $t' > t \in \varepsilon\mathbb{N}$，我们有

$$\widehat{\gamma}_{t'} = \left[\mathbf{g}^{-1}\Big(\underbrace{\mathcal{F}^{-1} \circ \overline{\mathcal{K}}_{\varepsilon}{}^{r} \circ \mathcal{F} \circ \mathbf{g}\left(\widehat{\gamma}_{[t-m\varepsilon,t]}\right)}_{\textbf{Parts 1-4}} + \underbrace{\mathcal{C} \circ \mathbf{g}\left(\widehat{\gamma}_{[t-m\varepsilon,t]}\right)}_{\textbf{Part 1 and part 5}}\Big)\right]^{\mathsf{T}}(m), \tag{26}$$

其中，$\widehat{\gamma}_{[t-m\varepsilon,t]}$ 是一个由 $m \in \mathbb{N}$ 定义的向量，表示延迟嵌入的维数。在图 1 中，我们展示了 KNO 的单层架构。与傅里叶神经算子类似， [9], 可通过将**第 2 部分5** x复制 x次级联，构建一个 x 层 KNO 架构。根据式 (26), KNO 的损失函数定义为

$$\mathcal{L} = \alpha\|\widehat{\gamma}_{t'} - \gamma_{t'}\|_F + \beta\sum_{i=0}^{m}\|\mathbf{g}^{-1} \circ \mathbf{g}\left(\widehat{\gamma}_{t-im\varepsilon}\right) - \gamma_{t-im\varepsilon}\|_F, \tag{27}$$

其中，$\alpha, \beta \in (0, \infty$ 分别控制损失函数中预测和重构过程的权重。

将损失函数的预测、重构、低频和高频贡献细分的思想，也可见于此前有关非线性动力系统稀疏辨识的研究[68–70]。

所提出的 KNO 框架与文献 [45] 类似，都旨在通过神经网络对 Koopman 算子进行参数化。KNO 与经典框架不同，因为它是为偏微分方程求解而开发的，并包含多个流程以降低 Koopman 算子近似的难度（例如，对高频和低频信息进行分而治之的处理，以及灵活设置内部时间分辨率）。下文中，我们将在具有代表性的

用于验证其有效性的任务。

III. 实验

A. 实验设计

1. *Data set information*

我们在一维 Bateman–Burgers 方程[71]， 、二维纳维–斯托克斯方程 [72], 、二维 Rayleigh-B´énard 对流（即一种湍流对流）[73], 、二维浅水方程 [74], 、Storm EVent ImagRy 数据集中的水汽红外卫星图像 [75], ，以及欧盟哥白尼海洋环境监测服务收集的西边界流数据 [76, 77]上开展实验。前四个数据集对应湍流分析中的典型问题。最后两个数据集包含由未知偏微分方程支配的真实数据，为探索 KNO 在真实降水预报（例如暴风雨和降雨）以及西边界流分析任务（例如黑潮和墨西哥湾流的动态建模）中的适用性提供了机会。完整的数据描述见附录 B。

2. *Experiment designs and baselines*

我们通过五项实验来验证模型：

(1) 网格无关实验。 正如先前研究 [5, 9, 19–23], 所指出的，神经算子模型应当具有网格无关性，因为它们学习的是整个偏微分方程族的解算子。因此，我们设计了一项实验来验证 KNO 的网格无关特性。

(2) 长期预测实验。 为验证 KNO 的长期预测能力，我们针对八个数据集设计了预测任务，其中包括湍流分析中的五个典型偏微分方程求解问题，以及三个复杂的动态系统建模问题。

(3) 零样本预测实验（离散化粒度）。参照文献 [9],的思路，我们通过在未经训练的离散化粒度上测试 KNO，验证其泛化能力（例如，采用类似超分辨率 [9] 的方式）。

(4) 零样本预测实验（预测区间）。除了验证 KNO 在未训练过的离散化粒度上的泛化能力外，我们还验证了 KNO 在未训练过的预测区间上的泛化能力（例如，零样本时间插值和外推）。

(5) 消融实验。为了展示 KNO 中学习得到的 Koopman 算子的重要性，我们进行了消融实验。

在我们的实验中，高频补偿可以通过单个卷积层 $\mathcal{C}_s$ 实现，也可以通过简单的三分支卷积网络 $\mathcal{C}_t$ 实现。除 KNO 外，我们还实现了以下模型进行比较：傅里叶神经算子（FNO） [9], 、U形神经算子（UNO） [78], 、卷积神经算子（CNO） [79], 、潜在谱模型（LSM）[80], 、U-Net [81], 以及残差神经网络（ResNet） [82]。在这些基线模型中，FNO [9]是经过广泛验证的神经算子模型。UNO [78] 和 CNO [79] 代表了近期提出的神经算子的性能。LSM [80] 是谱方法的神经网络参数化实例。U-Net [81] 和 ResNet [82]代表了深度学习领域时空建模的经典思路。其他常见的深度学习神经网络模型，例如卷积长短期记忆网络（ConvLSTM） [83], 和深度隐式物理模型（DHPM） [84], ，在复杂动态系统 [83] 上的效率较低，因此未纳入本研究。每个模型均使用其默认优化器进行训练（例如，KNO 和 FNO 使用 Adam 优化器训练）。

前四个偏微分方程数据集上的实验在一块配备 32GB 内存的 Nvidia V100 GPU 上进行。最后两个大规模真实数据集上的实验在一块配备 40GB 内存的 Nvidia A100 GPU 上进行。

B. 实验结果

1. *Mesh-independent experiment*

我们的网格无关实验采用在不同离散化条件（即网格的空间分辨率）下生成的一维 Bateman–Burgers 方程数据。最高分辨率的数据按照文献 [9] 中介绍的高斯初始化方法生成。较低分辨率的数据则直接从较高分辨率的数据中降采样得到。

我们选择 FNO 作为比较基线。至于其他在无网格实验中效率低于 FNO 的基线模型，例如图神经算子（GNO） [5] 和多极图神经算子（MGNO）[85] （参见文献 [9] 报告的结果），为方便起见，我们不再讨论。我们实现了多个采用不同超参数的 KNO 版本（例如，算子大小 o、频率模态数 f、Koopman 算子的迭代次数 r ，以及控制低频和高频信息贡献的相对权重 λ 。这些参数的含义请参见第 II D 节）。在所有条件下，这些模型均使用 1000 个样本进行训练，并在 200 个样本上进行 1 秒前向预测（即 $t'-t=1$）

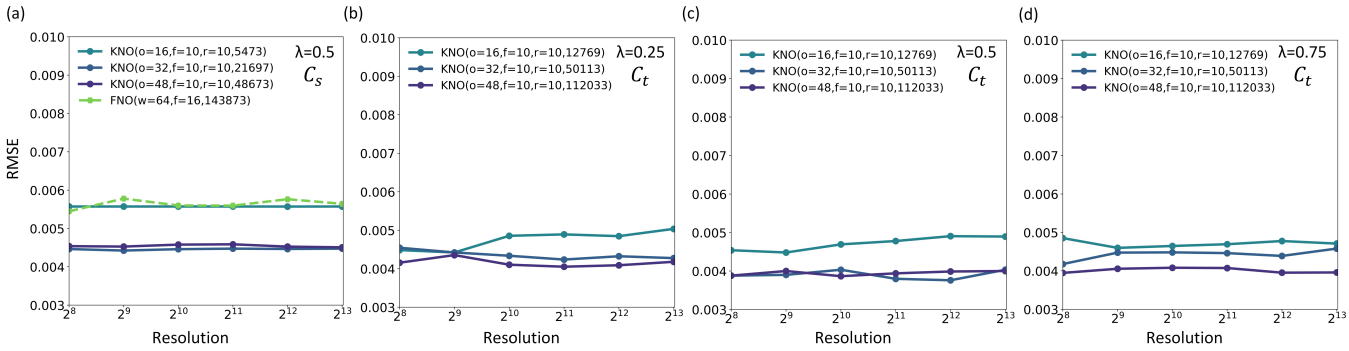

图 2. 网格无关性的实验结果。(a-d) 分别展示了 KNO 中采用不同高频补偿设计和 λ 时的结果。这里的高频补偿通过单个卷积层 $\mathcal{C}_s$ 或简单的三分支卷积网络 $\mathcal{C}_t$ 实现。参数 λ 控制低频和高频信息的贡献。模型性能采用均方根误差（RMSE）衡量。在 (a-d) 的图例中，括号内的数字表示参数设置和模型规模，其中最后一个数字对应模型规模，其余数字均表示参数。FNO 中的概念 w 表示宽度参数（即潜在空间的维度）[9]。请注意，(b-d) 中未重复展示 FNO 的结果，因为调整 λ 与 FNO 无关。因此，可将 (b-d) 中 KNO 模型的性能直接与 (a) 中 FNO 的性能进行比较。

以评估性能。训练期间，批量大小固定为 64。学习率初始化为 0.001，并且每 100 个训练轮次减半。损失函数中预测和重构的权重分别定义为 $\alpha = 5$ 和 $\beta = 0.5$。

如图 2 所示，KNO 在每种分辨率下的预测误差几乎保持不变。与 FNO 相比，KNO 模型在不同条件下的预测误差通常更小。值得注意的是，单元数为一的 KNO 只需约 5×10^3 个参数，就能达到与参数超过 1.4×10^5 个的单元数为一的 FNO 相近的性能。当 KNO 的模型规模增至 2×10^4 或 4×10^4 时，其性能可显著优于 FNO。这些结果表明，KNO 有望在偏微分方程求解中更好地兼顾准确度与效率（即以更少的参数实现更高的准确度）。同时，我们注意到，在高频与低频信息之间取得平衡的 KNO 模型（即 $\lambda = 0.5$）能够在不同分辨率下保持更稳健的性能（即性能波动相对较小）。因此，我们推测，$\lambda = 0.5$ 是处理跨分辨率预测任务的一种最佳经验选择。图 4 中后续的零样本实验也验证了这一推测。

2. *Long-term prediction experiment*

我们的长期预测实验在所有二维数据集上进行。在这些数据集中，二维纳维–斯托克斯方程有三种黏度情形，即 $\nu \in \{10^{-3}, 10^{-4}, 10^{-5}\}$。对于所有数据集，训练好的模型都需要预测未来 10 个时间帧（即 $\nu \in$，其中 $\{10^{-3}, 10^{-4}, 10^{-5}\}$用于二维瑞利–贝纳德对流数据，而所有其他数据集则设为 ）。来（即 $t' = t + z\varepsilon$，其中 $z \in \{1, \ldots, 40\}$ 是为二维瑞利–贝纳德对流数据选取的，而所有其他数据集均设为 $z \in \{1, \ldots, 10\}$ ）。请注意，相邻时间帧之间的时间差 ε 在不同数据集中含义各异。对于二维纳维–斯托克斯方程、二维瑞利–贝纳德对流和二维浅水方程，时间差表示一秒。对于西边界流数据（例如黑潮和墨西哥湾流），时间差对应一天。对于水汽红外卫星图像，时间差对应五分钟。因此，十个或五十个时间帧已足以涵盖相当长的物理时间，使所研究的系统呈现长期且高度非线性的演化。

这里介绍实验实现的详细信息。对于纳维–斯托克斯方程，训练集包含 1000 个样本。$\nu \in \{10^{-3}, 10^{-4}\}$ 的测试集包含 200 个样本，而$\nu = 10^{-5}$的测试集包含 100 个样本。对于瑞利–贝纳德对流，模型分别使用 1600 个样本进行训练，并使用 200 个样本进行测试。对于浅水方程，训练集和测试集分别包含 1600 个和 200 个样本。在所有西边界流数据预测任务中，模型均使用 1600 个样本进行训练，并使用 200 个样本进行测试。对于水汽预测任务，模型使用 3200 个样本进行训练后，使用 400 个样本进行测试。为说明 KNO 并不局限于小规模模型设计，我们展示了一种相对大型 KNO 模型的可行实现（使用 $o = 16$、$f = 12$、$\lambda = 0.5$ 和 $r = 6$ 定义。高频信息补充通过简单的三分支卷积网络$\mathcal{C}_t$实现。模型规模为 19871979）。所有数据集上的对比模型包括 FNO（模型规模为 926517）、UNO（模型规模为 30478033）、CNO（模型规模为

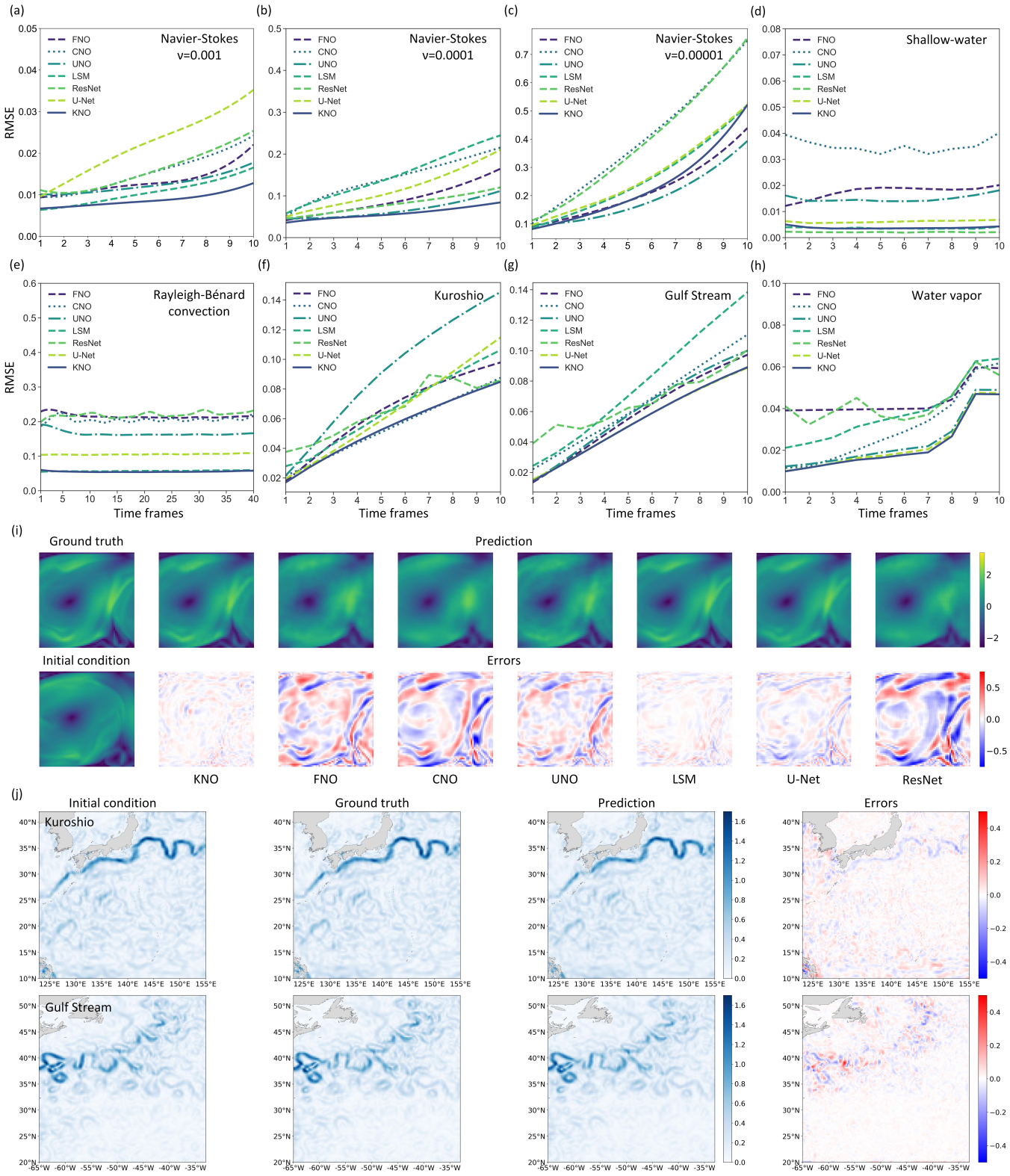

图 3. 长期预测的实验结果。(a-h) 展示了所有模型在不同数据集上的性能，其中每个预测步骤都会生成一个时间帧。(i) 展示了所有模型在瑞利-贝纳德对流上的预测结果（选取初始条件后的第二十六个时间帧作为示例）及其对应误差。请注意，(i) 中的色条为所有模型共用，因此可以直接比较不同模型的结果。(j) 展示了 KNO 在黑潮和墨西哥湾流上的预测结果（初始条件后的第三个时间帧）及其对应误差。

LSM（模型规模为 19188162）、U-Net （模型规模为 24950491）和 ResNet（模型规模为

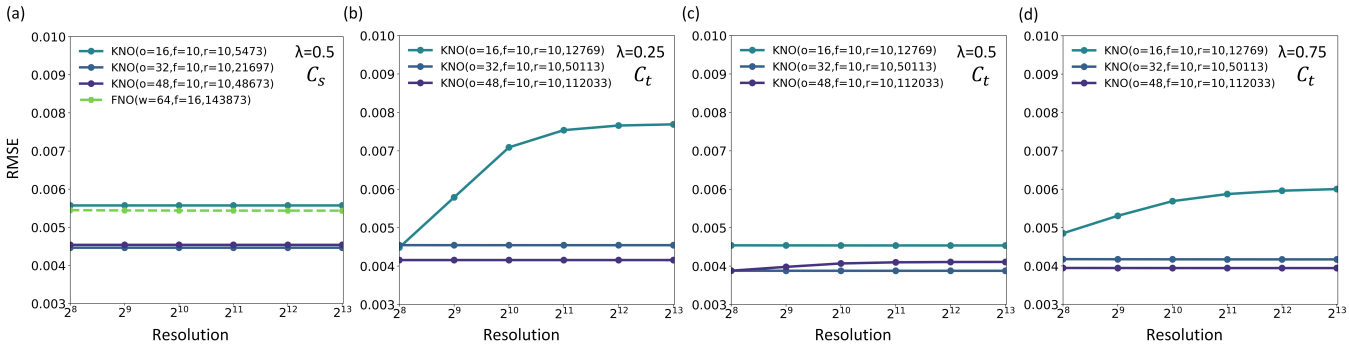

图 4. 一维 Bateman–Burgers 方程上的零样本实验（离散化粒度）结果。(a-d) 分别展示了 KNO 中采用不同高频补偿设计和 λ 时的结果。这里的高频补偿可以通过单个卷积层 $\mathcal{C}_s$ 或简单的三分支卷积网络 $\mathcal{C}_t$ 来实现。参数 λ 控制低频和高频信息的贡献。

20316490)。

如图 3(a-h) 所示，KNO 在所有数据集上均达到了最优或近乎最优的表现。KNO 的预测误差增长率通常低于其他对比模型。这些结果表明，采用 Koopman 算子建模方法有助于刻画偏微分方程或真实系统的长期演化，因为它能提升模型准确度。在图 3(i-j) 中，我们展示了预测结果示例。可以看出，KNO 所实现的较小误差确实确保了其预测结果具有更高质量的二维场可视化。其他模型的误差迅速增长，通常会导致可视化结果模糊且不准确。

3. *Zero-shot experiment on discretization granularity*

正如参考文献 [9]，所指出的，网格无关神经算子模型仅需使用低分辨率数据进行训练，便可预测高分辨率数据（称为零样本超分辨率 [9]）。在本研究中，我们通过实施零样本实验并比较 KNO 与 FNO 来验证这一特性。

模型设置与图 2 所示的网格无关性实验相同。我们使用一维 Bateman–Burgers 方程实施该实验。预测任务的大多数设置与图 2 中的相同。唯一的区别在于，所有模型均在较低分辨率下训练，并在较高分辨率下测试。具体而言，对于一维 Bateman–Burgers 方程，模型在 2^8 个网格上训练，并在 $\{2^9, \ldots, 2^{13}\}$ 个网格上测试。

如图 4 所示，在许多条件下，模型规模更小的 KNO 可以胜过模型规模更大的 FNO。同时，在比较 KNO 在不同λ设置下的性能后，我们发现，$\lambda = 0.5$ 是跨分辨率进行零样本预测的最佳选择，这与我们在图 2 中的观测一致。

4. *Zero-shot prediction experiment on prediction interval*

由于提出 KNO 的目的是对偏微分方程解的复杂动力学进行建模，因此自然会想到，KNO 是否能在预测区间未经训练的零样本预测任务中取得最佳性能。具体而言，可能有两种情形。第一种情形中，模型仅通过彼此不相邻、且被未训练时间帧隔开的时间帧进行监督。模型需要预测这些未受监督时间帧，即被夹在受监督时间帧之间的时间帧。例如，如果受监督时间帧之间隔着两帧未训练时间帧，那么模型在训练期间学习预测第 1、4、7 帧，依此类推；测试期间，模型则需要预测第 2、3、5、6 帧，依此类推。第二种情形中，模型在给定时间区间内接受监督，并学习预测该区间内的时间帧。测试期间，模型需要预测一个长得多的时间区间，其中涵盖多个未受监督时间帧。例如，模型经过训练，可以预测未来 10 帧；测试时则需要预测 40 帧。这两种零样本实验对于时间插值和外推都很有用。

我们使用二维 Rayleigh-B´enard 对流数据开展实验。直接采用图 3 实验中的模型参数设置。在第一类零样本实验中，模型通过间隔 2 或 4 个未训练时间帧的数据进行监督。在第二类零样本实验中，模型经过训练，可预测 10 或 20 个时间帧。测试时，要求模型预测未来 40 个时间帧（即有 30 或 20 个未监督时间帧用于零样本预测）。如图 5(a-d) 所示，KNO 通常能达到

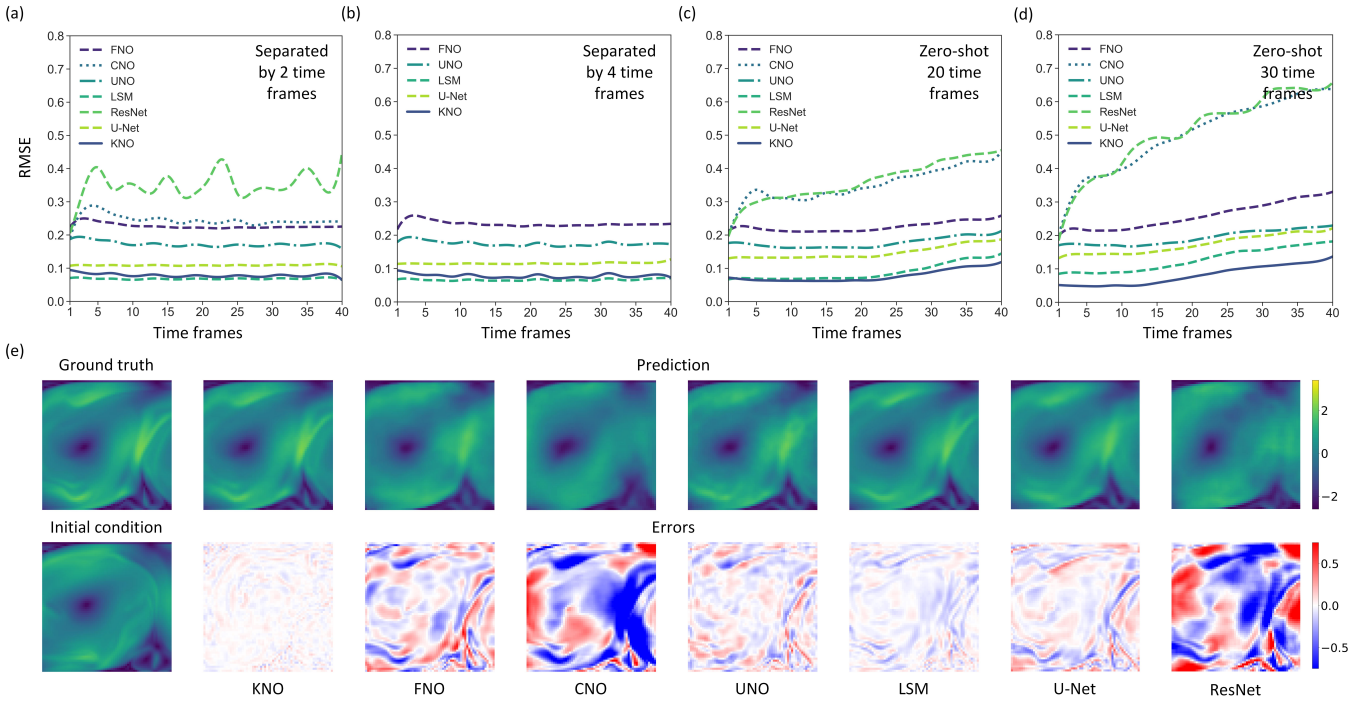

图 5. 二维 Rayleigh-B´énard 对流数据上的零样本实验（预测区间）结果。(a-b) 展示了第一类零样本预测的结果：训练期间，模型分别由间隔 2 个或 4 个未训练时间帧的时间帧进行监督。(b) 中未展示 ResNet 的结果，因为 ResNet 在训练期间未能很好地收敛。(c-d) 展示了第二类零样本预测的结果：测试期间分别有 20 个或 30 个未监督时间帧。请注意，(a-d) 中的曲线采用 SciPy [86] 提供的 B 样条基进行了平滑处理，因为 ResNet 和 CNO 的原始表现存在振荡。(e) 展示了图 (d) 所示零样本实验中所有模型的预测结果（选取初始条件之后的第二十六个时间帧作为示例进行说明）及相应误差。

两类零样本实验中的理想准确度。图 5(e) 展示了所有模型在包含 30 个未监督时间帧的零样本外推实验中的预测结果示例。由于这些示例对应于图 3(i) 所示的同一系统，因此可以直接比较这两种情况，并发现即使没有监督，KNO 的长期预测仍然稳健，而其他模型在零样本情况下的性能则显著下降。这些结果表明，近似 Koopman 算子所捕捉到的演化模式适用于动力系统建模。

5. *Ablation experiment*

为证明学习到的 Koopman 算子是确保 KNO 有效性的关键因素，我们设置了一项简单的消融实验，对比 KNO 框架中包含和不包含重构损失项时的预测性能。在前一种情况下，模型的时间预测主要依赖线性层的权重矩阵，该矩阵的作用相当于 Koopman 算子的 r 次幂。模型中其他可优化部分则会在重构损失的驱动下学习观测函数及其逆函数。在后一种情况下，整个模型只专注于时间预测，而线性层不一定以类似 Koopman 算子的方式发挥作用。

我们使用一维 Bate-man–Burgers 方程的数据进行消融实验。为方便起见，KNO 中的高频补偿通过单个卷积层 $\mathcal{C}_s$ 实现，并配有 $\lambda = 0.5$。如表I所示，采用有效重构过程的 KNO 通常优于仅由预测损失驱动的 KNO 框架。

IV. 结论

总之，我们开发了 Koopman 神经算子（KNO），这是一种基于神经网络、具有网格无关性的偏微分方程求解器。KNO 的基础代码以 C 语言提供，KNO 的官方工具箱见参考文献 [36]。与神经网络偏微分方程求解领域现有的先进方法相比，例如傅里叶神经算子 [9], U形神经算子 [78], 卷积神经算子 [79], 和潜在谱模型[80], , 我们提出的 KNO 展现出更强的能力，能够捕

算子大小 o	模态数 f	迭代次数 r	权重 α	权重 β	均方根误差
8	10	10	5.0	0.5	6.18×10^{-3}
8	10	10	5.0	0	$\mathbf{6.09 \times 10^{-3}}$
8	10	16	5.0	0.5	5.01×10^{-3}
8	10	16	5.0	0	$\mathbf{4.79 \times 10^{-3}}$
8	64	10	5.0	0.5	$\mathbf{4.93 \times 10^{-3}}$
8	64	10	5.0	0	5.17×10^{-3}
16	10	16	5.0	0.5	$\mathbf{3.24 \times 10^{-3}}$
16	10	16	5.0	0	3.32×10^{-3}
32	10	16	5.0	0.5	$\mathbf{2.32 \times 10^{-3}}$
32	10	16	5.0	0	2.46×10^{-3}
32	64	16	5.0	0.5	$\mathbf{2.34 \times 10^{-3}}$
32	64	16	5.0	0	2.40×10^{-3}
128	10	16	5.0	0.5	$\mathbf{1.92 \times 10^{-3}}$
128	10	16	5.0	0	2.08×10^{-3}

表I. 一维 Bateman–Burgers 方程消融实验的结果。

捉偏微分方程或真实动力系统的长期演化。同时，它通常能在不同网格分辨率下保持准确度。这一特性表明，KNO 有望成为构建大规模框架的基本单元，用于求解具有非线性动力学的复杂物理方程，或预测真实动力系统的未来状态。此外，稳健的网格无关性使 KNO 能够先在低分辨率数据上训练，再应用于高分辨率数据。这一特性可能为降低偏微分方程求解的计算成本带来更多可能。

致谢

作者 Y.T 和 P.S 感谢清华大学求是研究院人工智能与通用智能研究计划（2020GQG1017）以及华为创新研究计划（TC20221109044）提供的支持。作者 X.M.H 获得了国家自然科学基金委员会（42125503）和国家重点研发计划（2022YFE0195900, 2021YFC3101600、2020YFA0608000、2020YFA0607900）的资助。作者感谢就读于中国科学技术大学计算机科学与技术学院的吴浩先生提供的技术支持。此外，作者感谢清华大学地球系统科学系的高源先生和周舒怡先生在数据准备期间提供的帮助。

附录 A：Koopman 算子对应的李算子

除了正文中的式 (9)) 所示的线性系统外，我们还可以考虑此类 Koopman 算子的生成算子。由于它是沿向量场 $\gamma(\cdot)$ [87–89] 对 $\mathbf{g}(\cdot)$ 求导的李导数，因此称为李算子

$$\mathcal{L}_t \mathbf{g} = \lim_{t+\varepsilon \to t} \frac{\mathcal{K}_t^{t+\varepsilon} \mathbf{g}\left(\gamma_t\right) - \mathbf{g}\left(\gamma_t\right)}{t+\varepsilon - t}. \tag{A1}$$

该生成算子也定义了 $\mathbf{g}(\gamma_t)$ 的线性系统，因为

$$\partial_t \mathbf{g}\left(\gamma_t\right) = \lim_{\varepsilon \to 0} \frac{\mathbf{g}\left(\gamma_{t+\varepsilon}\right) - \mathbf{g}\left(\gamma_t\right)}{\varepsilon} = \lim_{t+\varepsilon \to t} \frac{\mathcal{K}_t^{t+\varepsilon} \mathbf{g}\left(\gamma_t\right) - \mathbf{g}\left(\gamma_t\right)}{\varepsilon} = \mathcal{L}_t \mathbf{g}\left(\gamma_t\right). \tag{A2}$$

尽管我们的工作主要聚焦于 Koopman 算子，也可以对 Lie 算子在应用中的情况进行建模。

附录 B：所实现数据集的详细信息

在我们的实验中，我们考虑一维 Bateman–Burgers 方程[71]，二维纳维–斯托克斯方程 [72], 二维 Rayleigh-B´énard 对流 [73], 二维浅水方程 [74], Storm EVent ImagRy 数据集中的水汽红外卫星图像 [75]，以及

由欧盟哥白尼海洋环境监测服务机构收集的西边界流数据 [76, 77]。下文将介绍它们的精确定义和预处理流程。

1. Bateman–Burgers 方程。

一维 Bateman–Burgers 方程为

$$\partial_t u\left(x_t\right)+\partial_x\left(\frac{u^2\left(x_t\right)}{2}\right)=\nu\partial_{xx}u\left(x_t\right),\ x_t\in(0,1)\times(0,1], \tag{B1}$$

$$u\left(x_0\right)=u_I,\ x_0\in(0,1)\times\{0\}, \tag{B2}$$

其中， $u(\cdot)$ 表示速度，记作 ω_I 的是周期初始条件，参数 ν 是黏度系数。方程 (B1-B2) 的数据集见参考文献 [9]。该方程的学习目标为 $u(\cdot)$。

2. 纳维–斯托克斯方程。

在数学上，不可压缩的二维纳维–斯托克斯方程的涡量形式定义为

$$\partial_t\omega+\mathbf{u}\cdot\nabla\omega=\nu\nabla^2\omega+\nabla\times\mathbf{f}, \tag{B3}$$

$$\nabla\cdot\mathbf{u}=0, \tag{B4}$$

其中， ω 是涡量标量场，$\mathbf{u}$ 定义速度向量场，$\mathbf{f}$ 表示与时间无关的强迫项。黏度系数设为 $\nu\in\{10^{-3},10^{-4},10^{-5}\}$。与 Bateman–Burgers 方程类似，方程 (B3-B4) 的数据见参考文献 [9]。在实验中，学习目标是涡量场 ω。

3. Rayleigh-B´enard 对流方程。

二维 Rayleigh-B´enard 对流定义为 [73]

$$\partial_t\mathbf{u}+\mathbf{u}\cdot\nabla\mathbf{u}+\frac{1}{\rho\left(\theta\right)}\nabla p=\nu\nabla^2\mathbf{u}+\mathbf{f}, \tag{B5}$$

$$\partial_t\theta+\mathbf{u}\cdot\nabla\theta=\kappa\nabla^2\theta, \tag{B6}$$

$$\nabla\cdot\mathbf{u}=0, \tag{B7}$$

$$\rho\left(\theta\right)=\rho\left(\theta_0\right)\left[1-a\left(\theta-\theta_0\right)\right], \tag{B8}$$

其中， p 是压力，标量 θ 是温度，系数 κ 表示热导率，而 $\rho(\theta)$表示温度为 θ时流体的密度。符号 θ_0 表示初始温度， a 表示热膨胀系数。二维瑞利-贝纳德对流数据由参考文献 [83]采用布辛涅斯克近似方法生成。在我们的实验中，学习目标是速度大小$\mathbf{u}$ （即标量场）。

4. 浅水方程。

二维浅水方程具有以下形式 [74]

$$\partial_t h+\partial_x u+\partial_y v=0, \tag{B9}$$

$$\partial_t hu+\partial_x\left(u^2h+\frac{1}{g_r}h^2\right)=-g_r h\partial_x b, \tag{B10}$$

$$\partial_t hv+\partial_y\left(v^2h+\frac{1}{g_r}h^2\right)=-g_r h\partial_y b, \tag{B11}$$

其中， h 表示水深标量场，符号 b 描述随空间变化的水深地形， g_r 表示重力加速度。符号 u 和 v 分别表示水平方向和垂直方向的速度。二维浅水方程的数据由参考文献 [74]提供。在我们的实验中，学习目标是深度场 h。

5. 水汽数据。

水汽数据的红外卫星图像取自 Storm EVent ImagRy 数据集 [75]，可用于降水分析和预报（例如风暴和降雨）。卫星图像的空间分辨率为 2 千米，相邻两幅图像数据之间的时间区间为 5 分钟。为方便起见，我们已将数据在空间域中从192×192 下采样至 128×128。

6. 西边界流数据。

西边界流数据来自欧盟哥白尼海洋环境监测服务 [76]。该数据集可用于分析西边界流的动力学特性，而西边界流是塑造全球海洋环流和气候变率的重要因素 [77]。具体而言，我们使用每日海表流速数据，其空间分辨率为 0.25×0.25 度。我们选取黑潮海域（10-42°N，123-155°E）和墨西哥湾流海域（20-52°N，33-65°W）的数据，研究黑潮和墨西哥湾流。数据时间区间为 2013/1/1 至 2018/12/31。

附录 C：代码实现

下面展示了基于 PyTorch 实现 KNO 的基本代码。KNO 的官方工具箱位于 https://github.com/Koopman-Laboratory/KoopmanLab（另见参考文献 [36]）。

```python
import torch
import numpy as np
import torch.nn as nn
```

```python
import torch.nn.functional as F

torch.manual_seed(0)

class encoder(nn.Module):
    def __init__(self, t_len, op_size):
        super(encoder, self).__init__()
        self.layer = nn.Linear(t_len, op_size)
    def forward(self, x):
        x = self.layer(x)
        x = torch.tanh(x)
        return x

class decoder(nn.Module):
    def __init__(self, t_len, op_size):
        super(decoder, self).__init__()
        self.layer = nn.Linear(op_size, t_len)
    def forward(self, x):
        x = torch.tanh(x)
        x = self.layer(x)
        return x

class Koopman_Operator1D(nn.Module):
    def __init__(self, op_size, modes_x = 16):
        super(Koopman_Operator1D, self).__init__
    ()
        self.op_size = op_size
        self.scale = (1 / (op_size * op_size))
        self.modes_x = modes_x
       self.koopman_matrix = nn.Parameter(self.


    # Complex multiplication
    def time_marching(self, input, weights):
       # (batch, t, x), (t, t+1, x) -> (batch,t+1, x)

        return torch.einsum("btx,tfx->bfx",



        # Fourier Transform
        x_ft = torch.fft.rfft(x)
        # Koopman Operator Time Marching
       out_ft = torch.zeros(x_ft.shape, dtype=

        out_ft[:, :, :self.modes_x] = self.
    time_marching(x_ft[:, :, :self.modes_x],
    self.koopman_matrix)
        #Inverse Fourier Transform
        x = torch.fft.irfft(out_ft, n=x.size(-1)
    )
        return x

class KNO1d(nn.Module):
    def __init__(self, op_size, modes_x = 16,


        # Layer Structure
       self.enc = encoder(t_len, op_size)


        x_reconstruct = torch.tanh(x_reconstruct
)

        # Predict
        x = self.enc(x) # Encoder


        for i in range(self.decompose):
            x1 = self.koopman_layer(x) # Koopman
     Operator
            x = x + x1



class Koopman_Operator2D(nn.Module):
    def __init__(self, op_size, modes):

()

        self.modes_x = modes

       self.koopman_matrix = nn.Parameter(self.


    # Complex multiplication
    def time_marching(self, input, weights):
       # (batch, t, x,y ), (t, t+1, x,y) -> (batch, t+1, x,y)

        return torch.einsum("btxy,tfxy->bfxy",input,


    def forward(self, x):
        batchsize = x.shape[0]
        # Fourier Transform
        x_ft = torch.fft.rfft2(x)
        # Koopman Operator Time Marching
       out_ft = torch.zeros(x_ft.shape, dtype=

       out_ft[:, :, :self.modes_x, :self.


       out_ft[:, :, -self.modes_x:, :self.



        #Inverse Fourier Transform
       x = torch.fft.irfft2(out_ft, s=(x.size

        return x

class KNO2d(nn.Module):

de



        # Layer Structure
       self.enc = encoder(t_len, op_size)


sel
```

```
            self.w0 = nn.Conv2d(op_size, op_size, 1)
        else:
            self.w0 = Convolutional_Module(op_size) # Define a convolutional structure
    def forward(self, x):
        # Reconstruct
        x_reconstruct = self.enc(x)
        x_reconstruct = torch.tanh(x_reconstruct)
        x_reconstruct = self.dec(x_reconstruct)
        # Predict
        x = self.enc(x) # Encoder
        x = x.permute(0, 3, 1, 2)
        x_w = x
        for i in range(self.decompose):
            x1 = self.koopman_layer(x) # Koopman Operator
            x = x + x1
        x = (1 - self.weight) * x + self.weight * self.w0(x_w)
        x = x.permute(0, 2, 3, 1)
        x = self.dec(x) # Decoder
        return x, x_reconstruct
```

---

[1] L. Debnath 和 L. Debnath, *Nonlinear partial differential equations for scientists and engineers* 2005 (施普林格,)。[2] H. Tanabe, *Functional analytic methods for partial differential equations* 2017 (CRC出版社,)。[3] M. S. Gockenbach, *Partial differential equations: analytical and numerical methods* 122 2005 , 第 卷（工业与应用数学学会, ）。[4] R. M. Mattheij、S. W. Rienstra 和 J. T. T. Boonkkamp, *Partial differential equations: modeling, analysis, computation* 2005 (工业与应用数学学会,)。[5] Z. Li、N. Kovachki、K. Azizzadenesheli、B. Liu、K. Bhat- tacharya、A. Stuart 和 A. Anandkumar，神经算子：用于偏微分方程的图核网络，arXiv 预印本 arXiv:2003.03485（2020）。[6] J. N. Reddy, *Introduction to the finite element method* (麦格劳-希尔教育, 2019)。[7] K. Lipnikov、G. Manzini 和 M. Shashkov，拟态有限差分法，《计算物理学杂志》**257**，1163（2014）。[8] E. Tadmor，非线性偏微分方程数值方法综述，《美国数学学会通报》**49**，507（2012）。[9] Z. Li、N. Kovachki、K. Azizzadenesheli、B. Liu、K. Bhat- tacharya、A. Stuart 和 A. Anandkumar，用于参数化偏微分方程的傅里叶神经算子，arXiv 预印本 arXiv:2010.08895（2020）。[10] M. Raissi、P. Perdikaris 和 G. E. Karniadakis，物理信息神经网络：一种用于求解涉及非线性偏微分方程的正问题和反问题的深度学习框架，《计算物理学杂志》**378**，686（2019）。[11] D. Kochkov、J. A. Smith、A. Alieva、Q. Wang、M. P. Brenner 和 S. Hoyer，机器学习加速的计算流体力学，《美国国家科学院院刊》**118**，e2101784118（2021）。[12] X. Guo、W. Li 和 F. Iorio，用于定常流近似的卷积神经网络，载于 *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* 2016 481 490 ( ) 第 – 页。[13] Y. Zhu 和 N. Zabaras，用于代理建模和不确定性量化的贝叶斯深度卷积编码器-解码器网络，《计算物理学杂志》**366**，415（2018）。[14] S. Bhatnagar、Y. Afshar、S. Pan、K. Duraisamy 和 S. Kaushik，使用卷积神经网络预测气动流场，《计算力学》**64**，525（2019）。[15] B. Yu *et al.*，深度 Ritz 方法：一种基于深度学习的数值算法，用于求解变分问题lems，《数学与统计学通讯》**6**，1 2018( )。[16] L. Bar 和 N. Sochen，基于偏微分方程的正问题与逆问题的无监督深度学习算法，arXiv 预印本 arXiv:1904.05417 (2019)。[17] S. Pan 和 K. Duraisamy，具有稳定性保证的非线性动力学线性嵌入的物理信息概率学习，《SIAM 应用动力系统杂志》**19**，480 (2020)。[18] M. Lienen 和 S. G¨unnemann，利用有限元网络从稀疏观测中学习物理系统的动力学，载于 *International Conference on Learning Representations* 2022 ( )。

[19] L. Lu、P. Jin 和 G. E. Karniadakis，DeepONet：基于算子通用逼近定理学习非线性算子以识别微分方程，arXiv 预印本 arXiv:1910.03193 (2019)。[20] K. Bhattacharya、B. Hosseini、N. B. Kovachki 和 A. M. Stuart，参数化偏微分方程的模型降阶与神经网络，arXiv 预印本 arXiv:2005.03180 (2020)。[21] N. H. Nelsen 和 A. M. Stuart，Banach 空间之间输入输出映射的随机特征模型，《SIAM科学计算期刊》**43**，A3212 (2021)。[22] N. Kovachki、S. Lanthaler 和 S. Mishra，关于傅里叶神经算子的通用逼近与误差界，《机器学习研究期刊》**22**，文章编号 (2021)。[23] Z. Li、D. Z. Huang、B. Liu 和 A. Anandkumar，利用学习得到的形变处理一般几何域上的偏微分方程的傅里叶神经算子，arXiv 预印本 arXiv:2207.05209 (2022)。[24] J. Guibas、M. Mardani、Z. Li、A. Tao、A. Anandkumar 和 B. Catanzaro，通过自适应傅里叶神经算子实现高效的 Transformer 令牌混合，载于 *International Conference on Learning Representations* 2021 ( )。

[25] L. Perko, *Differential equations and dynamical systems*, 第 7 卷（施普林格科学与商业媒体，2013）。[26] P. A. Fishwick, *Handbook of dynamic system modeling*（CRC 出版社，2007）。[27] P. C. Bressloff，连续神经场的时空动力学，《物理学杂志A：数学与理论》**45**，033001 (2011)。[28] J. E. Herrera-Estrada、Y. Satoh 和 J. Sheffield，全球干旱的时空动力学，《地球物理研究快报》**44**，2254 (2017)。[29] W. Wu、C. Meneveau 和 R. Mittal，湍流分离泡的时空动力学，《流体力学杂志》**883**，A45 (2020)。

[30] E. Sch¨oll, *Nonlinear spatio-temporal dynamics and chaos in semiconductors*，第 10 卷（剑桥大学出版社，2001）。[31] S. L. Brunton、M. Budisic、E. Kaiser 和 J. N. Kutz，动力系统的现代 Koopman 理论，SIAM评论**64**，229 (2022)。[32] J. Pathak、S. Subramanian、P. Harrington、S. Raja、A. Chattopadhyay、M. Mardani、T. Kurth、D. Hall、Z. Li、K. Azizzadenesheli、 *et al.*，FourCastNet：一种采用自适应傅里叶神经算子的全球数据驱动高分辨率天气模型，arXiv 预印本 arXiv:2202.11214 (2022)。[33] D. Cao、Y. Wang、J. Duan、C. Zhang、X. Zhu、C. Huang、Y. Tong、B. Xu、J. Bai、J. Tong、 *et al.*，用于多变量时间序列预测的谱时序图神经网络，《神经信息处理系统进展》**33**，17766 (2020)。[34] F. Aminian、E. D. Suarez、M. Aminian 和 D. T. Walz，利用神经网络预测经济数据，《计算经济学》**28**，71 (2006)。[35] L. Xu、N. Chen、Z. Chen、C. Zhang 和 H. Yu，地球系统科学中的时空预测：方法、不确定性、可预测性与未来方向，《地球科学评论》**222**，103828 (2021)。[36] W. Xiong、M. Ma、X. Huang、Z. Zhang、P. Sun 和 Y. Tian，KoopmanLab：一个基于 PyTorch 的 Koopman 神经算子库 (2023)，开源代码见 https://github.com/Koopman-Laboratory/KoopmanLab。[37] C. W. Rowley、I. Mezi´c、S. Bagheri、P. Schlatter 和 D. S. Henningson，非线性流动的谱分析，《流体力学杂志》**641**，115 (2009)。[38] I. Abraham 和 T. D. Murphey，利用 Koopman 算子主动学习动力学以实现数据驱动控制，IEEE机器人学汇刊**35**，1071 (2019)。[39] R. Taylor、J. N. Kutz、K. Morgan 和 B. A. Nelson，用于等离子体诊断与验证的动态模态分解，《科学仪器评论》**89**，053501 (2018)。[40] B. W. Brunton、L. A. Johnson、J. G. Ojemann 和 J. N. Kutz，利用动态模态分解提取大规模神经记录中的时空相干模式，《神经科学方法杂志》**258**，1 (2016)。[41] N. Takeishi、Y. Kawahara 和 T. Yairi，学习用于动态模态分解的 Koopman 不变子空间，《神经信息处理系统进展》**30** (2017)。[42] O. Azencot、N. B. Erichson、V. Lin 和 M. Mahoney，利用一致性Koopman自编码器预测序列数据，载于 *International Conference on Machine Learning* 2020 475 485 (PMLR,) pp. – 。[43] S. E. Otto 和 C. W. Rowley，用于学习动力学的线性递归自编码器网络，《SIAM 应用动力系统杂志》**18**，558 (2019)。[44] D. J. Alford-Lago、C. W. Curtis、A. T. Ihler 和 O. Issan，深度学习增强的动态模态分解，《混沌：非线性科学跨学科期刊》**32**，033116 (2022)。[45] B. Lusch、J. N. Kutz 和 S. L. Brunton，用于非线性动力学通用线性嵌入的深度学习，《自然·通讯》**9**，1 (2018)。[46] S. Macesic、N. Crnjaric-Zic 和 I. Mezic，非自治系统的 Koopman 算子族谱，《SIAM 应用动力系统杂志》**17**，2478（2018）。[47]A. Lasota 和 M. C. Mackey， *Probabilistic properties of deterministic systems*（剑桥大学出版社，1985）。[48] P. Gaspard、G. Nicolis、A. Provata 和 S. Tasaki，叉形分岔的谱特征：Liouville 方程方法，《物理评论E》51，74（1995）。[49] P. Gaspard，《混沌、散射与统计力学》，《混沌》（2005）。[50] A. Lasota 和 M. C. Mackey， *Chaos fractals and noise:* ，，， *stochastic aspects of dynamics* 97 卷（施普林格科学与商业媒体，1998）。[51] P. D. Lax，非线性演化方程的积分与孤立波，《纯粹与应用数学通讯》21，467（1968）。[52] P. D. Lax， *Linear algebra and its applications*，第 78 卷（约翰·威立父子公司，2007）。[53] J. P. Parker 和 J. Page，孤立前沿与孤子的 Koopman分析，《SIAM 应用动力系统杂志》19，2803（2020）。[54] H. Nakao 和 I. Mezi´c，偏微分方程的 Koopman 算子谱分析，《混沌：非线性科学跨学科期刊》30，113131（2020）。[55] C. Gin、B. Lusch、S. L. Brunton 和 J. N. Kutz，用于线性化偏微分方程的全局坐标变换的深度学习模型，《欧洲应用数学杂志》32，515（2021）。[56] J. Page 和 R. R. Kerswell，Burgers方程的 Koopman分析，《物理评论：流体》3，071901（2018）。[57] H. Arbabi 和 I. Mezic，遍历理论、动态模态分解与 Koopman 算子谱性质的计算，《SIAM 应用动力系统杂志》16，2096（2017）。[58] N. ˇCrnjari´c-ˇZic、S. Ma´ceˇsi´c 和 I. Mezi´c，随机动力系统的 Koopman 算子谱，《非线性科学杂志》30，2007（2020）。[59] S. L. Brunton、B. W. Brunton、J. L. Proctor、E. Kaiser 和 J. N. Kutz，作为间歇受迫线性系统的混沌，《自然·通讯》8，1（2017）。[60] Y. Saad， *Numerical methods for large eigenvalue problems: revised edition* 2011 （工业与应用数学学会，）。[61] M. Korda 和 I. Mezi´c，扩展动态模态分解向 Koopman 算子收敛的问题，《非线性科学杂志》28，687（2018）。[62] M. Li 和 L. Jiang，多尺度介质中非自治动力系统 Koopman 算子的降阶建模，arXiv 预印本 arXiv:2204.13180（2022）。[63] M. Li 和 L. Jiang，多尺度介质中非自治动力系统的数据驱动降阶建模，《计算物理学杂志》474，111799（2023）。[64] I. P. Cornfeld、S. V. Fomin 和 Y. G. Sinai， *Ergodic theory* 245 2012 ，卷（施普林格科学与商业媒体，）。[65] W. Xiong、M. Ma、X. Huang、Z. Zhang、P. Sun 和 Y. Tian，KoopmanLab：用于求解偏微分方程的 Koopman 神经算子系列 PyTorch 模块，arXiv 预印本 arXiv:2301.01104（2023）。[66] N. Park 和 S. Kim，视觉Transformer如何工作？，arXiv 预印本 arXiv:2202.06709（2022）。[67] C. Szegedy、W. Liu、Y. Jia、P. Sermanet、S. Reed、D. Anguelov、D. Erhan、V. Vanhoucke 和 A. Rabi-novich，更深入地理解卷积，载于 *Proceedings of the IEEE conference on computer vision and pattern recognition* 2015 1 9 （），第 – 页。

[68] K. Champion、B. Lusch、J. N. Kutz 和 S. L. Brunton，数据驱动的坐标与支配方程发现，美国国家科学院院刊 **116**，22445（2019）。[69] K. P. Champion、S. L. Brunton 和 J. N. Kutz，非线性多尺度系统的发现：采样策略与嵌入，《SIAM 应用动力系统杂志》**18**，312（2019）。[70] U. Fasel、J. N. Kutz、B. W. Brunton 和 S. L. Brunton，集成 SINDy：在低数据、高噪声条件下稳健地发现稀疏模型，并结合主动学习与控制，英国皇家学会会刊 A 辑 **478**，20210904（2022）。[71] E. R. Benton 和 G. W. Platzman，一维 Burgers 方程解一览表，《应用数学季刊》**30**，195（1972）。[72] C. Wang，Navier–Stokes 稳态方程的精确解，《流体力学年评》**23**，159（1991）。[73] E. Bodenschatz、W. Pesch 和 G. Ahlers，Rayleigh–B´énard 对流的最新进展，《流体力学年评》**32**，709（2000）。[74] M. Takamoto、T. Praditia、R. Leiteritz、D. MacKinlay、F. Alesiani、D. Pfl¨uger 和 M. Niepert，PDEBench：面向科学机器学习的大型基准测试，《神经信息处理系统进展》**35**，1596（2022）。[75] M. Veillette、S. Samsi 和 C. Mattioli，SEVIR：用于雷达与卫星气象学深度学习应用的风暴事件图像数据集，《神经信息处理系统进展》**33**，22009（2020）。[76] E.U. 哥白尼海洋环境监测服务，哥白尼海洋环境监测服务收集的每日卫星全球海平面数据（sealevel glo phy l4 observations 008 047）（2021）。[77] D. Hu、L. Wu、W. Cai、A. S. Gupta、A. Ganachaud、B. Qiu、A. L. Gordon、X. Lin、Z. Chen、S. Hu、 *et al.*，太平洋西边界流及其在气候中的作用，《自然》**522**，299（2015）。[78] M. A. Rahman、Z. E. Ross 和 K. Azizzadenesheli，U-NO：U形神经算子，arXiv 预印本 arXiv:2204.11127（2022）。[79] B. Raonic、R. Molinaro、T. De Ryck、T. Rohner、F. Bartolucci、R. Alaifari、S. Mishra 和 E. de B´ezenac，卷积——

卷积神经算子可实现对偏微分方程的稳健、准确学习，《神经信息处理系统进展》**36** (2024年)。[80] H. Wu、T. Hu、H. Luo、J. Wang 和 M. Long，利用潜在谱模型求解高维偏微分方程，载于 *Proceedings of the 40th International Conference on Machine Learning* 2023 37417 37438 ( ) 第 – 页。[81] O. Ronneberger、P. Fischer 和 T. Brox，U-Net：用于生物医学图像分割的卷积网络，载于 *International Conference on Medical image computing and computer-assisted intervention* 2015 (施普林格，)第 234–241 页。[82] K. He、X. Zhang、S. Ren 和 J. Sun，用于图像识别的深度残差学习，载于 *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016)第 770–778 页。[83] R. Wang、K. Kashinath、M. Mustafa、A. Albert 和 R. Yu，面向湍流预测的物理信息驱动深度学习，载于 *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* 2020 1457 1466 ( )第 – 页。[84] M. Raissi，深度隐式物理模型：非线性偏微分方程的深度学习，《机器学习研究杂志》**19**，932（2018）。[85] Z. Li、N. Kovachki、K. Azizzadenesheli、B. Liu、A. Stuart、K. Bhattacharya 和 A. Anandkumar，用于参数化偏微分方程的多极图神经算子，《神经信息处理系统进展》**33**，6755（2020）。[86] P. Virtanen、R. Gommers、T. E. Oliphant、M. Haberland、T. Reddy、D. Cournapeau、E. Burovski、P. Peterson、W. Weckesser、J. Bright、 *et al.*，SciPy 1.0：用于科学计算的基础算法，《自然方法》**17**，261（2020）。[87] B. O. Koopman，希尔伯特空间中的哈密顿系统与变换，《美国国家科学院院刊》**17**，315（1931）。[88] R. Abraham、J. E. Marsden 和 T. Ratiu， *Manifolds, tensor analysis and applications* 75 ，，第卷（施普林格科学与商业媒体，2012）。[89] C. Chicone 和 Y. Latushkin， *Evolution semigroups in dynamical systems and differential equations*，70（美国数学会，1999）。