

Jone Gorman AI Learning Guide: From Foundations to Agents, Physical AI, and RSI

2026-10-19

Contents

1. Understand computers	2
2. Learn your operating system	2
3. Learn Python	3
4. Manage development environments	3
5. Move from machine learning to deep learning	3
6. Understand large language models	4
7. Reinforcement learning: foundations and PPO	6
8. Explore agents and applications	6
9. RAG, memory, and evaluation	7
10. Multimodal models, generation, and world models	7
11. Physical AI: from perception to physical action	8
12. Scientific machine learning and AI for CAE	8
13. RSI: recursive self-improvement	9
14. Connect the path through a project	10
15. Build a sustainable learning practice	10

Expensive courses, unfamiliar terminology, and a constant stream of new technology can make getting started with AI feel overwhelming. This guide begins from the perspective of a learner with a limited budget and little computing experience, breaking “I want to learn AI” into manageable steps.

From understanding a computer to writing a first Python script, reproducing a model, and finishing a small project, each step can leave you with code and notes of your own. Courses provide an entry point; hands-on work connects the ideas.

The aim is conceptual familiarity backed by practical ability. Start with the relationship between AI, machine learning, deep learning, and generative models. Then explore tool-using agents, systems that interact with the physical world, and ways to evaluate and improve them. You can start a project before finishing every course.

Place the terms first: AI is the broader field; machine learning studies learning from data or experience; deep learning mainly uses multilayer neural networks; LLMs are a family of language models. Reinforcement learning is a learning and decision-making framework, while an agent organizes models, tools, and an environment around a goal. These are not all the same kind of model.

- **Starting from scratch:** Work through Sections 1–5 until you can write Python, manage an environment, and explain a small model’s training results.
- **Building agent applications:** Continue with Sections 6–9 and begin with cited question answering or a tool task whose result you can check.
- **Exploring physics and research:** Choose a direction from Sections 10–13, then use Section 14 to document a complete project. RSI here means Recursive Self-Improvement.

The material spans university courses, YouTube explanations, original papers, author code, official documentation, and research blogs. YouTube entries prioritize original releases by authors, universities, and institutions, with learning stages and goals. Use videos for intuition, papers for evidence, and code and documentation for implementation.

1. Understand computers

Start with the basic parts of a computer. Distinguish the CPU, memory, storage, and GPU, and develop a rough understanding of how programs, files, and hardware work together. You do not need high-performance hardware or a complete computer science education to begin.

- Introduction to computers, in Chinese

Practical checkpoint: Find your device’s processor, memory, and disk capacity, and locate frequently used files. Memory and disk storage are different; installing software primarily consumes storage space.

2. Learn your operating system

Begin with familiar Windows operations, then explore the operating system, terminal, directories, and permissions. Learn Linux when a course or project calls for it. Dual boot is one route included in the original document; a virtual machine or WSL can also provide a place to practice.

- Dual-boot installation tutorial, in Chinese
- Basic Linux operations, in Chinese

Practical checkpoint: Use a terminal to display and change directories, create a folder, and run a program. Understand what each command does.

3. Learn Python

Building a foundation in one language is an accessible starting point. This route uses Python: learn variables, conditionals, loops, functions, common data structures, imports, and basic debugging before moving into machine-learning code.

- Setting up Python and VS Code, in Chinese
- Python fundamentals, in Chinese

Practical checkpoint: Write a small script that reads a file, processes its contents, and saves a result. After following a tutorial, change the input or add a feature and explain the program's behavior.

4. Manage development environments

Projects often require different versions of Python and their dependencies. Miniconda provides a relatively small initial installation for creating project-specific environments; conda or pip within the environment can install the packages you need. The actual storage difference from Anaconda depends on what is installed, rather than a universal size ratio.

- Miniconda tutorial, in Chinese
- Official conda environment-management documentation

The original document uses Python 3.10 for its first environment. This is an example of the commands, not a version requirement for every project; follow the course or dependency file you are working with.

```
conda create -n aienv python=3.10
conda activate aienv
python --version
python -m pip --version
```

Practical checkpoint: Confirm that your terminal, editor, and package installation use the same environment. Record the project's Python version and dependency configuration.

5. Move from machine learning to deep learning

Mathematics, code, and experiments come together in AI. If a derivation is initially difficult, run a small example and then connect the equations with the inputs, outputs, and implementation. Courses move at different speeds; combine resources as needed without turning completion speed into the goal.

YouTube: intuition, mathematics, and code

- **Initial intuition:** 3Blue1Brown's But what is a neural network?. Connect networks, losses, and parameter updates, then sketch a two-layer network's inputs and outputs.

- **Mathematics as needed:** 3Blue1Brown's Vectors and The essence of calculus. Connect vector and matrix operations with derivatives and rates of change, then calculate a simple gradient yourself.
- **A first training implementation:** Karpathy's Building micrograd, alongside the course page. Basic Python and derivatives prepare you to study computation graphs, backpropagation, and gradient descent.
- **A broader overview:** MIT 6.S191's 2025 introductory lecture and course and labs. With introductory linear algebra and calculus, use it to map the field and run a selected lab.

Learn data quality, training/validation/test splits, overfitting, loss functions, and evaluation metrics alongside training. A linear model or small MLP can provide a baseline for deciding whether added complexity is worthwhile.

Courses and practical material

- Mu Li: Dive into Deep Learning, in Chinese
- Abhishek Thakur: Approaching (Almost) Any Machine Learning Problem, author's repository
- AAAMLP Chinese translation
- RethinkFun deep-learning tutorials, in Chinese

Git, GitHub, and hardware

Learn version control alongside programming: Git records changes, while GitHub can host and share repositories. Start with examples suited to your existing hardware, and evaluate GPU needs when a project requires it. The PC-building video from the original document is retained as supplementary hardware material.

- Introduction to GitHub, in Chinese
- Introduction to Git, in Chinese
- PC-building tutorial by Hardware Tea Talk, in Chinese

Practical checkpoint: Finish a modeling experiment with a data split, evaluation metric, and recorded results, then commit your implementation to a Git repository.

6. Understand large language models

Start with the Transformer and connect attention, tensor shapes, and the forward pass. Implementing a teaching model line by line before studying architectures such as Llama makes the modules in a paper more concrete. The sequence here is a learning path, rather than a ranking of technologies by recency or performance.

YouTube: from intuition to implementation

- **See the structure:** 3Blue1Brown's Transformers, the tech behind LLMs builds visual intuition for tokens, vectors, and attention.
- **Write a model:** Karpathy's Let's build GPT develops a small language model after you have learned tensors and automatic differentiation.

- **Reproduce each layer:** Umar Jamil's Coding a Transformer from scratch, paired with the author's implementation, connects the model, training, and inference.
- **Understand the input:** Karpathy's Let's build the GPT Tokenizer connects text with tokens and their implications for context and computation.
- **Go deeper into engineering:** Stanford CS336 (2025), starting with its first lecture and assignments. Approach its data, training, systems, and evaluation material after building Python, PyTorch, mathematics, and machine-learning foundations.

Transformer and Llama

- Attention Is All You Need walkthrough
- Implementing a Transformer in PyTorch
- Llama architecture walkthrough
- Implementing Llama in PyTorch

Model architecture

- RoPE: rotary position embeddings
- MoE: mixture of experts
- RMSNorm

Fine-tuning, inference, and efficiency

LoRA is a parameter-efficient fine-tuning method that can be used for supervised fine-tuning. KV caching, FlashAttention, and PagedAttention address different aspects of attention computation, inference, or memory management. Distinguish the problem each technique tackles.

- LoRA fine-tuning
- KV caching
- FlashAttention
- PagedAttention

From pretraining to post-training

Separate training objectives from parameter-update methods. Pretraining builds a base model; supervised fine-tuning (SFT) learns target behavior from examples; preference optimization or reinforcement learning uses different forms of feedback. LoRA is a parameter-efficient update method that can accompany different objectives. Repeatedly generating longer answers does not itself constitute training.

Continue with the Hugging Face LLM Course, the Smol Course instruction-tuning unit, and the original DPO paper. Explain the data examples, loss, and evaluation before running a teaching-scale experiment.

Practical checkpoint: Implement and explain an attention module at a teaching scale, recording key tensor shapes. For each subsequent technique, identify which part of the system it changes and what conditions it introduces.

7. Reinforcement learning: foundations and PPO

Understand states, actions, rewards, policies, and value functions before moving into algorithms and code. PPO is a focus of this route. Its performance depends on the task, implementation, and evaluation setup; it is not universally the best algorithm for every problem.

- Shusen Wang: Deep Reinforcement Learning, in Chinese
- PPO tutorial, in Chinese
- The 37 Implementation Details of Proximal Policy Optimization

YouTube, advanced: Berkeley CS285 (2023), with its official playlist and course site. Place policy gradients within a broader decision-making framework, including imitation learning, model-based methods, and offline RL. Complete a small RL experiment before selecting advanced lectures.

Learn implementation details alongside experiment tracking: inspect learning curves and record random seeds, hyperparameters, and the runtime environment. TensorBoard and Weights & Biases (W&B), mentioned in the original document, can support visualization. Begin with records you can save and revisit.

The original document also includes a community post about connecting to W&B. The post's entry URL is retained without personal sharing parameters. Its content and availability depend on the platform.

Distinguish inference engines from RL frameworks

- vLLM documentation: primarily for LLM inference and serving; it can also provide generation within an RL system.
- verl repository and official documentation: a framework for reinforcement learning with large language models.

Practical checkpoint: Complete a small PPO experiment with saved configuration and learning curves, then identify the separate roles of training, generation, and evaluation in an LLM reinforcement-learning system.

8. Explore agents and applications

Once you can read Python, manage dependencies, and understand basic model calls, explore how agents connect tools and external environments. This is the application stage of the original route; a small, clearly bounded project can bring the earlier material together.

- Hugging Face AI Agents Course: introduction
- Course livestream page included in the original document

Practical checkpoint: Choose a task whose result you can check. Implement a tool call, record the input, tool response, and final output, then verify that the model actually used the returned information.

YouTube and system design

- **Orient yourself:** Hugging Face's Agents Course introduction and Q&A explains the learning scope before you enter the course units.

- **Observe execution:** Hugging Face’s Agent process demo, paired with the first-agent tutorial. The video demonstrates execution; use the tutorial for the full implementation.
- **Understand system boundaries:** Anthropic’s Building Effective Agents distinguishes predefined workflows from agents that dynamically select their next steps, with different suitability and costs.

A working agent needs a goal, available tools, execution state, and stopping conditions. Begin with a small select–call–observe–continue-or-stop loop. Add input validation, explicit errors, bounded retries, and logs before introducing complex orchestration.

MCP provides a protocol for connecting model applications to tools and data. It can standardize interfaces; planning, permissions, and result verification still require system design. Start with one controlled tool and test malformed parameters, timeouts, and empty results.

9. RAG, memory, and evaluation

Move from getting an answer to completing a task with evidence. The original RAG paper combines generation with external retrieval. For practice, chunk and index your public notes, retrieve relevant passages, and require answers to cite actual text. Retrieving similar text and producing a correct answer need separate checks.

YouTube practice: LangChain’s RAG From Scratch playlist, paired with its official example repository. After learning model calls and file handling, study indexing, retrieval, and generation. Check current documentation for framework interfaces.

Distinguish context, persistent memory, and training. Saving information in files or databases changes what a later run can access; it does not by itself update model parameters. Specify each memory’s source, validity, and retrieval method before automating memory management.

Compare direct answering, RAG, and a tool-using agent on a small fixed question set. Record correctness, whether evidence supports conclusions, actual tool execution, time, and tokens. Separate questions used during development from final acceptance questions. The Hugging Face final project provides a course entry into benchmark-based agent evaluation.

Practical checkpoint: For 20 questions, save reference answers, source passages, execution traces, and costs. Include missing evidence, conflicting sources, and questions outside the material. This is a learning exercise, rather than industrial performance certification.

10. Multimodal models, generation, and world models

Multimodal learning connects forms of data such as images, text, audio, and actions. First ask what the model observes, then what it predicts: recognizing an image, generating a video, forecasting environment states, and selecting actions are different tasks.

YouTube foundations: MIT 6.S191’s Deep Generative Modeling (2025), with the course material, provides a generative-modeling overview after basic neural networks and probability.

Advanced author talk: Danijar Hafner’s Unsupervised Intelligent Agents. After reinforcement-learning foundations, study how world models enter learning and decision-making, then check the details against papers and code.

One practical approach to world models learns environment dynamics and uses predictions for

decision-making. The DreamerV3 author project page describes learning an environment model and improving behavior through imagined trajectories. Track observations, actions, latent states, rewards, and policies alongside visual results.

Practical checkpoint: In a simple simulator, compare one-step prediction error with multi-step roll-out error, then measure success on the same control task. A plausible-looking video alone does not establish accurate contact dynamics, long-horizon prediction, or closed-loop control; measure these abilities separately.

11. Physical AI: from perception to physical action

Here, Physical AI refers to intelligent systems interacting with a physical environment: sensor observations inform state estimation or a policy, which produces actions and receives environmental feedback. Robotics offers a concrete entry point, alongside control frequency, actuator limits, collisions, contact, and changing conditions.

YouTube and practical entry points

- **Conceptual entry:** NVIDIA Developer’s Introduction to Physical AI & Robotics at NVIDIA. Use it to orient yourself to physical interaction and simulation-based training, keeping product demonstrations separate from validation results.
- **Robotics practice:** Hugging Face’s LeRobot Tutorials playlist, paired with current LeRobot documentation. Begin with data recording, imitation learning, and policy evaluation. Check older videos’ hardware and installation steps against current documentation.
- **Simulation training:** Use Isaac Lab documentation to explore environments, observations, actions, rewards, and task configuration. Run a small existing task before changing a parameter, after checking hardware and software requirements.

Imitation learning learns behavior from demonstrations, while reinforcement learning optimizes policies around environmental feedback. For one task, compare data sources, costs, failure modes, and adaptation to environmental changes, alongside successful demonstrations.

Practical checkpoint: Complete a reaching or pushing task in simulation, then vary mass, friction, or sensor noise and record success rates and failures. Simulation performance does not replace hardware tests; transfer requires renewed checks of models, latency, and contact conditions.

12. Scientific machine learning and AI for CAE

Physical AI overlaps with using physics in machine learning, but the terms are not equivalent. Engineering simulation also requires differential equations, boundary conditions, numerical discretization, and conservation relationships to understand how learned models can work with solvers.

YouTube: physical constraints and operator learning

- **Orientation:** Steve Brunton’s Physics Informed Machine Learning playlist, after calculus, linear algebra, and introductory numerical methods.

- **Physical constraints:** Physics-Informed Neural Networks. Study how equation residuals and boundary conditions enter the learning objective, then reproduce a one-dimensional example.
- **Operator learning:** Fourier Neural Operator. Understand the aim of mapping input functions to output functions, then read the original FNO paper.

Compare these approaches on one problem: PINNs use equations as training constraints; neural operators learn mappings for families of inputs and solutions; MeshGraphNets investigates physical dynamics on mesh graphs. Their inputs, training data, and generalization assumptions differ. NVIDIA PhysicsNeMo provides an engineering entry point for scientific machine learning.

Practical checkpoint: Choose a small one-dimensional heat or elasticity problem, preserve a verifiable numerical solution, then train a small surrogate. Split complete load cases between training and testing. Compare field error, boundary satisfaction, and inference cost for unseen parameters or boundary conditions, specifying whether data generation, training, and inference are included in timing.

For contact and structural problems, also examine local peaks, mesh changes, and accumulated long-horizon error. Smooth visualizations do not establish engineering accuracy, and learned-model error differs from error in the underlying numerical model.

13. RSI: recursive self-improvement

Use this working question: does an improvement also strengthen the system's ability to produce subsequent improvements? When reading RSI research, specify who proposes a change, what changes, who evaluates it, and what persists.

Distinguish four mechanisms first

Mechanism	What changes	First check
Test-time feedback and revision	Output, context	Output quality
Learning from feedback	Model parameters	Held-out performance
Agent self-modification	Code, tools, execution logic	Execution and generalization
Evolutionary program search	Candidate programs, archive	Objective and constraint verification

This is a reading framework. Self-Refine demonstrates feedback and revision without additional training; learning from AI feedback includes training. Repeated answer revision, persistent memory, and weight updates need separate descriptions.

The Darwin Gödel Machine studies agent self-modification; the cited paper's experiments use frozen foundation models to improve coding-agent design. AlphaEvolve combines generation, automated evaluation, and evolutionary search over candidate programs. Gains on finite tasks alone do not demonstrate indefinitely accelerating improvement across arbitrary domains.

YouTube, papers, and code reading

- **Author technical talk:** Robert Lange’s ShinkaEvolve — AutoML Seminar, linked from his talks index. Approach it after agents, evaluation, and evolutionary search. This is an author’s seminar presentation, rather than a DGM-specific video.
- **Trace an actual modification:** Read the DGM author repository and follow a parent agent, code diff, evaluation, and retention decision rather than only the final score.
- **Verify an artifact first:** The AlphaEvolve results repository provides material for checking discoveries. Verifying a result and reproducing the complete search system are different tasks.

Practical checkpoint: With a fixed model and protected independent tests, compare one-shot generation, three feedback rounds, and equal-budget independent sampling. Record correctness, actual tool traces, tokens, time, and failures. Then allow changes to one restricted tool implementation and check transfer to unseen tasks. Keep the evaluator and test answers outside the candidate’s editable area.

This exercise explores what changes and how to trust evaluation. Even if results improve, separate the contributions of additional inference, memory, code changes, and training. Stronger RSI claims require a dedicated measurement of subsequent improvement capability.

14. Connect the path through a project

Finish one direction before extending to a second. These are proposed learning projects, not reported experimental results.

- **Evidence-based document agent:** Read public documents, return passage citations, and verify a number through a tool. Preserve retrieval, correctness, citation consistency, and cost records.
- **Physical-task policy:** Solve a small reaching or pushing task in simulation. Compare success across seeds and environmental perturbations, preserving failure videos and configurations.
- **Engineering surrogate:** Generate heat or elasticity data with a verified solver and predict unseen load cases. Report field and local errors, and define which stages the computational cost includes.
- **Traceable improvement loop:** Propose tool-code changes and accept them only after independent tests. Under a fixed budget, compare with the original tool and extra sampling, retaining each version diff.

Deliver a problem definition, data and splits, runnable code, baseline, evaluation method, resource records, and failure analysis. Aim to explain why the system produced its results and the conditions under which it remains unreliable.

15. Build a sustainable learning practice

Ongoing information sources

- **University courses and assignments:** Use MIT 6.S191, Stanford CS336, and Berkeley CS285 for structured knowledge, notes, and practice, selecting material that matches your preparation.
- **Original papers and discussion:** Follow preprints through arXiv’s machine-learning category, and use OpenReview for submissions and available review discussions. Check versions and publication status separately.

- **Conference proceedings:** Use PMLR and NeurIPS Proceedings to find published papers and supplementary material, then trace foundational methods for a specific problem.
- **Author code and engineering documentation:** Follow a paper or project page to its author repository. Read the README, dependencies, examples, and version history, and check implementation details against sources such as the PyTorch documentation.
- **Lab and author writing:** Use the DeepMind, Sakana AI, NVIDIA, and author project pages in this guide as research leads, checking claims against papers, code, experimental conditions, and failures.
- **Community discovery:** Hugging Face Daily Papers can surface material. Popularity, reposts, and comments help identify questions; return to original evidence for technical judgments.

For a new work, describe its problem, inputs and outputs, baseline, evaluation, and limits in your own words, then reproduce one minimal result. Keep source links, versions, code commits, and execution records so you can cross-check an idea between courses, papers, and implementation.

English reading and listening help you access courses, papers, and official documentation directly. When a translation is unclear, return to the original source, code, and actual execution results.

Creators mentioned in the original document include:

- **Umar Jamil:** algorithm implementations and paper walkthroughs.
- **Andrej Karpathy:** hands-on neural-network and language-model code.
- **Mu Li:** deep-learning courses, paper reading, and engineering practice.

Search around problems that interest you, read documentation, reproduce code, and learn to describe your questions clearly when asking for help. Preserve environments, changes, and results so that your work remains something you can revisit and build on.

LLMs are an important part of this path, and AI extends beyond them. As the focus of the field changes, the ability to find information, understand a problem, and implement an idea will continue to support the next thing you learn.

Start with curiosity, stay focused, and learn alongside people who share your interests. **Get your hands dirty.**