**Most Popular Common Software Design Patterns**

[**Creational Design Patterns**](https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns#creational-design-patterns)

There are 5 design patterns in the creational design patterns category.

| **Pattern Name** | **Description** |
| --- | --- |
| [Singleton](https://www.digitalocean.com/community/tutorials/java-singleton-design-pattern-best-practices-examples) | The singleton pattern restricts the initialization of a class to ensure that only one instance of the class can be created. |
| [Factory](https://www.digitalocean.com/community/tutorials/factory-design-pattern-in-java) | The factory pattern takes out the responsibility of instantiating a object from the class to a Factory class. |
| [Abstract Factory](https://www.digitalocean.com/community/tutorials/abstract-factory-design-pattern-in-java) | Allows us to create a Factory for factory classes. |
| [Builder](https://www.digitalocean.com/community/tutorials/builder-design-pattern-in-java) | Creating an object step by step and a method to finally get the object instance. |
| [Prototype](https://www.digitalocean.com/community/tutorials/prototype-design-pattern-in-java) | Creating a new object instance from another similar instance and then modify according to our requirements. |

[**Structural Design Patterns**](https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns#structural-design-patterns)

There are 7 structural design patterns defined in the Gangs of Four design patterns book.

| **Pattern Name** | **Description** |
| --- | --- |
| [Adapter](https://www.digitalocean.com/community/tutorials/adapter-design-pattern-java) | Provides an interface between two unrelated entities so that they can work together. |
| [Composite](https://www.digitalocean.com/community/tutorials/composite-design-pattern-in-java) | Used when we have to implement a part-whole hierarchy. For example, a diagram made of other pieces such as circle, square, triangle, etc. |
| [Proxy](https://www.digitalocean.com/community/tutorials/proxy-design-pattern) | Provide a surrogate or placeholder for another object to control access to it. |
| [Flyweight](https://www.digitalocean.com/community/tutorials/flyweight-design-pattern-java) | Caching and reusing object instances, used with immutable objects. For example, string pool. |
| [Facade](https://www.digitalocean.com/community/tutorials/facade-design-pattern-in-java) | Creating a wrapper interfaces on top of existing interfaces to help client applications. |
| [Bridge](https://www.digitalocean.com/community/tutorials/bridge-design-pattern-java) | The bridge design pattern is used to decouple the interfaces from implementation and hiding the implementation details from the client program. |
| [Decorator](https://www.digitalocean.com/community/tutorials/decorator-design-pattern-in-java-example) | The decorator design pattern is used to modify the functionality of an object at runtime. |

[**Behavioral Design Patterns**](https://www.digitalocean.com/community/tutorials/gangs-of-four-gof-design-patterns#behavioral-design-patterns)

There are 11 behavioral design patterns defined in the GoF design patterns.

| **Pattern Name** | **Description** |
| --- | --- |
| [Template Method](https://www.digitalocean.com/community/tutorials/template-method-design-pattern-in-java) | used to create a template method stub and defer some of the steps of implementation to the subclasses. |
| [Mediator](https://www.digitalocean.com/community/tutorials/mediator-design-pattern-java) | used to provide a centralized communication medium between different objects in a system. |
| [Chain of Responsibility](https://www.digitalocean.com/community/tutorials/chain-of-responsibility-design-pattern-in-java) | used to achieve loose coupling in software design where a request from the client is passed to a chain of objects to process them. |
| [Observer](https://www.digitalocean.com/community/tutorials/observer-design-pattern-in-java) | useful when you are interested in the state of an object and want to get notified whenever there is any change. |
| [Strategy](https://www.digitalocean.com/community/tutorials/strategy-design-pattern-in-java-example-tutorial) | Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime. |
| [Command](https://www.digitalocean.com/community/tutorials/command-design-pattern) | Command Pattern is used to implement lose coupling in a request-response model. |
| [State](https://www.digitalocean.com/community/tutorials/state-design-pattern-java) | State design pattern is used when an Object change it’s behavior based on it’s internal state. |
| [Visitor](https://www.digitalocean.com/community/tutorials/visitor-design-pattern-java) | Visitor pattern is used when we have to perform an operation on a group of similar kind of Objects. |
| [Interpreter](https://www.digitalocean.com/community/tutorials/interpreter-design-pattern-java) | defines a grammatical representation for a language and provides an interpreter to deal with this grammar. |
| [Iterator](https://www.digitalocean.com/community/tutorials/iterator-design-pattern-java) | used to provide a standard way to traverse through a group of Objects. |
| [Memento](https://www.digitalocean.com/community/tutorials/memento-design-pattern-java) | The memento design pattern is used when we want to save the state of an object so that we can restore later on. |

### Machine Learning Design Patterns Reference (2024)

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Design Pattern** | **Problem Solved** | **Solution** |
| [**Data Representation**](https://medium.com/geekculture/data-representation-design-patterns-d0714debb23f) | Hashed Feature (??) | Problems associated with categorical features such as incomplete vocabulary, model size due to cardinality, and cold start. | Bucket a deterministic and portable hash of string representation and accept the trade-off of collisions in the data representation. |
|  | **Embeddings (4)** | **High-cardinality features where closeness relationships are important to preserve.** | **Learn a data representation that maps high-cardinality data into a lower-dimensional space in such a way that the information relevant to the learning problem is preserved.** |
|  | **Feature Cross (14)** | **Model complexity insufficient to learn feature relationships.** | **Help models learn relationships between inputs faster by explicitly making each combination of input values a separate feature.** |
|  | Multimodal Input (23) | How to choose between several potential data representations. | Concatenate all the available data representations. |
| [**Problem Representation**](https://medium.datadriveninvestor.com/problem-representational-design-pattern-dea210e583b2) | Reframing (21) | Several problems including confidence for numerical prediction, ordinal categories, restricting prediction range, and multitask learning. | Change the representation of the output of a machine learning problem; for example, representing a regression problem as a classification (and vice versa). |
|  | Multilabel (22) | More than one label applies to a given training example. | Encode the label using a multi-hot array, and use k sigmoids as the output layer. |
|  | **Ensembles (3)** | **Bias-variance trade-off on small- and medium-scale problems.** | **Combine multiple machine learning models and aggregate their results to make predictions.** |
|  | Cascade (26) | Maintainability or drift issues when a machine learning problem is broken into a series of ML problems. | Treat an ML system as a unified workflow for the purposes of training, evaluation, and prediction. |
|  | Neutral Class (27) | The class label for some subset of examples is essentially arbitrary. | Introduce an additional label for a classification model, disjoint from the current labels. |
|  | **Rebalancing (15)** | **Heavily imbalanced data.** | **Downsample, upsample, or use a weighted loss function depending on different considerations.** |
| **Patterns That Modify Model Training** | Useful Overfitting (28) | Using machine learning methods to learn a physics-based model or dynamical system. | Forgo the usual generalization techniques in order to intentionally overfit on the training dataset. |
|  | Checkpoints (17) | Lost progress during long-running training jobs due to machine failure. | Store the full state of the model periodically, so that partially trained models are available and can be used to resume training from an intermediate point, instead of starting from scratch. |
|  | **Transfer Learning (2)** | **Lack of large datasets that are needed to train complex machine learning models.** | **Take part of a previously trained model, freeze the weights, and use these nontrainable layers in a new model that solves a similar problem.** |
|  | Distribution Strategy (18) | Training large neural networks can take a very long time, which slows experimentation. | Carry the training loop out at scale over multiple workers, taking advantage of caching, hardware acceleration, and parallelization. |
|  | **Hyperparameter Tuning (1)** | **How to determine the optimal hyperparameters of a machine learning model.** | **Insert the training loop into an optimization method to find the optimal set of model hyperparameters.** |
| **Resilience** | **Stateless Serving Function (10)** | **Production ML system must be able to synchronously handle thousands to millions of prediction requests per second.** | **Export the machine learning model as a stateless function so that it can be shared by multiple clients in a scalable way.** |
|  | **Batch Serving (12)** | **Carrying out model predictions over large volumes of data using an endpoint that is designed to handle requests one at a time will overwhelm the model.** | **Use software infrastructure commonly used for distributed data processing to carry out inference asynchronously on a large number of instances at once.** |
|  | **Continued Model Evaluation (11)** | **Model performance of deployed models degrades over time either due to data drift, concept drift or other changes to the pipelines which feed data to the model.** | **Detect when a deployed model is no longer fit-for-purpose by continually monitoring model predictions and evaluating model performance.** |
|  | Two-Phase Predictions (19) | Large, complex models must be kept performant when they are deployed at the edge or on distributed devices. | Split the use case into two phases with only the simpler phase being carried out on the edge. |
|  | Keyed Predictions (20) | How to map the model predictions that are returned to the corresponding model input when submitting large prediction jobs. | Allow the model to pass through a client-supported key during prediction that can be used to join model inputs to model predictions. |
| **Reproducibility** | **Transform (9)** | **The inputs to a model must be transformed to create the features the model expects and that process must be consistent between training and serving.** | **Explicitly capture and store the transformations applied to convert the model inputs into features.** |
|  | **Repeatable Splitting (13)** | **When creating data splits, it's important to have a method that is lightweight and repeatable regardless of the programming language or random seeds.** | **Identify a column that captures the correlation relationship between rows and use the Farm Fingerprint hashing algorithm to split the available data into training, validation, and testing datasets.** |
|  | Bridged Schema (24) | As new data becomes available, any changes to the data schema could prevent using both the new and old data for retraining. | Adapt the data from its older, original data schema to match the schema of the newer, better data. |
|  | Windowed Inference (25) | Some models require an ongoing sequence of instances to run inference, or features must be aggregated across a time window in such a way that avoids training-serving skew. | Externalize the model state and invoke the model from a stream analytics pipeline to ensure that features calculated in a dynamic, time-dependent way can be correctly repeated between training and serving. |
|  | **Workflow Pipeline (5)** | **When scaling the ML workflow, run trials independently and track performance for each step of the pipeline.** | **Make each step of the ML workflow a separate, containerized service that can be chained together to make a pipeline that can be run with a single REST API call.** |
|  | **Feature Store (8)** | **The ad hoc approach to feature engineering slows model development and leads to duplicated effort between teams as well as work stream inefficiency.** | **Create a feature store, a centralized location to store and document feature datasets that will be used in building machine learning models and can be shared across projects and teams.** |
|  | **Model Versioning (7)** | **It is difficult to carry out performance monitoring and split test model changes while having a single model in production or to update models without breaking existing users.** | **Deploy a changed model as a microservice with a different REST endpoint to achieve backward compatibility for deployed models.** |
| **Responsible AI** | Heuristic Benchmark (29) | Explaining model performance using complicated evaluation metrics does not provide the intuition that business decision makers need. | Compare an ML model against a simple, easy-to-understand heuristic. |
|  | **Explainable Predictions (6)** | **Sometimes it is necessary to know why a model makes certain predictions either for debugging or for regulatory and compliance standards.** | **Apply model explainability techniques to understand how and why models make predictions and improve user trust in ML systems.** |
|  | Fairness Lens (16) | Bias can cause machine learning models to not treat all users equally and can have adverse effects on some populations. | Use tools to identify bias in datasets before training and evaluate trained models through a fairness lens to ensure model predictions are equitable across different groups of users and different scenarios. |

**Design Patterns for Machine Learning by Popularity (2024). The popularity for 2023 are shown in brackets.**

1. Hyperparameter Tuning (1)
2. Transfer Learning (2)
3. Ensembles (3)
4. Embeddings (4)
5. Workflow Pipeline (7)
6. Explainable Predictions (8)
7. Model Versioning (9)
8. Feature Store (16)
9. Transform (15)
10. Stateless Serving Function (10)
11. Continued Model Evaluation (11)
12. Repeatable Splitting (12)
13. Feature Cross (6)
14. Batch Serving (14)
15. Rebalancing (5)
16. Fairness Lens (29)
17. Checkpoints (19)
18. Distribution Strategy (21)
19. Two-Phase Predictions (20)
20. Keyed Predictions (22)
21. Reframing (17)
22. Multilabel (18)
23. Multimodal Input (13)
24. Bridged Schema (24)
25. Windowed Inference (25)
26. Cascade (23)
27. Neutral Class (26)
28. Useful Overfitting (27)
29. Heuristic Benchmark (28)