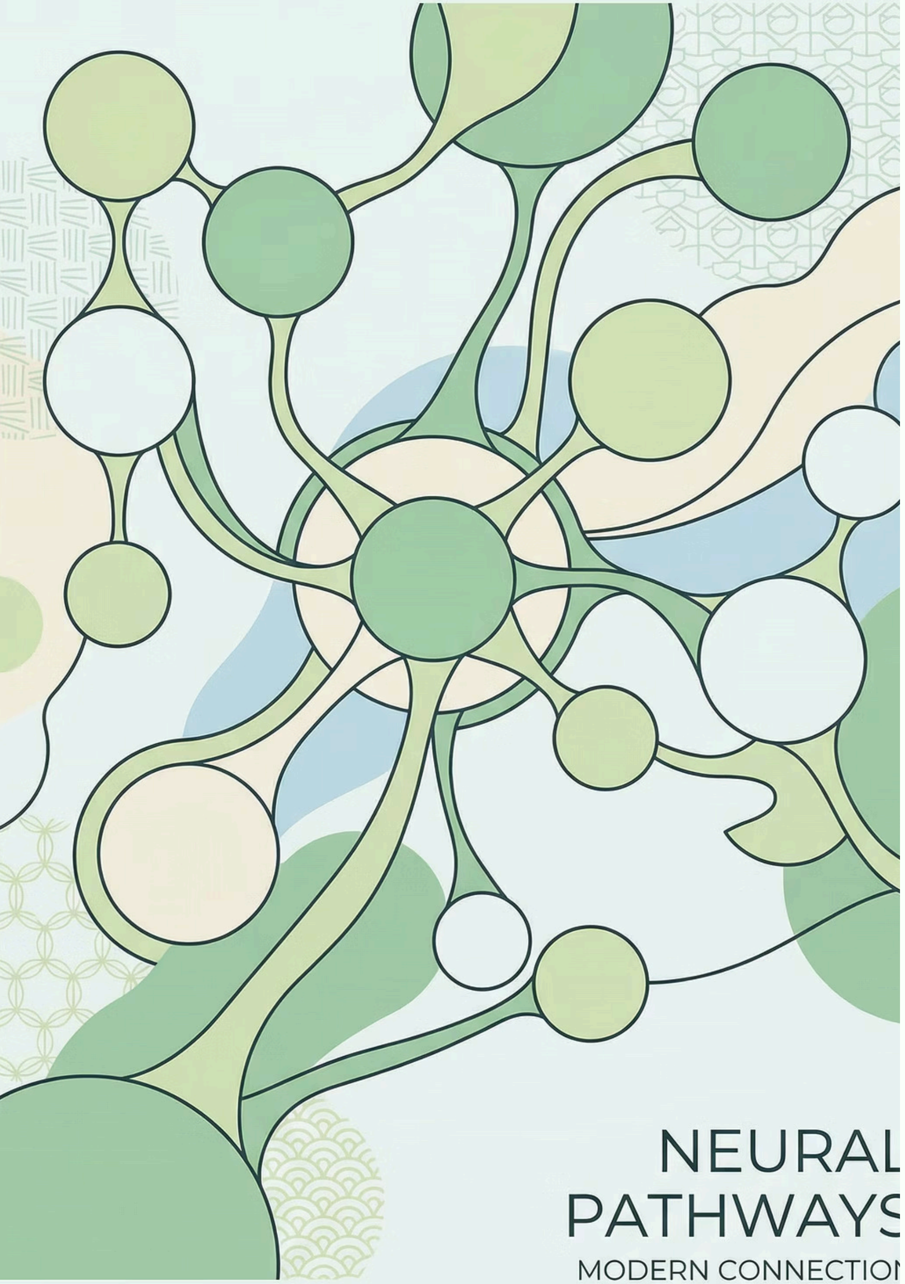
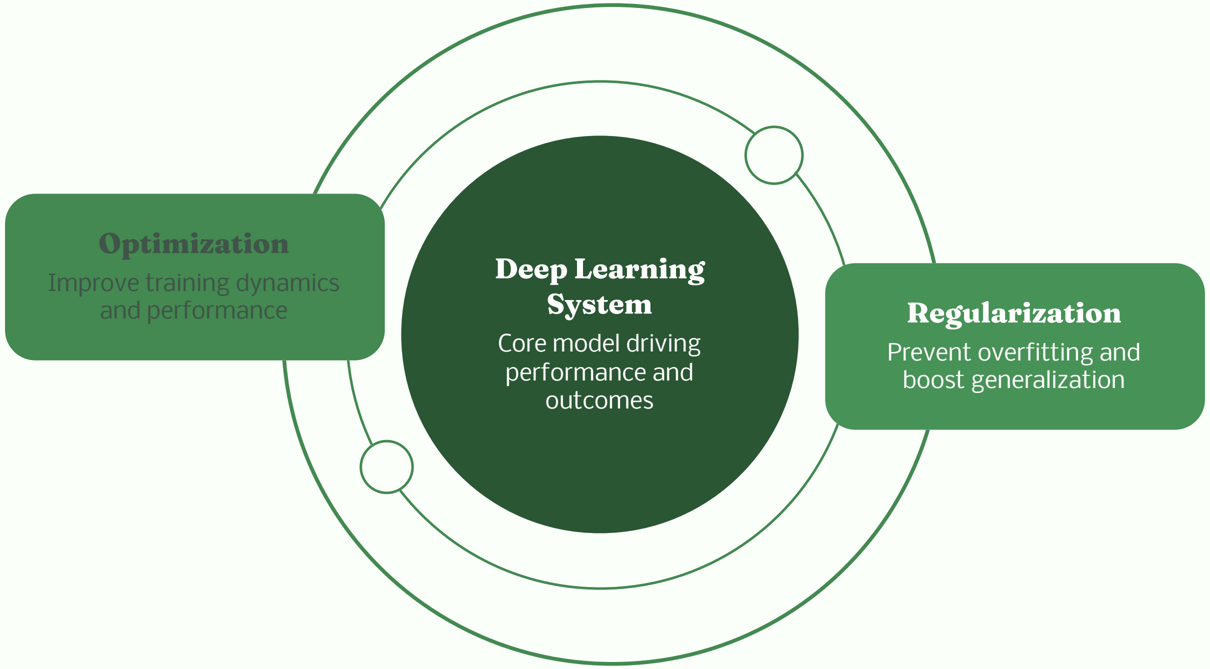


Optimisation and Regularisation in Deep Learning

Essential techniques to improve model performance, prevent overfitting, and ensure robust generalisation across production systems–from recommendation engines to computer vision applications.



Made with GAMMA

The Balance: Overfitting vs Underfitting

Overfitting

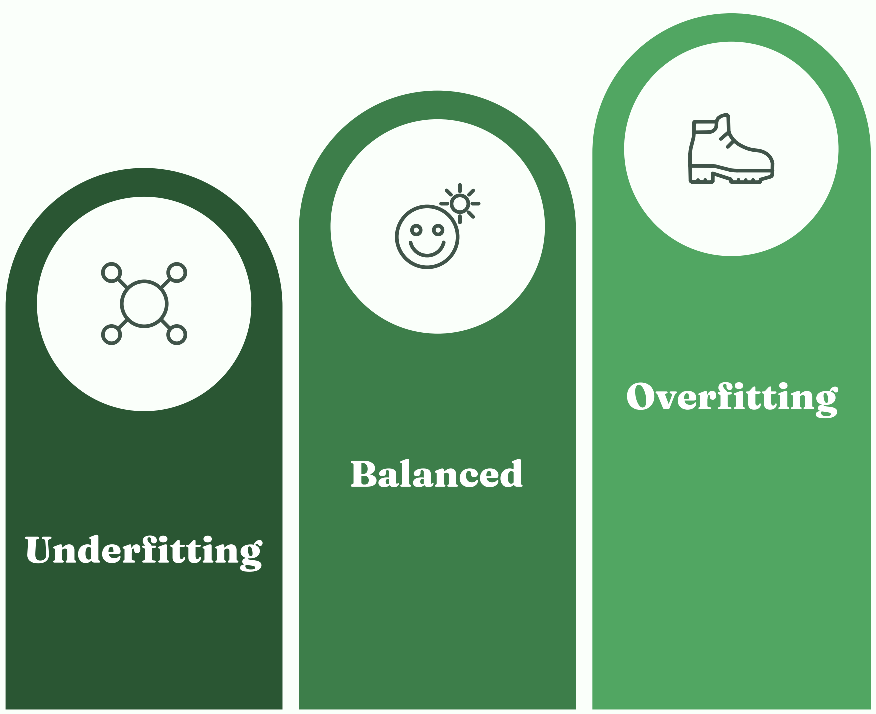
The model learns noise and irrelevant details from training data, performing brilliantly on examples it has seen but failing on new, unseen data.

Real-world example: A recommendation algorithm that memorises specific user interactions but cannot predict preferences for new users or novel content.

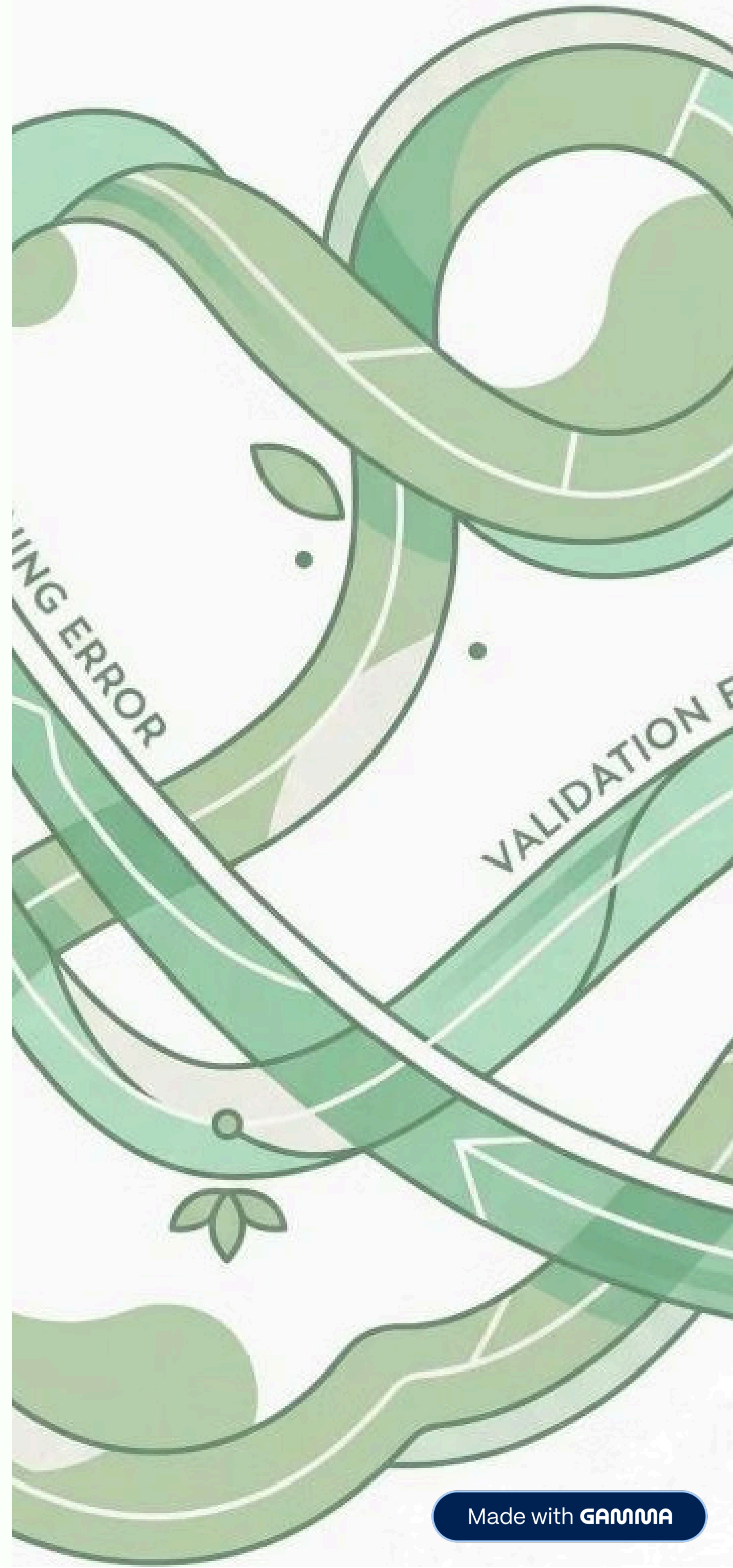
Underfitting

The model is too simple to capture the underlying patterns in the data, lacking the capacity to learn important relationships and trends.

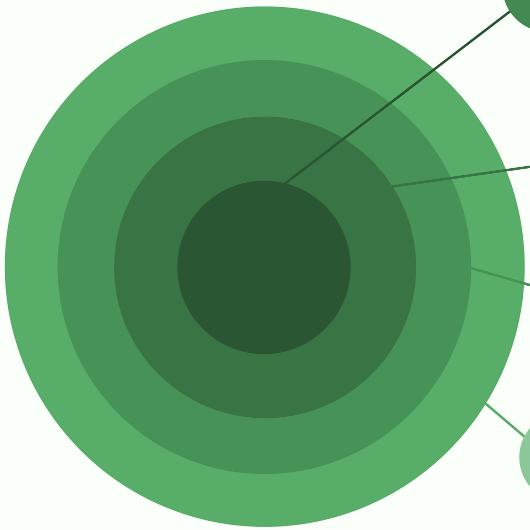
Real-world example: Using simple linear regression to predict house prices when features like rooms, area, and location exhibit complex non-linear behaviour.



The goal is achieving balance: capturing essential trends (low bias) whilst maintaining strong generalisation to new data (low variance).



Finding the Sweet Spot

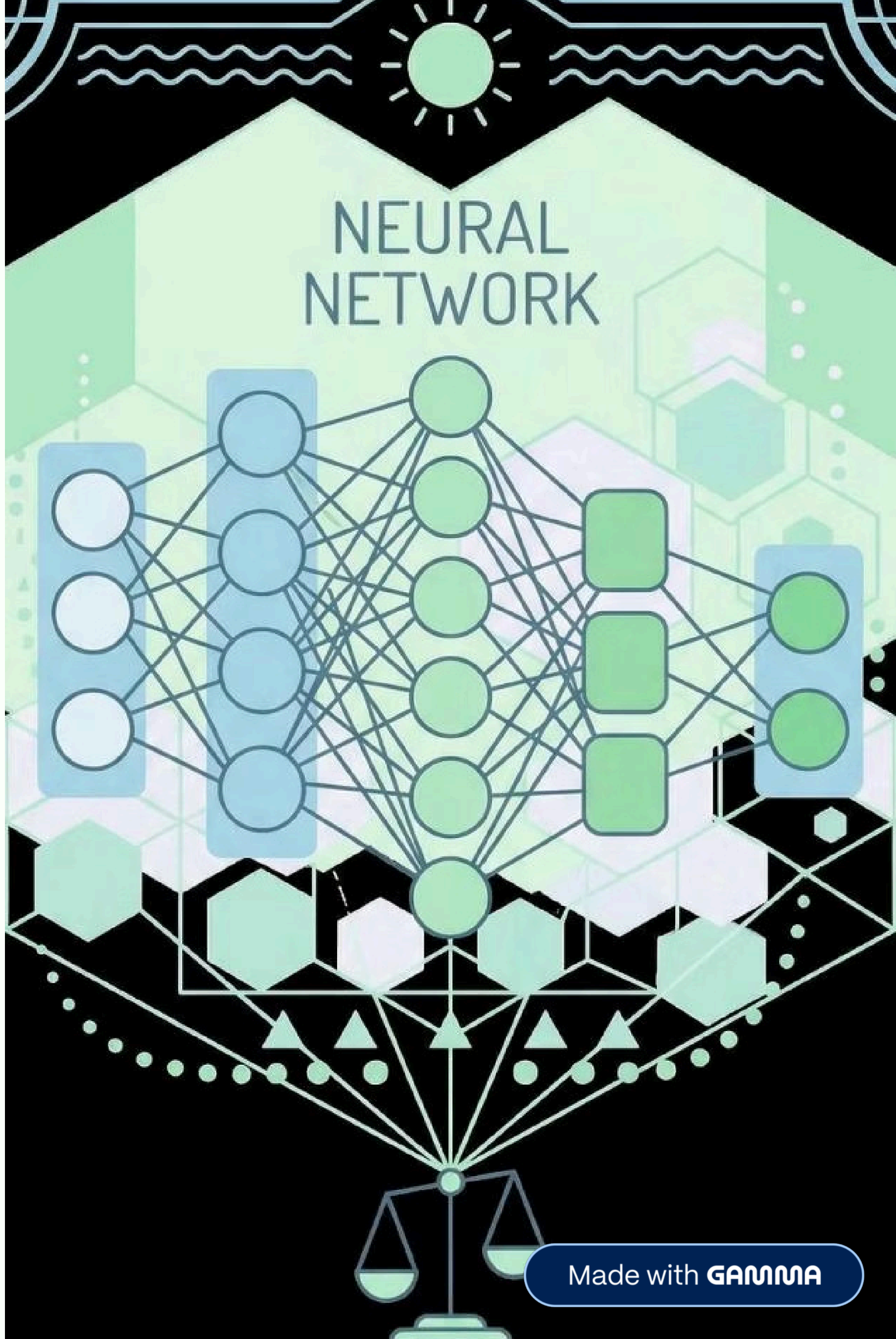


Sweet Spot
Optimal model performance.

Low Bias
Accurate on average.

Low Variance
Consistent across data.

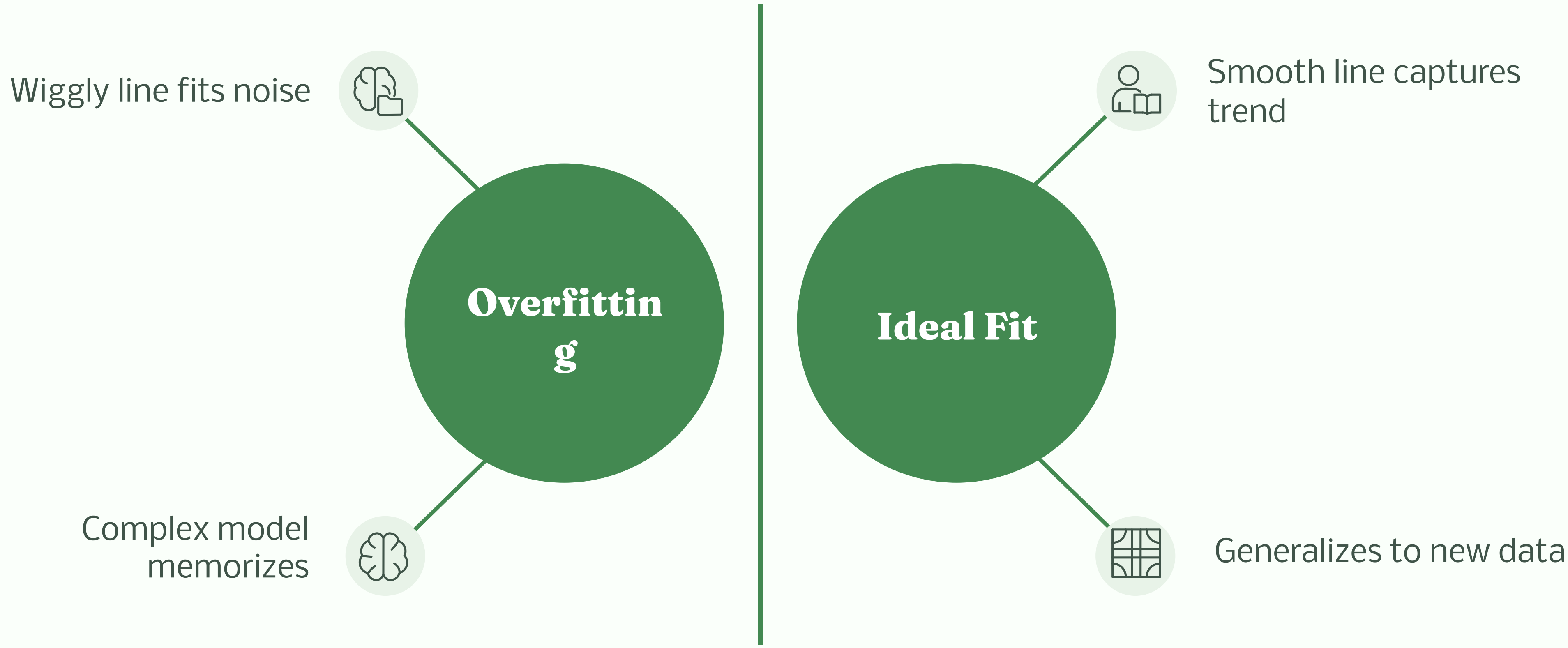
Strong Generalization
Performs on new data.



Regularization: Preventing Overfitting

Techniques to prevent models from memorizing training data instead of learning general patterns.

Deep neural networks, with their vast number of parameters and complex architectures, possess an immense capacity to learn. However, this power comes with a risk: they can easily memorize the training data, including noise and irrelevant details, rather than learning the underlying general patterns. This leads to poor performance on new, unseen data–a phenomenon known as overfitting.



High Model Complexity

Too many parameters allow models to fit noise in training data.

Limited Training Data

Insufficient data makes it easier for models to simply memorize examples.

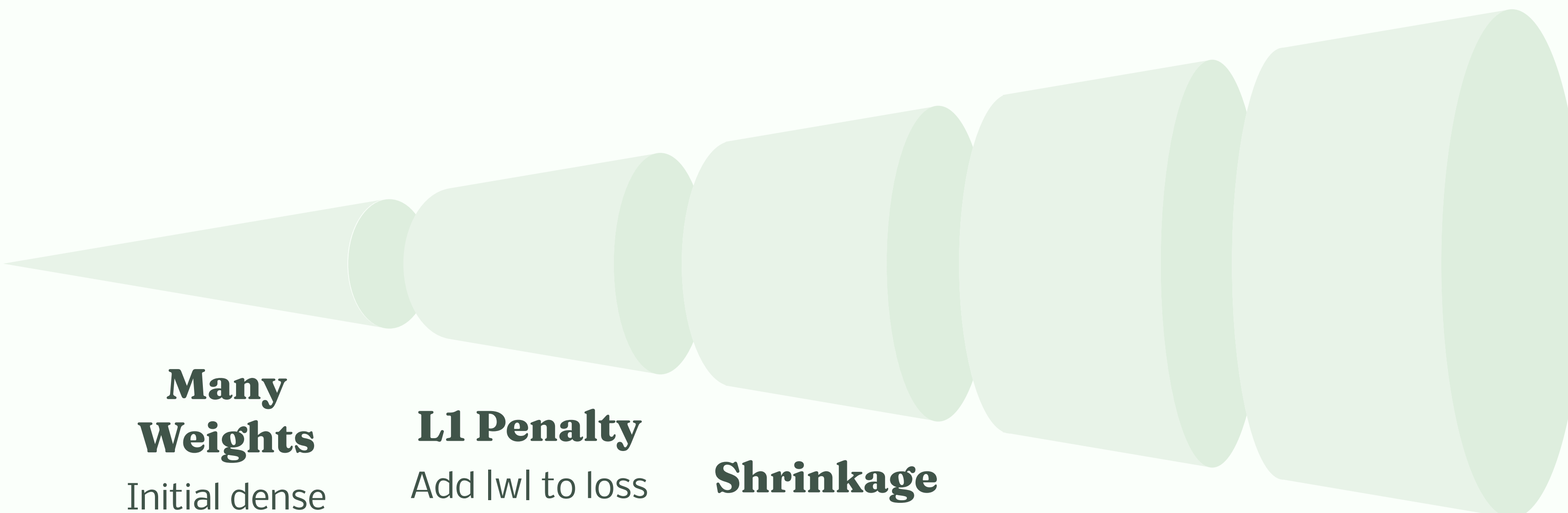
Absence of Constraints

Without penalties, weights can grow arbitrarily large, increasing sensitivity to minor fluctuations.

L1 Regularization (Lasso)

L1 Regularization, also known as Lasso, is a technique used to prevent overfitting by adding a penalty proportional to the absolute value of the magnitude of coefficients to the loss function.

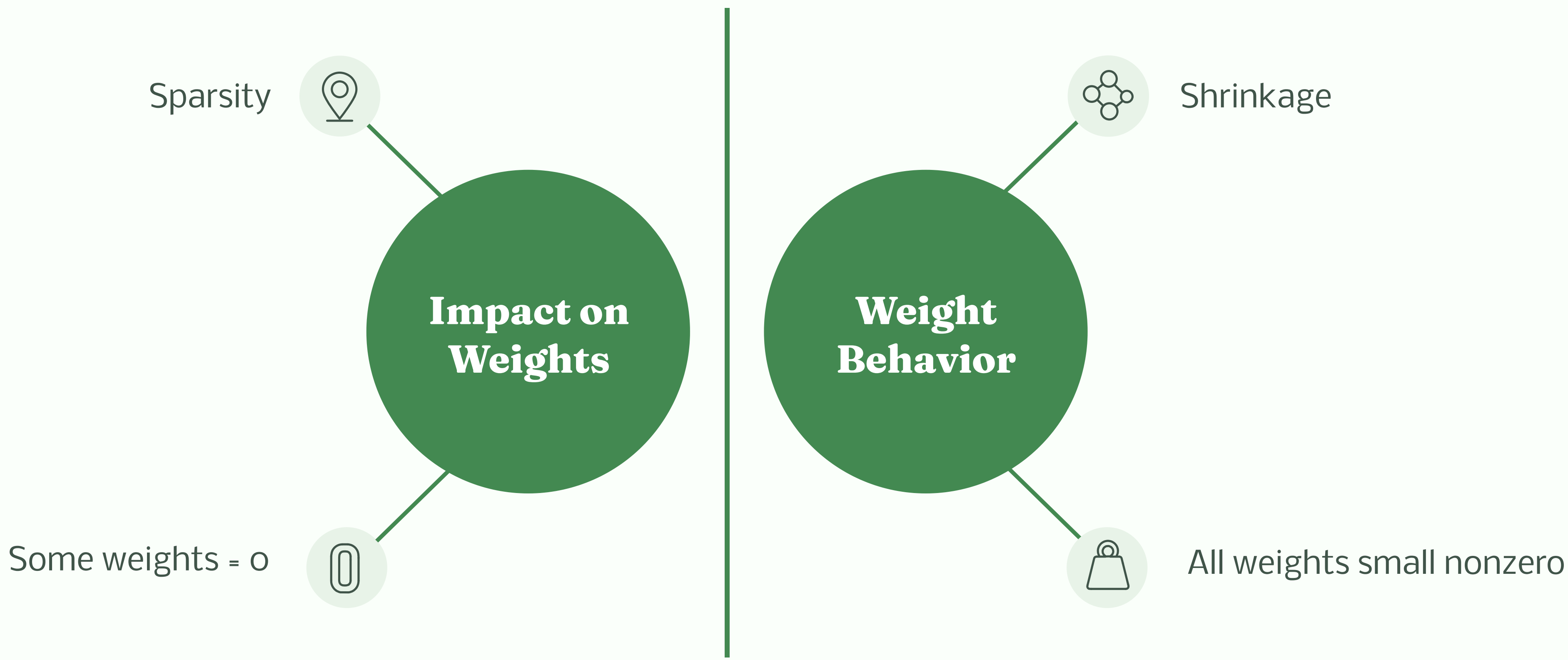
- Adds the absolute value of weights to the loss function
- Formula: Loss = Original Loss + $\lambda \Sigma |w|$
- Encourages sparse weights (many weights become zero)
- Useful for feature selection



This regularization method is particularly effective when dealing with high-dimensional datasets where many features might be irrelevant. By forcing some weights to zero, L1 regularization inherently performs feature selection, simplifying the model and improving its interpretability.

L2 Regularization (Ridge / Weight Decay)

- Adds the squared value of weights to the loss
- Formula: Loss = Original Loss + $\lambda \Sigma w^2$
- Penalizes large weights
- Most commonly used regularization in deep learning



Early Stopping

A critical regularization technique that stops model training when the performance on a validation dataset begins to degrade, preventing overfitting.

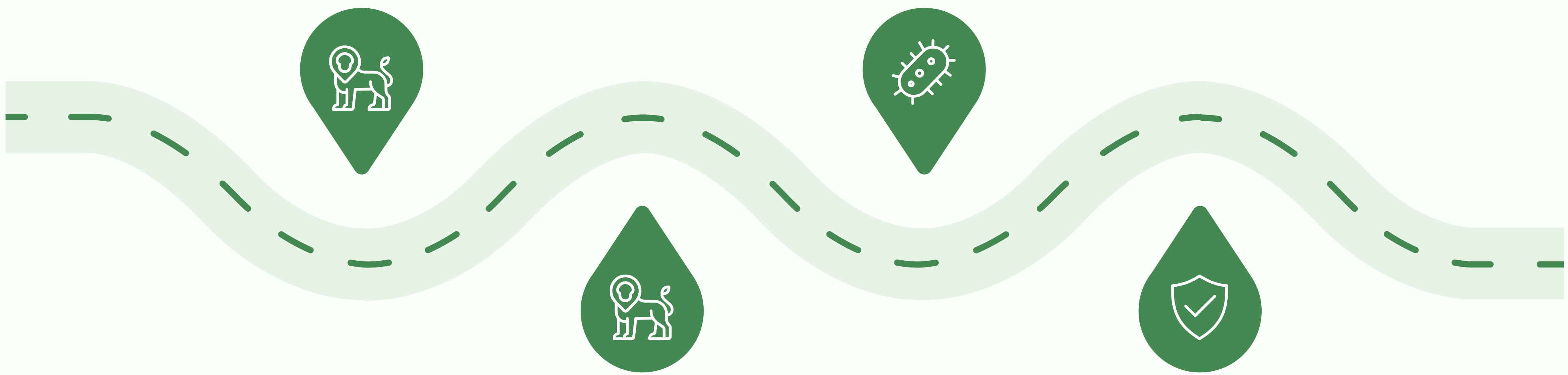
- **Stops training when validation loss stops improving**
- **Prevents over-training**
- **Simple yet effective technique**

Training Loss

Steadily decreases

Early Stopping

Stop at validation minimum



Validation Loss

Decreases then rises

Overfitting

Avoid by stopping early

This method monitors a model's performance on a separate validation dataset during training. When the validation loss, which is a measure of the model's error on unseen data, stops improving or starts to increase, it indicates that the model is beginning to overfit the training data. At this 'early stopping point', training is halted to preserve the model's ability to generalize to new data effectively.

Data Augmentation

Artificially increases training data (rotations, flips, noise, etc.)

- Common in computer vision
- Helps model generalize better by exposing it to variations

Original Image

Base dog photo used for augmentation

Rotated 30°

Introduces viewpoint variation

Horizontally Flipped

Mimics mirrored perspectives

Increased Brightness

Simulates lighting changes

Added Random Noise

Builds robustness to artifacts

Effective Dataset

More diverse examples for training

Optimizers: Training the Model

Algorithms that update weights to minimize the loss function.

Optimizers are the driving force behind training machine learning models. They dictate how a model's internal parameters (weights and biases) are adjusted after each training iteration, based on the gradients computed from the loss function. This process is crucial for guiding the model from a random initial state to one that performs optimally on a given task.

01

Interpret Gradients

Optimizers use gradients to understand the landscape of the loss function, identifying the direction in which the loss decreases most rapidly.

02

Calculate Step Size

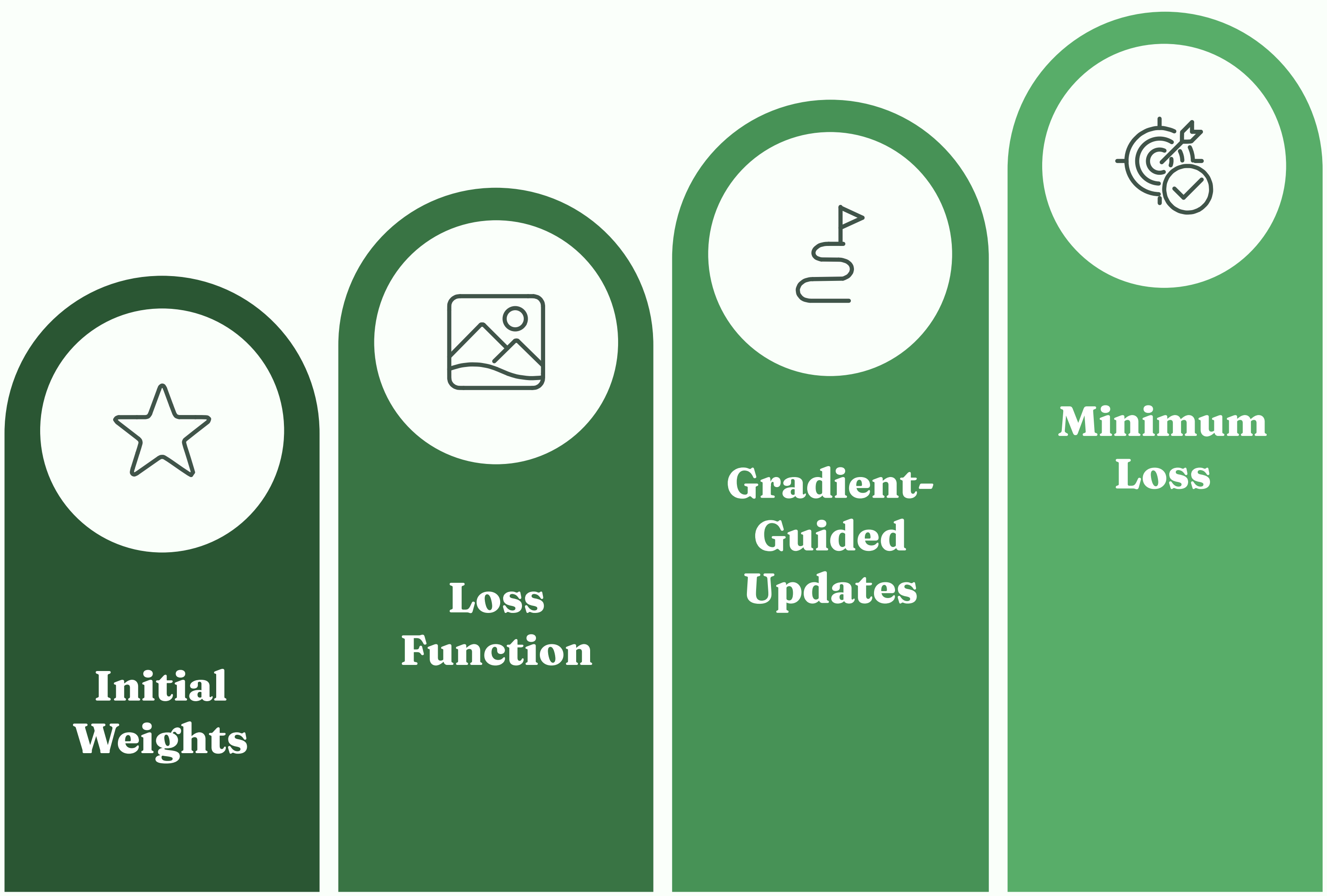
They determine the appropriate learning rate or step size for each weight update, balancing rapid convergence with avoiding overshooting the minimum.

03

Minimize Loss

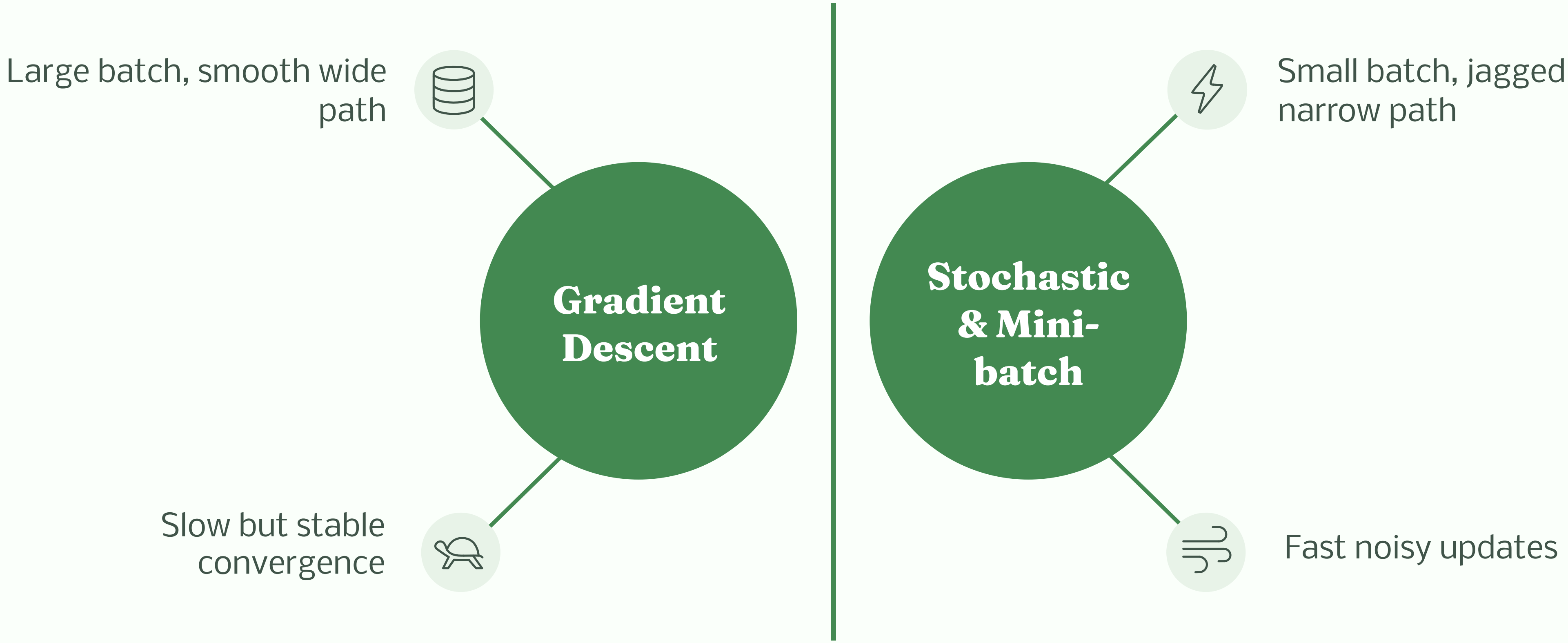
Through iterative adjustments, optimizers guide the model to find the set of weights that minimizes the discrepancy between predictions and actual targets.

The choice of optimizer significantly impacts training speed, stability, and the final performance of the model. Different optimizers employ various strategies to navigate the complex loss landscapes of deep neural networks, each with its own strengths and weaknesses.



Gradient Descent & SGD

- Gradient Descent: Updates weights using the full dataset, slow for large datasets
- Stochastic Gradient Descent (SGD): Updates weights using one sample at a time, faster but noisy updates
- Mini-batch Gradient Descent: Uses small batches of data, most commonly used in practice



Advanced Optimizers

Building upon basic gradient descent, advanced optimizers employ sophisticated strategies to navigate complex loss landscapes more efficiently and effectively.

SGD with Momentum

- Adds velocity to updates to speed up convergence
- Helps escape local minima

AdaGrad

- Adapts learning rate for each parameter
- Good for sparse data
- Learning rate keeps decreasing (can become too small)

RMSProp

- Fixes AdaGrad's learning rate problem
- Works well for RNNs

Adam (Most Popular)

- Combines Momentum + RMSProp
- Adaptive learning rates
- Fast convergence and easy to use

These optimizers have become fundamental tools in deep learning, allowing models to train faster and achieve better performance on a wide array of tasks.

SGD Momentum

Speed: adds velocity to updates.

AdaGrad

Sparsity: adapts per-parameter rates.

RMSProp

Adaptivity: scales by recent gradients.

Adam

Hybrid: combines momentum and RMSProp.

Regularization vs Optimizers: Key Differences

Aspect	Regularization	Optimizers
Purpose	Prevent overfitting by reducing model complexity	Minimize the loss function during training
Affects	Model generalization and interpretability	Training speed, stability, and convergence
Acts on	Loss function (penalty terms) / Model architecture (Dropout)	Model's internal weights and biases
Examples	L1, L2 (Weight Decay), Dropout, Early Stopping, Data Augmentation	SGD, Adam, RMSProp, AdaGrad, Momentum

In simple terms: Regularization controls how complex the model becomes, ensuring it learns general patterns rather than memorizing noise. Optimizers control how the model learns, guiding the adjustment of weights to efficiently reach an optimal solution.

L2 Regularisation (Ridge Regression)

L2 regularisation penalises large weights in the model, keeping it simpler and preventing overfitting by adding a penalty term to the loss function.

$$L_{new} = L_{original} + \lambda \sum_i w_i^2$$

The parameter λ controls regularisation strength—higher values impose stronger penalties on large weights.

Weight Shrinkage

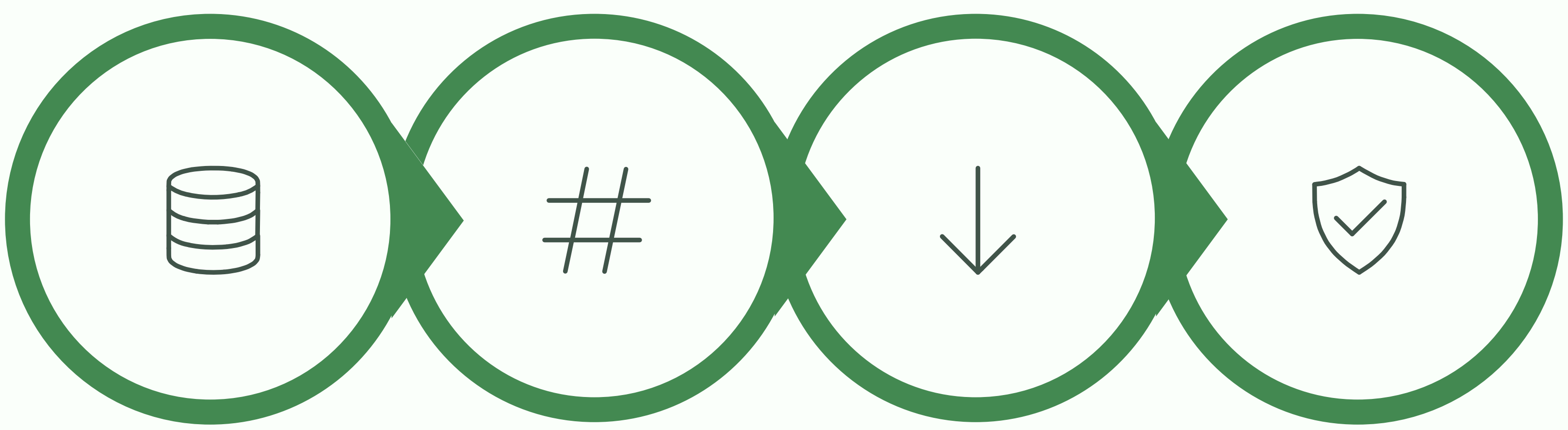
Reduces large weights without forcing them to zero, maintaining model expressiveness

Stability

Encourages stable, generalisable models that perform consistently

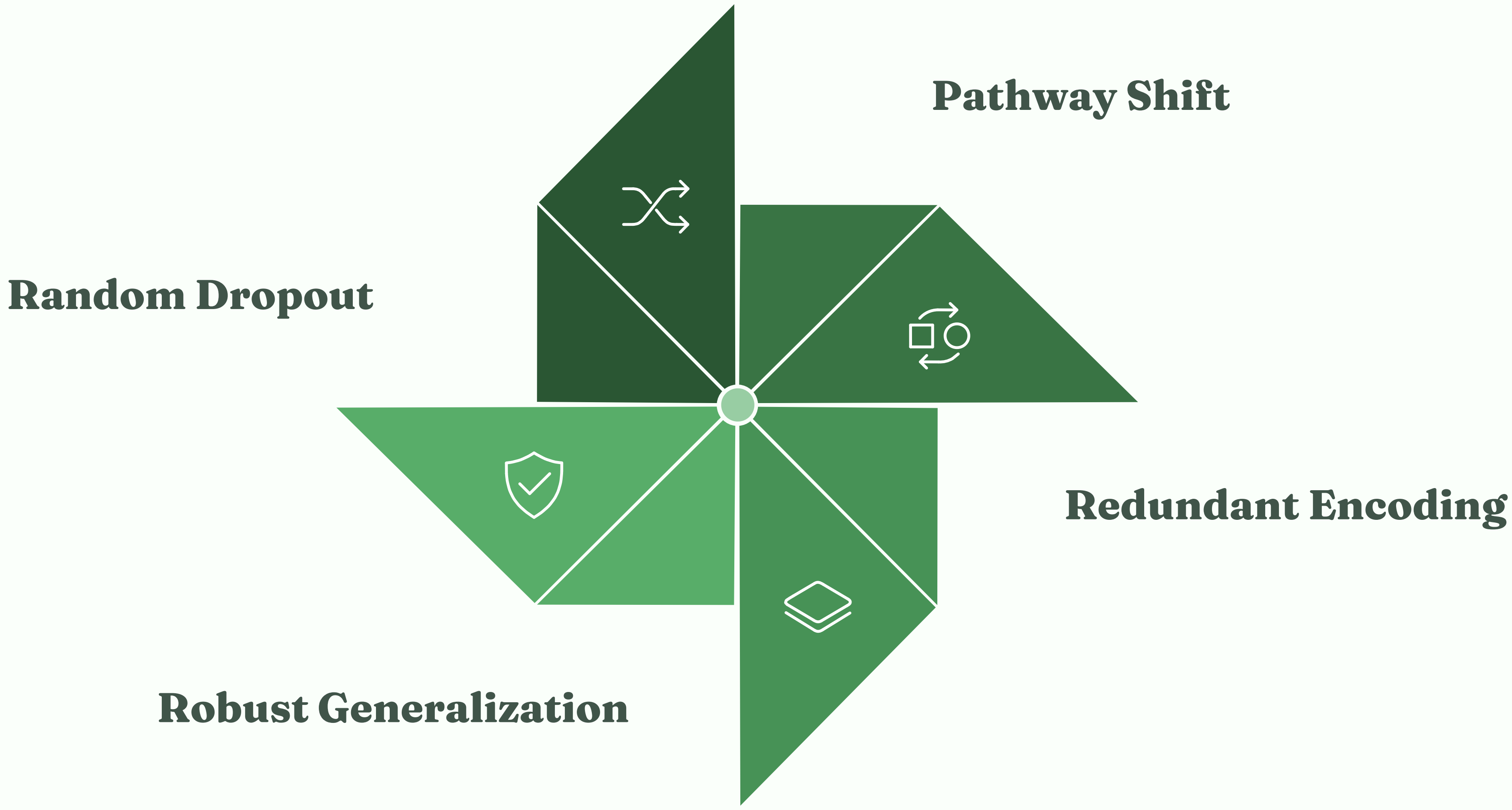
Production Example

In financial forecasting, L2 regularisation prevents over-reliance on volatile short-term market fluctuations



Initial Weights | **Penalty Applied** | **Gradual Shrinkage** | **Optimal Weights**

Dropout: Selective Neuron Deactivation

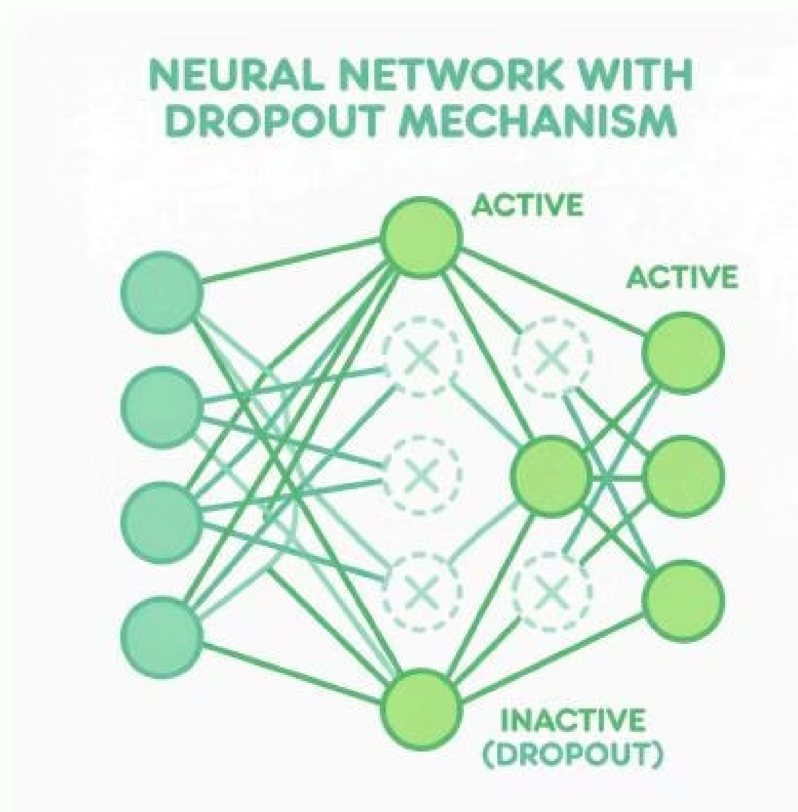


Dropout randomly deactivates a fraction of neurons during training, forcing the network to learn redundant, robust representations rather than relying on specific pathways.

Key Benefits

- Reduces over-reliance on specific neurons or features
- Prevents co-adaptation of neurons during training
- Significantly improves model generalisation
- Acts as an ensemble method within a single network

Typical dropout rates: 0.2 - 0.5 during training (disabled during inference)



In speech recognition models, dropout prevents memorisation of specific speakers' voices and dramatically improves performance across diverse accents and audio conditions.

Common Optimizers Visualized

Optimization algorithms determine how models update weights during training. Each optimizer has unique characteristics that affect convergence speed and final performance.

SGD (Stochastic Gradient Descent)

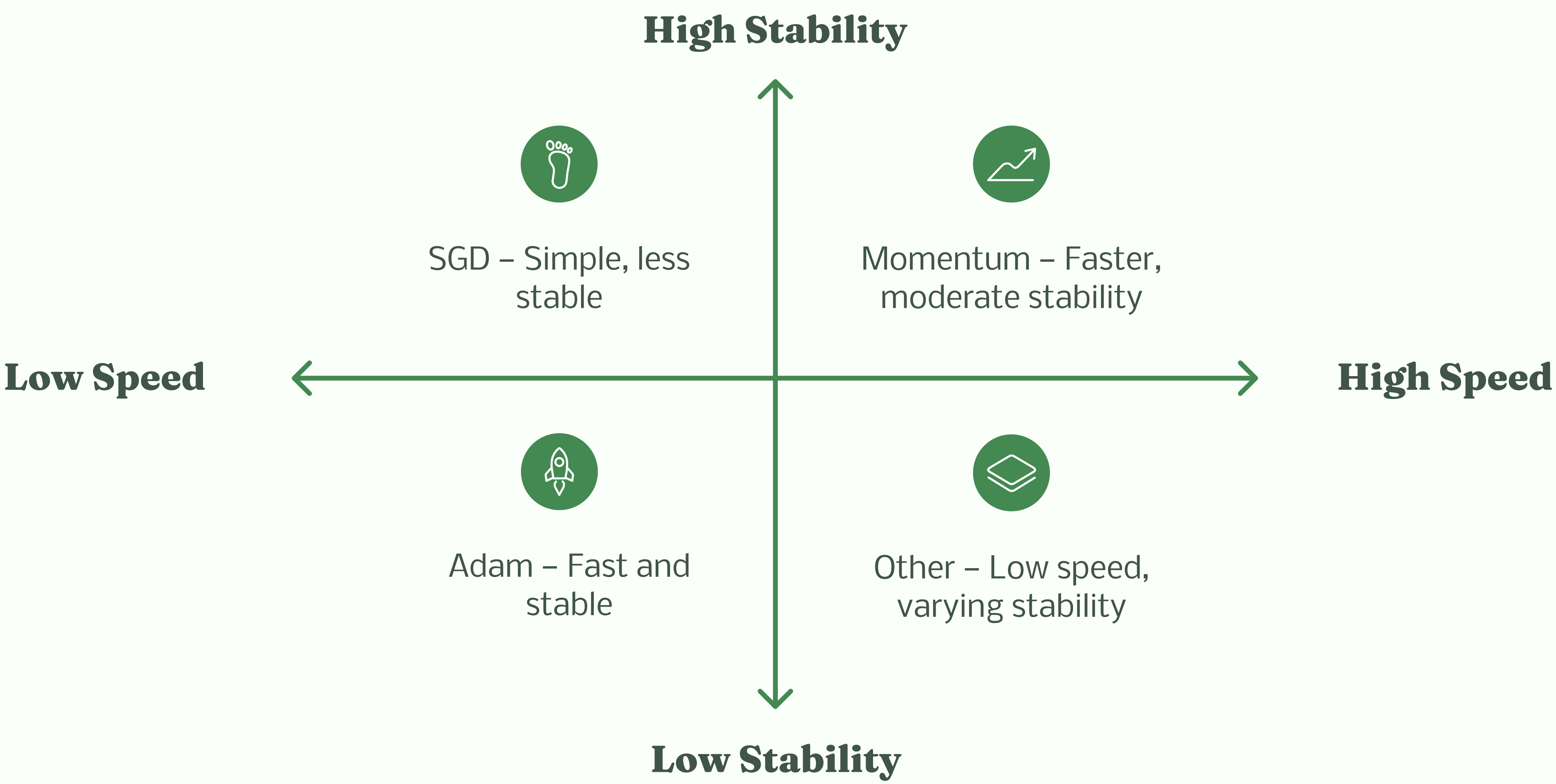
Basic optimizer that updates weights using gradients. Simple but can be slow and get stuck in local minima.

Momentum

Adds velocity to SGD, accelerating convergence and smoothing oscillations by accumulating past gradients.

Adam

Combines momentum with adaptive learning rates. Most popular optimizer for deep learning due to fast, stable convergence.



Batch Normalisation: Stabilising Deep Networks

Batch Normalisation normalises layer outputs to maintain consistent activation distributions throughout training, enabling faster convergence and higher learning rates.

Compute Statistics

Calculate mean and variance of activations within each mini-batch

Normalise

Standardise activations to zero mean and unit variance

Scale and Shift

Apply trainable parameters to restore network expressiveness

Compute Statistics

Calculate mean and variance of activations within each mini-batch

Normalise

Standardise activations to zero mean and unit variance

Scale and Shift

Apply trainable parameters to restore network expressiveness

Benefits of Batch Normalisation

Faster Convergence

Models train 2-3x faster with stable gradients and higher learning rates

Reduced Covariate Shift

Minimises internal distribution changes as parameters update during training

Mild Regularisation

Adds slight noise that acts as implicit regularisation, reducing overfitting

In image classification architectures like ResNet, batch normalisation is essential for training very deep networks (50+ layers) whilst maintaining stability and achieving state-of-the-art accuracy.

Hyperparameter Tuning Strategies

Hyperparameters are configuration settings that define model behaviour–learning rate, number of layers, batch size, and regularisation strength. Finding optimal combinations is crucial for production performance.

01

Grid Search

Exhaustively evaluates all possible parameter combinations–thorough but computationally expensive for large search spaces

02

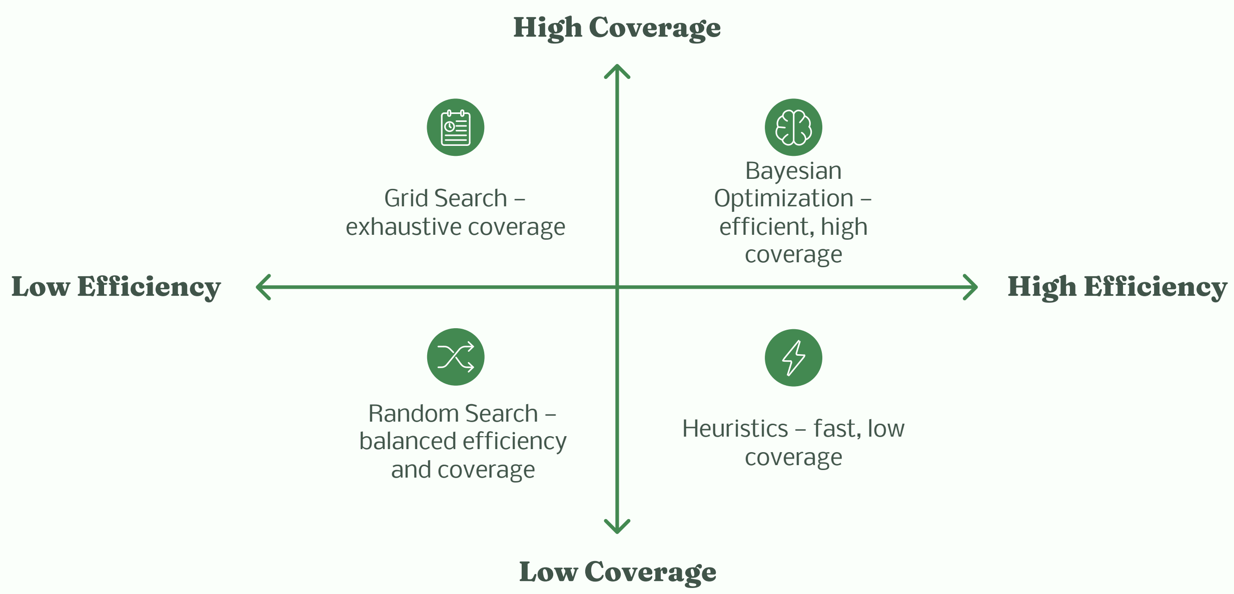
Random Search

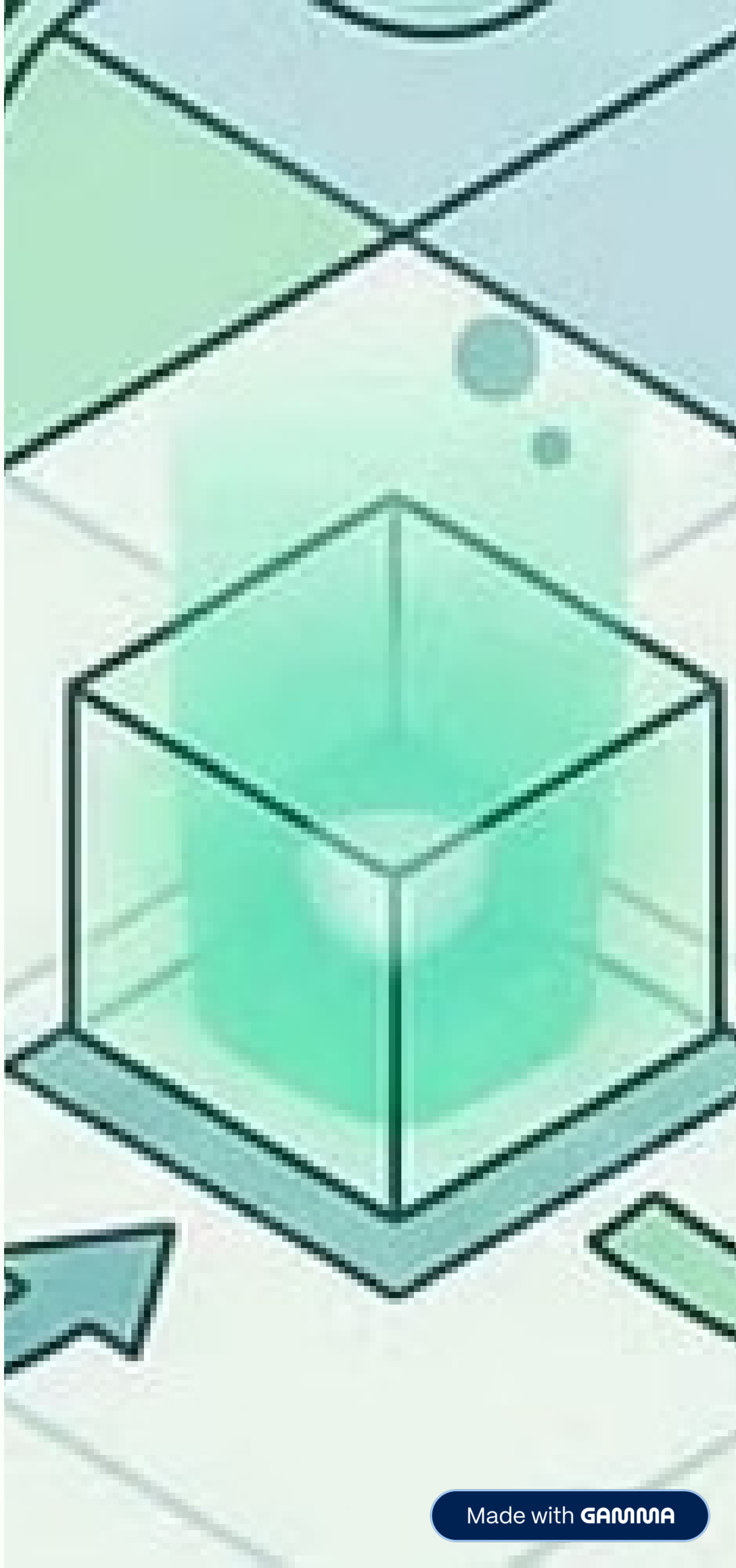
Samples random parameter combinations–surprisingly effective and significantly faster than grid search for high-dimensional spaces

03

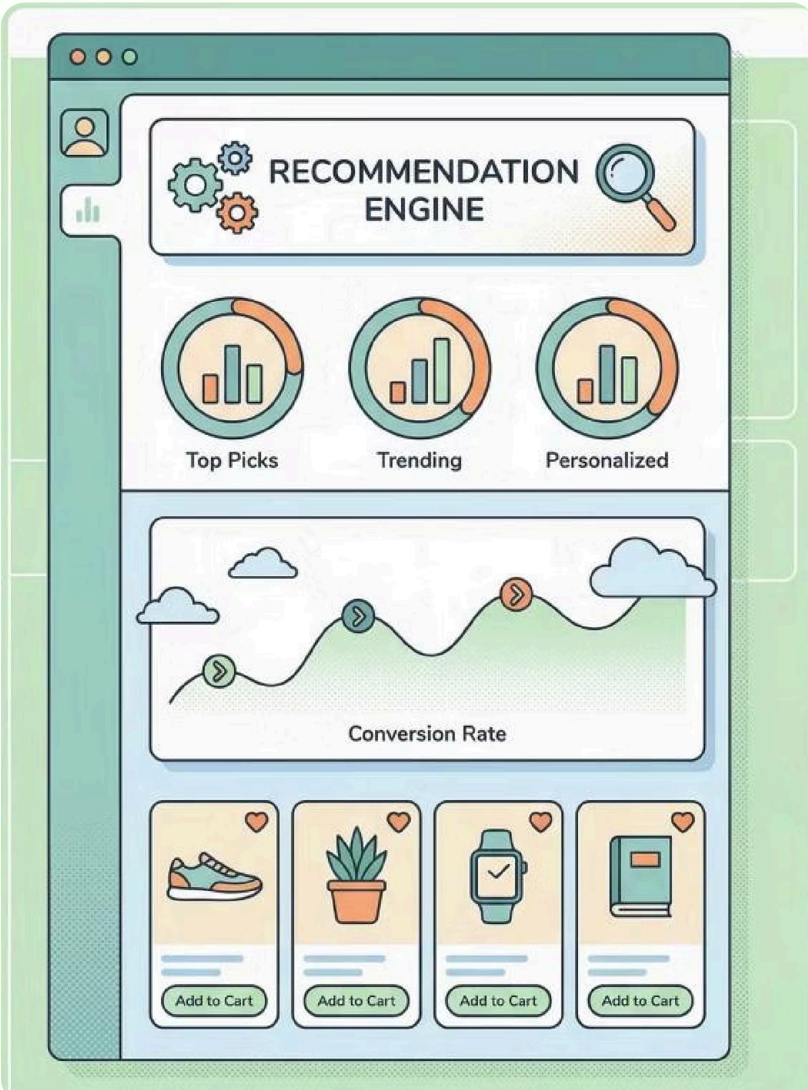
Bayesian Optimisation

Intelligently explores the parameter space based on past results–most efficient for expensive model training





Real-World Tuning Impact



In an e-commerce recommendation system, systematic tuning of learning rate and network architecture can improve accuracy by 15-25% whilst reducing training time by up to 40%.

3x

Speed Improvement

Proper learning rate tuning accelerates convergence

20%

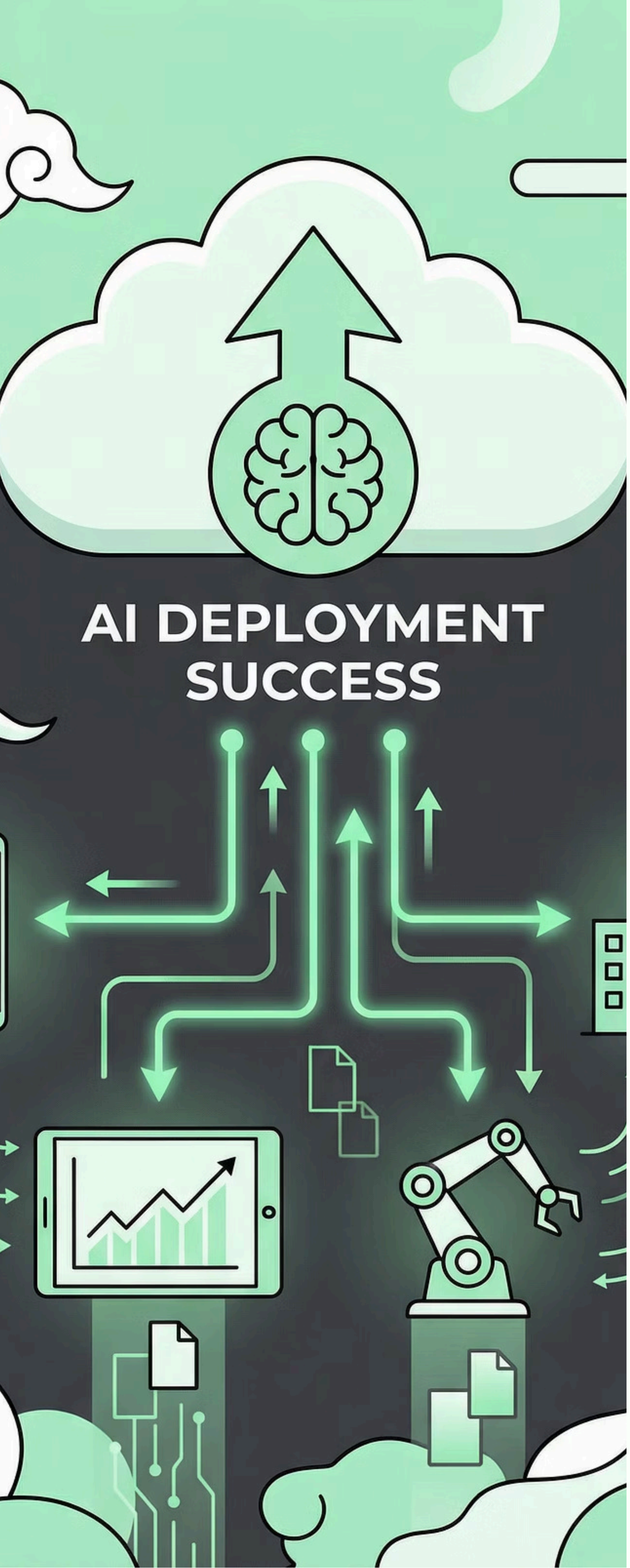
Accuracy Gains

Optimal architecture selection boosts performance

40%

Time Reduction

Efficient configurations cut training costs



Key Takeaways for Production Systems

Balance
Avoid overfitting and underfitting

Regularisation
Use L2 and dropout techniques

Tuning
Optimize hyperparameters carefully

Normalisation
Scale inputs for stable training

Balance is Everything

Avoid both overfitting and underfitting through careful model selection and validation strategies

Regularisation is Essential

Use L2 regularisation and dropout to ensure models generalise beyond training data

Systematic Tuning Pays Off

Invest time in hyperparameter optimisation–it dramatically impacts real-world performance

Normalisation Enables Scale

Batch normalisation is crucial for training deep networks efficiently and reliably