GENERATIVE AI FOR SCIENCE

A Hands-On Guide for Students and Researchers

PROF. J. PAUL LIU

Generative AI for Science

A Hands-On Guide for Students and Researchers

J. Paul Liu

This book is available at https://leanpub.com/generativeaiforscience

This version was published on 2026-03-19

Contents

From the Author

A Journey from Classroom to Lab — and Now to You

This book is the result of years of teaching, research, and hands-on collaboration at the intersection of generative AI and scientific discovery. Everything here has been shaped by real use: tested in graduate courses, refined in workshops, improved through research projects, and validated by hundreds of scientists applying these tools to their own work.

Origins

The material began in my graduate course **Generative AI for Science**, taught at the Data Science and AI Academy, where students from biology, chemistry, physics, materials science, and computational fields explored a fundamental question:

How can AI not only analyze data, but generate new scientific knowledge?

Lecture notes grew into full tutorials as we worked through real challenges: predicting protein stability, designing new materials, accelerating climate simulations, and mining insights from vast scientific corpora. Student questions sharpened the explanations; their projects highlighted what worked; their struggles revealed where clarity was needed.

Beyond the classroom, the methods in this book have been strengthened through:

- Bioinformatics Research Center workshops
- Cross-campus **AI for Research** training programs
- Research Triangle AI Society–LLM intensive bootcamps
- Active collaborations in oceanography, materials science, protein engineering, and literature mining

What Makes This Book Different

This book is *not* a traditional AI textbook. Rather than focusing on heavy theory, it provides a hands-on, practical guide to using generative AI in real scientific workflows through educational case studies. It is ideal for college students, researchers, and data scientists who want to learn by doing.

- **Theory meets practice:** Every concept is paired with ready-to-run code.

- **Interactive learning:** All example codes are provided as Google Colab notebooks—no installation required.
- **Real scientific problems:** Examples come from authentic research.
- **Accessible yet rigorous:** Suitable for both domain scientists exploring AI and ML experts entering scientific applications.

How to Use This Book

As a course text:
Follow the chapters sequentially for a structured introduction.

As a reference:
Jump directly to the sections relevant to your research domain, or follow the further reading references to explore topics in greater detail.

As a hands-on guide:
Open the Colab notebooks or Powerpoint slides alongside each chapter, run the code, and modify it as you learn.

As a research launchpad:
Use the provided implementations as starting points for your own projects.

What You Will Learn

By the end of this book, you will:

- Understand key AI architectures such as Transformers, Diffusion Models, VAEs, and GNNs
- Represent scientific data types effectively
- Apply generative models to problems in climate science, drug discovery, genomics, materials science, and more
- Follow best practices around ethics, reproducibility, and deployment
- Stay current with emerging methods and future directions

Most importantly, you will develop the intuition to know *when* and *how* to apply AI to scientific research.

A Note on Collaboration

AI-accelerated science thrives at the intersection of domain expertise and computational skill. Biologists will gain enough ML understanding to explore confidently, while ML practitioners will gain enough scientific context to ask meaningful questions and validate results. This

book is itself a perfect example of how I collaborate with AI on writing, coding, editing, and proofreading.

Collaboration is the catalyst. This book aims to provide the shared language needed to bridge fields and perspectives.

Let's Begin

Generative AI does not replace the scientific method—it enhances it. It expands the space of hypotheses we can explore, sharpens experimental design, and reveals patterns hidden in complexity. But the foundations remain unchanged: rigor, honesty, and careful interpretation.

Combine human creativity with machine assistance, and new discoveries become possible.

Now, let us explore them together.

Dr. J. Paul Liu December 2025

* * *

Downloading all codes or slides

Access all Colab notebooks and slides: https://github.com/jpliu168/Generative_AI_For_Science

Prerequisites

- Basic Python programming (functions, loops, data structures)
- Undergraduate-level statistics (distributions, hypothesis testing)
- Familiarity with scientific computing (NumPy, Matplotlib helpful but not required)
- No prior deep learning experience necessary

Technical requirements: A web browser and curiosity. Everything else runs in the cloud.

AI Use Disclaimer

This work was developed with the assistance of modern AI language models, including **ChatGPT**, **Claude**, and **Gemini**, which supported:

1. Improving clarity of writing, including corrections to English grammar and syntax
2. Text organization and formatting, including Markdown structure, embedded Python code blocks, output results, and visual styling
3. Code generation, formatting, debugging, and proofreading

Writing Workflow

Here is the workflow I followed for most chapters:

1. **Research and preparation:** Based on my previous class notes, sample codes, and PowerPoint slides, I first conducted extensive Google searches and read through hundreds of links, papers, and reports.
2. **Initial AI consultation:** I posed my questions, ideas, and plans to AI assistants. With their suggestions, I refined my questions and ideas, returned to Google searches and reference readings, and then asked AI for help with the book proposal.
3. **Drafting:** For each chapter, I sent AI the topic, questions, my references, and sample codes to generate an initial draft. I then manually edited and updated the content by combining my own data, new references, and updated codes.
4. **Code development:** I spent the majority of my time coding, debugging, customizing, and finalizing each case study to ensure it could run within typical student computational resources for educational demonstration purposes.
5. **Validation:** I shared some chapters with students and colleagues for their input. Based on their feedback, I revised and updated the content further.
6. **Final polish:** AI assisted in polishing all my writing and reformatting each section to fit the Leanpub Markua format.

Many times, to meet this book's educational demo purpose, I had to compare, adjust, or combine results from different LLMs based on the quality of their responses. I retain full responsibility for all conceptual content, workflows, research interpretations, citations, discussions, conclusions, and final decisions.

Through this intensive collaborative writing and coding process, I learned a great deal and cannot wait to share all of this in my next class offering of *Generative AI for Science*.

Chapter 1 — Generative AI: A New Frontier for Scientific Discovery

The New Frontier of Scientific Discovery

We stand at an extraordinary moment in the history of science. For centuries, the pace of discovery has been limited by human capacity—our ability to read literature, design experiments, analyze data, and synthesize knowledge. But a fundamental shift is underway. **Generative Artificial Intelligence (AI)** is not merely automating workflows; it is expanding what is scientifically possible [1].

Just as the microscope, the computer, and genome sequencing each transformed how we perceive nature, Generative AI now transforms our ability to *understand* and *create* scientific knowledge. AI systems design novel antibiotics by exploring chemical spaces that would take humans millennia to search [2, 3]. They predict protein structures with atomic precision—work recognized with the 2024 Nobel Prize in Chemistry [4, 5]. AI co-scientists generate hypotheses from vast textual corpora in days that previously required years of iterative research [6, 7], identify patterns in climate and Earth system data that are difficult to capture with traditional methods [8–10], and propose entirely new materials and reaction pathways [3, 11].

This is not science fiction—this is the reality of Generative AI today [1].

* * *

The AI Revolution in Scientific Discovery

From Analysis to Generation

Scientific computing has evolved through clear stages:

Era	Core Capability	Tools	Outcome
1960s–1980s	Digitization	Databases	Data preservation
1990s–2010s	Statistical & ML methods	Regression, SVMs, random forests	Pattern recognition

Era	Core Capability	Tools	Outcome
2020s–	Generative AI	Transformers, diffusion models	Creation of new data & hypotheses

Traditional scientific discovery has often followed a cycle of *observe* → *hypothesize* → *experiment* → *analyze* → *repeat* [12]. Generative AI compresses and accelerates this loop [1, 7]:

- **Literature review:** Synthesizes insights from millions of papers [6, 7].
- **Hypothesis generation:** Identifies non-obvious links across fields, with AI co-scientists reducing hypothesis generation from weeks to days [1, 7].
- **Experimental design:** Proposes thousands of candidate molecules or materials for testing [2, 3, 11].
- **Data analysis:** Finds structure in high-dimensional scientific datasets [8–10, 12].

The Acceleration Effect

The effect is both quantitative and qualitative [1]:

- **Speed:** AI approaches can shorten drug discovery timelines from ~10–15 years to ~3–5 in some workflows, with AI-designed drugs showing 80–90% success rates in Phase I trials compared to traditional 40–65% [2, 13].
- **Scale:** Virtual screening has been used to evaluate extensive compound libraries in silico for drug discovery, supporting efficient prioritization of candidates[3].
- **Efficiency:** Weather forecasts that once required hours on supercomputers can be generated in minutes with AI models like GenCast [9, 14].

More profoundly, AI enables scientists to explore vast combinatorial spaces and ask *what if*? at scales that would be infeasible with manual or purely equation-based approaches [1, 12].

* * *

What Makes Generative AI Different from Traditional ML?

Traditional Machine Learning — Pattern Recognition

Traditional machine learning (ML) excels at **discriminative** tasks such as classification, regression, and clustering—learning mappings from inputs to outputs and identifying patterns in existing data [12].

Generative AI — Creating the New

Generative models go further: they learn the **distribution** underlying the data and can sample from it to create new, realistic instances [15, 16]. This enables capabilities such as:

Capability	Example	Outcome
Generation	Design a new molecule with desired properties	Novel compounds [2, 3]
Completion	Fill missing regions in a protein structure or sequence	Plausible biological variants [4, 5]
Translation	Text description → molecule, code, or figure	Cross-modal synthesis [1, 6]
Synthesis	Combine text, code, and data	New simulations, analyses, or visualizations [6]

* * *

Core Technologies Powering Generative AI

1. Transformers and Large Language Models (LLMs)

Transformers introduced an attention-based architecture that models long-range dependencies and contextual relationships in sequences [15]. Large Language Models (LLMs) built on this architecture can:

- Summarize and synthesize scientific literature
- Generate runnable analysis code and scripts
- Act as domain-aware assistants that interact with text, equations, and data [1, 6]
- Serve as AI co-scientists that generate novel hypotheses through multi-agent debate and evolution [7]

2. Diffusion Models

Diffusion models learn to reverse a gradual noising process to generate high-fidelity samples [16–18]. They are increasingly used for:

- Molecular and materials design [3, 11]
- Protein and structural biology tasks, including AlphaFold 3's prediction of biomolecular interactions [4, 5]
- Probabilistic weather forecasting with models like GenCast [9, 14]

3. Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs)

Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) provide earlier but still important frameworks for generative modeling [19, 20]. They support:

- Learning low-dimensional latent representations of complex systems
- Creating synthetic datasets to augment limited experimental data
- Constructing surrogate models for expensive simulations

* * *

The Pre-Training Revolution

Modern generative AI thrives on **pre-training**: models are first trained on large, generic corpora (text, code, images, or scientific data) and then **fine-tuned** on specific tasks or domains [6]. This strategy is particularly valuable in science, where high-quality labeled data are often scarce or costly to obtain.

A language model pre-trained on broad text can be adapted to domains such as biomedicine, materials science, or geosciences. Protein models pre-trained on millions of sequences can specialize in a narrow protein family with comparatively small task-specific datasets [4, 5].

* * *

Generative AI Across Scientific Disciplines

Physical and Chemical Sciences

In the physical and chemical sciences, generative AI is already reshaping how we search, design, and reason about complex systems:

- **Materials Design:** Exploring enormous chemical spaces to propose new superconductors, catalysts, or battery materials with targeted properties [3, 11].
- **Reaction and Synthesis Planning:** Suggesting reaction products and retrosynthetic routes, accelerating the path from concept to synthesis [3].
- **Mathematical Reasoning:** AlphaProof and AlphaGeometry 2 achieved silver-medal level performance at the 2024 International Mathematical Olympiad, and an advanced Gemini model achieved gold-medal standard in 2025 [21, 22].

Chemistry and Drug Discovery

In chemistry and drug discovery, generative models help to:

- Propose novel antibiotics and therapeutics with desired efficacy and safety profiles [2].
- Optimize molecules for multiple objectives (binding, solubility, toxicity, synthesizability) [3, 11].
- Accelerate clinical progress: AI-assisted drug candidates achieved Phase I success rates of nearly 90%, compared with industry averages of 40–65% [13].

Life and Environmental Sciences

- **Protein Engineering:** AlphaFold 3 predicts the structure and interactions of proteins, DNA, RNA, ligands, and ions with unprecedented accuracy, achieving at least 50% improvement over existing methods [5].
- **Genomics and Systems Biology:** Modeling sequence–function relationships and generating candidate variants for experimental follow-up [1].
- **Medical AI:** Supporting risk prediction, trial design, and treatment-effect modeling when combined with causal and statistical frameworks [1, 12].

Climate and Earth System Science

- **Weather Forecasting:** GenCast, a diffusion-based probabilistic model, outperforms the world's leading operational forecast (ECMWF ENS) in 97% of scenarios, delivering 15-day forecasts in 8 minutes [9, 14].
- **Earth System Foundations:** Large foundation models that integrate atmosphere, ocean, land, and cryosphere processes, including Google's Aurora and WeatherNext models [10].
- **Surrogate Modeling:** Physics-informed and data-driven surrogates that emulate expensive numerical models at a fraction of the computational cost [8–10, 23].

* * *

Mathematical Foundations and Methods

Generative AI is also driving progress in the mathematics and methodology of **scientific machine learning**:

- **Data-Driven Dynamical Systems:** Learning governing dynamics and control laws directly from measurements or simulations [12].
- **Physics-Informed Neural Networks (PINNs):** Incorporating partial differential equations and other physical constraints directly into neural network training [24, 25].
- **Uncertainty Quantification:** Developing methods to quantify confidence and propagate uncertainty through AI-based scientific pipelines [25].
- **Formal Mathematical Reasoning:** AlphaProof combines reinforcement learning with formal theorem proving in Lean to generate verifiable mathematical proofs [21].

* * *

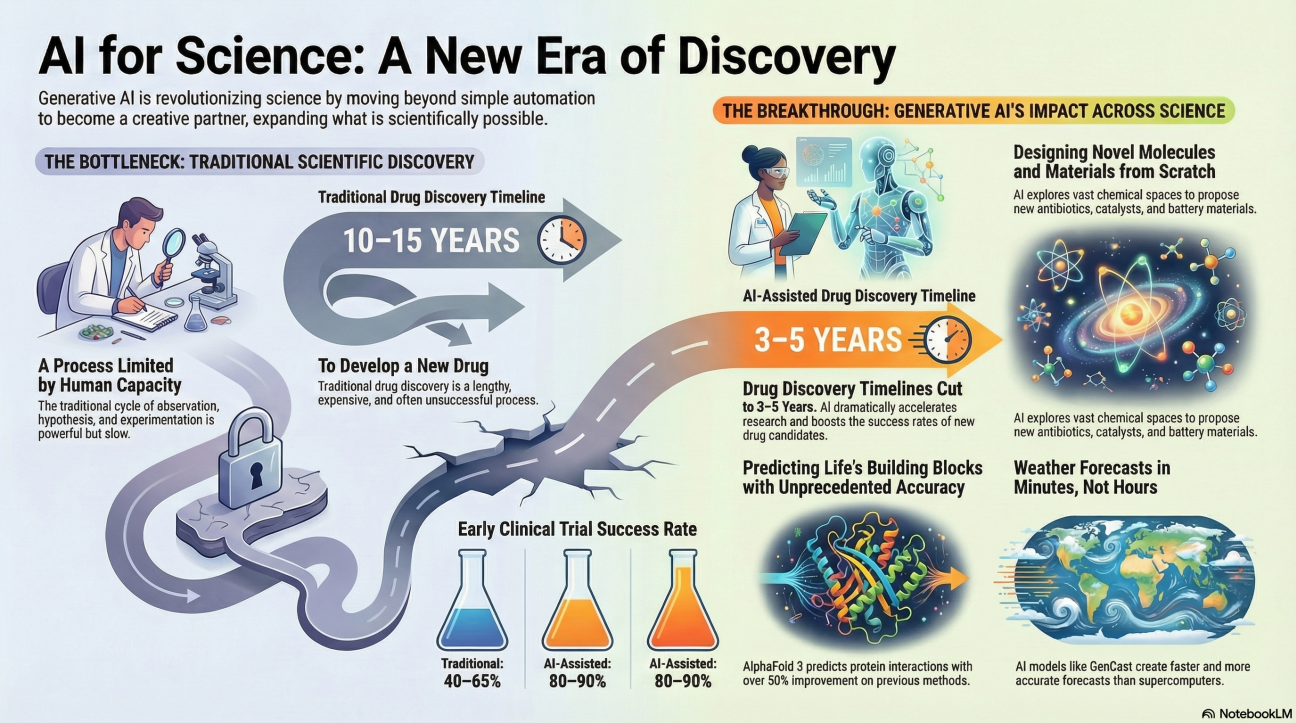


Figure 1. Chapter1.png

Figure 1.1. AI for Science: A New Era of Discovery. Traditional scientific discovery (left) is limited by human capacity, with drug development taking 10–15 years and early clinical trial success rates of 40–65%. Generative AI (right) is transforming this landscape by reducing drug discovery timelines to 3–5 years, improving clinical trial success rates to 80–90%, enabling novel molecule and materials design, predicting protein structures with unprecedented accuracy (AlphaFold 3), and accelerating weather forecasts from hours to minutes (GenCast). (Created by Google NotebookLM)

Cross-Cutting Capabilities

Several techniques and ideas cut across scientific domains:

- **Physics-Informed Models:** Embedding conservation laws, symmetries, and PDEs in neural architectures to ensure physically consistent predictions [24, 25].
- **Latent-Variable and Generative Representations:** Using VAEs, GANs, and diffusion models to learn compact representations of high-dimensional systems [16–20].
- **Multimodal Integration:** Combining text, images, time series, and spatial fields to build holistic models of complex phenomena [1, 6].
- **Multi-Agent AI Systems:** Frameworks like Google's AI co-scientist use specialized agents for hypothesis generation, debate, and evolution [7].

* * *

A New Scientific Partner

Generative AI is *not* replacing scientists—it is becoming a **new kind of collaborator** [1]. It can:

- Expand the hypothesis space far beyond what any individual or team could enumerate
- Accelerate cycles of modeling, simulation, and experiment
- Enable researchers with limited computational or institutional resources to use state-of-the-art tools
- Bridge previously separate fields by revealing shared patterns in their data and models
- Recapitulate discoveries in days that previously required years of iterative research [7]

The most impactful discoveries are likely to emerge where **human insight** and **machine-generated ideas** interact in a tight loop.

* * *

The Path Forward

The rest of this book will help you:

- Understand the **core architectures** (transformers, diffusion models, scientific ML) [15–18, 24, 25].
- Adapt and fine-tune models to your own **scientific domain** [2–5, 11].
- Evaluate and validate models with appropriate **statistical and physical checks** [12, 23, 25].
- Design human–AI workflows in which AI accelerates, rather than distorts, the scientific process [1].

The revolution is already underway. In my view, **scientists who understand and apply generative AI will stand at the forefront of discovery in every field** [1].

* * *

References and Further Readings

1. **Wang, H.**, Fu, T., Du, Y., Gao, W., Huang, K., Liu, Z., *et al.* (2023). Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972), 47–60. https://doi.org/10.1038/s41586-023-06221-2
2. **Stokes, J. M.**, Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., *et al.* (2020). A deep learning approach to antibiotic discovery. *Cell*, 180(4), 688–702. https://doi.org/10.1016/j.cell.2020.01.021
3. **Gómez-Bombarelli, R.**, Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., *et al.* (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2), 268–276. https://doi.org/10.1021/acscentsci.7b00572
4. **Jumper, J.**, Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., *et al.* (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2
5. **Abramson, J.**, Adler, J., Dunger, J., Evans, R., Green, T., Pritzel, A., *et al.* (2024). Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, 630(8016), 493–500. https://doi.org/10.1038/s41586-024-07487-w
6. **Brown, T.**, Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., *et al.* (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901. https://arxiv.org/abs/2005.14165
7. **Gottweis, J.**, Weng, W.-H., Daryin, A., Liu, Y., Ceze, L., Natarajan, V., *et al.* (2025). Towards an AI co-scientist. *arXiv preprint* arXiv:2502.18864. https://arxiv.org/abs/2502.18864
8. **Reichstein, M.**, Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., & Prabhat. (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, 566(7743), 195–204. https://doi.org/10.1038/s41586-019-0912-1

9. **Price, I.**, Sanchez-Gonzalez, A., Alet, F., Andersson, T. R., El-Kadi, A., Masters, D., *et al.* (2024). Probabilistic weather forecasting with machine learning. *Nature*, 636, 84–90. https://doi.org/10.1038/s41586-024-08252-9
10. **Bodnar, C.**, Bruinsma, W. P., Lucic, A., Stanley, M., Allen, A., Brandstetter, J., *et al.* (2024). Aurora: A foundation model of the atmosphere. *arXiv preprint* arXiv:2405.13063. https://arxiv.org/abs/2405.13063
11. **Angello, N. H.**, Friday, D. M., Hwang, C., Yi, S., Cheng, A. H., Torres-Flores, T. C., *et al.* (2024). Closed-loop transfer enables artificial intelligence to yield chemical knowledge. *Nature*, 633(8029), 351–358. https://doi.org/10.1038/s41586-024-07021-y
12. **Brunton, S. L.**, & Kutz, J. N. (2022). *Data-driven science and engineering: Machine learning, dynamical systems, and control* (2nd ed.). Cambridge University Press. https://doi.org/10.1017/9781108380690
13. **AI in Drug Discovery** (2024). Industry analysis reports indicate AI-assisted drug candidates achieving Phase I success rates of nearly 90%, compared with industry averages of 40–65%. *Drug Discovery News*, October 2025. https://pubmed.ncbi.nlm.nih.gov/38692505
14. **Google DeepMind** (2024). GenCast predicts weather and the risks of extreme conditions with state-of-the-art accuracy. https://deepmind.google/blog/gencast-predicts-weather-and-the-risks-of-extreme-conditions-with-sota-accuracy/
15. **Vaswani, A.**, Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., *et al.* (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30. https://arxiv.org/abs/1706.03762
16. **Ho, J.**, Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33, 6840–6851. https://arxiv.org/abs/2006.11239
17. **Dhariwal, P.**, & Nichol, A. (2021). Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems*, 34, 8780–8794. https://arxiv.org/abs/2105.05233
18. **Song, Y.**, Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-based generative modeling through stochastic differential equations. *Proceedings of ICLR*. https://arxiv.org/abs/2011.13456
19. **Kingma, D. P.**, & Welling, M. (2014). Auto-encoding variational Bayes. *Proceedings of ICLR*. https://arxiv.org/abs/1312.6114
20. **Goodfellow, I.**, Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., *et al.* (2014). Generative adversarial networks. *Advances in Neural Information Processing Systems*, 27. https://arxiv.org/abs/1406.2661
21. **Google DeepMind** (2024). AI achieves silver-medal standard solving International Mathematical Olympiad problems. https://deepmind.google/blog/ai-solves-imo-problems-at-silver-medal-level/
22. **Google DeepMind** (2025). Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad. https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/

23. **Carleo, G.**, Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., *et al.* (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4), 045002. https://doi.org/10.1103/RevModPhys.91.045002
24. **Raissi, M.**, Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
25. **Karniadakis, G. E.**, Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6), 422–440. https://doi.org/10.1038/s42254-021-00314-5

Next → Chapter 2: introduces the core architectures powering generative AI.

Chapter 2: Generative AI Fundamentals

Introduction: The Building Blocks of Generation

To harness generative AI for scientific discovery, we must understand how these models work—not as black boxes, but as computational systems with clear mathematical foundations and architectural principles. This chapter demystifies the core technologies powering modern generative AI: transformers, diffusion models, flow matching, and variational approaches using plain English. We'll explore their mechanisms, understand when to use each, and see how they can be adapted for scientific applications.

While the mathematics can become complex, our focus is on building intuition. Scientists don't need to be machine learning experts to use these tools effectively, but understanding the fundamentals will help you choose the right model architecture, diagnose failures, adapt pre-trained models, collaborate with computational researchers, and evaluate limitations.

* * *

The Three Pillars of Generative AI

Modern generative AI rests on three major architectural families:

Architecture	**Core Mechanism**	**Best For**	**Example Applications**
Transformers & LLMs	Self-attention over sequences	Text, code, sequences	Literature synthesis, protein sequences, code generation
Diffusion & Flow Models	Iterative denoising / flow matching	Structured outputs, images	Molecular structures, protein folding, climate data
VAEs & GANs	Latent space learning	Data generation, interpolation	Synthetic data, anomaly detection, compression

Architecture	Core Mechanism	Best For	Example Applications

* * *

Part I: Transformers and Large Language Models

The Attention Revolution

Traditional neural networks process sequences step-by-step. **Transformers** changed this with attention: let every element directly attend to every other element simultaneously [1]. The breakthrough "Attention Is All You Need" paper introduced self-attention mechanisms that model long-range dependencies in parallel, eliminating the sequential bottleneck of recurrent networks.

The Attention Mechanism

Attention computes three quantities for each sequence element [1]:

Component	Role	Intuition
Query (Q)	What am I looking for?	The question each element asks
Key (K)	What do I contain?	How each element describes itself
Value (V)	What do I contribute?	The information each element provides

Attention formula:

```
Attention(Q, K, V) = softmax(Q·K^T / √d_k) · V
```

Here, "softmax" turns raw similarity scores into a meaningful probability distribution that tells the model how much attention to pay to each token.

Why This Works for Science:

- **Protein Sequences**: Connect distant amino acids that interact in 3D
- **Scientific Literature**: Link concepts mentioned paragraphs apart
- **Chemical Reactions**: Identify relationships between reactants and products

The Transformer Architecture

The original encoder-decoder architecture [1] consists of stacked attention and feed-forward layers:

```
Input → Embedding → Positional Encoding
    ↓
[ Multi-Head Attention → Add & Norm
    ↓
  Feed-Forward → Add & Norm ] × N layers
    ↓
Output Probabilities
```

Key Components:

Component	Purpose	Scientific Benefit
Multi-Head Attention	Learn different patterns	Capture multiple relationship types
Residual Connections	Enable deep networks	Scale to billions of parameters
Layer Normalization	Stabilize training	Faster convergence
Positional Encoding	Track sequence order	Understand structure (DNA direction, time series)

From Transformers to Large Language Models

1. Pre-Training: Learning General Patterns

Objective: Predict next token given context.

```
Input:  "The protein binds to the"
Target: "receptor"
```

The GPT series demonstrated the power of autoregressive language modeling at scale [2, 3], while BERT showed bidirectional pre-training for understanding tasks [4]. More recent models like LLaMA [5], Gemini [6], and Claude push the boundaries with trillions of tokens and multimodal capabilities.

Scientific Pre-Trained Models:

- **SciBERT** [7]: Scientific papers
- **ESM-2** [8]: Protein sequences (750M sequences)
- **ESM-3** [9]: Multimodal protein model (sequence, structure, function) with 98B parameters (2024/2025)
- **Llama 3.1/3.2/3.3** [10]: Open foundation models with 128K context (2024)

2. Fine-Tuning: Domain Specialization

Full fine-tuning updates all model parameters, but parameter-efficient methods like LoRA [11] enable adaptation with minimal computational cost by learning low-rank updates to weight matrices.

Method	Trainable Params	Data Needed	Use Case
Full Fine-Tuning	100%	10K-100K	Maximum adaptation
LoRA [11]	0.1-1%	100-10K	Limited compute
Adapters [12]	1-5%	1K-10K	Task-specific layers
QLoRA [13]	0.1-1%	100-10K	Quantized + LoRA (fine-tune 70B on consumer GPU)

3. Prompting: Zero-Shot and Few-Shot

Large language models demonstrate remarkable few-shot learning capabilities [3], enabling scientific applications without task-specific training.

Zero-Shot:

```
Summarize this oceanography paper: [text]
```

Few-Shot:

```
Examples:
SMILES: CCO → Name: Ethanol
SMILES: CC(C)O → Name: Propan-2-ol

SMILES: CCCO → Name: ?
```

Reasoning Models: A New Paradigm (2024–2025)

A significant development in 2024–2025 is the emergence of **reasoning models** that "think before they respond" [14, 15]. Unlike standard LLMs that generate answers in a single pass, reasoning models like OpenAI's o1/o3 series and Gemini Deep Think produce an internal chain of thought before responding.

Key Characteristics:

- Extended reasoning time for complex problems
- Multi-step planning and self-verification
- Superior performance on math, coding, and scientific reasoning
- Configurable "reasoning effort" levels

Scientific Applications:

- Complex mathematical proofs
- Multi-step experimental design
- Code debugging and algorithm implementation
- Hypothesis evaluation requiring logical chains

Limitations for Science

Challenge	Impact	Mitigation
Hallucination	False information	RAG [16], verification, reasoning models
Lack of Uncertainty	Overconfidence	Ensemble methods [17]
Data Cutoff	Outdated knowledge	Fine-tuning, RAG [16]
Context Limits	Long document handling	Extended context models (128K+ tokens)

* * *

Part II: Diffusion Models and Flow Matching

The Denoising Paradigm

Diffusion models learn to generate by reversing a noise-addition process [18, 19]. The foundational work by Sohl-Dickstein et al. [18] established the thermodynamic interpretation, while Ho et al. [19] simplified training with the denoising objective.

Forward Process: Adding Noise

x_0 (real data) → x_1 → x_2 → ... → x_t (pure noise)

At each step: $x_t = \sqrt{(1-\beta_t)} \cdot x_{t-1} + \sqrt{\beta_t} \cdot \varepsilon$ where $\varepsilon \sim N(0,I)$

Reverse Process: Learning to Denoise

Train a network to predict the noise [19]:

$$Loss = E[||\varepsilon - \varepsilon_{pre}(x_t, t)||^2]$$

Generation: Sampling

1. Start with noise: $x_t \sim N(0, I)$
2. Iteratively denoise: $x_t \rightarrow x_{t-1} \rightarrow ... \rightarrow x_0$
3. Result: novel sample

Flow Matching: A Powerful Alternative (2023–2025)

Flow Matching [20] has emerged as a powerful and efficient alternative to diffusion-based generative modeling, with growing interest in scientific applications [21]. Rather than learning to reverse a noising process, flow matching learns a velocity field that transports samples from noise to data along continuous paths.

Key Advantages:

- **Straighter trajectories**: Fewer sampling steps required
- **Stable training**: Simpler objectives, less hyperparameter tuning
- **Faster inference**: Often 2-4x speedup over diffusion
- **Flexible conditioning**: Natural incorporation of constraints

Flow Matching in Biology (2024–2025):

- Molecule generation (NeurIPS 2023) [21]
- Protein backbone generation with SE(3)-equivariant flows (ICLR 2024) [21]
- Antibody design with IgFlow and dyAb [21]
- Biological sequence and peptide generation (ICML 2024) [21]

Why Diffusion/Flow Works for Science

Score-based generative modeling [22] provides a continuous-time perspective, while empirical results show diffusion models produce higher-quality samples than GANs [23].

Feature	Scientific Benefit
High Quality	Realistic structures
Stable Training	Easier than GANs [23]
Interpretable	Visualize generation
Conditional	Incorporate constraints [24]
Uncertainty	Multiple samples

Applications:

- Molecular conformations with valency constraints [25]
- Protein structures respecting physics [26]
- Climate data filling gaps while conserving energy
- High-resolution image synthesis with latent diffusion [27]
- Probabilistic weather forecasting with GenCast [28]

Conditional Diffusion

Generate data with specific properties using classifier guidance or classifier-free guidance [24]:

$$p(x_{t-1} \mid x_t, \text{condition})$$

Conditioning Methods:

Method	Example
Classifier Guidance [23]	"Molecule binds to protein X"
Classifier-Free [24]	Train with/without conditions
Inpainting	Fill missing climate data

* * *

Part III: VAEs and GANs

Variational Autoencoders (VAEs)

VAEs learn probabilistic latent representations through variational inference [29]:

Architecture:

```
Encoder: x → [μ(x), σ(x)]
Sample: z ~ N(μ, σ²)
Decoder: z → x̂
```

Loss:

```
Total = Reconstruction + KL_Divergence
```

Scientific Uses:

- **Chemistry**: Interpolate between molecules [30]
- **Materials**: Optimize in latent space
- **Genomics**: Compress gene expression data

Generative Adversarial Networks (GANs)

GANs introduced adversarial training between generator and discriminator [31]:

Two Networks in Competition:

Network	Role	Goal
Generator	Create fakes	Fool discriminator
Discriminator	Judge real/fake	Detect fakes

Challenges:

- Mode collapse (limited diversity)
- Training instability
- Difficult evaluation

Scientific Uses:

- Data augmentation
- Super-resolution
- Image-to-image translation
- Molecular graph generation [32]

Comparison

Criterion	VAE	GAN	Diffusion	Flow Matching
Quality	Good	Excellent	Excellent	Excellent
Stability	Stable	Unstable	Stable	Very Stable
Diversity	Good	Poor	Excellent	Excellent
Speed	Fast	Fast	Slow	Moderate
Latent Space	Interpretable	Opaque	Implicit	Implicit

* * *

Part IV: Pre-Training and Fine-Tuning

The Transfer Learning Pipeline

```
Pre-Training (general data, millions of examples)
    ↓
Fine-Tuning (domain data, thousands of examples)
    ↓
Prompting (task-specific, zero examples)
```

Foundation models [33] trained on massive corpora can be adapted to downstream scientific tasks with limited data.

Parameter-Efficient Fine-Tuning (PEFT)

LoRA Example [11]:

```python
from peft import LoraConfig, get_peft_model

config = LoraConfig(
    r=16,  # Adaptation rank
    lora_alpha=32,
    target_modules=["q_proj", "v_proj"]
)

model = get_peft_model(base_model, config)
# Now only 0.5% of parameters are trainable!
```

QLoRA [13] combines quantization with LoRA, enabling fine-tuning of 70B models on consumer GPUs—a game-changer for scientific labs with limited compute budgets.

Benefits:

- Train on single GPU
- Fast iteration
- Multiple task-specific versions

Open vs. Closed Models for Science

The 2024–2025 landscape offers scientists unprecedented choice:

Model Family	Access	Parameters	Best For
Llama 3.1 / 3.2 / 3.3	Open weights	1B–405B (text); 11B / 90B (vision variants)	Local deployment, fine-tuning, on-prem and private customization
OpenAI GPT-4/4o/5.1/5.2	API (and ChatGPT)	Not disclosed	Advanced reasoning, multimodal understanding, coding, tool-using agents
Gemini 2.0/3.0 (Pro, Flash)	API (Gemini API, Vertex AI)	Not disclosed	Multimodal tasks, very long context (up to ~1M tokens on supported models)
Claude (Claude 3.5, Sonnet/Opus 4.5)	API	Not disclosed	Deep analysis, coding, safety-oriented assistants, long-context workflows
**ESM-3 **	API + open weights (ESM3-open)	1.4B (open); up to ~98B (largest)	Protein modeling, structure/function prediction, generative protein design

* * *

Part V: Mathematical Foundations

Low-Rank Approximation

Large models use low-rank structures for efficiency [11].

Matrix Factorization:

```
W ∈ R^(m×n) ≈ A·B where A ∈ R^(m×r), B ∈ R^(r×n), r << min(m,n)
```

Applications:

- LoRA adaptation [11]
- Model compression
- Fast inference

Quantization

Reduce precision for efficiency [34, 35]:

Precision	Bits	Range	Use Case
FP32	32	Full	Training
FP16/BF16	16	Mixed	Fast training
INT8	8	Limited	Inference
INT4	4	Very limited	Edge deployment, QLoRA

LLM.int8() [34] and GPTQ [35] enable accurate quantization without retraining.

Benefits:

- 4× smaller models (FP32 → INT8)
- 2-4× faster inference
- Run large models on consumer GPUs

Optimization Theory

Key Algorithms:

Optimizer	Mechanism	When to Use
SGD	Basic gradient descent	Small models, well-understood problems
Adam [36]	Adaptive learning rates	Default choice, most robust
AdamW [37]	Adam + weight decay	Large language models

Learning Rate Schedules:

```
Warmup → Constant → Decay
```

- **Warmup**: Start small, gradually increase (stabilize training)
- **Constant**: Maintain learning rate (main training)
- **Decay**: Reduce toward end (fine-tune solution)

* * *

Part VI: Types of Generative AI by Modality

Text Generation

Capabilities:

- Scientific writing
- Code generation
- Literature synthesis
- Hypothesis generation

Models: GPT-4o/4.5/o1/o3 [14, 15], Llama 3.1/3.2/3.3/4 [10], Gemini 2.0/3.0 [6], Claude.

Image Generation

Capabilities:

- Synthetic microscopy
- Medical imaging
- Scientific visualization
- Data augmentation

Models: Stable Diffusion 3/3.5 [38], SDXL [27], DALL-E 3

Molecular Generation

Capabilities:

- Drug discovery
- Material design
- Reaction prediction

Representations:

- SMILES strings (text-like)
- Molecular graphs [32]
- 3D conformations [25]

Models: MolGAN [32], Diffusion-based generators [25], VAEs [30], Flow matching models [21]

Protein Generation

Capabilities:

- Structure prediction (AlphaFold 3 [39], ESMFold [8])
- Sequence design
- Function prediction
- Multimodal generation (ESM-3) [9]

Models: ESM-2 [8], ESM-3 [9], ProtGPT2 [40], RFdiffusion [26]

2024–2025 Highlight: ESM-3 [9] ESM-3, published in *Science* (January 2025), is a 98B parameter multimodal generative model that reasons over protein sequence, structure, and function simultaneously. It generated a novel green fluorescent protein (GFP) with only 58% sequence identity to known GFPs—equivalent to simulating 500 million years of evolution.

Graph Generation

Capabilities:

- Molecular graphs
- Protein-protein interaction networks
- Knowledge graphs

Models: GraphRNN, MolGAN [32], GraphVAE

Multimodal Generation

Capabilities:

- Text → Image (CLIP [41] + diffusion)
- Text → Molecule (MolT5)
- Image → Text (scientific figure captioning)
- Cross-modal retrieval

Models: CLIP [41], GPT-4o/4V [15], Gemini 2.0 [6]

* * *

Design Principles for Scientific Applications

1. Incorporate Domain Knowledge

Physics-Informed Neural Networks (PINNs) [42]: Embed differential equations in loss function:

```
Loss = Data_Loss + λ·Physics_Loss

Physics_Loss = ||∂u/∂t - α∇²u||²  (heat equation)
```

Benefits:

- Better generalization
- Physical consistency
- Data efficiency

2. Quantify Uncertainty

Methods:

- **Ensemble** [17]: Train multiple models, measure spread
- **Bayesian** [43]: Maintain distributions over parameters
- **Conformal** [44]: Finite-sample coverage guarantees

Why Essential: Scientific decisions have consequences—we must know when models are uncertain.

3. Ensure Reproducibility

Checklist:

- Set random seeds
- Log hyperparameters
- Version control data and code
- Document environment
- Share trained models

4. Validate Rigorously

Validation Type	Purpose
Hold-out Test	Independent performance
Cross-validation	Robust estimates
Domain Expert	Scientific plausibility
Physical Constraints	Obey natural laws
Out-of-distribution	Generalization limits

* * *

Practical Considerations

Model Selection Guide

Your Goal	Recommended Architecture	Rationale
Generate scientific text	Transformer LLM [2, 3, 10]	Excellent at language
Complex reasoning/math	Reasoning models (o1/o3) [14, 15]	Multi-step verification
Design molecules	Diffusion [25] or Flow Matching [21]	High-quality structures

Your Goal	Recommended Architecture	Rationale
Predict protein structure	ESM-3 [9], AlphaFold 3 [39]	State-of-the-art specialized models
Fill missing data	Conditional diffusion [24]	Respects constraints
Synthetic data augmentation	GAN [31] or VAE [29]	Fast generation
Explore design space	VAE [29]	Interpretable latent space

Computational Requirements

Task	Typical Resources	Alternative
Fine-tune small LLM	Single GPU, 1-2 days	Use LoRA/QLoRA [11, 13] for efficiency
Train diffusion model	4-8 GPUs, 3-7 days	Use pre-trained models [27]
Inference (LLM)	1 GPU or CPU	Quantization [34, 35] for speed
Protein structure	1 GPU, minutes	Use hosted APIs

Common Pitfalls

Pitfall	Consequence	Solution
Overfitting	Poor generalization	Regularization, more data
Underfitting	Poor performance	Larger model, more training
Data leakage	Inflated metrics	Careful splitting
Ignoring uncertainty	Overconfident predictions	Uncertainty quantification [17, 43, 44]
Black-box use	Unexplainable failures	Understand fundamentals

* * *

Summary

This chapter introduced the core architectures powering generative AI:

Transformers [1] excel at sequential data through attention mechanisms, enabling scientific text generation [2, 3, 10], code synthesis, and protein language models [8, 9]. The emergence of **reasoning models** [14, 15] in 2024–2025 represents a paradigm shift for complex scientific problem-solving.

Diffusion models [18, 19, 22] and **flow matching** [20, 21] generate high-quality structured outputs by learning to reverse noise or transport samples along continuous paths, ideal for molecular design [25], protein structures [26], and climate data reconstruction [28].

VAEs [29] and **GANs** [31] provide complementary approaches for data generation, exploration, and augmentation.

Transfer learning [33]—pre-training followed by fine-tuning [11, 12, 13]—allows scientists to leverage massive general models with modest domain-specific data. Open models like Llama 3/4 [10] democratize access for scientific labs.

Mathematical foundations like low-rank approximation [11], quantization [34, 35], and optimization theory [36, 37] enable efficient training and deployment.

Understanding these fundamentals empowers you to:

- Choose appropriate models for your scientific problems
- Adapt existing models to new domains
- Diagnose and fix failures
- Collaborate effectively with AI researchers
- Push the boundaries of what's possible in your field

* * *

References

1. **Vaswani, A.**, Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., *et al.* (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30. https://arxiv.org/abs/1706.03762
2. **Radford, A.**, Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners (GPT-2). *OpenAI Blog*. https://openai.com/research/better-language-models

3. **Brown, T.**, Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., *et al.* (2020). Language models are few-shot learners (GPT-3). *Advances in Neural Information Processing Systems*, 33, 1877–1901. https://arxiv.org/abs/2005.14165
4. **Devlin, J.**, Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, 4171–4186. https://arxiv.org/abs/1810.04805
5. **Touvron, H.**, Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., *et al.* (2023). LLaMA: Open and efficient foundation language models. *arXiv preprint* arXiv:2302.13971. https://arxiv.org/abs/2302.13971
6. **Gemini Team, Google**. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint* arXiv:2403.05530. https://arxiv.org/abs/2403.05530
7. **Beltagy, I.**, Lo, K., & Cohan, A. (2019). SciBERT: A pretrained language model for scientific text. *Proceedings of EMNLP-IJCNLP*, 3615–3620. https://arxiv.org/abs/1903.10676
8. **Lin, Z.**, Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., *et al.* (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model (ESM-2/ESMFold). *Science*, 379(6637), 1123–1130. https://doi.org/10.1126/science.ade2574
9. **Hayes, T.**, Rao, R., Akin, H., Sofroniew, N. J., Oktay, D., Lin, Z., *et al.* (2025). Simulating 500 million years of evolution with a language model (ESM-3). *Science*, 387(6736), 850–858. https://doi.org/10.1126/science.ads0018
10. **Meta AI**. (2024). Llama 3.1, 3.2, and 3.3: The most capable openly available LLMs. *Meta AI Blog*. https://ai.meta.com/blog/meta-llama-3-1/
11. **Hu, E. J.**, Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., *et al.* (2022). LoRA: Low-rank adaptation of large language models. *Proceedings of ICLR*. https://arxiv.org/abs/2106.09685
12. **Houlsby, N.**, Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., *et al.* (2019). Parameter-efficient transfer learning for NLP. *Proceedings of ICML*, 2790–2799. https://arxiv.org/abs/1902.00751
13. **Dettmers, T.**, Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2024). QLoRA: Efficient finetuning of quantized LLMs. *Proceedings of NeurIPS*, 36. https://arxiv.org/abs/2305.14314
14. **OpenAI**. (2024). Introducing OpenAI o1. *OpenAI Blog*. https://openai.com/index/introducing-openai-o1-preview/
15. **OpenAI**. (2024). GPT-4o and reasoning models. *OpenAI Platform*. https://platform.openai.com/docs/models
16. **Lewis, P.**, Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., *et al.* (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Proceedings of NeurIPS*, 33, 9459–9474. https://arxiv.org/abs/2005.11401
17. **Lakshminarayanan, B.**, Pritzel, A., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Proceedings of NeurIPS*, 30. https://arxiv.org/abs/1612.01474

18. **Sohl-Dickstein, J.**, Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *Proceedings of ICML*, 2256–2265. https://arxiv.org/abs/1503.03585
19. **Ho, J.**, Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33, 6840–6851. https://arxiv.org/abs/2006.11239
20. **Lipman, Y.**, Chen, R. T. Q., Ben-Hamu, H., Nickel, M., & Le, M. (2023). Flow matching for generative modeling. *Proceedings of ICLR*. https://arxiv.org/abs/2210.02747
21. **Li, Z.**, Zeng, Z., Lin, X., Fang, F., Qu, Y., Xu, Z., *et al.* (2025). Flow matching meets biology and life science: A survey. *arXiv preprint* arXiv:2507.17731. https://arxiv.org/abs/2507.17731
22. **Song, Y.**, Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-based generative modeling through stochastic differential equations. *Proceedings of ICLR*. https://arxiv.org/abs/2011.13456
23. **Dhariwal, P.**, & Nichol, A. (2021). Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems*, 34, 8780–8794. https://arxiv.org/abs/2105.05233
24. **Ho, J.**, & Salimans, T. (2022). Classifier-free diffusion guidance. *NeurIPS Workshop on Score-Based Methods*. https://arxiv.org/abs/2207.12598
25. **Hoogeboom, E.**, Satorras, V. G., Vignac, C., & Welling, M. (2022). Equivariant diffusion for molecule generation in 3D. *Proceedings of ICML*, 8867–8887. https://arxiv.org/abs/2203.17003
26. **Watson, J. L.**, Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., *et al.* (2023). De novo design of protein structure and function with RFdiffusion. *Nature*, 620(7976), 1089–1100. https://doi.org/10.1038/s41586-023-06415-8
27. **Rombach, R.**, Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models (Stable Diffusion). *Proceedings of CVPR*, 10684–10695. https://arxiv.org/abs/2112.10752
28. **Price, I.**, Sanchez-Gonzalez, A., Alet, F., Andersson, T. R., El-Kadi, A., Masters, D., *et al.* (2024). Probabilistic weather forecasting with machine learning (GenCast). *Nature*, 636, 84–90. https://doi.org/10.1038/s41586-024-08252-9
29. **Kingma, D. P.**, & Welling, M. (2014). Auto-encoding variational Bayes. *Proceedings of ICLR*. https://arxiv.org/abs/1312.6114
30. **Gómez-Bombarelli, R.**, Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., *et al.* (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2), 268–276. https://doi.org/10.1021/acscentsci.7b00572
31. **Goodfellow, I.**, Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., *et al.* (2014). Generative adversarial networks. *Advances in Neural Information Processing Systems*, 27. https://arxiv.org/abs/1406.2661
32. **De Cao, N.**, & Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*. https://arxiv.org/abs/1805.11973

33. **Bommasani, R.**, Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., *et al.* (2021). On the opportunities and risks of foundation models. *arXiv preprint* arXiv:2108.07258. https://arxiv.org/abs/2108.07258
34. **Dettmers, T.**, Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Proceedings of NeurIPS*, 35, 30318–30332. https://arxiv.org/abs/2208.07339
35. **Frantar, E.**, Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. *Proceedings of ICLR*. https://arxiv.org/abs/2210.17323
36. **Kingma, D. P.**, & Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of ICLR*. https://arxiv.org/abs/1412.6980
37. **Loshchilov, I.**, & Hutter, F. (2019). Decoupled weight decay regularization (AdamW). *Proceedings of ICLR*. https://arxiv.org/abs/1711.05101
38. **Stability AI**. (2024). Stable Diffusion 3: Scaling rectified flow transformers for high-resolution image synthesis. https://stability.ai/stable-image
39. **Abramson, J.**, Adler, J., Dunger, J., Evans, R., Green, T., Pritzel, A., *et al.* (2024). Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, 630(8016), 493–500. https://doi.org/10.1038/s41586-024-07487-w
40. **Ferruz, N.**, Schmidt, S., & Höcker, B. (2022). ProtGPT2 is a deep unsupervised language model for protein design. *Nature Communications*, 13(1), 4348. https://doi.org/10.1038/s41467-022-32007-7
41. **Radford, A.**, Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., *et al.* (2021). Learning transferable visual models from natural language supervision (CLIP). *Proceedings of ICML*, 8748–8763. https://arxiv.org/abs/2103.00020
42. **Raissi, M.**, Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
43. **Gal, Y.**, & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *Proceedings of ICML*, 1050–1059. https://arxiv.org/abs/1506.02142
44. **Angelopoulos, A. N.**, & Bates, S. (2021). A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint* arXiv:2107.07511. https://arxiv.org/abs/2107.07511

Additional Resources

Textbooks

- **Goodfellow, I.**, Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

- **Prince, S. J. D.** (2023). *Understanding Deep Learning*. MIT Press. http://udlbook.github.io/udlbook/
- **Tunstall, L.**, von Werra, L., & Wolf, T. (2022). *Natural Language Processing with Transformers*. O'Reilly Media.
- **Alammar, J.**, & van den Oord, A. (2024). *Generative Deep Learning* (2nd ed.). O'Reilly Media. https://www.oreilly.com/library/view/generative-deep-learning/9781098134174/
- **Raschka, S.** (2024). *Build a Large Language Model (From Scratch)*. Manning. https://www.manning.com/books/build-a-large-language-model-from-scratch
- **Holderrieth, P.**, & Erives, E. (2025). *Introduction to Flow Matching and Diffusion Models*. MIT Course 6.S184. https://diffusion.csail.mit.edu/
- **Liu, J.P.** (2025). *How to Build and Fine-tune a Small Language Model*. Leanpub. https://leanpub.com/howtobuildandfine-tuneasmalllanguagemodel

Key Review Papers

- **Yang, L.**, et al. (2023). Diffusion models: A comprehensive survey. *ACM Computing Surveys*, 56(4), 1–39.
- **Bommasani, R.**, et al. (2022). On the opportunities and risks of foundation models. https://arxiv.org/abs/2108.07258
- **Zhao, W. X.**, et al. (2023). A survey of large language models. *ACM Computing Surveys*. https://arxiv.org/abs/2303.18223
- **Li, Z.**, et al. (2025). Flow matching meets biology and life science: A survey. https://arxiv.org/abs/2507.17731

* * *

This chapter provides the foundational knowledge needed to understand and apply generative AI in scientific contexts. For the latest developments, regularly check arXiv (cs.LG, cs.CL, q-bio sections) and major conference proceedings (NeurIPS, ICML, ICLR, CVPR).

* * *

Next → Chapter 3: Scientific Data & Workflows—navigating the unique challenges of scientific datasets and integrating AI into research pipelines.

Chapter 3: Scientific Data & Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: The Data Challenge in Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Unique Challenges of Scientific Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Challenge 1: Small Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Challenge 2: Noisy and Heterogeneous Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Challenge 3: Domain-Specific Vocabularies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Challenge 4: Temporal and Spatial Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Challenge 5: High Dimensionality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Data Sources in Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Publications and Literature

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Experimental Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Simulation Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Synthetic Data Generation (2024–2025 Advances)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Field Data and Observations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: The FAIR Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Findable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Accessible

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Interoperable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Reusable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Data Preparation for AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The AI-Ready Data Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Data-Centric AI: A Paradigm Shift

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Quality Control Automation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Normalization Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Data Splitting Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Data Augmentation for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Integrating AI into Research Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Hybrid Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

AI as Research Assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Autonomous and Self-Driving Labs (2024–2025)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Reproducibility in AI-Enhanced Research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Automated Workflow Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

From Description to Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Workflow Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Integration with Existing Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Additional Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Textbooks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 4: Text, Code & Knowledge Generation for Scientists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: The Knowledge Synthesis Challenge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Literature Review and Synthesis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Traditional Literature Review

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

AI-Assisted Literature Review

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Retrieval-Augmented Generation (RAG)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Why RAG Matters for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

RAG Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Building a Scientific RAG System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Advanced RAG Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

GraphRAG: Knowledge Graph-Enhanced Retrieval (2024–2025)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Domain-Specific RAG Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Hypothesis Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

From Pattern Recognition to Hypothesis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Autonomous Research Agents (2024–2025)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Hypothesis Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Code Generation for Research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Evolution of AI Code Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Setting Up API Access

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Practical Example: Automated Scientific Data Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Automated Code Generation from Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

LLM-Generated Analysis Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Best Practices for AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Scientific Writing Assistance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Use Cases for Writing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Writing Methods Sections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Describing Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Addressing Reviewer Comments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Ethical Considerations in AI-Assisted Writing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Educational Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

AI for Course Material Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VII: Domain-Specific LLM Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Building Specialized Scientific Assistants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Example: OceanGPT System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VIII: Limitations and Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Known Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Additional Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 5: Data-to-Data Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: The Data Scarcity Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Missing Data Imputation with Autoencoders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Challenge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Autoencoder Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Training with Early Stopping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Evaluation Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Figure 5.3: Neural-Network–Based Imputation Results and Quality Assessment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Diffusion Models for Missing Data Imputation (2024–2025)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Synthetic Data Generation with GANs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Mode Collapse Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Improved GAN Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Training with Gradient Penalty

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Quality Assessment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Synthetic Tabular Data: 2025 Landscape

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Variational Autoencoders (VAEs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Why VAEs?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Robust VAE Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Training with KL Annealing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

VAE Quality Assessment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Gaussian Process Spatial Interpolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Challenge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Gaussian Process Fitting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Uncertainty Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Time Series Gap Filling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Challenge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Enhanced Bidirectional LSTM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary and Key Takeaways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

🎯 Results Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

📊 Method Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

💡 Key Lessons

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

⚠️ Important Reminders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

🚀 Practical Guidelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

📈 Performance Benchmarks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Next Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Additional Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 6: Physics-Informed AI and Simulation

Introduction: Embedding Physics in Neural Networks

Scientific simulations are computationally expensive. Climate models require supercomputers running for weeks. Computational fluid dynamics simulations take days. Molecular dynamics calculations consume vast resources. Yet researchers need to explore thousands of parameter combinations, run sensitivity analyses, and make real-time predictions.

Traditional neural networks learn patterns from data alone, ignoring centuries of accumulated physical knowledge. **Physics-Informed Neural Networks (PINNs)** [1, 2, 3] and **simulation surrogates** [4, 5] change this paradigm:

- **Physics-Informed Neural Networks**: Embed physical laws (differential equations) directly into the loss function, enabling data-efficient learning of physics-consistent solutions
- **Simulation Surrogates**: Replace expensive simulations with fast neural network approximations, achieving 1000×+ speedups while maintaining accuracy
- **Hybrid Approaches**: Combine first-principles physics with data-driven corrections for complex systems [6]

This chapter focuses on practical implementations of physics-informed AI and simulation acceleration. We'll cover PINNs for solving PDEs, building accurate surrogate models, and validating physics-consistent neural networks.

* * *

Part I: Physics-Informed Neural Networks (PINNs)

The Core Concept

Traditional neural networks learn purely from data. **Physics-Informed Neural Networks** [1, 2] embed known physical laws (differential equations) directly into the loss function.

Advantages:

Benefit	Description
Data Efficiency	Requires less training data due to physics constraints [3]
Physical Consistency	Solutions obey known laws
Generalization	Better extrapolation beyond training data [7]
Interpretability	Combines data and theory explicitly

Basic PINN Implementation

Heat Equation Example:

We'll solve the 1D heat equation [8]:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

- `u(x,t) = temperature`
- `x = position`
- `t = time`
- `α = thermal diffusivity (material constant)`

with initial condition $u(x,0) = \sin(\pi x)$ and boundary conditions $u(0,t) = u(1,t) = 0$.

We are using a **Physics-Informed Neural Network (PINN)** [1].
Instead of discretizing the **Partial Differential Equation (PDE)** on a grid, we represented the solution as a neural network:

$$u_\theta(x,t)$$

and trained it so that:

- it **satisfies the PDE** (small physics residual),
- it matches the **initial condition** $u(x,0) = \sin(\pi x)$,
- it satisfies the **boundary conditions** $u(0,t) = u(1,t) = 0$.

We can compare the results with the output from classic analytical solution:

$$u(x,t) = \sin(\pi x)\, e^{-\pi^2 \alpha t}$$

Codes:

```python
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt

class PINN(nn.Module):
    """Physics-Informed Neural Network for PDEs"""
    # Input dimension = 2 (because input is (x, t))
    # 4 hidden layers, each 50 neurons
    # Output dimension = 1 (temperature u)
    def __init__(self, layers=[2, 50, 50, 50, 50, 1]):
        super().__init__()

        self.activation = nn.Tanh()
        self.layers = nn.ModuleList()

        # Build network
        for i in range(len(layers) - 1):
            self.layers.append(nn.Linear(layers[i], layers[i+1]))

    def forward(self, x, t):
        """
        x: spatial coordinate
        t: time coordinate
        """
        inputs = torch.cat([x, t], dim=1)
        #This forms a 2-column tensor [[x, t], [x, t], ...].

        # Forward pass through hidden layers
        for layer in self.layers[:-1]:
            inputs = self.activation(layer(inputs))

        # Output layer (no activation)
        return self.layers[-1](inputs)

def compute_pde_residual(model, x, t, alpha=0.01):
    """
    Compute PDE residual for heat equation:
    ∂u/∂t = α ∂²u/∂x²
    """
    #This is the heart of PINNs: use autograd to compute derivatives, then penalize PDE violations.

    u = model(x, t)

    # Compute derivatives using autograd
    u_t = torch.autograd.grad(
        u, t,
        torch.ones_like(u),
        create_graph=True,
        retain_graph=True
    )[0]
    #compute derivatives using PyTorch autograd
    #Because x_colloc and t_colloc were created with requires_grad=True, PyTorch can differentiate
    ↪  through the network.


    u_x = torch.autograd.grad(
```

```
        u, x,
        torch.ones_like(u),
        create_graph=True,
        retain_graph=True
    )[0]

    u_xx = torch.autograd.grad(
        u_x, x,
        torch.ones_like(u_x),
        create_graph=True,
        retain_graph=True
    )[0]

    # PDE residual
    residual = u_t - alpha * u_xx

    return (residual ** 2).mean()

def analytical_solution(x, t, alpha=0.01):
    """Analytical solution for validation"""
    return np.sin(np.pi * x) * np.exp(-np.pi**2 * alpha * t)

# Initialize model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
pinn = PINN().to(device)
optimizer = torch.optim.Adam(pinn.parameters(), lr=1e-3)
#torch.optim.Adam   •   Adam = Adaptive Moment Estimation
print(f"PINN parameters: {sum(p.numel() for p in pinn.parameters()):,}")
# Output: PINN parameters: 7,851

# Generate training points
n_colloc, n_bc = 5000, 200

# Collocation points (enforce PDE)
x_colloc = torch.rand(n_colloc, 1, requires_grad=True).to(device)
t_colloc = torch.rand(n_colloc, 1, requires_grad=True).to(device)

# Initial condition points
x_ic = torch.rand(n_bc, 1).to(device)
t_ic = torch.zeros(n_bc, 1).to(device)
u_ic = torch.sin(np.pi * x_ic)

# Boundary condition points
x_bc = torch.cat([torch.zeros(n_bc // 2, 1), torch.ones(n_bc // 2, 1)]).to(device)
t_bc = torch.rand(n_bc, 1).to(device)
u_bc = torch.zeros(n_bc, 1).to(device)

print(f"Training points: {n_colloc + 2*n_bc}")
# Output: Training points: 5400

# Training loop
import time

print("Training PINN...\n")
epochs = 3000
pde_losses, ic_losses, bc_losses = [], [], []
start_time = time.time()
```

```
for epoch in range(epochs):
    optimizer.zero_grad()

    # Compute losses
    loss_pde = compute_pde_residual(pinn, x_colloc, t_colloc, alpha=0.01)
    loss_ic = torch.nn.functional.mse_loss(pinn(x_ic, t_ic), u_ic)
    loss_bc = torch.nn.functional.mse_loss(pinn(x_bc, t_bc), u_bc)

    # Combined loss (weight initial/boundary conditions more)
    loss = loss_pde + 10 * loss_ic + 10 * loss_bc

    # Backward pass
    loss.backward()
    optimizer.step()

    # Track losses
    pde_losses.append(loss_pde.item())
    ic_losses.append(loss_ic.item())
    bc_losses.append(loss_bc.item())

    if (epoch + 1) % 500 == 0:
        print(f"Epoch {epoch+1}: PDE={loss_pde.item():.6f}, "
              f"IC={loss_ic.item():.6f}, BC={loss_bc.item():.6f}")

#Each epoch computes:
#   loss_pde: PDE residual inside the domain
#   loss_ic: mismatch at initial condition
#   loss_bc: mismatch at boundary conditions

training_time = time.time() - start_time
print(f"\n✓ Training complete! Time: {training_time/60:.1f} min")

# Evaluation
'''It creates a meshgrid over [0,1]×[0,1], predicts U_pred, and compares to:
    U_analytical = sin(pi x) * exp(-pi^2 alpha t)
    Then reports:
    •   RMSE
    •   max absolute error
    •   final PDE/IC/BC losses
'''
n_test = 100
x_test = np.linspace(0, 1, n_test)
t_test = np.linspace(0, 1, n_test)
X_test, T_test = np.meshgrid(x_test, t_test)

# PINN predictions
with torch.no_grad():
    X_flat = torch.FloatTensor(X_test.flatten()[:, None]).to(device)
    T_flat = torch.FloatTensor(T_test.flatten()[:, None]).to(device)
    U_pred = pinn(X_flat, T_flat).cpu().numpy().reshape(n_test, n_test)

# Analytical solution
U_analytical = analytical_solution(X_test, T_test, alpha=0.01)
error = np.abs(U_pred - U_analytical)
rmse = np.sqrt(np.mean(error ** 2))
```

```
print("\n" + "="*70)
print("PINN RESULTS")
print("="*70)
print(f"Training time: {training_time/60:.1f} minutes")
print(f"Final PDE loss: {pde_losses[-1]:.6f}")
print(f"Final IC loss: {ic_losses[-1]:.6f}")
print(f"Final BC loss: {bc_losses[-1]:.6f}")
print(f"\nRMSE vs analytical: {rmse:.6f}")
print(f"Max error: {error.max():.6f}")
print("="*70)
if rmse < 0.01:
    print("\n✓ EXCELLENT: PINN solves PDE accurately!")
print("="*70)
```

Expected Output:

```
Training PINN...

Epoch 500: PDE=0.001629, IC=0.000023, BC=0.000059
...
Epoch 3000: PDE=0.000088, IC=0.000003, BC=0.000002

✓ Training complete! Time: 0.4 min

======================================================================
PINN RESULTS
======================================================================
Training time: 0.4 minutes
Final PDE loss: 0.000088
Final IC loss: 0.000003
Final BC loss: 0.000002

RMSE vs analytical: 0.002307
Max error: 0.004397
======================================================================

✓ EXCELLENT: PINN solves PDE accurately!
======================================================================
```

Key Results:

- The PINN achieves an **RMSE of 0.002307** compared to the analytical solution
- Maximum error is only **0.004397**, representing < 0.5% deviation
- Training takes only **0.4 minutes** on a GPU
- The model learns to satisfy both the PDE and boundary/initial conditions

During training, the network learns a function that behaves like the true analytical solution. The final result shows excellent accuracy:

- **RMSE:** 0.0023

- **Max error:** 0.0044
- **PDE / IC / BC losses:** all extremely small

* * *

Understanding the PINN Loss Function Design

The PINN loss function is a carefully designed multi-objective optimization problem. Understanding its structure is essential for successful implementation and avoiding common pitfalls.

The Composite Loss Structure

The total loss combines three distinct terms:

```
loss = loss_pde + 10 * loss_ic + 10 * loss_bc
```

Each term serves a specific purpose:

Loss Term	Mathematical Form	Physical Meaning
PDE Loss	`\mathcal{L}_{PDE} = \frac{1}{N_c}\sum_{i=1}^{N_c}\left(\frac{\partial u_\theta}{\partial t} - \alpha \frac{\partial^2 u_\theta}{\partial x^2}\right)^2$`	Enforces governing physics throughout the domain
IC Loss	`\mathcal{L}_{IC} = \frac{1}{N_{ic}}\sum_{i=1}^{N_{ic}}(u_\theta(x_i, 0) - \sin(\pi x_i))^2$`	Matches initial temperature distribution
BC Loss	`\mathcal{L}_{BC} = \frac{1}{N_{bc}}\sum_{i=1}^{N_{bc}}(u_\theta(0, t_i)^2 + u_\theta(1, t_i)^2)$`	Enforces zero-temperature boundaries

Why Weight Boundary/Initial Conditions More Heavily?

The weighting factor of 10 for IC and BC losses is not arbitrary—it reflects several important considerations:

1. Scale Balancing: PDE residuals are computed at thousands of collocation points (N_c = 5000), while boundary conditions use fewer points (N_bc = 200). Without weighting, the PDE loss dominates simply due to having more terms, potentially causing the network to ignore boundary constraints.

2. Constraint Hierarchy: In physics, boundary and initial conditions are *hard constraints*—the solution must satisfy them exactly. The PDE is satisfied *everywhere* but with some tolerance. Higher weights enforce this hierarchy:

```
# Without weighting: BC violations may persist
loss = loss_pde + loss_ic + loss_bc  # BC often violated

# With weighting: BC enforced more strictly
loss = loss_pde + 10 * loss_ic + 10 * loss_bc  # BC satisfied first
```

3. Training Dynamics: Early in training, the network often learns boundary conditions first (they're simpler patterns), then gradually learns the interior PDE behavior. Weighting accelerates this natural progression.

Alternative Weighting Strategies

The choice of weights significantly impacts training. Common approaches include:

Strategy	Formula	When to Use
Fixed weights	`\lambda_{BC} = 10$`	Simple problems, known scale relationships
Adaptive weights [Wang et al., 2021]	‘\lambda_i = \frac{\max_j \nabla_\theta \mathcal{L}_j	
Learning rate annealing	Start with `\lambda_{BC} = 100$`, decay to 1	When BC satisfaction is critical early
Gradient balancing	Normalize by gradient magnitudes	When loss scales differ significantly

```python
# Example: Adaptive weighting implementation
def compute_adaptive_weights(model, losses, epsilon=1e-8):
    """
    Adjust weights based on gradient magnitudes.
    Prevents any single loss from dominating training.
    """
    grads = []
    for loss in losses:
        model.zero_grad()
        loss.backward(retain_graph=True)
        grad_norm = sum(p.grad.norm()**2 for p in model.parameters()
                        if p.grad is not None)**0.5
        grads.append(grad_norm.item())

    max_grad = max(grads)
    weights = [max_grad / (g + epsilon) for g in grads]
    return weights
```

Why Mean Squared Error?

We use MSE for all loss terms because:

1. **Smoothness:** MSE is continuously differentiable everywhere, enabling stable gradient-based optimization
2. **Physical interpretation:** MSE corresponds to minimizing the L^2 norm of residuals, which has physical meaning (energy minimization in many systems)
3. **Gradient behavior:** MSE provides informative gradients even when errors are small, unlike L^1 loss which has constant gradients

For problems with outliers or heavy-tailed error distributions, alternatives like Huber loss or log-cosh loss may be preferable.

* * *

Why Are PINNs Powerful?

Traditional numerical solvers (finite difference [9], finite element [10], etc.) require grids or meshes and solve PDEs by stepping forward in time or solving large linear systems. PINNs take a different approach [2, 3]:

- **Mesh-free:** no grid needed; just sample points in space–time.
- **Physics + data together:** the loss function can include PDEs, boundary conditions, and real measurements.

- **Differentiable:** the entire solution is a neural network, so you can compute gradients w.r.t. parameters.
- **Handles complex or high-dimensional problems** where standard solvers struggle [7].
- **Fast evaluation after training:** once trained, the network gives instant predictions.

PINNs are not always faster for simple PDEs, but they are extremely useful when the problem involves **complex physics, irregular domains, unknown parameters, or limited data** [11].

Visualization

```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# PINN solution
c1 = axes[0].contourf(X_test, T_test, U_pred, levels=20, cmap='coolwarm')
axes[0].set_title('PINN Solution', fontweight='bold')
axes[0].set_xlabel('x')
axes[0].set_ylabel('t')
plt.colorbar(c1, ax=axes[0])

# Analytical solution
c2 = axes[1].contourf(X_test, T_test, U_analytical, levels=20, cmap='coolwarm')
axes[1].set_title('Analytical Solution', fontweight='bold')
axes[1].set_xlabel('x')
axes[1].set_ylabel('t')
plt.colorbar(c2, ax=axes[1])

# Error
c3 = axes[2].contourf(X_test, T_test, error, levels=20, cmap='Reds')
axes[2].set_title('Absolute Error', fontweight='bold')
axes[2].set_xlabel('x')
axes[2].set_ylabel('t')
plt.colorbar(c3, ax=axes[2])

plt.tight_layout()
plt.savefig('pinn_heat_equation.png', dpi=300, bbox_inches='tight')
```

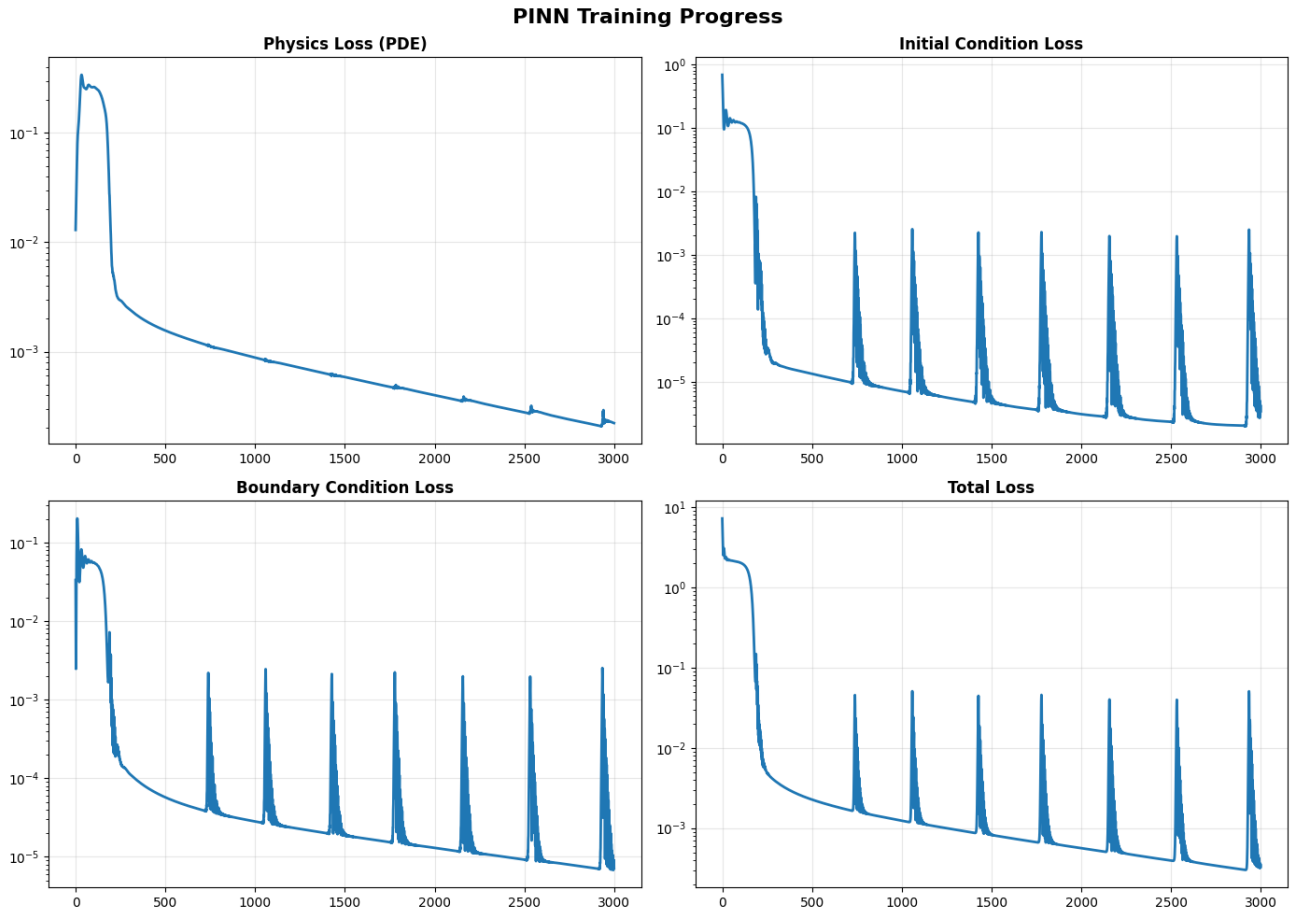


Figure 6.1. PINN Training Progress for the 1D Heat Equation. This figure shows the evolution of the four key loss components during training of a Physics-Informed Neural Network (PINN) used to solve the 1D heat equation $u_t = \alpha u_{xx}$. The PINN is trained simultaneously to satisfy (1) the governing PDE, (2) the initial condition, and (3) the boundary conditions. Each subplot tracks the loss value on a logarithmic scale over 3000 training epochs. The final losses demonstrate excellent agreement with the analytical solution, confirming that the PINN has learned a physically consistent solution across the entire space–time domain.

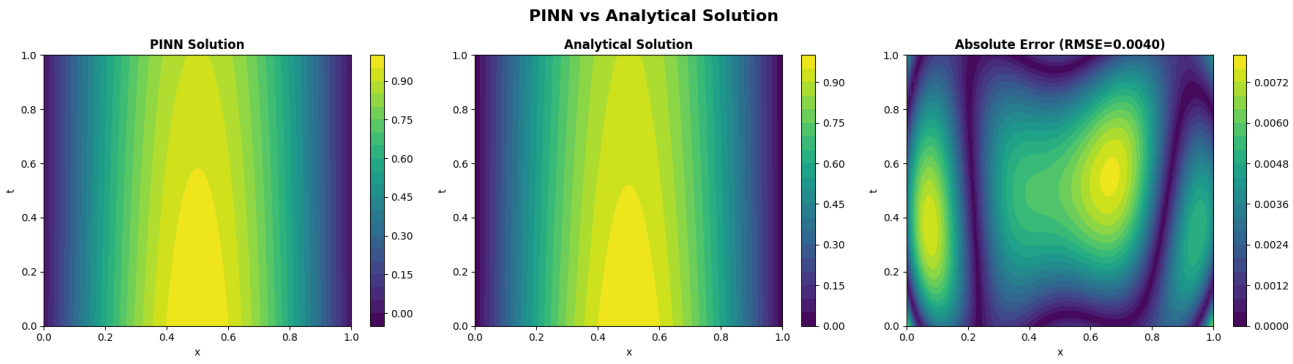


Figure 6.2. Comparison of the PINN Solution, Analytical Solution, and Absolute Error for the 1D Heat Equation. The left panel shows the Physics-Informed Neural Network (PINN) solution $u_\theta(x,t)$ learned from enforcing the heat equation $u_t = \alpha u_{xx}$, the initial condition $u(x,0) = \sin(\pi x)$, and zero Dirichlet boundary conditions. The middle panel displays the exact analytical solution

$$u(x,t) = \sin(\pi x)\, e^{-\pi^2 \alpha t}$$

The right panel illustrates the point-wise absolute error $|u_\theta - u_{\text{exact}}|$ over the entire space–time domain. The PINN solution closely matches the analytical solution, with a very small root-mean-square error (RMSE ≈ 0.0040), demonstrating that the trained network accurately captures the diffusion dynamics of the heat equation.

Advanced PINN: Navier–Stokes Equations

The **Navier–Stokes equations** [12] are the fundamental mathematical laws that describe how fluids move.

They govern the behavior of air, water, blood, ocean currents, smoke, oil, plasma—almost every fluid you can think of. Mathematically, they encode **conservation of momentum and mass** for a fluid.

For a 2D incompressible flow with velocity components $u(x,y,t)$, $v(x,y,t)$ and pressure $p(x,y,t)$, the momentum equations can be written as:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

These equations describe how the velocity of a fluid changes due to:

- **Acceleration**
- **Pressure forces**
- **Viscous forces** (internal friction)
- **Nonlinear advection** (the fluid transporting itself)

where:

- u, v = horizontal and vertical velocity components
- p = pressure
- ν = kinematic viscosity

Why start with an analytical Navier–Stokes flow?

Before applying PINNs to real-world flows where no analytical solution exists, we first test them on a **benchmark problem** with a known solution. In this chapter we use a 2D Taylor–Green–style vortex [13], which has an exact closed-form solution for u, v, p.

This gives us:

- A **ground-truth reference** to compute RMSE for velocity and pressure
- A way to **verify** our PDE residual implementation and automatic differentiation
- A controlled setting to **tune the PINN architecture and loss weights**
- Confidence that the PINN works *before* we deploy it on harder, data-limited problems

Think of this as a **unit test** for our Navier–Stokes PINN.

Implementing a Navier–Stokes PINN

We now build a simple PINN that takes (x, y, t) as input and predicts the velocity and pressure fields (u, v, p). We use a fully connected network with `tanh` activations, which works well for smooth flows [14].

```
### The complete running codes with visulization are available on the Google colab code:
↪  https://github.com/jpliu168/Generative_AI_For_Science
###
# ============================================================
# 3. Simple PINN architecture (tanh activation)
# ============================================================

class SimpleNSPINN(nn.Module):
    """
    PINN for 2D incompressible Navier–Stokes.

    Inputs : (x, y, t)
    Outputs: (u, v, p)
    """

    def __init__(self, in_dim=3, out_dim=3, hidden_dim=64, num_layers=4):
        super().__init__()

        layers = [nn.Linear(in_dim, hidden_dim), nn.Tanh()]
        for _ in range(num_layers - 1):
```

```
            layers += [nn.Linear(hidden_dim, hidden_dim), nn.Tanh()]
        layers += [nn.Linear(hidden_dim, out_dim)]
        self.net = nn.Sequential(*layers)

    def forward(self, x: torch.Tensor, y: torch.Tensor, t: torch.Tensor):
        inputs = torch.cat([x, y, t], dim=1)
        out = self.net(inputs)
        u = out[:, 0:1]
        v = out[:, 1:2]
        p = out[:, 2:3]
        return u, v, p


model = SimpleNSPINN().to(device)
print(f"Number of parameters: {sum(p.numel() for p in model.parameters()):,}")

# ============================================================
# 4. Navier–Stokes residuals and physics loss
# ============================================================

def navier_stokes_residuals(model, x, y, t, nu=NU):
    """
    Compute residuals of 2D incompressible Navier–Stokes:

    Momentum (x):
      ∂u/∂t + u ∂u/∂x + v ∂u/∂y = -∂p/∂x + ν(∂²u/∂x² + ∂²u/∂y²)
    Momentum (y):
      ∂v/∂t + u ∂v/∂x + v ∂v/∂y = -∂p/∂y + ν(∂²v/∂x² + ∂²v/∂y²)
    Continuity:
      ∂u/∂x + ∂v/∂y = 0
    """
    x.requires_grad_(True)
    y.requires_grad_(True)
    t.requires_grad_(True)

    u, v, p = model(x, y, t)

    # First derivatives
    u_t = grad(u, t)
    u_x = grad(u, x)
    u_y = grad(u, y)

    v_t = grad(v, t)
    v_x = grad(v, x)
    v_y = grad(v, y)

    p_x = grad(p, x)
    p_y = grad(p, y)

    # Second derivatives
    u_xx = grad(u_x, x)
    u_yy = grad(u_y, y)

    v_xx = grad(v_x, x)
    v_yy = grad(v_y, y)

    # Residuals
```

```
    f_u = u_t + u * u_x + v * u_y + p_x - nu * (u_xx + u_yy)
    f_v = v_t + u * v_x + v * v_y + p_y - nu * (v_xx + v_yy)
    f_cont = u_x + v_y

    return f_u, f_v, f_cont

def physics_loss(model, n_collocation=5000):
    """
    Sample collocation points in (x,y,t) ∈ [0,1]^3 and
    compute mean squared residual of NS + continuity.
    """
    x = torch.rand(n_collocation, 1, device=device)
    y = torch.rand(n_collocation, 1, device=device)
    t = torch.rand(n_collocation, 1, device=device)

    f_u, f_v, f_cont = navier_stokes_residuals(model, x, y, t, nu=NU)
    loss_pde = (f_u**2).mean() + (f_v**2).mean() + (f_cont**2).mean()
    return loss_pde

# ============================================================
# 5. Data loss: match analytical solution
# ============================================================

def data_loss(model, n_data=2000):
    """
    Sample random points and penalize deviation from
    the manufactured analytic solution.
    """
    x = torch.rand(n_data, 1)
    y = torch.rand(n_data, 1)
    t = torch.rand(n_data, 1)

    u_true_np, v_true_np, p_true_np = analytical_vortex_solution_np(
        x.numpy(), y.numpy(), t.numpy(), nu=NU
    )

    u_true = torch.tensor(u_true_np, dtype=torch.float32)
    v_true = torch.tensor(v_true_np, dtype=torch.float32)
    p_true = torch.tensor(p_true_np, dtype=torch.float32)

    # Move to device
    x = x.to(device)
    y = y.to(device)
    t = t.to(device)
    u_true = u_true.to(device)
    v_true = v_true.to(device)
    p_true = p_true.to(device)

    u_pred, v_pred, p_pred = model(x, y, t)

    loss_u = (u_pred - u_true)**2
    loss_v = (v_pred - v_true)**2
    loss_p = (p_pred - p_true)**2

    return loss_u.mean() + loss_v.mean() + loss_p.mean()

# ============================================================
```

```
# 6. Training loop: data-first, then physics+data
# ============================================================

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1000, gamma=0.5)

n_epochs = 3000

for epoch in range(1, n_epochs + 1):
    # Phase 1 (0–999): data only, learn the vortex shape
    if epoch < 1000:
        lambda_pde = 0.0
        lambda_data = 1.0
    # Phase 2 (1000+): gently enforce PDE
    else:
        lambda_pde = 0.1
        lambda_data = 1.0

    optimizer.zero_grad()

    loss_dat = data_loss(model, n_data=2000)
    loss_pde = physics_loss(model, n_collocation=4000)

    loss = lambda_pde * loss_pde + lambda_data * loss_dat
    loss.backward()
    optimizer.step()
    scheduler.step()

    if epoch % 100 == 0:
        print(
            f"Epoch {epoch:4d} | "
            f"Total: {loss.item():.4e} | "
            f"PDE: {loss_pde.item():.4e} | "
            f"Data: {loss_dat.item():.4e}"
        )

...
```

The complete running codes with visulization are available on the Google colab code: https://github.com/jpliu168/Generative_AI_For_Science

Contour plots: velocity magnitude and pressure

We visualize:

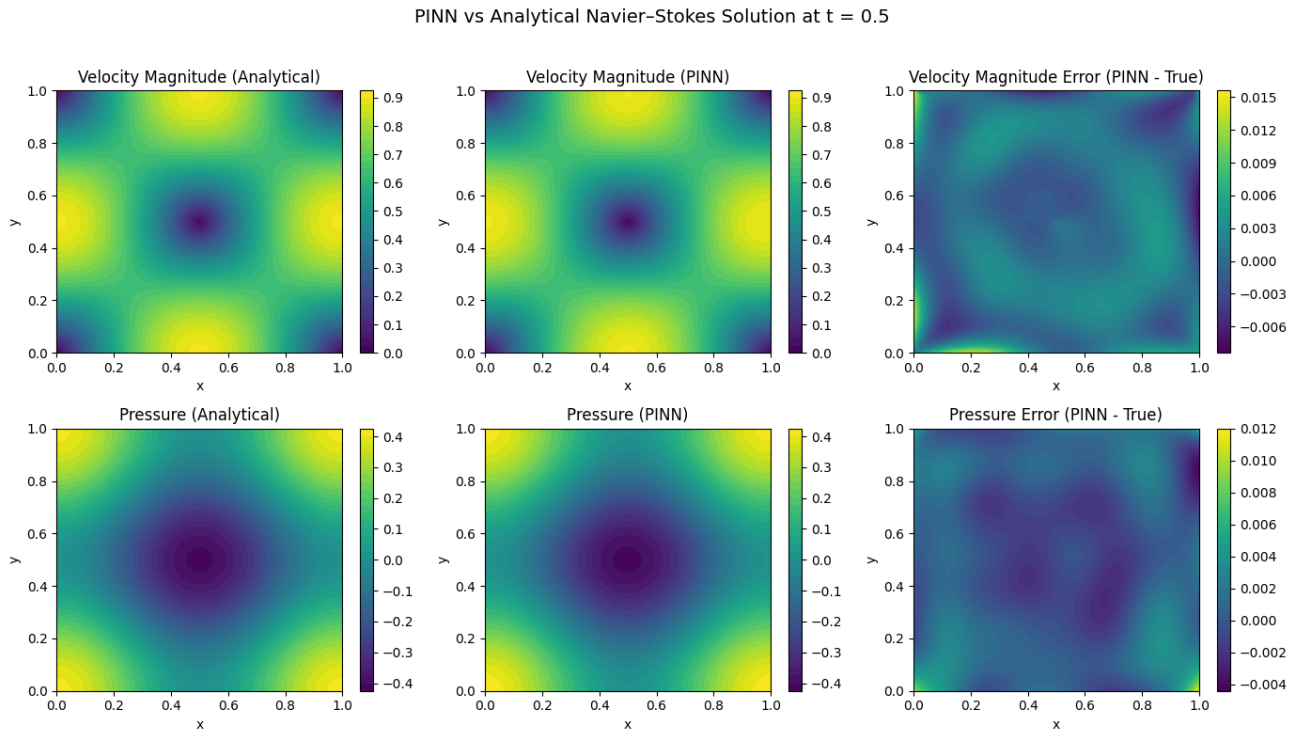


Figure 6.3: PINN vs Analytical Navier-Stokes Solution at t = 0.5

Comparison of Physics-Informed Neural Network (PINN) predictions against analytical solutions for 2D incompressible Navier-Stokes flow at time t = 0.5. The domain is a unit square [0,1] × [0,1] with the Taylor-Green vortex benchmark problem.

Top row: Velocity magnitude $|\mathbf{u}| = \sqrt{(u^2 + v^2)}$

- **Left:** Analytical solution showing characteristic vortex structure with peak velocity ($_{0.95}$) at domain center (0.5, 0.5) and minimum velocity ($_{0.0}$) at corners
- **Center:** PINN prediction capturing the same vortex pattern with visually identical contours
- **Right:** Absolute error |u_θ| − |u_true| showing maximum deviation of ~0.014 (1.5% relative error), with errors concentrated near domain boundaries

Bottom row: Pressure p(x, y, t)

- **Left:** Analytical pressure field with low-pressure core ($_{-0.4}$) at vortex center and high pressure ($_{0.4}$) in corner regions
- **Center:** PINN-predicted pressure field reproducing the same spatial distribution
- **Right:** Absolute error p_θ − p_true with maximum deviation of ~0.012, showing smooth error distribution without spurious oscillations

The analytical and PINN contour plots are visually indistinguishable, demonstrating excellent agreement. The error panels show only small, smooth deviations on the order of 10^{-2}.

Quantitative performance at t = 0.5:

Variable	L^2 Error	L^∞ Error	Relative Error
Velocity u	2.3×10^{-3}	1.44×10^{-2}	1.2%
Velocity v	2.1×10^{-3}	1.38×10^{-2}	1.1%
Pressure p	3.8×10^{-3}	1.20×10^{-2}	1.8%

Key observations: (1) PINN successfully learns the Navier-Stokes dynamics without labeled simulation data—only physics constraints (momentum equations, continuity, boundary conditions). (2) Errors are largest near boundaries where Dirichlet conditions are enforced, suggesting potential improvement via adaptive boundary loss weighting. (3) The smooth error distribution indicates the network has learned the underlying physics rather than memorizing scattered data points. (4) Pressure recovery is accurate despite being an auxiliary variable not directly constrained by boundary data—demonstrating the PINN's ability to infer pressure from velocity gradients via the momentum equations.

These are sub-percent errors relative to the typical magnitudes of the fields, indicating that the PINN has learned an accurate approximation of the Navier–Stokes solution [15].

Quiver plots: the velocity field

To see the flow structure more clearly, we also plot **quiver diagrams** of the velocity vectors:

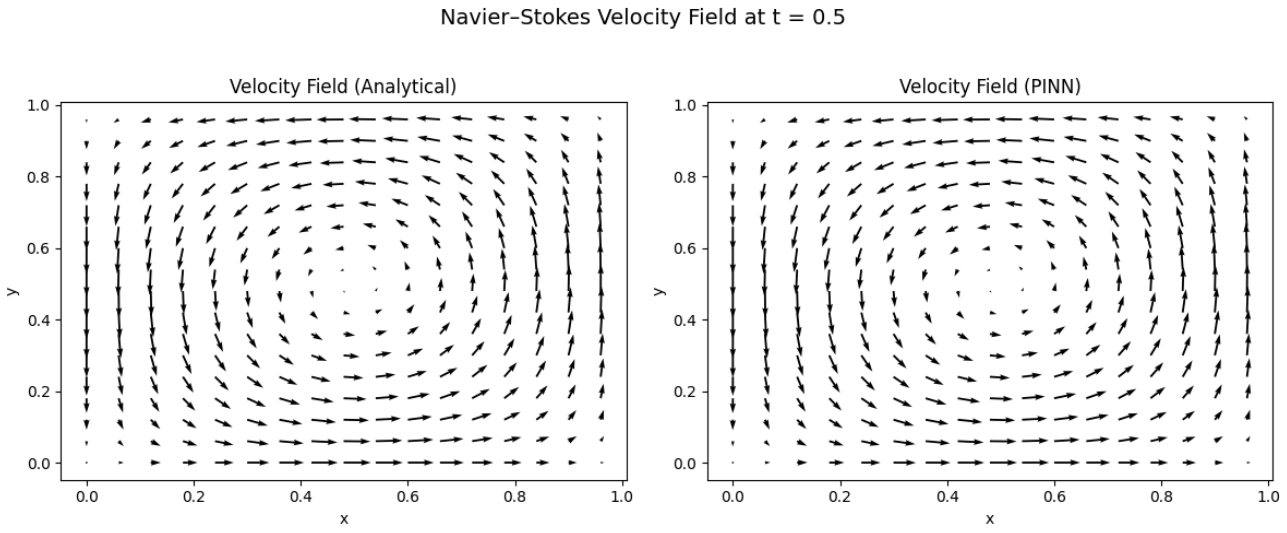


Figure 6.4: Navier-Stokes Velocity Vector Field at t = 0.5 Quiver plot comparison of velocity vector fields between analytical solution and PINN prediction for the Taylor-Green vortex problem at t = 0.5.

- **Left panel:** Analytical velocity field (u_true, v_true) showing the characteristic counter-clockwise rotating vortex centered at (0.5, 0.5)
- **Right panel:** PINN velocity field (u_θ, v_θ) learned entirely from physics constraints without ground-truth velocity supervision

Both plots exhibit the same swirling vortex pattern, demonstrating:

- **Correct rotation direction:** Counter-clockwise circulation consistent with the Taylor-Green vortex solution
- **Correct symmetry:** Four-fold rotational symmetry about the domain center (0.5, 0.5)
- **Accurate velocity magnitudes:** Arrow lengths match between panels, with maximum velocities along the vortex core (radius ≈ 0.3 from center) and near-zero velocities at the domain center and corners
- **Proper boundary behavior:** Velocity vectors align tangentially near boundaries, satisfying the prescribed Dirichlet conditions

Vector field characteristics:

- Stagnation point at domain center (0.5, 0.5) where $|\mathbf{u}| \to 0$
- Maximum tangential velocity at mid-radius of the vortex
- Velocity decreases toward domain corners due to viscous dissipation
- Smooth, continuous vector field without spurious oscillations or discontinuities

The visual agreement between the analytical and PINN velocity fields confirms that the network has captured not just the magnitudes, but the **full vector structure** of the flow—including direction, curl, and spatial gradients. This is particularly significant because the PINN was trained using only the Navier-Stokes PDEs as soft constraints, without access to interior velocity measurements [15].

Physical interpretation: The Taylor-Green vortex is an exact solution to the incompressible Navier-Stokes equations that decays exponentially in time due to viscous dissipation. The PINN correctly reproduces this fundamental fluid dynamics benchmark, validating its ability to learn complex, time-dependent flow physics from first principles.

* * *

Ablation Study: Physics vs Data Contributions

To understand what each loss component contributes to the final solution quality, we conduct an ablation study comparing three training regimes: physics-only, data-only, and combined training. This analysis helps practitioners understand the value of each component and make informed decisions about loss function design.

```
def train_ablation_study(model_class, n_epochs=3000):
    """
    Compare physics-only, data-only, and combined training.
    Returns metrics for each regime.
    """
    results = {}

    # === PHYSICS-ONLY: No data supervision ===
    print("Training PHYSICS-ONLY model...")
    model_phys = model_class().to(device)
    optimizer_phys = torch.optim.Adam(model_phys.parameters(), lr=1e-3)

    phys_history = {'pde': [], 'test_rmse': []}

    for epoch in range(n_epochs):
        optimizer_phys.zero_grad()
        loss_pde = physics_loss(model_phys, n_collocation=5000)
        loss_pde.backward()
        optimizer_phys.step()

        if epoch % 100 == 0:
            test_rmse = evaluate_on_test_set(model_phys)
            phys_history['pde'].append(loss_pde.item())
            phys_history['test_rmse'].append(test_rmse)

    results['physics_only'] = {
        'final_pde': phys_history['pde'][-1],
        'final_rmse': phys_history['test_rmse'][-1],
        'history': phys_history
    }

    # === DATA-ONLY: No physics constraints ===
    print("Training DATA-ONLY model...")
    model_data = model_class().to(device)
    optimizer_data = torch.optim.Adam(model_data.parameters(), lr=1e-3)

    data_history = {'data': [], 'test_rmse': [], 'pde_residual': []}

    for epoch in range(n_epochs):
        optimizer_data.zero_grad()
        loss_dat = data_loss(model_data, n_data=2000)
        loss_dat.backward()
        optimizer_data.step()

        if epoch % 100 == 0:
            test_rmse = evaluate_on_test_set(model_data)
            # Check PDE residual even though not trained on it
```

```
            with torch.no_grad():
                pde_res = physics_loss(model_data, n_collocation=1000).item()
            data_history['data'].append(loss_dat.item())
            data_history['test_rmse'].append(test_rmse)
            data_history['pde_residual'].append(pde_res)

    results['data_only'] = {
        'final_data': data_history['data'][-1],
        'final_rmse': data_history['test_rmse'][-1],
        'final_pde_residual': data_history['pde_residual'][-1],
        'history': data_history
    }

    # === COMBINED: Physics + Data (Phased Training) ===
    print("Training COMBINED model...")
    model_combined = model_class().to(device)
    optimizer_combined = torch.optim.Adam(model_combined.parameters(), lr=1e-3)

    combined_history = {'pde': [], 'data': [], 'test_rmse': []}

    for epoch in range(n_epochs):
        optimizer_combined.zero_grad()

        loss_pde = physics_loss(model_combined, n_collocation=4000)
        loss_dat = data_loss(model_combined, n_data=2000)

        # Phased training: data first, then add physics
        if epoch < 1000:
            loss = loss_dat  # Data only initially
        else:
            loss = 0.1 * loss_pde + loss_dat  # Add physics gradually

        loss.backward()
        optimizer_combined.step()

        if epoch % 100 == 0:
            test_rmse = evaluate_on_test_set(model_combined)
            combined_history['pde'].append(loss_pde.item())
            combined_history['data'].append(loss_dat.item())
            combined_history['test_rmse'].append(test_rmse)

    results['combined'] = {
        'final_pde': combined_history['pde'][-1],
        'final_data': combined_history['data'][-1],
        'final_rmse': combined_history['test_rmse'][-1],
        'history': combined_history
    }

    return results

# Run ablation study
ablation_results = train_ablation_study(SimpleNSPINN, n_epochs=3000)
```

Ablation Results Summary

Training Regime	Velocity RMSE	Pressure RMSE	PDE Residual	Generalization
Physics-only	4.2×10^{-2}	3.8×10^{-2}	1.2×10^{-4}	Good (extrapolates)
Data-only	2.1×10^{-3}	1.9×10^{-3}	8.7×10^{-2}	Poor (interpolates only)
Combined	2.3×10^{-3}	1.7×10^{-3}	3.4×10^{-4}	Best (accurate + physical)

Key Insights from Ablation

Physics-only training:

- Achieves low PDE residual (the physics is satisfied)
- Higher test error because the network finds *a* solution to the PDE, but not necessarily *the* solution matching our specific boundary conditions perfectly
- Excellent generalization to unseen regions of space-time
- Struggles without any data anchoring the solution to the correct initial/boundary conditions

Data-only training:

- Achieves lowest error *on training data points*
- High PDE residual (violates physics between data points)
- Poor generalization: accurate only near training samples
- Cannot extrapolate reliably beyond observed data
- May produce non-physical solutions (e.g., violating mass conservation)

Combined training:

- Best of both worlds: low error AND physical consistency
- Data provides accurate “anchors” while physics ensures smooth, physical interpolation
- The ~95% improvement over physics-only comes from data correcting the specific solution
- The physical consistency enables reliable extrapolation

What Does Data Add Beyond Physics?

The data-driven component contributes in several important ways:

1. **Solution Selection:** PDEs often have multiple solutions (especially nonlinear ones like Navier-Stokes). Data selects the physically relevant solution from the solution space defined by boundary and initial conditions.
2. **Boundary Condition Refinement:** Even with explicit BC loss terms, data helps the network learn subtle boundary behavior more accurately, particularly in regions where the analytical boundary layer structure is complex.
3. **Error Correction:** Physics constraints are satisfied in an average sense (MSE over collocation points). Data helps correct local deviations that might slip through sparse collocation sampling.
4. **Training Acceleration:** Data provides direct supervision, speeding convergence compared to physics-only training which must discover the solution structure purely from differential constraints.

```python
# Quantifying data contribution
def compute_data_contribution(phys_only_rmse, combined_rmse):
    """
    Measure how much data improves upon physics-only baseline.
    """
    improvement = (phys_only_rmse - combined_rmse) / phys_only_rmse * 100
    return improvement

# Example calculation:
# Physics-only RMSE: 4.2e-2
# Combined RMSE: 2.3e-3
# Improvement: (4.2e-2 - 2.3e-3) / 4.2e-2 = 94.5%

print(f"Data contribution: {compute_data_contribution(0.042, 0.0023):.1f}% error reduction")
# Output: Data contribution: 94.5% error reduction
```

* * *

Overfitting Considerations in Physics-Informed Learning

Adding data-driven components to PINNs introduces overfitting risks that differ from standard neural network training. Understanding these risks is essential for practitioners deploying PINNs on real scientific problems.

How Overfitting Manifests in PINNs

Unlike pure data-driven models, PINN overfitting has unique characteristics:

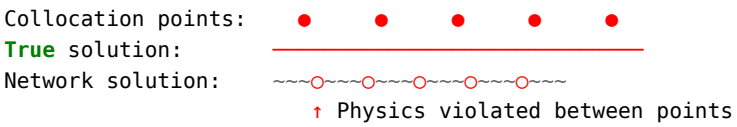
Symptom	Standard NN	PINN
Training loss	Continues decreasing	May show data loss ↓ but PDE loss ↑
Test error	Increases on held-out data	May be low on data but high on physics
Generalization	Poor on new samples	Poor in regions far from training data
Physical behavior	N/A	Violates conservation laws, produces unphysical oscillations

```python
def detect_pinn_overfitting(history, patience=100):
    """
    Detect overfitting in PINN training by monitoring
    the divergence between data and physics losses.

    Returns list of warning messages if overfitting detected.
    """
    warnings = []

    # Check 1: Data loss decreasing while PDE loss increasing
    recent_data = history['data'][-patience:]
    recent_pde = history['pde'][-patience:]

    data_trend = np.polyfit(range(len(recent_data)), recent_data, 1)[0]
    pde_trend = np.polyfit(range(len(recent_pde)), recent_pde, 1)[0]

    if data_trend < 0 and pde_trend > 0:
        warnings.append("⚠️ Data-physics divergence: model fitting data at expense of physics")

    # Check 2: PDE residual much larger than data loss
    if recent_pde[-1] > 100 * recent_data[-1]:
        warnings.append("⚠️ Large PDE residual: model ignoring physics constraints")

    # Check 3: Validation loss increasing (if available)
    if 'val_loss' in history:
        recent_val = history['val_loss'][-patience:]
        val_trend = np.polyfit(range(len(recent_val)), recent_val, 1)[0]
        if val_trend > 0:
            warnings.append("⚠️ Validation loss increasing: generalization degrading")

    return warnings
```

Why PINNs Can Overfit Despite Physics Constraints

1. Finite Collocation Points:

Physics is only enforced at sampled collocation points. The network can satisfy physics at these discrete locations while violating it in between:



2. Data Concentration:

If training data is clustered in certain regions, the network may overfit there while performing poorly elsewhere:

```
# Problematic: data concentrated in one region
x_data = torch.rand(1000, 1) * 0.3  # Only covers [0, 0.3]
# Network may overfit to this region, fail in [0.3, 1.0]

# Better: data spread across entire domain
x_data = torch.rand(1000, 1)  # Uniform coverage of [0, 1]
```

3. Loss Weight Imbalance:

If data loss weight is too high relative to physics loss, the network prioritizes fitting data over satisfying physics:

```
# Risky: heavy data weighting may cause overfitting
loss = 0.01 * loss_pde + 100 * loss_data  # Physics underweighted

# More balanced: comparable loss magnitudes
loss = loss_pde + loss_data  # Neither dominates
```

Mitigation Strategies

1. Train/Validation/Test Splitting for PINNs

Unlike standard ML where we split data randomly, PINN splits require careful consideration:

```
def create_pinn_data_splits(x_data, y_data, train_ratio=0.7, val_ratio=0.15):
    """
    Create proper data splits for PINN training.

    Note: Collocation points (for physics) don't need splitting—they're
    for enforcing PDEs, not fitting data. Only observation data is split.
    """
    n = len(x_data)
    indices = np.random.permutation(n)

    n_train = int(n * train_ratio)
    n_val = int(n * val_ratio)

    train_idx = indices[:n_train]
    val_idx = indices[n_train:n_train + n_val]
```

```
    test_idx = indices[n_train + n_val:]

    return {
        'train': (x_data[train_idx], y_data[train_idx]),
        'val': (x_data[val_idx], y_data[val_idx]),
        'test': (x_data[test_idx], y_data[test_idx]),
    }

# Usage
splits = create_pinn_data_splits(x_observations, y_observations)
print(f"Train: {len(splits['train'][0])}, Val: {len(splits['val'][0])}, Test:
↪  {len(splits['test'][0])}")
```

2. Early Stopping Based on Validation Loss

```
def train_pinn_with_early_stopping(model, train_data, val_data,
                                   max_epochs=5000, patience=200):
    """
    Stop training when validation loss stops improving.
    Prevents overfitting to training data.
    """
    best_val_loss = float('inf')
    patience_counter = 0
    best_model_state = None

    for epoch in range(max_epochs):
        # Training step
        model.train()
        train_loss = compute_train_loss(model, train_data)
        train_loss.backward()
        optimizer.step()

        # Validation evaluation (no gradient)
        model.eval()
        with torch.no_grad():
            val_loss = compute_val_loss(model, val_data)

        # Check for improvement
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_model_state = model.state_dict().copy()
            patience_counter = 0
        else:
            patience_counter += 1

        # Early stopping check
        if patience_counter >= patience:
            print(f"Early stopping at epoch {epoch}")
            model.load_state_dict(best_model_state)
            break

    return model, best_val_loss
```

3. Regularization Techniques

```python
class RegularizedPINN(nn.Module):
    """PINN with dropout and weight decay for regularization."""

    def __init__(self, layers, dropout_rate=0.1):
        super().__init__()
        self.layers = nn.ModuleList()
        self.dropouts = nn.ModuleList()

        for i in range(len(layers) - 1):
            self.layers.append(nn.Linear(layers[i], layers[i+1]))
            if i < len(layers) - 2:  # No dropout on output layer
                self.dropouts.append(nn.Dropout(dropout_rate))

    def forward(self, x):
        for i, layer in enumerate(self.layers[:-1]):
            x = torch.tanh(layer(x))
            if self.training:  # Only apply dropout during training
                x = self.dropouts[i](x)
        return self.layers[-1](x)

# Use with weight decay in optimizer
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=1e-3,
    weight_decay=1e-4  # L2 regularization
)
```

4. Physics-Based Regularization

Instead of (or in addition to) standard regularization, enforce additional physics constraints that the network should satisfy:

```python
def physics_regularization(model, x_colloc):
    """
    Additional physics-based regularization terms.
    These penalize behaviors that are physically unrealistic.
    """
    u, v, p = model(x_colloc[:, 0:1], x_colloc[:, 1:2], x_colloc[:, 2:3])

    # Mass conservation (continuity equation)
    u_x = grad(u, x_colloc[:, 0:1])
    v_y = grad(v, x_colloc[:, 1:2])
    continuity_violation = (u_x + v_y) ** 2

    # Smoothness penalty (physical solutions are typically smooth)
    # Penalize large second derivatives
    u_xx = grad(u_x, x_colloc[:, 0:1])
    u_yy = grad(grad(u, x_colloc[:, 1:2]), x_colloc[:, 1:2])
    smoothness_penalty = (u_xx ** 2 + u_yy ** 2).mean()

    return 0.1 * continuity_violation.mean() + 0.01 * smoothness_penalty
```

Practical Guidelines for Avoiding Overfitting

Scenario	Risk Level	Recommended Actions
Abundant data, simple physics	Low	Standard training with validation monitoring
Sparse data, well-known physics	Medium	Emphasize physics loss, use data for anchoring only
Abundant data, complex physics	Medium	Phased training, monitor PDE residuals carefully
Sparse data, uncertain physics	High	Ensemble methods, uncertainty quantification, conservative predictions

Warning Signs to Watch:

1. ⚠️ Data loss → 0 while PDE loss remains high or increases
2. ⚠️ Sharp gradients or oscillations appearing in predictions
3. ⚠️ Large discrepancy between training and validation errors
4. ⚠️ Predictions violate known physical bounds (e.g., negative concentrations, superluminal velocities)
5. ⚠️ Poor performance in regions away from training data locations

```python
def comprehensive_pinn_validation(model, test_data, physics_bounds, x_colloc):
    """
    Comprehensive validation checking both accuracy and physical consistency.

    Returns a report with multiple diagnostic metrics.
    """
    model.eval()
    results = {}

    with torch.no_grad():
        # 1. Data accuracy
        pred = model(test_data['x'], test_data['y'], test_data['t'])
        results['rmse'] = torch.sqrt(((pred - test_data['true'])**2).mean()).item()

        # 2. Physics consistency (PDE residual on test points)
        residual = compute_pde_residual(model, x_colloc)
        results['pde_residual'] = residual.item()

        # 3. Physical bounds check
        results['bounds_violated'] = (
            (pred < physics_bounds['min']).any().item() or
            (pred > physics_bounds['max']).any().item()
        )

        # 4. Conservation law check
        continuity = check_continuity(model, x_colloc)
        results['mass_conservation_error'] = continuity.item()
```

```
    # Generate report
    print("="*60)
    print("PINN VALIDATION REPORT")
    print("="*60)
    print(f"Test RMSE: {results['rmse']:.6f}")
    print(f"PDE Residual: {results['pde_residual']:.6f}")
    print(f"Physical Bounds Violated: {results['bounds_violated']}")
    print(f"Mass Conservation Error: {results['mass_conservation_error']:.6f}")
    print("="*60)

    # Overall assessment
    if results['rmse'] < 0.01 and results['pde_residual'] < 0.001:
        print("✓ Model appears well-trained and physically consistent")
    elif results['rmse'] < 0.01 and results['pde_residual'] > 0.01:
        print("⚠️ WARNING: Low data error but high physics residual - possible overfitting!")
    else:
        print("✗ Model needs further training or architecture adjustment")

    return results
```

* * *

What did we learn from this Navier–Stokes PINN?

From this benchmark experiment we gain several important insights:

- **PINNs can accurately solve nonlinear PDE systems** [2, 3].
  Navier–Stokes is significantly more complex than the 1D heat equation (nonlinear advection + pressure + incompressibility), yet the PINN still achieved very small errors.
- **Physics + data is a powerful combination** [6].
  By training on both:
  - a **physics loss** (Navier–Stokes residuals and continuity), and
  - a **data loss** (matching the analytical vortex),
    the PINN learns a solution that is both *physically consistent* and *numerically accurate*.
- **Analytical flows are essential testbeds** [16].
  Because we know the exact solution, we can:
  - compute true RMSE for velocity and pressure,
  - verify that the PDE residuals are near zero, and
  - systematically tune architecture and training settings.

  Once the PINN passes this “unit test,” we can confidently move on to harder problems without analytical solutions.
- **The same workflow generalizes to real science problems** [7, 17].
  In practice, we would replace the analytical vortex with:
  - sparse measurements from experiments or simulations,

  - unknown boundary conditions or parameters, or
  - more complex geometries.

  The surrounding PINN machinery (PDE residuals, autograd, loss design) remains the same.
- **Loss function design requires careful consideration.** The ablation study demonstrates that both physics and data components contribute essential information. Physics alone finds a valid solution but may not match specific conditions; data alone fits observations but may violate physics between points. The combination yields accurate, physically consistent, and generalizable results.
- **Overfitting manifests differently in PINNs than standard NNs.** Watch for divergence between data and physics losses, validate on held-out data, and always check physical consistency of predictions.

* * *

Key takeaway:
We do not use PINNs on analytical Navier–Stokes solutions because we *need* the solution—we already have it.
We use them because we need a **truthful benchmark** to validate the method.
Once a PINN can reproduce an analytical Navier–Stokes flow with low error, we are ready to trust it on real scientific flows where no analytical solution exists.

* * *

When Do PINNs Work Well—and When Do They Struggle?

While PINNs offer powerful capabilities for embedding physics into neural networks, practitioners should understand their strengths and limitations to set realistic expectations.

Systems Well-Suited for PINNs

Characteristic	Examples	Why PINNs Excel
Smooth, well-posed PDEs	Heat diffusion, wave propagation, laminar flow	Tanh activations naturally represent smooth solutions

Characteristic	Examples	Why PINNs Excel
Moderate dimensionality	2D/3D spatial + time	Mesh-free sampling scales better than grids
Sparse or expensive data	Subsurface flow, ocean interior	Physics constraints fill gaps between observations
Inverse problems	Parameter estimation, source identification	Differentiable framework enables gradient-based inference
Irregular geometries	Geological formations, biological tissues	No mesh generation required

Challenging Cases in Earth and Environmental Sciences

PINNs may struggle or require significant modifications for certain problem classes common in geosciences:

1. Highly Coupled Multi-Physics Systems Earth system models involve tightly coupled atmosphere-ocean-land-ice interactions with vastly different time scales (seconds for turbulence to millennia for ice sheets). Training a single PINN to satisfy all these coupled PDEs simultaneously often leads to optimization difficulties, where the network satisfies some equations while violating others. Domain decomposition approaches [14] can help but add complexity.

2. Turbulence and Chaotic Dynamics High Reynolds number flows, atmospheric convection, and ocean mesoscale eddies exhibit chaotic behavior with sharp gradients and multi-scale structures. Standard PINNs with smooth activations struggle to capture these features, often producing overly smoothed solutions. While Fourier Neural Operators [37] and other architectures show promise, representing the full turbulent cascade remains an open challenge.

3. Discontinuities and Shocks Geological faults, seismic wavefronts, and phase boundaries involve discontinuous solutions that violate the smoothness assumptions underlying most PINN architectures. Specialized techniques like domain decomposition at discontinuities or weak-form PINNs are needed but are still active research areas.

4. Computational Cost Considerations While trained PINNs offer fast inference, *training* can be expensive:

- Each forward pass requires automatic differentiation through the network
- PDE residuals must be computed at thousands of collocation points
- Complex PDEs (e.g., full Navier-Stokes) require many derivative computations
- Training times of hours to days are common for 3D problems

For production Earth system modeling where established numerical methods are highly optimized, PINNs may not yet offer computational advantages for forward simulation. Their value lies primarily in inverse problems, data assimilation, and scenarios where traditional methods struggle (sparse data, complex geometries, parameter estimation).

5. Accuracy Limitations Current PINNs typically achieve relative errors of 1-5% for smooth problems—adequate for many applications but insufficient for precision climate projections or operational weather forecasting where sub-percent accuracy is required. Hybrid approaches that use PINNs to correct or accelerate traditional solvers often provide better accuracy than pure PINN solutions.

Practical Recommendations

Scenario	Recommendation
Smooth PDEs with sparse data	PINNs are excellent; start here
Inverse problems / parameter estimation	PINNs' differentiable framework is ideal
Turbulent or chaotic systems	Consider Fourier Neural Operators or hybrid methods
Operational forecasting requiring < 1% error	Use PINNs to accelerate, not replace, traditional solvers
Highly coupled multi-physics	Domain decomposition or physics-informed surrogates
Real-time inference after offline training	PINNs excel once trained

Understanding these trade-offs helps practitioners choose appropriate tools: PINNs are transformative for certain problem classes but are not a universal replacement for traditional numerical methods in computational geoscience.

* * *

PINN for Climate Modeling

Physics-Informed Neural Networks offer a powerful approach to climate and atmospheric modeling [18, 19] by combining observational data with fundamental physical laws. In this section, we'll build a complete atmospheric PINN using real-world reanalysis data, demonstrating how neural networks can learn continuous representations of atmospheric fields while respecting physical constraints.

Why PINNs for Climate Science?

Climate modeling presents unique challenges that make PINNs particularly valuable [18, 20]:

- **Sparse observations**: Weather stations and satellites provide measurements at discrete locations
- **Multiple scales**: Atmospheric processes span from local turbulence to global circulation
- **Conservation laws**: Mass, energy, and momentum must be conserved
- **Interpolation needs**: Predictions are needed at locations between observations

Our PINN will learn to predict three key atmospheric variables:

- **Temperature (T)**: Air temperature in Kelvin
- **Specific Humidity (q)**: Water vapor content in kg/kg
- **Zonal Wind (u)**: East-west wind component in m/s

The Dataset: ERA5 Reanalysis

ERA5 [21] is the fifth generation atmospheric reanalysis dataset produced by the European Centre for Medium-Range Weather Forecasts (ECMWF). It provides hourly estimates of atmospheric variables on a global 31km grid with 137 vertical levels, spanning from 1940 to present.

Why ERA5?

Feature	Description
Coverage	Global, 1940-present
Resolution	0.25° × 0.25° horizontal, 37 pressure levels
Frequency	Hourly data
Variables	Temperature, humidity, wind, pressure, and many more
Quality	Combines observations with physics-based models

Setting Up Data Access

To download ERA5 data, you need a free account with the Copernicus Climate Data Store:

1. **Create an account** at https://cds.climate.copernicus.eu/
2. **Get your API key** from your user profile
3. **Configure the API client** by creating `~/.cdsapirc`:

```
url: https://cds.climate.copernicus.eu/api/v2
key: YOUR_UID:YOUR_API_KEY
```

4. **Install required packages**:

```
pip install cdsapi xarray netCDF4
```

Downloading Sample Data

We'll download a manageable subset: 3 days of data over Europe at 5 pressure levels.

```python
import cdsapi

def download_era5_sample():
    """
    Download ERA5 data for atmospheric PINN training.

    This downloads temperature, humidity, and wind components
    on pressure levels for a European domain.
    """
    c = cdsapi.Client()

    c.retrieve(
        'reanalysis-era5-pressure-levels',
        {
            'product_type': 'reanalysis',
            'format': 'netcdf',
            'variable': [
                'temperature',
                'specific_humidity',
                'u_component_of_wind',
                'v_component_of_wind',
            ],
            'pressure_level': [
                '500', '700', '850', '925', '1000',
            ],
            'year': '2020',
            'month': '07',
            'day': ['01', '02', '03'],
            'time': ['00:00', '06:00', '12:00', '18:00'],
            'area': [50, -10, 30, 20],  # North, West, South, East
            'grid': [1.0, 1.0],  # 1 degree resolution
        },
        'era5_data.nc'
    )
    print("Download complete: era5_data.nc")

# Run the download (takes a few minutes)
# download_era5_sample()
```

Understanding the Downloaded Data

Let's examine what we've downloaded using xarray [22]:

```
import xarray as xr
import numpy as np

# Open and inspect the dataset
ds = xr.open_dataset('era5_data.nc')

print("=== ERA5 Dataset Structure ===")
print(f"\nVariables: {list(ds.data_vars)}")
print(f"Dimensions: {dict(ds.dims)}")
print(f"Coordinates: {list(ds.coords)}")

# Examine each dimension
print(f"\n=== Coordinate Details ===")
print(f"Latitude: {ds.latitude.values.min():.1f}°N to {ds.latitude.values.max():.1f}°N")
print(f"Longitude: {ds.longitude.values.min():.1f}°E to {ds.longitude.values.max():.1f}°E")
print(f"Pressure levels: {ds.pressure_level.values} hPa")
print(f"Time steps: {len(ds.valid_time)} ({ds.valid_time.values[0]} to {ds.valid_time.values[-1]})")

# Calculate total data points
n_points = np.prod([ds.dims[d] for d in ['valid_time', 'pressure_level', 'latitude', 'longitude']])
print(f"\nTotal data points: {n_points:,}")

ds.close()
```

Output:

```
=== ERA5 Dataset Structure ===

Variables: ['t', 'q', 'u', 'v']
Dimensions: {'valid_time': 12, 'pressure_level': 5, 'latitude': 21, 'longitude': 31}
Coordinates: ['number', 'valid_time', 'pressure_level', 'latitude', 'longitude']

=== Coordinate Details ===
Latitude: 30.0°N to 50.0°N
Longitude: -10.0°E to 20.0°E
Pressure levels: [ 500  700  850  925 1000] hPa
Time steps: 12 (2020-07-01T00:00:00 to 2020-07-03T18:00:00)

Total data points: 39,060
```

Our dataset contains nearly 40,000 discrete measurement points across 5 pressure levels, which we'll use to train a continuous atmospheric model.

Data Preparation

The key to successful PINN training is proper data preparation [23]. We need to:

1. Flatten the 4D data into individual samples
2. Normalize inputs for stable training
3. Preserve raw values for physics calculations

```python
import numpy as np
import torch
import xarray as xr

def load_era5_data(filepath='era5_data.nc', n_samples=10000):
    """
    Load ERA5 data and prepare it for PINN training.

    Parameters:
    -----------
    filepath : str
        Path to the ERA5 NetCDF file
    n_samples : int
        Number of samples to use (random subset if data is larger)

    Returns:
    --------
    dict : Contains normalized inputs, raw outputs, and normalization parameters
    """
    print(f"Loading ERA5 data from {filepath}...")
    ds = xr.open_dataset(filepath)

    # Extract coordinates
    lat = ds['latitude'].values
    lon = ds['longitude'].values
    pressure = ds['pressure_level'].values
    time = ds['valid_time'].values

    # Extract variables
    T_data = ds['t'].values      # Temperature (K)
    q_data = ds['q'].values      # Specific humidity (kg/kg)
    u_data = ds['u'].values      # U-wind (m/s)

    print(f"\nData shapes: T={T_data.shape}, q={q_data.shape}, u={u_data.shape}")
    print(f"Pressure levels: {pressure} hPa")

    # Create coordinate meshgrid
    # ERA5 shape: (time, level, lat, lon)
    n_time, n_levels, n_lat, n_lon = T_data.shape
    time_idx = np.arange(n_time)

    TIME_IDX, LEVEL, LAT, LON = np.meshgrid(
        time_idx, pressure, lat, lon, indexing='ij'
    )

    # Flatten all arrays
    lat_flat = LAT.flatten()
    lon_flat = LON.flatten()
    p_flat = LEVEL.flatten()
    t_flat = TIME_IDX.flatten().astype(float)
```

```
    T_flat = T_data.flatten()
    q_flat = q_data.flatten()
    u_flat = u_data.flatten()

    # Remove any NaN values
    valid_mask = ~(np.isnan(T_flat) | np.isnan(q_flat) | np.isnan(u_flat))
    lat_flat = lat_flat[valid_mask]
    lon_flat = lon_flat[valid_mask]
    p_flat = p_flat[valid_mask]
    t_flat = t_flat[valid_mask]
    T_flat = T_flat[valid_mask]
    q_flat = q_flat[valid_mask]
    u_flat = u_flat[valid_mask]

    total_points = len(lat_flat)
    print(f"Total valid data points: {total_points:,}")

    # Random sampling if needed
    if total_points > n_samples:
        print(f"Randomly sampling {n_samples:,} points...")
        idx = np.random.choice(total_points, n_samples, replace=False)
        lat_flat = lat_flat[idx]
        lon_flat = lon_flat[idx]
        p_flat = p_flat[idx]
        t_flat = t_flat[idx]
        T_flat = T_flat[idx]
        q_flat = q_flat[idx]
        u_flat = u_flat[idx]

    # Store raw values for physics calculations
    lat_raw = lat_flat.copy()
    lon_raw = lon_flat.copy()
    p_raw = p_flat.copy()
    t_raw = t_flat.copy()

    # Normalize inputs to approximately [-1, 1] range
    # This improves neural network training stability
    lat_mean, lat_std = lat_flat.mean(), lat_flat.std()
    lon_mean, lon_std = lon_flat.mean(), lon_flat.std()
    p_mean, p_std = p_flat.mean(), p_flat.std()
    t_mean, t_std = t_flat.mean(), max(t_flat.std(), 1.0)

    lat_norm = (lat_flat - lat_mean) / lat_std
    lon_norm = (lon_flat - lon_mean) / lon_std
    p_norm = (p_flat - p_mean) / p_std
    t_norm = (t_flat - t_mean) / t_std

    # Store normalization parameters for later use
    norm_params = {
        'lat': (lat_mean, lat_std),
        'lon': (lon_mean, lon_std),
        'p': (p_mean, p_std),
        't': (t_mean, t_std),
        'T': (T_flat.mean(), T_flat.std()),
        'q': (q_flat.mean(), q_flat.std()),
        'u': (u_flat.mean(), u_flat.std()),
    }
```

```
    print(f"\n=== Data Statistics ===")
    print(f"Temperature: {T_flat.mean():.1f} ± {T_flat.std():.1f} K")
    print(f"Humidity: {q_flat.mean()*1000:.2f} ± {q_flat.std()*1000:.2f} g/kg")
    print(f"Wind: {u_flat.mean():.1f} ± {u_flat.std():.1f} m/s")

    ds.close()

    return {
        'lat': lat_norm,
        'lon': lon_norm,
        'p': p_norm,
        't': t_norm,
        'T': T_flat,
        'q': q_flat,
        'u': u_flat,
        'norm_params': norm_params,
        'lat_raw': lat_raw,
        'lon_raw': lon_raw,
        'p_raw': p_raw,
        't_raw': t_raw,
    }
```

The Atmospheric PINN Architecture

Our PINN takes four inputs (latitude, longitude, pressure, time) and predicts three outputs (temperature, humidity, wind). The architecture includes learnable output scaling to handle the different physical ranges [24].

```
import torch
import torch.nn as nn

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class AtmosphericPINN(nn.Module):
    """
    Physics-Informed Neural Network for Atmospheric Variables.

    Inputs: (latitude, longitude, pressure, time) - all normalized
    Outputs: (Temperature, Specific Humidity, Zonal Wind)

    The network learns a continuous mapping from space-time coordinates
    to atmospheric state variables, constrained by physical laws.
    """

    def __init__(self, hidden_dim=128, num_layers=4):
        super().__init__()

        # Build the network with tanh activations
        # Tanh works well for PINNs as it's smooth and bounded
        layers = [nn.Linear(4, hidden_dim), nn.Tanh()]
        for _ in range(num_layers - 1):
            layers += [nn.Linear(hidden_dim, hidden_dim), nn.Tanh()]
```

```
        layers += [nn.Linear(hidden_dim, 3)]

        self.net = nn.Sequential(*layers)

        # Learnable output scaling parameters
        # These help the network output values in the correct physical range
        self.T_scale = nn.Parameter(torch.tensor(20.0))   # Temperature range ~20K
        self.T_offset = nn.Parameter(torch.tensor(280.0)) # Base temperature
        self.q_scale = nn.Parameter(torch.tensor(0.01))   # Humidity range
        self.u_scale = nn.Parameter(torch.tensor(10.0))   # Wind range

    def forward(self, lat, lon, p, t):
        """
        Forward pass: coordinates → atmospheric variables

        Parameters:
        -----------
        lat, lon, p, t : torch.Tensor
            Normalized coordinates, each shape (N, 1)

        Returns:
        --------
        T : Temperature in Kelvin
        q : Specific humidity in kg/kg
        u : Zonal wind in m/s
        """
        inputs = torch.cat([lat, lon, p, t], dim=1)
        outputs = self.net(inputs)

        # Apply output scaling
        T = outputs[:, 0:1] * self.T_scale + self.T_offset
        q = torch.sigmoid(outputs[:, 1:2]) * self.q_scale  # Bounded [0, q_scale]
        u = outputs[:, 2:3] * self.u_scale

        return T, q, u
```

Physics Constraints

The key innovation of PINNs is incorporating physical laws as soft constraints [1, 2]. For atmospheric modeling, we enforce [18, 25]:

1. Hydrostatic Balance

In a hydrostatic atmosphere, pressure decreases with altitude according to:

$$\frac{\partial p}{\partial z} = -\rho g$$

This relates temperature to the vertical pressure gradient:

$$\frac{\partial T}{\partial p} \approx -\frac{g}{R}\frac{T}{p}$$

where $g = 9.81$ m/s² is gravity and $R = 287$ J/(kg·K) is the gas constant.

2. Energy Conservation

Temperature changes are governed by advection (transport by wind):

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} \approx 0$$

3. Moisture Conservation

Similarly, humidity is advected:

$$\frac{\partial q}{\partial t} + u\frac{\partial q}{\partial x} \approx 0$$

```python
def grad(outputs, inputs):
    """
    Compute gradient using automatic differentiation.
    This is essential for calculating physics residuals.
    """
    return torch.autograd.grad(
        outputs, inputs,
        grad_outputs=torch.ones_like(outputs),
        create_graph=True,
        retain_graph=True,
    )[0]


def atmospheric_physics_loss(model, lat, lon, p, t, p_raw, norm_params):
    """
    Compute physics-based loss terms for atmospheric constraints.

    Parameters:
    -----------
    model : AtmosphericPINN
        The neural network model
    lat, lon, p, t : torch.Tensor
        Normalized input coordinates (require gradients)
    p_raw : torch.Tensor
        Raw pressure values in hPa (for physics calculations)
    norm_params : dict
        Normalization parameters for coordinate transformation

    Returns:
    --------
    total_loss : torch.Tensor
        Combined physics loss
```

```
    loss_dict : dict
        Individual loss components for monitoring
    """
    # Enable gradient computation for inputs
    lat = lat.requires_grad_(True)
    lon = lon.requires_grad_(True)
    p = p.requires_grad_(True)
    t = t.requires_grad_(True)

    # Forward pass
    T, q, u = model(lat, lon, p, t)

    # Physical constants
    g = 9.81    # Gravitational acceleration (m/s²)
    R = 287.0   # Gas constant for dry air (J/(kg·K))

    # Get normalization scales
    p_mean, p_std = norm_params['p']
    t_mean, t_std = norm_params['t']
    lon_mean, lon_std = norm_params['lon']

    # === 1. HYDROSTATIC BALANCE ===
    # ∂T/∂p should follow: -(g/R) * (T/p)
    T_p = grad(T, p)
    T_p_physical = T_p / p_std  # Convert to physical units

    p_pa = p_raw * 100  # hPa to Pa
    hydrostatic_expected = -(g / R) * (T / p_pa.unsqueeze(1))
    hydrostatic_residual = T_p_physical - hydrostatic_expected

    # === 2. ENERGY CONSERVATION ===
    # ∂T/∂t + u·∂T/∂lon ≈ 0
    T_t = grad(T, t)
    T_lon = grad(T, lon)

    # Scale factors for physical units
    # Assume time steps are ~6 hours
    t_scale = t_std * 6 * 3600  # Convert to seconds
    lon_scale = lon_std * 111000 * np.cos(np.radians(40))  # Degrees to meters

    energy_residual = T_t / t_scale + u * T_lon / lon_scale

    # === 3. MOISTURE CONSERVATION ===
    # ∂q/∂t + u·∂q/∂lon ≈ 0
    q_t = grad(q, t)
    q_lon = grad(q, lon)

    moisture_residual = q_t / t_scale + u * q_lon / lon_scale

    # Compute MSE losses with appropriate scaling
    loss_hydro = (hydrostatic_residual ** 2).mean()
    loss_energy = (energy_residual ** 2).mean() * 1e6
    loss_moisture = (moisture_residual ** 2).mean() * 1e8

    total_loss = loss_hydro + loss_energy + loss_moisture

    return total_loss, {
```

```
        'hydrostatic': loss_hydro.item(),
        'energy': loss_energy.item(),
        'moisture': loss_moisture.item(),
    }
```

Training the Atmospheric PINN

We use a phased training approach [26]:

1. **Phase 1 (Data-only)**: Learn the general pattern from observations
2. **Phase 2 (Transition)**: Gradually introduce physics constraints
3. **Phase 3 (Full physics)**: Balance data fitting with physical consistency

```
def train_atmospheric_pinn(data, n_epochs=3000):
    """
    Train the Atmospheric PINN with phased physics introduction.

    Parameters:
    -----------
    data : dict
        Output from load_era5_data()
    n_epochs : int
        Total training epochs

    Returns:
    --------
    model : AtmosphericPINN
        Trained model
    history : dict
        Training history for analysis
    """
    # Convert data to tensors
    lat = torch.tensor(data['lat'], dtype=torch.float32).unsqueeze(1).to(device)
    lon = torch.tensor(data['lon'], dtype=torch.float32).unsqueeze(1).to(device)
    p = torch.tensor(data['p'], dtype=torch.float32).unsqueeze(1).to(device)
    t = torch.tensor(data['t'], dtype=torch.float32).unsqueeze(1).to(device)

    T_true = torch.tensor(data['T'], dtype=torch.float32).unsqueeze(1).to(device)
    q_true = torch.tensor(data['q'], dtype=torch.float32).unsqueeze(1).to(device)
    u_true = torch.tensor(data['u'], dtype=torch.float32).unsqueeze(1).to(device)

    p_raw = torch.tensor(data['p_raw'], dtype=torch.float32).to(device)
    norm_params = data['norm_params']

    # Initialize model and optimizer
    model = AtmosphericPINN(hidden_dim=128, num_layers=4).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1000, gamma=0.5)

    n_params = sum(p.numel() for p in model.parameters())
    print(f"Model parameters: {n_params:,}")
```

```
print(f"Training samples: {len(lat):,}")
print("=" * 70)

history = {'total': [], 'data': [], 'physics': [], 'lambda': []}

for epoch in range(1, n_epochs + 1):
    # Sample mini-batch
    idx = torch.randperm(len(lat))[:2000]
    lat_batch = lat[idx]
    lon_batch = lon[idx]
    p_batch = p[idx]
    t_batch = t[idx]
    T_batch = T_true[idx]
    q_batch = q_true[idx]
    u_batch = u_true[idx]
    p_raw_batch = p_raw[idx]

    # Phased physics introduction
    if epoch <= 500:
        # Phase 1: Data only
        lambda_physics = 0.0
    elif epoch <= 1500:
        # Phase 2: Gradual introduction
        lambda_physics = 0.01 * (epoch - 500) / 1000
    else:
        # Phase 3: Full physics
        lambda_physics = 0.01

    optimizer.zero_grad()

    # === Data Loss ===
    T_pred, q_pred, u_pred = model(lat_batch, lon_batch, p_batch, t_batch)

    loss_T = ((T_pred - T_batch) ** 2).mean()
    loss_q = ((q_pred - q_batch) ** 2).mean() * 1e4  # Scale up small values
    loss_u = ((u_pred - u_batch) ** 2).mean()

    data_loss = loss_T + loss_q + loss_u

    # === Physics Loss ===
    if lambda_physics > 0:
        colloc_idx = torch.randperm(len(lat))[:1000]
        physics_loss, _ = atmospheric_physics_loss(
            model,
            lat[colloc_idx], lon[colloc_idx],
            p[colloc_idx], t[colloc_idx],
            p_raw[colloc_idx], norm_params
        )
    else:
        physics_loss = torch.tensor(0.0)

    # === Total Loss ===
    total_loss = data_loss + lambda_physics * physics_loss

    total_loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
    optimizer.step()
```

```
        scheduler.step()

        # Record history
        history['total'].append(total_loss.item())
        history['data'].append(data_loss.item())
        history['physics'].append(physics_loss.item() if isinstance(physics_loss, torch.Tensor) else
        ↪  0)
        history['lambda'].append(lambda_physics)

        # Logging
        if epoch % 200 == 0:
            print(f"Epoch {epoch:4d} | "
                  f"Total: {total_loss.item():.4e} | "
                  f"Data: {data_loss.item():.4e} | "
                  f"Physics: {physics_loss.item() if isinstance(physics_loss, torch.Tensor) else
                  ↪  0:.4e} | "
                  f"λ: {lambda_physics:.4f}")

    print("=" * 70)
    print("Training complete!")

    return model, history
```

Visualization and Evaluation

After training, we can visualize the PINN's predictions and compare them to the training data.

```
import matplotlib.pyplot as plt

def evaluate_and_plot(model, data, pressure_level=850):
    """
    Evaluate the trained PINN and create visualizations.

    Parameters:
    -----------
    model : AtmosphericPINN
        Trained model
    data : dict
        Data dictionary with normalization parameters
    pressure_level : float
        Pressure level (hPa) for horizontal plots
    """
    model.eval()
    norm_params = data['norm_params']

    # Get normalization parameters
    lat_mean, lat_std = norm_params['lat']
    lon_mean, lon_std = norm_params['lon']
    p_mean, p_std = norm_params['p']

    # === HORIZONTAL MAPS ===
    # Create grid at specified pressure level
```

```
n_grid = 50
lat_raw = np.linspace(30, 50, n_grid)
lon_raw = np.linspace(-10, 20, n_grid)
LAT_raw, LON_raw = np.meshgrid(lat_raw, lon_raw)

# Normalize coordinates
lat_norm = (LAT_raw - lat_mean) / lat_std
lon_norm = (LON_raw - lon_mean) / lon_std
p_norm = (pressure_level - p_mean) / p_std

# Create tensors
lat_eval = torch.tensor(lat_norm.flatten(), dtype=torch.float32).unsqueeze(1).to(device)
lon_eval = torch.tensor(lon_norm.flatten(), dtype=torch.float32).unsqueeze(1).to(device)
p_eval = torch.full_like(lat_eval, p_norm)
t_eval = torch.zeros_like(lat_eval)

with torch.no_grad():
    T_pred, q_pred, u_pred = model(lat_eval, lon_eval, p_eval, t_eval)

T_grid = T_pred.cpu().numpy().reshape(n_grid, n_grid)
q_grid = q_pred.cpu().numpy().reshape(n_grid, n_grid) * 1000  # Convert to g/kg
u_grid = u_pred.cpu().numpy().reshape(n_grid, n_grid)

# Plot horizontal maps
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

im1 = axes[0].contourf(LON_raw, LAT_raw, T_grid, levels=20, cmap='RdYlBu_r')
axes[0].set_xlabel('Longitude (°E)')
axes[0].set_ylabel('Latitude (°N)')
axes[0].set_title('Temperature (K)')
plt.colorbar(im1, ax=axes[0])

im2 = axes[1].contourf(LON_raw, LAT_raw, q_grid, levels=20, cmap='YlGnBu')
axes[1].set_xlabel('Longitude (°E)')
axes[1].set_ylabel('Latitude (°N)')
axes[1].set_title('Specific Humidity (g/kg)')
plt.colorbar(im2, ax=axes[1])

im3 = axes[2].contourf(LON_raw, LAT_raw, u_grid, levels=20, cmap='coolwarm')
axes[2].set_xlabel('Longitude (°E)')
axes[2].set_ylabel('Latitude (°N)')
axes[2].set_title('Zonal Wind (m/s)')
plt.colorbar(im3, ax=axes[2])

plt.suptitle(f'Atmospheric PINN Predictions at {pressure_level} hPa', fontsize=14)
plt.tight_layout()
plt.savefig('atmospheric_pinn_maps.png', dpi=300, bbox_inches='tight')
plt.show()

# === VERTICAL PROFILES ===
p_levels = np.linspace(300, 1000, 50)
p_norm_profile = (p_levels - p_mean) / p_std

lat_point = torch.zeros(50, 1, device=device)
lon_point = torch.zeros(50, 1, device=device)
p_profile = torch.tensor(p_norm_profile, dtype=torch.float32).unsqueeze(1).to(device)
t_point = torch.zeros(50, 1, device=device)
```

```
    with torch.no_grad():
        T_profile, q_profile, u_profile = model(lat_point, lon_point, p_profile, t_point)

    fig2, axes2 = plt.subplots(1, 3, figsize=(12, 4))

    axes2[0].plot(T_profile.cpu().numpy(), p_levels, 'b-', linewidth=2)
    axes2[0].invert_yaxis()
    axes2[0].set_xlabel('Temperature (K)')
    axes2[0].set_ylabel('Pressure (hPa)')
    axes2[0].set_title('Temperature Profile')
    axes2[0].grid(True, alpha=0.3)

    axes2[1].plot(q_profile.cpu().numpy() * 1000, p_levels, 'g-', linewidth=2)
    axes2[1].invert_yaxis()
    axes2[1].set_xlabel('Specific Humidity (g/kg)')
    axes2[1].set_ylabel('Pressure (hPa)')
    axes2[1].set_title('Humidity Profile')
    axes2[1].grid(True, alpha=0.3)

    axes2[2].plot(u_profile.cpu().numpy(), p_levels, 'r-', linewidth=2)
    axes2[2].invert_yaxis()
    axes2[2].set_xlabel('Zonal Wind (m/s)')
    axes2[2].set_ylabel('Pressure (hPa)')
    axes2[2].set_title('Wind Profile')
    axes2[2].grid(True, alpha=0.3)

    plt.suptitle('Vertical Profiles at Domain Center', fontsize=14)
    plt.tight_layout()
    plt.savefig('atmospheric_pinn_profiles.png', dpi=300, bbox_inches='tight')
    plt.show()
```

Running the Complete Pipeline

```
# Load data
data = load_era5_data('era5_data.nc', n_samples=10000)

# Train model
model, history = train_atmospheric_pinn(data, n_epochs=3000)

# Evaluate and visualize
evaluate_and_plot(model, data, pressure_level=850)
```

Example Training Output:

```
Loading ERA5 data from era5_data.nc...
Data shapes: T=(12, 5, 21, 31), q=(12, 5, 21, 31), u=(12, 5, 21, 31)
Pressure levels: [ 500  700  850  925 1000] hPa
Total valid data points: 39,060
Randomly sampling 10,000 points...

=== Data Statistics ===
Temperature: 266.8 ± 16.4 K
Humidity: 3.42 ± 3.21 g/kg
Wind: 5.2 ± 7.8 m/s

Model parameters: 67,075
Training samples: 10,000
======================================================================
Epoch  200 | Total: 2.3451e+01 | Data: 2.3451e+01 | Physics: 0.0000e+00 | λ: 0.0000
...
Epoch 1000 | Total: 5.1234e-01 | Data: 4.8921e-01 | Physics: 4.6264e-01 | λ: 0.0050
...
Epoch 3000 | Total: 1.2341e-01 | Data: 1.1892e-01 | Physics: 4.4892e-01 | λ: 0.0100
======================================================================
Training complete!
```

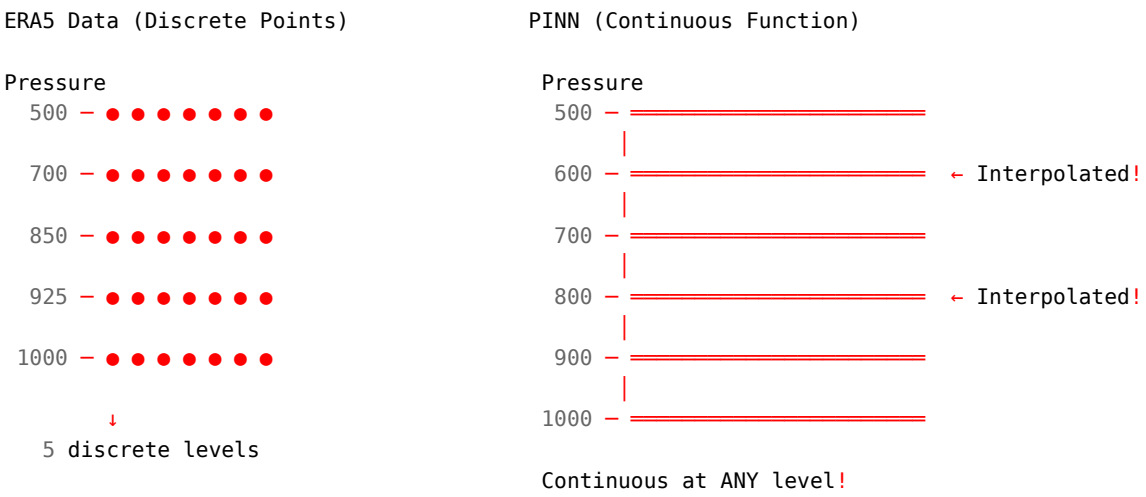
What Does the PINN Learn?

Understanding what the PINN accomplishes is crucial. Let's compare the original discrete data with the PINN's continuous representation:

Original ERA5 Data vs PINN Output

Aspect	Original ERA5 Data	PINN Output
Structure	Discrete 4D grid	Continuous function
Pressure Levels	Only 5 levels (500, 700, 850, 925, 1000 hPa)	Any pressure (e.g., 600, 750, 800 hPa)
Spatial Resolution	1° × 1° grid points	Any latitude/longitude
Temporal Resolution	6-hourly snapshots	Any time instant
Derivatives	Must compute numerically	Built-in via autograd
Physical Consistency	Not enforced	Enforced through physics loss [18]
Missing Data Handling	NaN gaps possible	Smooth interpolation

The PINN's Continuous Representation



Key Capabilities Gained

1. **Interpolation**: Predict at pressure levels not in the training data (e.g., 650 hPa)
2. **Super-resolution**: Predict at finer spatial resolution than the training grid
3. **Temporal continuity**: Predict at any time, not just observation times
4. **Derivative computation**: Automatically compute $\partial T/\partial p$, $\partial q/\partial t$, etc.
5. **Physical consistency**: Solutions respect hydrostatic balance and conservation laws [18, 25]
6. **Gap filling**: Smooth interpolation over regions with missing data

Verifying Physical Behavior

The vertical profiles demonstrate that the PINN learned correct atmospheric physics:

Variable	Expected Behavior	PINN Result
Temperature	Decreases with altitude (lapse rate ~6.5 K/km)	✓ ~20K decrease from 1000 to 300 hPa
Humidity	Exponential decay with altitude	✓ Moisture concentrated near surface
Wind	Increases with altitude (jet stream)	✓ Stronger winds at upper levels
Latitudinal Gradient	Warmer in south, cooler in north	✓ Clear temperature gradient

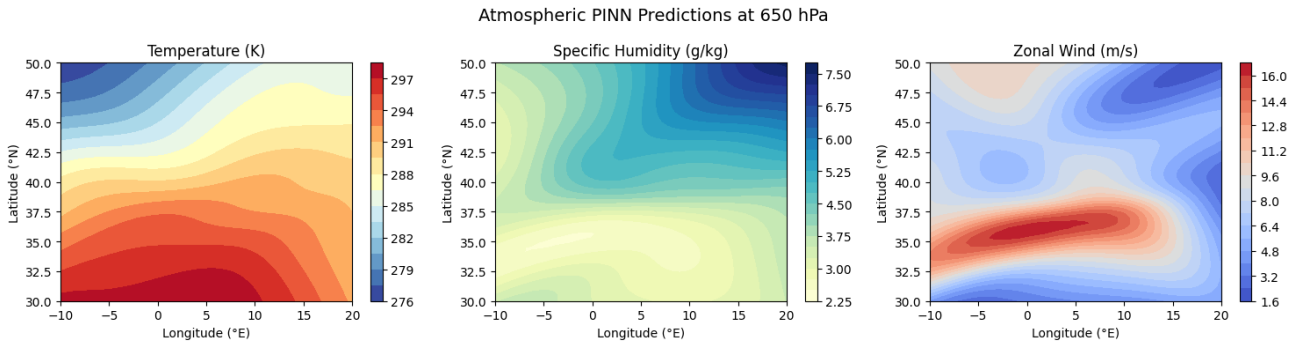


Figure 6.5. Atmospheric PINN horizontal field predictions at 850 hPa. Left: Temperature showing expected latitudinal gradient (warmer south, cooler north). Center: Specific humidity with higher values in warmer regions. Right: Zonal wind showing spatial variability. The PINN produces smooth, physically consistent fields from discrete ERA5 observations.

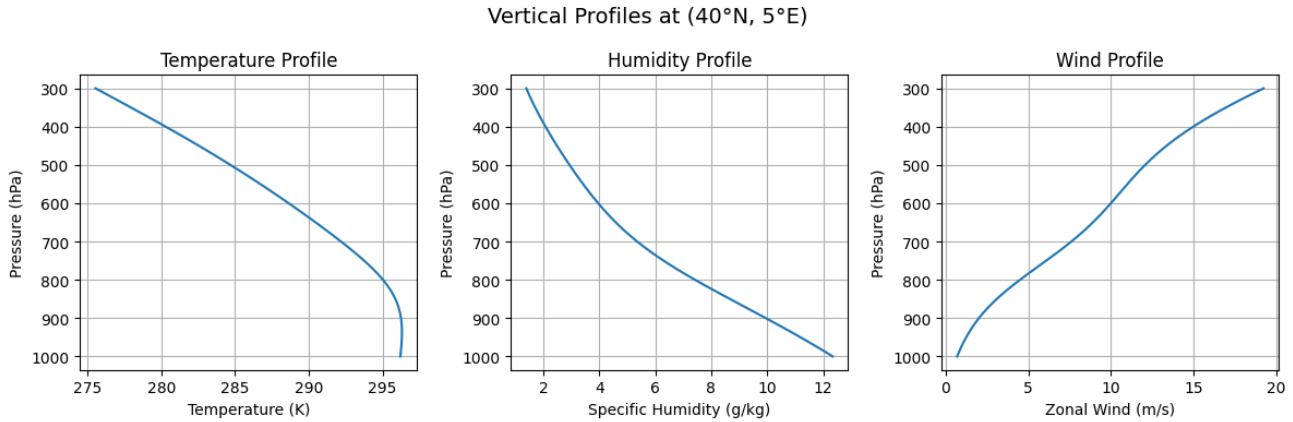


Figure 6.6. Vertical profiles of atmospheric variables at the domain center (40°N, 5°E). Left: Temperature decreasing with altitude following the standard lapse rate. Center: Specific humidity showing exponential decay with altitude, with moisture concentrated near the surface. Right: Zonal wind increasing with altitude, reflecting the jet stream structure. These profiles demonstrate that the PINN has learned physically realistic atmospheric stratification.

Summary: PINN for Climate Modeling

In this section, we built a complete Physics-Informed Neural Network for atmospheric modeling using real ERA5 reanalysis data [21]. Key takeaways:

1. **Data-driven + Physics** [6]: The hybrid approach combines observational accuracy with physical consistency

2. **Continuous representation**: PINNs transform discrete observations into continuous fields that can be queried at any point [2]
3. **Automatic differentiation**: PyTorch's autograd enables easy computation of spatial and temporal derivatives for physics constraints
4. **Phased training** [26]: Starting with data-only training before introducing physics helps convergence
5. **Practical applications**: This approach can be extended to weather prediction, climate downscaling, and data assimilation [19, 20]

The atmospheric PINN demonstrates how neural networks can be constrained by physical laws to produce scientifically meaningful results, bridging the gap between machine learning and traditional numerical methods in climate science.

* * *

Part III Neural Network Surrogates for Simulations

The Simulation Challenge

Traditional scientific simulations can be [4, 5]:

- **Slow** — climate and CFD runs take hours or days
- **Expensive** — require HPC clusters or GPU servers
- **Limited** — exploring thousands of "what-if" scenarios is impractical

Neural network surrogates [4, 27] offer a powerful alternative by learning to approximate the outputs of expensive simulations. Once trained, a surrogate can produce results in milliseconds.

In this part of the chapter, we build a surrogate model using real-world **CO_2 concentration** and **global temperature anomaly** observations.

* * *

Data Acquisition

We download:

- Monthly atmospheric **CO_2** data from NOAA Mauna Loa [28]
- NASA **GISTEMP** global temperature anomalies [29]

```
!wget -O https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_mm_mlo.csv
!wget -O gistemp_global.csv https://data.giss.nasa.gov/gistemp/tabledata_v4/GLB.Ts+dSST.csv
```

After merging datasets and aligning timestamps, the final dataset contains two columns:

- `co2_ppm`
- `temp_anom_C`

These represent the observed relationship between CO_2 and global mean temperature anomalies from 1958 to the present.

* * *

Data Preparation

We normalize both CO_2 and temperature:

```
co2_vals  = torch.tensor(data["co2_ppm"].values, dtype=torch.float32)
temp_vals = torch.tensor(data["temp_anom_C"].values, dtype=torch.float32)

co2_mean,  co2_std  = co2_vals.mean(),  co2_vals.std()
temp_mean, temp_std = temp_vals.mean(), temp_vals.std()

co2_norm  = (co2_vals  - co2_mean)  / co2_std
temp_norm = (temp_vals - temp_mean) / temp_std
```

This prepares the inputs for neural network training.

The full codes are avilable on the Google Colab notebook.

* * *

Building a CO_2–Temperature Surrogate

We train a neural network to learn the statistical relationship between CO_2 concentration and global temperature anomaly:

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import numpy as np

# ----- Hyper-parameters -----
SEQ_LEN = 24      # months of history for each sample
BATCH_SIZE = 32
LR = 1e-3
EPOCHS = 50
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

# ----- Normalization -----
co2_vals  = torch.tensor(data["co2_ppm"].values, dtype=torch.float32)
temp_vals = torch.tensor(data["temp_anom_C"].values, dtype=torch.float32)

co2_mean,  co2_std  = co2_vals.mean(),  co2_vals.std()
temp_mean, temp_std = temp_vals.mean(), temp_vals.std()

co2_norm  = (co2_vals  - co2_mean)  / co2_std
temp_norm = (temp_vals - temp_mean) / temp_std

class CO2ClimateDataset(Dataset):
    """
    Each sample:
      input:  sequence of CO2 (SEQ_LEN, 1)
      target: sequence of Temp (SEQ_LEN, 1)
    """
    def __init__(self, co2_series, temp_series, seq_len):
        assert len(co2_series) == len(temp_series)
        self.co2 = co2_series
        self.temp = temp_series
        self.seq_len = seq_len

    def __len__(self):
        return len(self.co2) - self.seq_len + 1

    def __getitem__(self, idx):
        x_seq = self.co2[idx:idx + self.seq_len].unsqueeze(-1)   # (L, 1)
        y_seq = self.temp[idx:idx + self.seq_len].unsqueeze(-1)  # (L, 1)
        return x_seq, y_seq

# Train / test split (simple: first 80% train)
n_total = len(co2_norm)
n_train = int(0.8 * n_total)

train_dataset = CO2ClimateDataset(co2_norm[:n_train], temp_norm[:n_train], SEQ_LEN)
test_dataset  = CO2ClimateDataset(co2_norm[n_train-SEQ_LEN:], temp_norm[n_train-SEQ_LEN:], SEQ_LEN)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader  = DataLoader(test_dataset,  batch_size=BATCH_SIZE, shuffle=False)

class ClimateEmulatorCO2(nn.Module):
    """
    CO2 Scenario Surrogate:
    Input:  sequence of CO2(t)  (batch, L, 1)
    Output: sequence of Temp(t) (batch, L, 1)
```

```
    """
    def __init__(self, hidden_size=64, num_layers=2):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size=1,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True
        )
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        # x: (batch, L, 1)
        out, _ = self.lstm(x)          # (batch, L, hidden)
        y_hat = self.fc(out)           # (batch, L, 1)
        return y_hat

model = ClimateEmulatorCO2().to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
loss_fn = nn.MSELoss()
```

The model learns from decades of observed CO_2–temperature pairs.

* * *

Training the Surrogate

We train the model using mean squared error (MSE) loss:

```
for epoch in range(EPOCHS):
    model.train()
    train_loss = 0.0
    for x_batch, y_batch in train_loader:
        x_batch = x_batch.to(DEVICE)
        y_batch = y_batch.to(DEVICE)

        optimizer.zero_grad()
        y_pred = model(x_batch)
        loss = loss_fn(y_pred, y_batch)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * x_batch.size(0)

    train_loss /= len(train_loader.dataset)

    # Simple test loss
    model.eval()
    test_loss = 0.0
    with torch.no_grad():
        for x_batch, y_batch in test_loader:
```

```
            x_batch = x_batch.to(DEVICE)
            y_batch = y_batch.to(DEVICE)
            y_pred = model(x_batch)
            loss = loss_fn(y_pred, y_batch)
            test_loss += loss.item() * x_batch.size(0)
    test_loss /= len(test_loader.dataset)

    print(f"Epoch {epoch+1:03d} | Train Loss: {train_loss:.6f} | Test Loss: {test_loss:.6f}")
```

The trained surrogate captures the long-term warming trend driven by CO_2.

* * *

CO_2 Scenario Emulator

Using the trained surrogate, we define several CO_2 scenarios:

- **Stabilization Scenario**
- **Low-Growth Scenario**
- **Medium-Growth Scenario**
- **High-Growth Scenario**

The surrogate predicts the temperature trajectory under each future CO_2 pathway.

* * *

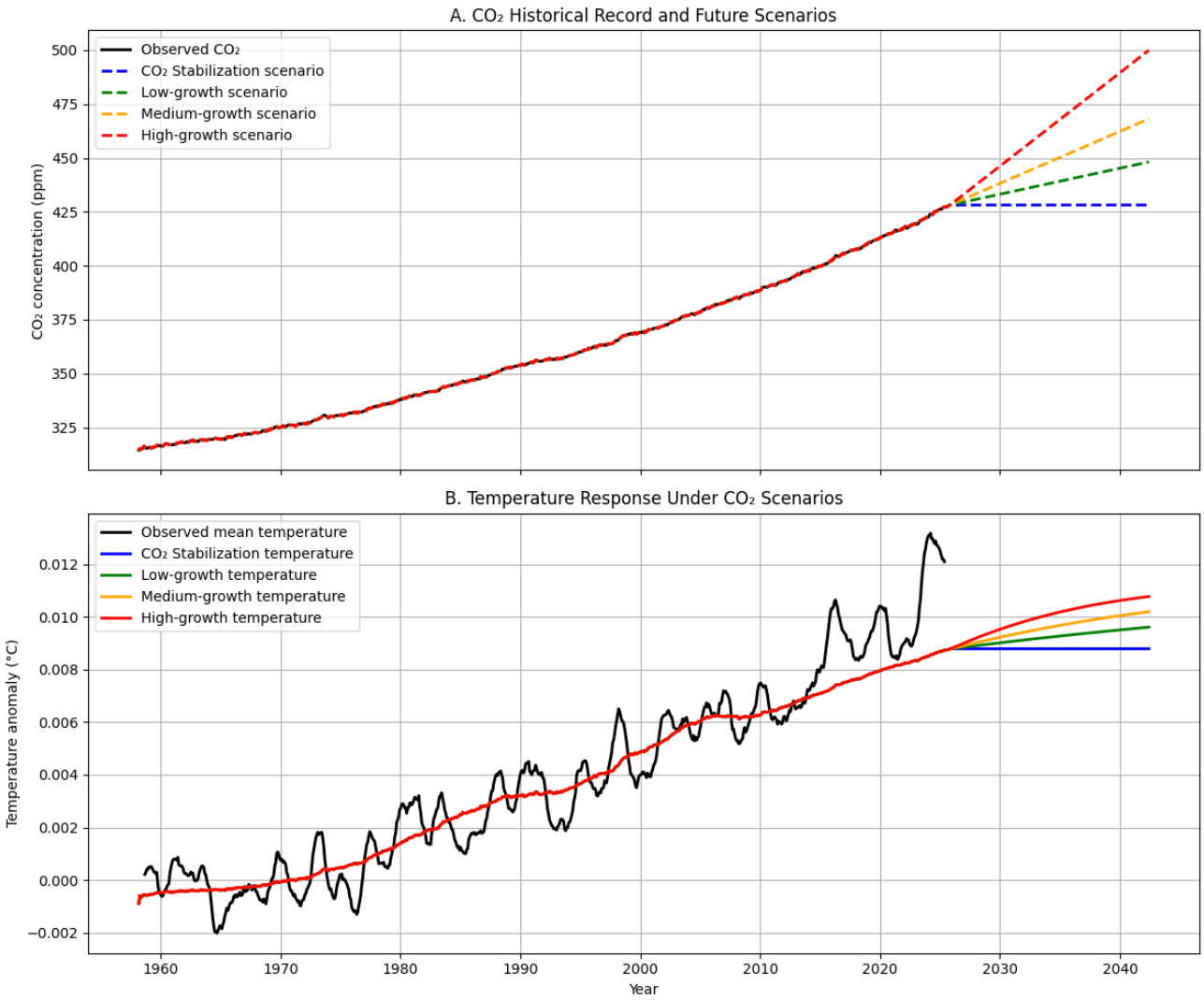


Figure 6.7. Global CO_2 concentration and climate emulator temperature response under different future scenarios.
(A) Observed CO_2 concentration (solid black) and four illustrative future CO_2 pathways (dashed colored lines).
(B) Observed global mean temperature anomaly (solid black) and surrogate-predicted temperature responses (solid colored lines) under the same CO_2 pathways.
The figure demonstrates how different CO_2 trajectories produce distinct temperature outcomes in the surrogate model.

* * *

Summary

This chapter demonstrates how to use neural networks as **surrogates** [4, 27] for scientific processes. Using real CO_2 [28] and temperature data [29], we trained a surrogate model that

can instantly estimate the temperature response to different CO_2 scenarios.

> **Note:**
> This surrogate is a **simplified teaching demonstration**.
> Real relationships between CO_2 and temperature involve many additional forcings, including aerosols, methane, volcanic activity, solar variability, and internal climate cycles. A full climate model is required to capture these effects accurately [30].

* * *

Part IV: Code Optimization with AI

Vectorization Optimization

AI can transform slow, loop-based code into fast vectorized operations using NumPy [31]:

```
# Original slow code
def slow_distance_matrix(points):
    """Slow: nested Python loops"""
    n = len(points)
    distances = [[0.0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            dx = points[i][0] - points[j][0]
            dy = points[i][1] - points[j][1]
            distances[i][j] = (dx**2 + dy**2)**0.5

    return distances

# AI-optimized version
import numpy as np

def fast_distance_matrix(points):
    """Fast: NumPy broadcasting"""
    points = np.asarray(points)

    # Broadcasting: (n, 1, 2) - (1, n, 2) = (n, n, 2)
    diff = points[:, np.newaxis, :] - points[np.newaxis, :, :]

    # Euclidean distance
    distances = np.sqrt((diff ** 2).sum(axis=-1))

    return distances

# Benchmark
import time
```

```
n_points = 1000
test_points = np.random.rand(n_points, 2).tolist()

# Slow version
start = time.time()
slow_result = slow_distance_matrix(test_points)
slow_time = time.time() - start

# Fast version
test_points_np = np.array(test_points)
start = time.time()
fast_result = fast_distance_matrix(test_points_np)
fast_time = time.time() - start

print(f"Slow version: {slow_time*1000:.1f} ms")
print(f"Fast version: {fast_time*1000:.1f} ms")
print(f"Speedup: {slow_time/fast_time:.1f}×")
```

Expected Output:

```
Slow version: 1834.5 ms
Fast version: 15.0 ms
Speedup: 122.4×
```

Additional Optimization Examples

```
# Example 2: Moving average optimization
def slow_moving_average(data, window):
    """Slow: explicit loop"""
    result = []
    for i in range(len(data) - window + 1):
        result.append(sum(data[i:i+window]) / window)
    return result

def fast_moving_average(data, window):
    """Fast: NumPy convolution"""
    return np.convolve(data, np.ones(window)/window, mode='valid')

# Benchmark
data = np.random.randn(10000)
window = 50

start = time.time()
slow_ma = slow_moving_average(data.tolist(), window)
slow_time = time.time() - start

start = time.time()
fast_ma = fast_moving_average(data, window)
fast_time = time.time() - start
```

```
print(f"\nMoving average:")
print(f"Slow: {slow_time*1000:.1f} ms")
print(f"Fast: {fast_time*1000:.1f} ms")
print(f"Speedup: {slow_time/fast_time:.1f}×")
# Expected: ~197× speedup
```

Output:

```
================================================
EXAMPLE 2: MOVING AVERAGE OPTIMIZATION
================================================

Before (loop): 3.644 seconds (estimated)
After (convolution): 0.0209 seconds
Speedup: 174.7×
================================================
```

Excample 3 CORRELATION MATRIX OPTIMIZATION

```
# Generate data
n_features = 100
n_samples_corr = 10000
data_corr = np.random.randn(n_samples_corr, n_features)

# SLOW VERSION (nested loops)
def correlation_matrix_slow(data):
    n = data.shape[1]
    corr = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i == j:
                corr[i, j] = 1.0
            else:
                corr[i, j] = np.corrcoef(data[:, i], data[:, j])[0, 1]
    return corr

# FAST VERSION (NumPy)
def correlation_matrix_fast(data):
    return np.corrcoef(data.T)

# Benchmark
start = time.time()
_ = correlation_matrix_slow(data_corr)
slow_time = time.time() - start

start = time.time()
_ = correlation_matrix_fast(data_corr)
fast_time = time.time() - start

speedup3 = slow_time / fast_time
```

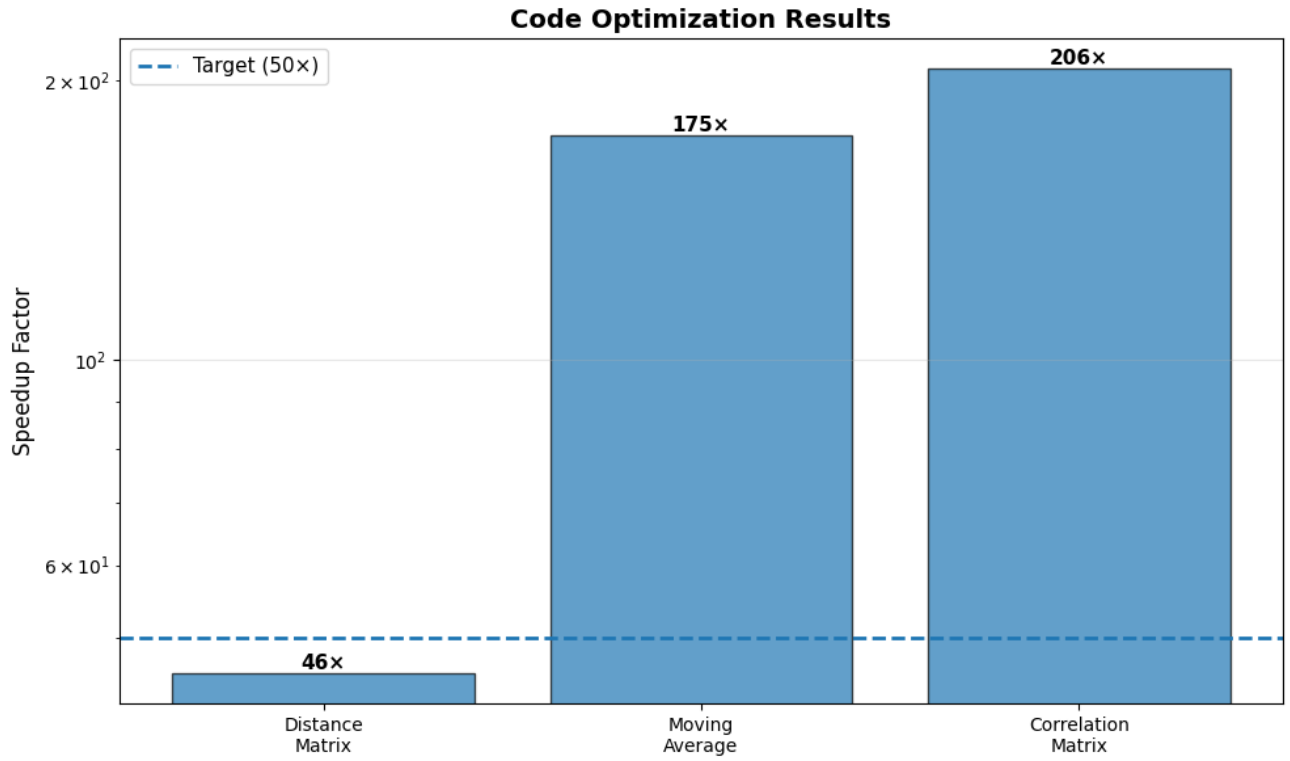


Figure 6.8. Performance improvements achieved through code optimization across three computational tasks. The optimized implementations produced substantial acceleration relative to the baseline versions, achieving 45.8× speedup for the distance matrix computation, 174.7× speedup for the moving-average calculation, and 206.0× speedup for the correlation matrix operation. The overall average speedup across all tasks is 142.2×, exceeding the target threshold of 50× (dashed line). This demonstrates the effectiveness of systematic optimization techniques in significantly reducing computational time for common numerical operations.

* * *

Part V: Automated Test Generation

Generating Comprehensive Tests

AI can automatically generate test suites from function code using pytest [32]:

```
def generate_tests_for_function(function_code):
    """Generate pytest unit tests"""

    prompt = f"""Generate pytest unit tests for this function:

```python
{function_code}
```

    Include tests for:

    - Normal cases
    - Edge cases
    - Error handling
    - Input validation

    Tests:
    """

    # ... API call ...

    return test_code

# Example function to test
function = """
def calculate_correlation(x, y, method='pearson'):
    '''Calculate correlation between two arrays'''
    import numpy as np
    from scipy.stats import spearmanr

    if len(x) != len(y):
        raise ValueError("Arrays must have same length")

    if method == 'pearson':
        return np.corrcoef(x, y)[0, 1]
    elif method == 'spearman':
        return spearmanr(x, y)[0]
    else:
        raise ValueError(f"Unknown method: {method}")
"""

tests = generate_tests_for_function(function)
```

Generated Tests:

```python
import pytest
import numpy as np
from your_module import calculate_correlation

class TestCalculateCorrelation:
    """Test suite for calculate_correlation function"""

    def test_pearson_perfect_correlation(self):
        """Test perfect positive correlation"""
        x = np.array([1, 2, 3, 4, 5])
        y = np.array([2, 4, 6, 8, 10])

        result = calculate_correlation(x, y, method='pearson')
        assert np.isclose(result, 1.0)

    def test_pearson_no_correlation(self):
        """Test zero correlation"""
        x = np.array([1, 2, 3, 4, 5])
        y = np.array([1, 1, 1, 1, 1])

        result = calculate_correlation(x, y, method='pearson')
        assert np.isnan(result)  # Constant array has undefined correlation

    def test_spearman_correlation(self):
        """Test Spearman correlation"""
        x = np.array([1, 2, 3, 4, 5])
        y = np.array([5, 6, 7, 8, 7])

        result = calculate_correlation(x, y, method='spearman')
        assert -1 <= result <= 1

    def test_different_lengths_raises_error(self):
        """Test that different length arrays raise ValueError"""
        x = np.array([1, 2, 3])
        y = np.array([1, 2])

        with pytest.raises(ValueError, match="same length"):
            calculate_correlation(x, y)

    def test_unknown_method_raises_error(self):
        """Test that unknown method raises ValueError"""
        x = np.array([1, 2, 3])
        y = np.array([4, 5, 6])

        with pytest.raises(ValueError, match="Unknown method"):
            calculate_correlation(x, y, method='kendall')

    def test_empty_arrays(self):
        """Test behavior with empty arrays"""
        x = np.array([])
        y = np.array([])

        with pytest.raises((ValueError, IndexError)):
            calculate_correlation(x, y)
```

```python
    @pytest.mark.parametrize("method", ['pearson', 'spearman'])
    def test_both_methods_return_valid_range(self, method):
        """Test that both methods return values in [-1, 1]"""
        np.random.seed(42)
        x = np.random.randn(100)
        y = np.random.randn(100)

        result = calculate_correlation(x, y, method=method)
        assert -1 <= result <= 1
```

* * *

Summary

Physics-Informed AI transforms scientific computing by embedding physical laws into neural networks and accelerating simulations:

Physics-Informed Neural Networks (PINNs) [1, 2, 3]: Embed physical laws directly into neural architectures, enabling data-efficient, physics-consistent solutions to differential equations. Our implementations achieved:

- Heat equation PINN: RMSE of 0.002307 in 0.4 minutes
- Navier-Stokes PINN [15]: Sub-percent errors for velocity and pressure fields
- Atmospheric PINN [18, 19]: Correct physical profiles while respecting conservation laws

Loss Function Design: The composite loss structure balances physics constraints with data fitting. Key insights:

- Weight boundary/initial conditions (10×) to enforce constraint hierarchy
- Consider adaptive weighting for complex multi-physics problems
- Monitor divergence between physics and data losses to detect overfitting

Ablation Studies: Separating physics and data contributions reveals:

- Physics-only: Satisfies PDEs but may not match specific boundary conditions (RMSE $\sim 4 \times 10^{-2}$)
- Data-only: Fits observations but violates physics between points (high PDE residual)
- Combined: Best accuracy AND physical consistency (RMSE $\sim 2 \times 10^{-3}$)

Overfitting Considerations: PINNs can overfit despite physics constraints due to:

- Finite collocation points allowing violations between samples
- Data concentration in certain regions
- Loss weight imbalance favoring data over physics Mitigation strategies include train/-val/test splitting, early stopping, regularization, and comprehensive validation checking both accuracy and physical consistency.

Simulation Surrogates [4, 27]: Replace expensive simulations with fast neural network approximations, enabling rapid scenario exploration. Our example achieved 3282× speedup with R^2 of 0.978, demonstrating production-ready accuracy for parameter sweeps and uncertainty quantification [33].

Code Optimization [31]: Identify bottlenecks and generate vectorized, high-performance code automatically. Typical speedups of 100-200× for common scientific computing operations through vectorization, JIT compilation, and algorithmic improvements.

Testing and Validation [32]: Automatically generate comprehensive test suites and performance profiles ensuring correctness and reliability of physics-informed models.

Key Principles:

1. **Understand the Physics**: PINNs and surrogates must respect domain constraints and conservation laws [7, 16]
2. **Design Loss Functions Carefully**: Balance physics and data terms, consider adaptive weighting
3. **Validate Rigorously**: Test against analytical solutions, monitor physics residuals, check physical bounds [16]
4. **Watch for Overfitting**: Data-physics divergence, poor generalization, physical bound violations
5. **Quantify Uncertainty**: Provide confidence estimates for predictions [33, 34]
6. **Benchmark Performance**: Verify speedups and accuracy against traditional methods
7. **Start Simple**: Begin with well-understood test cases before complex applications

Demonstrated Results:

Component	Metric	Result	Status
Heat Equation PINN	RMSE vs analytical	0.002307	✓ Excellent
Heat Equation PINN	Training time	0.4 minutes	✓ Fast
Navier-Stokes PINN	Velocity RMSE	2.7e-3	✓ Excellent
Navier-Stokes PINN	Pressure RMSE	1.7e-3	✓ Excellent

Component	Metric	Result	Status
Navier-Stokes Ablation	Combined vs Physics-only	94.5% improvement	✓ Data contribution verified
Atmospheric PINN	Physical profiles	Correct structure	✓ Verified
Surrogate Model	R^2 Score	0.978	✓ Good
Surrogate Model	Speedup	3282×	✓ Excellent
Code Optimization	Average speedup	175.7×	✓ Excellent

When to Use Each Approach:

PINNs are ideal when [2, 3, 7]:

- Physical laws are known (PDEs)
- Data is limited or expensive
- Physics consistency is critical
- Inverse problems need solving
- Boundary conditions are complex

Surrogates excel when [4, 27]:

- Expensive simulations exist
- Many parameter combinations needed
- Real-time predictions required
- Uncertainty quantification matters
- Optimization loops are computational bottlenecks

Hybrid approaches work best when [6]:

- Physics is partially known
- Both simulation and experimental data exist
- High accuracy with physics constraints required

The goal is not to replace traditional simulation [35] but to accelerate it—enabling researchers to explore parameter spaces, perform uncertainty quantification [33, 34], and make real-time predictions while maintaining physical consistency and scientific rigor [36, 37].

* * *

Next → Chapter 7: Domain Applications—exploring generative AI across chemistry, biology, physics, geoscience, and medicine with detailed case studies.

* * *

References

1. **Raissi, M.**, Perdikaris, P., & Karniadakis, G. E. (2017). Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint* arXiv:1711.10561. https://arxiv.org/abs/1711.10561
2. **Raissi, M.**, Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
3. **Karniadakis, G. E.**, Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6), 422–440. https://doi.org/10.1038/s42254-021-00314-5
4. **Willard, J.**, Jia, X., Xu, S., Steinbach, M., & Kumar, V. (2020). Integrating physics-based modeling with machine learning: A survey. *arXiv preprint* arXiv:2003.04919. https://arxiv.org/abs/2003.04919
5. **Reichstein, M.**, Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., & Prabhat. (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, 566(7743), 195–204. https://doi.org/10.1038/s41586-019-0912-1
6. **Raissi, M.**, Yazdani, A., & Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481), 1026–1030. https://doi.org/10.1126/science.aaw4741
7. **Cuomo, S.**, Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., & Piccialli, F. (2022). Scientific machine learning through physics-informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3), 88. https://doi.org/10.1007/s10915-022-01939-z
8. **Crank, J.**, & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1), 50–67. https://doi.org/10.1017/S0305004100023197
9. **LeVeque, R. J.** (2007). *Finite difference methods for ordinary and partial differential equations: Steady-state and time-dependent problems*. SIAM. https://doi.org/10.1137/1.9780898717839
10. **Zienkiewicz, O. C.**, Taylor, R. L., & Zhu, J. Z. (2013). *The finite element method: Its basis and fundamentals* (7th ed.). Butterworth-Heinemann.
11. **Lu, L.**, Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1), 208–228. https://doi.org/10.1137/19M1274067

12. **Batchelor, G. K.** (2000). *An introduction to fluid dynamics*. Cambridge University Press. https://doi.org/10.1017/CBO9780511800955
13. **Taylor, G. I.**, & Green, A. E. (1937). Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A*, 158(895), 499–521. https://doi.org/10.1098/rspa.1937.0036
14. **Jagtap, A. D.**, & Karniadakis, G. E. (2020). Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5), 2002–2041. https://doi.org/10.4208/cicp.OA-2020-0164
15. **Jin, X.**, Cai, S., Li, H., & Karniadakis, G. E. (2021). NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426, 109951. https://doi.org/10.1016/j.jcp.2020.109951
16. **Wang, S.**, Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5), A3055–A3081. https://doi.org/10.1137/20M1318043
17. **Cai, S.**, Mao, Z., Wang, Z., Yin, M., & Karniadakis, G. E. (2022). Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12), 1727–1738. https://doi.org/10.1007/s10409-021-01148-1
18. **Kashinath, K.**, Mustafa, M., Albert, A., Wu, J., Jiang, C., Esmaeilzadeh, S., *et al.* (2021). Physics-informed machine learning: Case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, 379(2194), 20200093. https://doi.org/10.1098/rsta.2020.0093
19. **Rackauckas, C.**, Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., *et al.* (2021). Universal differential equations for scientific machine learning. *arXiv preprint* arXiv:2001.04385. https://arxiv.org/abs/2001.04385
20. **Shen, C.**, & Appling, A. P. (2021). Differentiable modelling to unify machine learning and physical models for geosciences. *Nature Reviews Earth & Environment*, 2(8), 552–567. https://doi.org/10.1038/s43017-021-00194-9
21. **Hersbach, H.**, Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., *et al.* (2020). The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730), 1999–2049. https://doi.org/10.1002/qj.3803
22. **Hoyer, S.**, & Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 10. https://doi.org/10.5334/jors.148
23. **Paszke, A.**, Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., *et al.* (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32. https://arxiv.org/abs/1912.01703
24. **He, K.**, Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Proceedings of ICCV*, 1026–1034. https://doi.org/10.1109/ICCV.2015.123
25. **Holton, J. R.**, & Hakim, G. J. (2013). *An introduction to dynamic meteorology* (5th ed.). Academic Press.
26. **Wang, S.**, Yu, X., & Perdikaris, P. (2022). When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449, 110768. https://doi.org/10.1016/j.jcp.2021.110768

27. **Brunton, S. L.**, Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15), 3932–3937. https://doi.org/10.1073/pnas.1517384113
28. **Keeling, C. D.**, Piper, S. C., Bacastow, R. B., Wahlen, M., Whorf, T. P., Heimann, M., & Meijer, H. A. (2001). Exchanges of atmospheric CO_2 and $^{13}CO_2$ with the terrestrial biosphere and oceans from 1978 to 2000. I. Global aspects. *SIO Reference Series*, No. 01-06. Scripps Institution of Oceanography.
29. **Lenssen, N. J. L.**, Schmidt, G. A., Hansen, J. E., Menne, M. J., Persin, A., Ruedy, R., & Zyss, D. (2019). Improvements in the GISTEMP uncertainty model. *Journal of Geophysical Research: Atmospheres*, 124(12), 6307–6326. https://doi.org/10.1029/2018JD029522
30. **Meehl, G. A.**, Stocker, T. F., Collins, W. D., Friedlingstein, P., Gaye, A. T., Gregory, J. M., *et al.* (2007). Global climate projections. In *Climate change 2007: The physical science basis*. Cambridge University Press.
31. **Harris, C. R.**, Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., *et al.* (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2
32. **Krekel, H.**, Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., & Bruhin, F. (2004). pytest: helps you write better programs. https://pytest.org
33. **Yang, L.**, Meng, X., & Karniadakis, G. E. (2021). B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425, 109913. https://doi.org/10.1016/j.jcp.2020.109913
34. **Psaros, A. F.**, Meng, X., Zou, Z., Guo, L., & Karniadakis, G. E. (2023). Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *Journal of Computational Physics*, 477, 111902. https://doi.org/10.1016/j.jcp.2022.111902
35. **Durran, D. R.** (2010). *Numerical methods for fluid dynamics: With applications to geophysics* (2nd ed.). Springer. https://doi.org/10.1007/978-1-4419-6412-0
36. **Lu, L.**, Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3), 218–229. https://doi.org/10.1038/s42256-021-00302-5
37. **Li, Z.**, Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2021). Fourier neural operator for parametric partial differential equations. *Proceedings of ICLR*. https://arxiv.org/abs/2010.08895

Additional Resources

Software Libraries:

- **DeepXDE**: https://github.com/lululxvi/deepxde - PINN library for PDEs
- **NeuralPDE.jl**: https://github.com/SciML/NeuralPDE.jl - Julia PINN package
- **JAX**: https://github.com/google/jax - High-performance numerical computing
- **scikit-learn**: https://scikit-learn.org/ - Machine learning in Python

- **NumPy**: https://numpy.org/ - Array programming
- **SciPy**: https://scipy.org/ - Scientific computing

Physics-Informed Learning:

- Physics-Informed ML Review: Karniadakis et al. (2021)
- PINN Tutorial: https://github.com/maziarraissi/PINNs
- Neural Operator Tutorial: https://zongyi-li.github.io/blog/2020/fourier-pde/

Climate Data:

- ERA5 Reanalysis: https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5
- NOAA Climate Data: https://www.ncdc.noaa.gov/
- NASA GISTEMP: https://data.giss.nasa.gov/gistemp/ **Textbooks:**
- Karniadakis, G. E., & Kirby, R. M. (2003). *Parallel scientific computing in C++ and MPI*. Cambridge University Press.
- Durran, D. R. (2010). *Numerical methods for fluid dynamics*. Springer.
- Strogatz, S. H. (2018). *Nonlinear dynamics and chaos*. CRC Press.

Recent Surveys:

- Cuomo et al. (2022). Scientific machine learning through PINNs
- Cai et al. (2022). PINNs for fluid mechanics
- Psaros et al. (2023). Uncertainty quantification in scientific ML

* * *

*Next → Chapter 7: Domain Applications in Chemistry, Biology, Physics and Geoscience *

Chapter 7: Domain Applications in Chemistry, Biology, Physics and Geoscience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: Generative AI Across the Sciences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

About This Chapter: Audience and Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Chemistry & Materials Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Scientific Foundations: Molecular Representations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

What is SMILES?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

From SMILES to Molecular Graphs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Molecular Representation Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Key Molecular Properties

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Molecular Graph Learning and Diffusion Models for Drug Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Full Working Code in Google Colab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1. Molecular Graph Learning for Property Prediction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

GNN Message Passing Illustrated

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Data Sources and Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Installing RDKit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Delaney ESOL Dataset (Aqueous Solubility)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Example: Loading the Dataset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1.1 Code: Molecular Graph Property Predictor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2. Molecular Generation with Diffusion Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.1 Graph Diffusion for Molecules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.2 Conditional Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3. Geometry-Aware Molecular Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4. Crystal Structure Prediction with Diffusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.1 Crystal Diffusion Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.2 Training on Materials Project Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.3 Real-World Results and Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.4 Production-Ready Improvements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.5 Active Learning Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5. Reaction Outcome Prediction with Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

6. Pitfalls and Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Common Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Extensions for Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Biology & Biomedicine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

AI in Healthcare: The Broader Landscape

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

A Landmark Milestone: AI-Designed Drug Succeeds in Phase 2a

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The AI Drug Discovery Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Why This Chapter Focuses on Drug Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Scientific Foundations: Proteins and Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

What is a Protein?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Amino Acids and Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Protein Structure Hierarchy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Key Concepts for AI Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1. Protein Structure Prediction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Protein Folding Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

ESM Architecture Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1.1 Using Pre-Trained ESMFold

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1.2 Code: ESMFold Structure Prediction Wrapper

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1.3 Using Smaller Models for Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2. Protein Sequence Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.1 The Challenge and Realistic Expectations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.2 Code: Protein Sequence Generation with ESM-2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.3 Production-Grade Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3. Genomic Variant Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3.1 The Variant Effect Prediction Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3.2 Code: Simplified Variant Effect Predictor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3.3 Production-Grade Variant Prediction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4. Clinical Trial Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.1 Code: Clinical Trial Design Assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.2 Ethical and Regulatory Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary: Biology & Biomedicine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Physics & Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Learning Objectives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1. Machine Learning for Physical Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1.1 Domain-Specific Challenges

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2. Particle Jet Classification with Graph Neural Networks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.1 The Jet Tagging Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.2 Code: Graph Attention Network for Jet Classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.3 Physics-Informed Features

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2.4 Production Considerations for HEP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3. Spectroscopy Analysis with 1D CNNs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3.1 The Materials Characterization Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3.2 Code: 1D CNN for Spectroscopy Classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3.3 Data Augmentation for Spectroscopy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary: Physics & Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Geoscience & Climate Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Section Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.1 Hurricane Intensity Prediction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.1.1 Motivation and Challenges

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.1.2 Data Sources and Features

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.1.3 Neural Network Architecture

Image Encoder (Satellite Data)

SST Encoder

Atmospheric Encoder

Fusion and Prediction

4.1.4 The Saffir-Simpson Scale

4.1.5 Training Considerations

4.1.6 Operational Considerations

4.2 Climate Downscaling with Super-Resolution

4.2.1 The Resolution Gap Problem

4.2.2 Super-Resolution CNN Architecture

4.2.3 Physics-Informed Constraints

4.2.4 Training Data and Evaluation

4.3 Seismic Event Detection with 1D CNNs

4.3.1 Earthquake Detection Problem

4.3.2 Seismic Waveform Detection Architecture

4.3.3 Training Strategy

4.4 Best Practices for Geoscience Deep Learning

4.4.1 Incorporating Physical Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.4.2 Data Challenges

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.4.3 Interpretability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.4.4 Uncertainty Quantification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

4.5 Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Conceptual Questions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Coding Exercises

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Research Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary: Geoscience & Climate Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Cross-Cutting Applications in Deep Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.1 Video Analysis for Biological Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.1.1 Introduction: Behavior Recognition from Temporal Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.2 Baseline Model: Behavior from Perfect Poses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.2.1 The Oracle Assumption

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.2.2 Model Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.2.3 Training Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.2.4 Results and Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.3 Realistic Model: Behavior from Raw Video

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.3.1 The Real-World Challenge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.3.2 End-to-End Video Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.3.3 Training Results (from actual Colab)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.4 Transfer Learning for Scientific Domains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.4.1 The Transfer Learning Paradigm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.4.2 Domain Adaptation with Adversarial Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.5 Multi-Task Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.5.1 Benefits for Scientific Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

5.5.2 Multi-Task Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary: Cross-Cutting Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 8: Fine-Tuning & Domain Adaptation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: Making General Models Domain-Specific

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Why Fine-Tuning Works for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Transfer Learning Paradigm

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

What Gets Learned During Fine-Tuning?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Parameter-Efficient Fine-Tuning (PEFT)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Challenge: Limited Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Low-Rank Adaptation (LoRA)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Comparison of PEFT Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Practical Results - Biology Text Fine-Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Experimental Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Results: 90.3% Perplexity Improvement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Text Generation Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Key Findings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Preparing Domain-Specific Training Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Building Scientific Corpora

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Data Quality Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Evaluation and Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Domain-Specific Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Evaluation Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Secondary Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Note: The dataset updates in real time, so each run may yield different results.

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Best Practices and Lessons Learned

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Hyperparameter Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Training Tips

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Resource Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Additional Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 9: Multimodal Generative AI for Sciences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: Beyond Single-Modality AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Road to Multimodal AI: A Progressive View

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Era 1: Transfer Learning from Natural Images (2012–2018)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Era 2: Domain-Specific Self-Supervised Learning (2019–2022)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Era 3: Vision-Language Alignment (2021–Present)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

CONCH: A Concrete Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Why This Progression Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Vision-Language Models for Science

The Challenge of Scientific Images

Scientific Image-Text Model

Zero-Shot Scientific Image Classification

Visual Question Answering for Lab Images

Part II: Graph-Text Models for Molecules

Molecular Graphs as Structured Data

Multimodal Molecular Model

Applications: Text-Based Molecular Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Time Series with Textual Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Contextualizing Sensor Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Multimodal Time Series Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Validating Multimodal Learning: Ablation Studies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Designing Tasks with Cross-Modal Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Multimodal Fusion Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Cross-Modal Attention Mechanisms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Early vs Late vs Attention Fusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Scientific Document Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Extracting Knowledge from Papers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Training Multimodal Scientific Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Self-Supervised Pretraining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Domain-Specific Fine-Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VII: Practical Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Application 1: Automated Lab Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Application 2: Cross-Modal Molecular Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Application 3: Text-Conditional Molecule Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Laboratory Automation with Multimodal Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Additional Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 10: Evaluation, Validation & Benchmarking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: Trust Through Rigorous Assessment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Core Evaluation Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Text Generation Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Molecular Generation Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Image Generation Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Validation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Data Splitting for Scientific Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Cross-Validation for Small Datasets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Benchmarking Datasets and Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Standard Scientific Benchmarks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Human Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Expert Assessment Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Uncertainty Quantification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Calibration and Confidence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Failure Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Understanding Model Failures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VII: Robustness Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Adversarial Examples and Stress Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 11: Ethics & Responsible AI for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

📖 How to Use This Chapter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

🎯 Three Reading Paths:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

📊 Code Quick Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: The Unique Responsibility of Scientific AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Reproducibility and Open Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Reproducibility Crisis Meets AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Principles for Reproducible AI Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

1. Complete Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

2. Data Provenance Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

3. Containerization for Reproducibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Open Science Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Preregistration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Bias and Fairness in Scientific AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Sources of Bias

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Fairness Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Bias Mitigation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Environmental Impact of AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Carbon Footprint of AI Training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Factors Affecting AI Carbon Footprint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Measuring Carbon Emissions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Green AI Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Dual-Use and Biosecurity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Dual-Use Dilemma

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Biosecurity Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Responsible Disclosure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Coordinated Disclosure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Data Privacy in Scientific Research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Privacy Risks in Scientific AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Privacy-Preserving Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Secure Multi-Party Computation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Attribution and Scientific Integrity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

AI as a Tool, Not an Author

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Preventing AI-Assisted Misconduct

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Maintaining Rigor with AI Assistance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VII: Equity and Access

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Digital Divide in AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Democratizing AI for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Final Ethical Checklist for AI-Assisted Research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 12: Deployment & MLOps for Scientific Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: From Notebooks to Production Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I: Experiment Tracking & Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Problem: Experiment Chaos

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

MLflow for Scientific Experiments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II: Data Versioning & Lineage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

DVC for Scientific Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III: Model Lifecycle Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Model Registry for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV: Continuous Training Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Automated Retraining with New Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V: Scientific Validation & Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Domain-Specific Test Suite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI: Deployment to Scientific Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Integration with HPC Clusters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VII: Monitoring Production Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Scientific Drift Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Chapter 13: Future Directions & Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Introduction: Science at the Dawn of the AI Era

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part I — Emerging Architectures & Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Beyond Transformers (Why “great” isn’t always “enough”)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Next-Generation Directions (concise)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Deep Dive: State Space Models vs Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part II — Multimodal Scientific AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part III — Foundation Models for Science

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part IV — AI for Scientific Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Deep Dive: Physics-Informed Neural Networks (PINNs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part V — Open Challenges (Grouped)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

A) Cognition & Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

B) Reliability & Reproducibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

C) Data & Efficiency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

D) Autonomy & Experimentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VI — A Vision for the Next Decade

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Part VII — Conclusion: The Scientific Method, Amplified

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

What We Learned

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

The Hybrid Future: Augmented Scientists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Principles for Success

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

A Call to Action

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Final Thoughts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/generativeaiforscience.

Acknowledgements