

1.1 — The AI Zoo: What Counts as AI?

Gavin · AI-assisted course design

Contents

1	Learning goals	3
2	1. Start by separating the layers	3
3	2. Explicit knowledge and searching possible futures	4
	3.1 Symbolic and rule-based systems	4
	3.2 Search and planning	5
4	3. Optimization and evolution	5
5	4. Learning a mapping from examples	6
6	5. Ask where the learning signal comes from	6
	6.1 Supervised learning	6
	6.2 Unsupervised learning	7
	6.3 Self-supervised learning	7
	6.4 Reinforcement learning	7
7	6. Where large language models fit	7
8	7. Images, video, speech, and multimodal systems	8
	8.1 Image generation	8
	8.2 Video generation	9
	8.3 Speech and audio	9
	8.4 Multimodal models and multimodal products	9
9	8. Assembling an agent around models	9
10	9. Use a map, not a single family tree	11
11	Check your understanding	11
	11.1 Discussion	12
12	More Learning	12
13	References	12

Imagine asking an assistant: “Make a poster for a community astronomy night, then explain the design.” The answer appears in one conversation. Behind that interface, a language model might interpret your request, a separate image model might generate the artwork, and ordinary software might save the file. Another system could combine those capabilities differently.

Calling the whole experience “AI” tells us very little about how it works. This lesson builds a map for asking better questions: What computation is happening? What was learned? Which component produced the behavior?

AI is a broad field concerned with systems that perform tasks involving abilities such as perception, learning, reasoning, planning, and action. Its boundaries overlap with statistics, optimization, and computer science. A useful engineering description identifies mechanisms rather than trying to settle whether every individual program deserves the label.

1 Learning goals

By the end of this lesson, you should be able to:

- Explain how explicit rules, search, optimization, and learned models solve different kinds of problems.
- Distinguish an algorithm, an architecture, trained weights, an inference runtime, orchestration, and a product.
- Identify the learning signal in supervised, unsupervised, self-supervised, and reinforcement learning.
- Explain how language, image, video, speech, and multimodal models can participate in one system.
- Trace an agent’s proposed action through the software that actually executes it.

i Math to know / refresh

Needed now: Basic functions and probability intuition. A function maps inputs to outputs; a probability distribution assigns probabilities to possible outcomes. No calculus is required.

Useful refresh: Optimization means looking for inputs or parameters that make an objective smaller or larger. For a finite set of mutually exclusive and exhaustive outcomes, probabilities are nonnegative and sum to one.

Side trail: Graphs describe states and possible transitions. Gradients describe how a numerical objective changes locally. Both will become useful later; neither needs a formal treatment here.

2 1. Start by separating the layers

Suppose we build the astronomy-poster assistant. Six distinctions help us describe it precisely.

An **algorithm** is a procedure: follow rules to perform a computation. Searching candidate routes, updating a model during training, and choosing an output during generation all involve algorithms. “Algorithm” is therefore broader than “machine learning.”

A neural-network **architecture** specifies the organization of a model’s computation: which components connect, which operations they perform, and the shapes of their inputs and outputs. A

Transformer is an architecture family. It leaves many design choices open and does not specify one set of learned capabilities.

Trained weights are numerical parameters produced by training. Think of the architecture as defining a family of possible functions and the weights as selecting a particular member. A saved checkpoint contains these parameters and may contain other training state. Two checkpoints with the same architecture can behave differently.

An **inference runtime**, or inference engine, loads the model and executes it. Hardware, numerical precision, generation settings, and implementation choices affect speed, memory use, and sometimes outputs. Inference means using a model; training means adjusting its learned parameters. The two involve computation for different purposes.

Orchestration connects model calls with surrounding software: assembling context, retrieving documents, calling tools, checking results, and deciding whether to continue. An agent harness is one kind of orchestration. A **product** packages the system into an experience with an interface, accounts, storage, and operational controls.

These are analytical layers, not necessarily six separate programs. “Model” itself is used loosely, so ask whether a speaker means an architecture, a checkpoint, or a hosted service.

Consider four changes to our assistant. Replacing its checkpoint changes learned parameters. Adding an event brief to its context changes the input. Adjusting a sampling setting changes generation behavior. Giving it a file-saving tool changes its available actions. Similar improvements at the interface can have very different causes.

Our recurring question is: **Where does this behavior live?**

3 2. Explicit knowledge and searching possible futures

3.1 Symbolic and rule-based systems

A symbolic system represents information using explicit objects, facts, relationships, or rules. A rule engine can match current facts against rule conditions and apply the matching rules. CLIPS is a concrete, maintained example of software for building rule-based expert systems. CLIPS documentation

Here is a small illustrative system:

- Fact: the event is outdoors.
- Fact: the weather forecast says rain.
- Rule: if an event is outdoors and rain is forecast, recommend the indoor backup venue.

The conclusion follows because the conditions match. No training dataset is necessary for this rule. The system can show which facts and rule led to its recommendation.

That transparency has limits. If nobody supplied a rule for strong wind, the system may miss that problem. If the forecast is wrong, a correctly executed rule can still recommend the wrong action. Larger knowledge bases also require careful handling of conflicting rules and missing information.

An ordinary conditional statement does not, by itself, make a program an interesting AI system. The connection to symbolic AI becomes useful when explicit representations support reasoning over a

domain. Rule-based components also remain useful inside systems containing learned models, especially where a precise condition must be enforced.

3.2 Search and planning

Search explores possible alternatives. Planning asks which sequence of actions can move a system from its current state to a goal. A planner needs some representation of available actions and their consequences; those representations may be supplied by people or learned.

Suppose a delivery robot can reach the venue by two routes. Route A has legs costing 3 and 6 minutes, for a total of 9. Route B has three legs costing 2 minutes each, for a total of 6. Choosing the route with fewer legs gives the wrong answer. The objective concerns total travel time.

On a large map, enumerating every complete route is expensive. Heuristic search uses estimates to prioritize promising partial routes. A* combines cost already incurred with an estimate of remaining cost; its guarantees depend on the heuristic and search conditions. Hart, Nilsson, and Raphael's original paper

Search can work without learning. A learned estimate can also guide search. These categories already overlap.

Retrieval check: In the venue example, which information came from explicit facts, which came from a rule, and which would require exploring possible action sequences?

4 3. Optimization and evolution

Optimization begins with a choice and an objective. A schedule might minimize lateness while respecting room capacities. A delivery plan might minimize distance while meeting time windows. Tools such as OR-Tools supply several solver families for these constrained problems. OR-Tools introduction

A tiny numerical example makes the idea concrete. Let the cost of choosing a number x be:

$$L(x) = (x - 3)^2.$$

Choosing 0 costs 9; choosing 2 costs 1; choosing 3 costs 0. We can solve this one by inspection. More complicated objectives require procedures that propose, evaluate, and improve candidates. An optimizer's answer depends on the objective, constraints, search method, and available computation. "Optimized" does not automatically mean "globally best."

Evolutionary algorithms search using populations of candidates. A typical genetic algorithm evaluates fitness, selects candidates, creates variants through mutation and often crossover, and repeats. Crossover combines parts of parent candidates; mutation changes a candidate. The DEAP library makes these operators explicit. DEAP operator tutorial

For an illustrative four-bit candidate, define fitness as the number of ones. The candidate 1010 has fitness 2. Flipping its second bit produces 1110 with fitness 3. That is a worked calculation, not an experimental result. In a real scheduling problem, the candidate might encode assignments, and an evaluation might be costly. Some changes would make a candidate worse or violate a constraint.

Evolutionary search is useful when candidates can be evaluated even if a convenient gradient is unavailable. It can optimize a design directly or help choose parts of a learning system. A neural network can itself be optimized using evolutionary methods. Conversely, many neural networks are trained with gradient-based optimization. “Evolutionary” describes a search strategy; “neural network” describes a model family.

Prediction check: If a timetable optimizer rewards only short walking distances, what undesirable schedule could score well? Write down a missing constraint before continuing.

5 4. Learning a mapping from examples

Machine learning builds useful behavior from data rather than specifying every decision rule manually. A learning procedure fits a model; the fitted model is then evaluated on cases beyond those used for fitting. Classical model families include linear models, decision trees, nearest-neighbor methods, and ensembles such as random forests. [scikit-learn’s supervised-learning guide](#)

Imagine predicting attendance from the number of advance registrations. An illustrative fitted model is $f(x) = 4 + 2x$. If $x = 3$, it predicts 10 attendees. The formula’s form and the numerical values play different roles: $a + bx$ describes a model family, while $a = 4$ and $b = 2$ specify this member.

In an actual project, data would determine whether that relationship was useful. An excellent fit to old events could fail on a new kind of event. Evaluation on held-out examples helps expose that problem; preprocessing must also avoid leaking information from evaluation data into training. [scikit-learn’s fit, predict, and evaluation workflow](#)

A **neural network** composes parameterized transformations, commonly including weighted combinations and nonlinear functions. Training adjusts parameters so these transformations become useful for the objective. A multilayer perceptron is one example; convolutional, recurrent, and Transformer architectures organize computation differently. A neural “unit” is a mathematical component, not a miniature biological brain. [Neural-network documentation](#)

Neural networks can learn intermediate representations as well as final outputs. That helps with rich inputs such as images, but it does not eliminate choices about data, architecture, objectives, or evaluation. “Classical” and “neural” are useful descriptions, not a ranking that determines which approach will work best for every task.

6 5. Ask where the learning signal comes from

A model family and a learning setup answer different questions. The same architecture can participate in several setups, sometimes at different stages of its training.

6.1 Supervised learning

Supervised learning uses example inputs paired with desired targets. An input might be an event description and its target a topic category, or registration features and a measured attendance count. Targets need not be manually written by a human; they can come from records or other measurement processes. Classification predicts categories, while regression predicts numerical quantities. [scikit-learn’s supervised-learning guide](#)

For example, a spam detector can learn from messages paired with spam/not-spam labels. The learning procedure receives information about what the answer should be for each training example.

6.2 Unsupervised learning

Unsupervised learning seeks structure without supplied target labels for the task. We might cluster event descriptions by similarity, then inspect whether the groups are useful. K-means, for example, fits cluster centers using a distance-based objective. “Unsupervised” does not mean the algorithm has no objective or that its groups are guaranteed to match meaningful human categories. Clustering documentation

If a cluster contains astronomy and photography events, perhaps their shared vocabulary dominates the chosen representation. Calling the cluster “science” is a human interpretation that still needs checking.

6.3 Self-supervised learning

Self-supervised learning constructs targets from the data itself. Hide part of a sentence and predict the hidden content, or use the beginning of a passage to predict what follows. The original text supplies the target without a separate person labeling each training example. BERT’s masked-language-model objective is a well-known example. BERT paper

For “The telescope is on the roof,” we can hide “roof” and use the rest as input. We have created a supervised-style prediction problem from an existing sentence. Terminology varies: self-supervised methods are sometimes grouped under unsupervised learning. The important distinction here is how the target was obtained.

6.4 Reinforcement learning

Reinforcement learning concerns an agent choosing actions in an environment and learning from rewards to improve expected return. Feedback may arrive after several actions, so the learner must address which choices contributed to the outcome. A reward evaluates what happened; it need not specify the correct action at every step. OpenAI’s introduction to RL

A robot might receive a reward for completing a delivery and penalties for collisions. A reward based only on speed could encourage behavior the designer did not intend. Defining a measurable reward and specifying the real goal are separate tasks.

RL need not use neural networks. Nor does every deployed agent learn while it runs. AlphaZero is a useful hybrid example: its training combines self-play reinforcement learning with a neural network, while tree search helps select moves. AlphaZero paper

Retrieval check: An image model predicts noise that software added to training images. Is the target supplied by a human labeler, derived from the data and corruption process, or delivered as an environmental reward? Explain the distinction rather than relying on the word “image.”

7 6. Where large language models fit

A language model learns statistical relationships in language sequences. Generative LLMs commonly produce text by repeatedly predicting a distribution over the next small piece of text and choosing

from that distribution. Lesson 1.2 explains those pieces, called tokens, and how text becomes numbers. The GPT-3 paper provides a concrete autoregressive language-model example. GPT-3 paper

“Large” has no timeless parameter threshold. More importantly, the label does not describe a complete assistant product. Many LLMs use Transformer-based architectures, but architecture and training objective remain separate concepts. The original Transformer was introduced for sequence-to-sequence tasks including machine translation; later systems adapted its components in different ways. Original Transformer paper

Training a model to predict language can produce abilities useful across many tasks. Additional training can shape instruction following and other assistant behavior. One published recipe combines supervised demonstrations with reinforcement learning from human feedback. That recipe is an example, not a claim that every assistant follows the same process. InstructGPT paper

A prompt can change behavior without changing weights. In the poster assistant, “Use a calm, scientific tone” becomes part of the current input. Providing examples in context can also change performance without a training update, as demonstrated in GPT-3’s few-shot evaluations. GPT-3 paper

Distinguish three things: stored parameters, information in the current context, and records saved by the surrounding product. Remembering something during a conversation is not sufficient evidence that model weights changed. Similarly, fluent prose is not evidence that a claim was checked against an external source.

8 7. Images, video, speech, and multimodal systems

The input and output medium provides another axis of classification. It does not uniquely identify an architecture or learning algorithm.

8.1 Image generation

An image classifier assigns a label to an image; an image generator produces image content. One influential generative approach is diffusion. During training, images are corrupted with noise and a network learns a denoising-related prediction. During generation, a sampling procedure repeatedly uses the trained network to transform noise into a sample. Those inference steps normally reuse fixed weights rather than retraining the network for each picture. Denoising diffusion paper

Latent diffusion performs this process in a learned compressed representation, then decodes the result into pixels. Text can condition the generation process. “Latent” here means an internal representation, not a hidden original photograph waiting to be uncovered. Latent diffusion paper

Image generation also includes adversarial methods, which train a generator against a discriminator, and flow-matching methods, which learn a field that guides transformations between distributions. These are examples of different generative approaches, not a complete catalog. GAN paper, flow-matching paper

Architecture cuts across these labels: a diffusion model can use a Transformer as its neural backbone. “Transformer” therefore does not imply “language model.” Diffusion Transformer paper

8.2 Video generation

Video adds time. Independently generating plausible frames does not ensure that an object keeps its identity or moves coherently. Video models must represent temporal relationships as well as visual appearance. Make-A-Video illustrates one published approach that extends text-to-image generation with temporal components and learning from video. It is a historical architectural example, not a description of every current video product. Make-A-Video paper

8.3 Speech and audio

Speech recognition maps recorded speech toward text. Text-to-speech goes in the other direction. Other audio tasks include generating music, separating sources, or classifying sounds. Their outputs, data, and evaluation criteria differ.

Whisper is a concrete speech-recognition example trained with audio and corresponding transcripts. WaveNet is a generative audio example that predicts waveform samples conditioned on earlier samples. Autoregressive generation therefore also applies beyond written language. Whisper paper, WaveNet paper

8.4 Multimodal models and multimodal products

A multimodal model works across more than one modality. A vision-language model might accept an image and a question and produce text. That capability does not automatically imply it can generate images.

For a documented specimen, Gemma 3’s 4B, 12B, and 27B variants combine a vision encoder with a language-model architecture; the 1B variant lacks that vision encoder. This is a specific published model-family distinction. Gemma 3 technical report, Sections 2–2.1

A multimodal product can also connect separate models: speech recognition, a text model, and speech synthesis, for example. Hearing a spoken answer alone cannot tell us whether one model or several produced it. We need documentation, configuration, or execution traces to distinguish these designs.

9 8. Assembling an agent around models

In the broad AI sense, an agent receives observations and selects actions. In LLM engineering, “agent” often describes a system that repeatedly uses a model to choose actions, calls tools, observes results, and continues toward a goal. Usage varies. A useful distinction is between a predefined workflow and a loop in which the model helps choose the next step. Anthropic’s workflow and agent distinction

Return to the astronomy poster. Here is one possible design, created for this lesson. It makes no claim about the hidden internals of a particular commercial assistant.

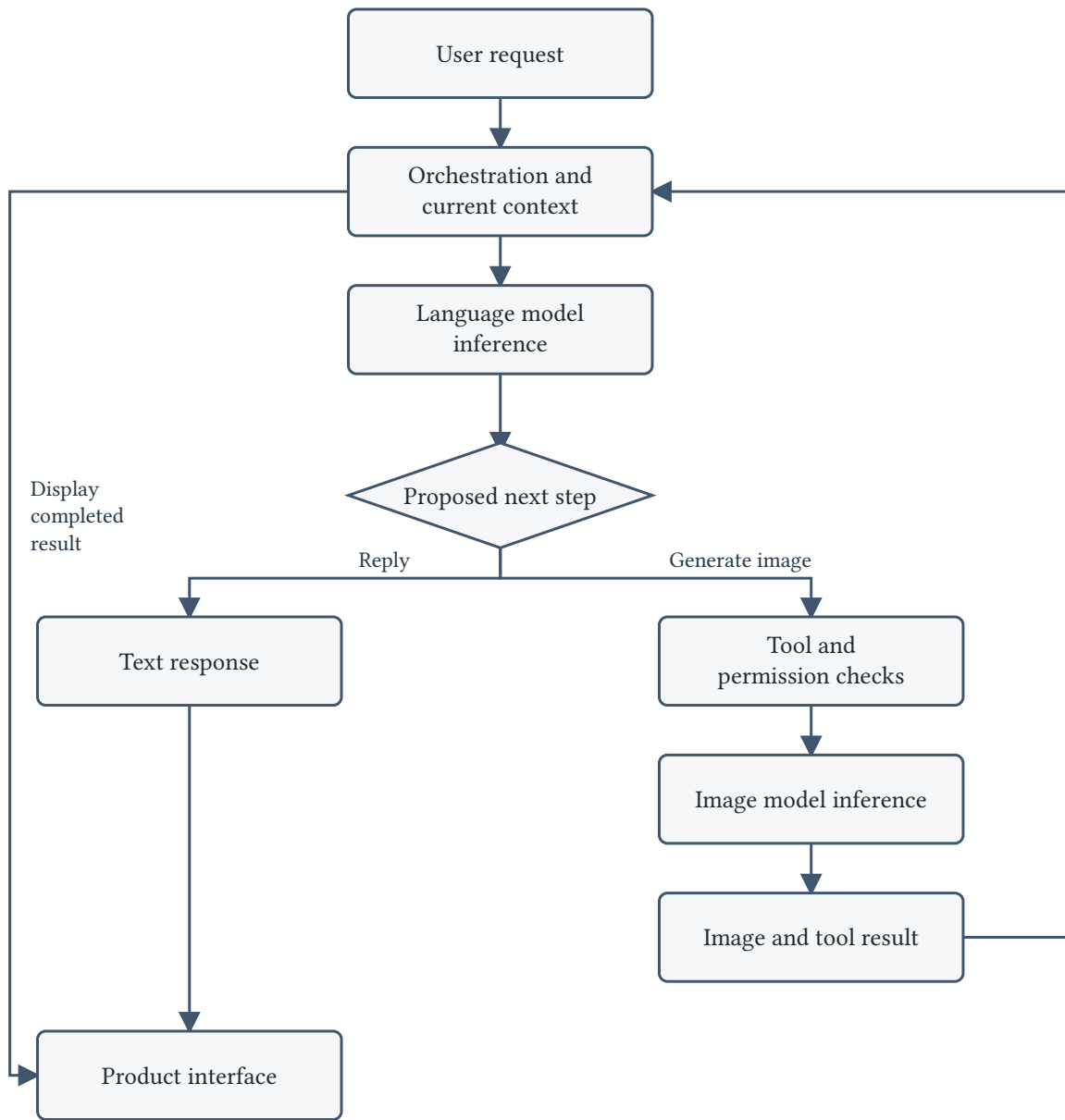


Figure 1. An illustrative language-model-plus-image-tool system. Each model call runs an architecture with trained weights in an inference runtime. The loop connecting calls belongs to orchestration.

Prose alternative: The product receives the request and supplies it to an orchestrator. The orchestrator prepares context for the language model. The model proposes either a response or an image-generation action. Software checks the action and, if permitted, calls the image model. The resulting image and tool status become an observation available for another step. The interface displays the completed output.

The LLM might turn “astronomy night” into a more detailed image prompt specifying a telescope, a dark sky, and space for event information. In this design, the LLM produces instructions; another

model executes the image-generation task. The orchestration layer must actually dispatch the call. Text that looks like a tool request has no external effect by itself.

If image generation fails, the model could receive the failure, revise the request, or explain the problem. ReAct is an early published example of interleaving language-model reasoning with actions and observations. The surrounding execution environment still determines which actions are available. ReAct paper

An operational agent also needs stopping conditions and limits. “Stop when the requested artifact exists,” “stop after the budget is exhausted,” and “ask before publishing” govern different risks. A model’s ability to propose publication does not grant permission to publish. Tools, permissions, and runtime checks help determine what can actually happen.

💡 Lab recommended here

Work through Lab 01 — Map AI Systems. Classify systems along several axes and separate models from surrounding software. Record your initial predictions before checking the supplied evidence. Pay special attention to claims that a user interface alone cannot establish.

10 9. Use a map, not a single family tree

A useful system description answers several questions at once:

- **Task and modality:** What goes in, and what comes out?
- **Mechanism:** Rules, search, learned prediction, generation, or a combination?
- **Learning signal:** Labels, derived targets, structure in data, or rewards?
- **Model family:** A tree, linear model, neural architecture, or no trained model?
- **System boundary:** One model call, a workflow, an agent loop, or a complete product?
- **Evidence:** Which details are documented, observed, inferred, or unknown?

An image-capable coding agent could be neural, Transformer-based, multimodal, trained through several learning stages, and embedded in tool-using orchestration. All of those descriptions can be true simultaneously.

This map is enough to narrow our focus. Next, we examine the representations that let a language model turn text into computation.

11 Check your understanding

Answer before reading the discussion below.

1. A route finder uses a hand-written map and A*. Must it have trained weights?
2. Two applications use the same model checkpoint. One retrieves documents and one does not. Must their answers be equally informed?
3. An image generator takes 30 sampling steps. Does that establish that it performed 30 training updates?

4. A text model was trained using RL, then deployed with fixed weights and no tools. Is it necessarily a tool-using agent?
5. A product returns both text and pictures. What evidence would distinguish one multimodal generator from a language model connected to an image tool?

11.1 Discussion

1. No. Search can operate on explicit representations without a learned model.
2. No. Retrieved information can change the supplied context even when weights stay fixed.
3. No. Sampling can repeatedly evaluate fixed weights. Training updates must be established separately.
4. No. Its training method does not determine its deployed orchestration or available actions.
5. Model documentation, system documentation, and execution traces can help. The visible output alone does not settle the architecture.

For the earlier checks: the optimizer could schedule back-to-back events without enough preparation time; denoising targets can be constructed from the training data and known added noise. Both answers concern the design of the learning or optimization problem, not the marketing label attached to the system.

12 More Learning

Choose the resource that addresses your biggest remaining question; these are optional.

- CLIPS documentation — Read the opening of the User’s Guide to see explicit facts and rules in a real expert-system language. No neural-network background is needed.
- OR-Tools introduction — Compare routing, scheduling, and constraint solving. Identify the decisions, objective, and constraints in one example.
- scikit-learn Getting Started — Follow the separation between fitting a model and using it to predict. Notice where evaluation enters the workflow.
- AlphaZero paper — Read the algorithm description first. Locate the neural network, tree search, self-play data, and training objective as separate components.
- Latent diffusion paper — Read the abstract and introduction for an example of separating image representation from generative computation. The derivations are a later side trail.
- ReAct paper — Follow one action-and-observation example. Ask what the model writes and what the environment actually executes.

13 References

Primary papers and official documentation support the claims linked above. The numerical examples and system diagram are original instructional examples, not reported experiments. Sources checked 7 October 2026.

- CLIPS. Official documentation.
- Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths.
- Google OR-Tools. Introduction.
- DEAP. Operators and Algorithms.

- scikit-learn. Supervised learning, Getting Started, neural networks, and clustering.
- Devlin, J., et al. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
- OpenAI Spinning Up. Key Concepts in RL.
- Silver, D., et al. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.
- Brown, T. B., et al. (2020). Language Models are Few-Shot Learners.
- Vaswani, A., et al. (2017). Attention Is All You Need.
- Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising Diffusion Probabilistic Models.
- Rombach, R., et al. (2022). High-Resolution Image Synthesis with Latent Diffusion Models.
- Goodfellow, I. J., et al. (2014). Generative Adversarial Networks.
- Lipman, Y., et al. (2023). Flow Matching for Generative Modeling.
- Peebles, W., and Xie, S. (2023). Scalable Diffusion Models with Transformers.
- Singer, U., et al. (2022). Make-A-Video: Text-to-Video Generation without Text-Video Data.
- Radford, A., et al. (2022). Robust Speech Recognition via Large-Scale Weak Supervision.
- van den Oord, A., et al. (2016). WaveNet: A Generative Model for Raw Audio.
- Gemma Team (2025). Gemma 3 Technical Report.
- Anthropic (2024). Building effective agents.
- Yao, S., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models.