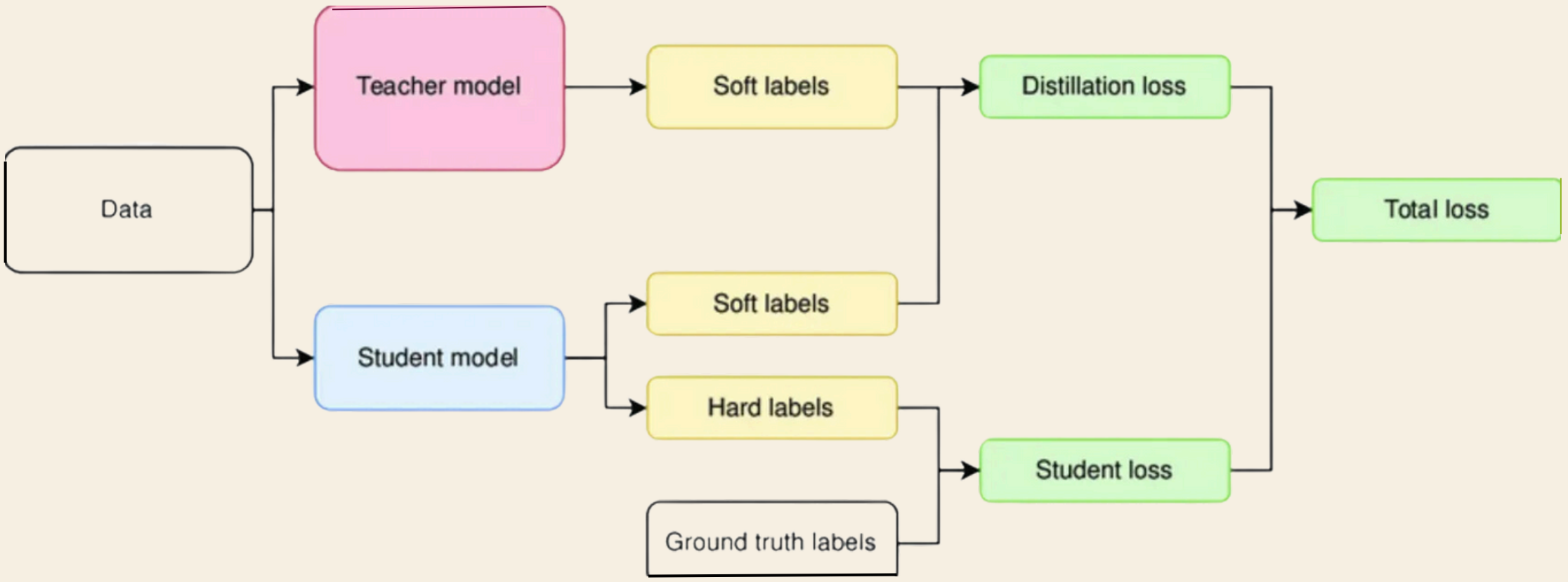
KNOWLEDGE DISTILLATION

In Large Language Models

WHAT IS KNOWLEDGE DISTILLATION?



Knowledge Distillation is a model compression technique where a smaller model (student) is trained to mimic the behavior of a larger, more complex model (teacher) by learning from the teacher's output distributions rather than just the ground truth labels.

```
# Traditional Training
loss = cross_entropy(model(x), ground_truth_labels)

# Knowledge Distillation
teacher_probabilities = teacher_model(x)  # Soft targets from teacher
student_probabilities = student_model(x)  # Student's predictions

# Combined loss using both hard labels and soft targets
distillation_loss = (1-α) * cross_entropy(student_probabilities, ground_truth_labels)
+                   α * KL_divergence(student_probabilities, teacer_probabilities/T)
```

Where:

- T is the temperature parameter (controls softness of probability distribution)
- α is the weighting factor between hard and soft targets
- KL_divergence measures how different the student's predictions are from the teacher's

WHY DO WE NEED IT?

Environmental Impact
Distilled models significantly reduce energy consumption and carbon footprint

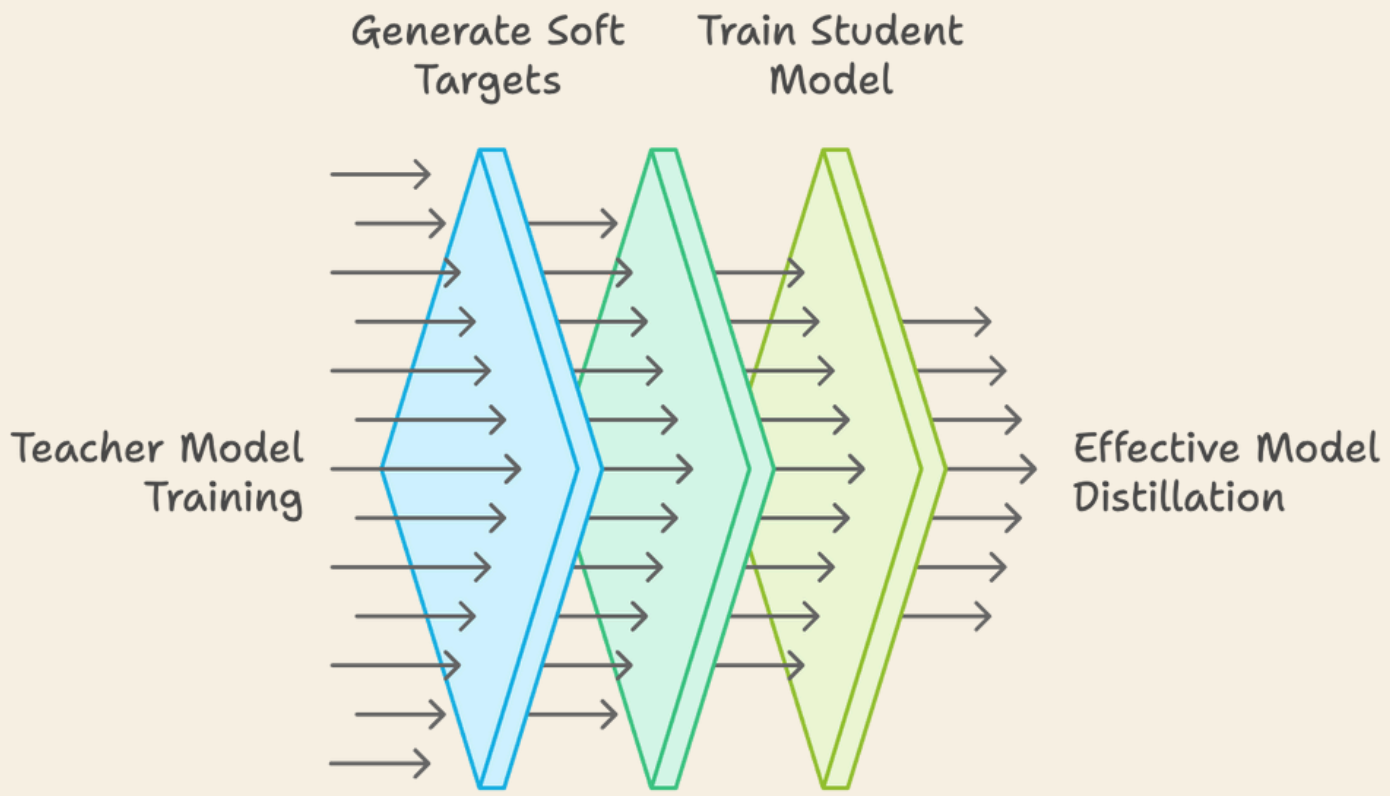
Model Efficiency & Costs
Distilled models require less computing power and retain high performance

Resource Accessibility
Distilled models lower the cost of AI infrastructure, making it more accessible

Production Deployment
Distilled models enable faster response times for real-time applications

- **Model Efficiency & Costs** → Original models require massive computing power (100B+ parameters, multiple GPUs, 500GB+ RAM) . Distilled models run on standard hardware while retaining 90-95% performance (10B parameters or less)
- **Production Deployment** → Large models have 1000ms+ inference time, making real-time applications impossible. Distilled models achieve 100ms or less response times, enabling chatbots, live translation, etc.
- **Resource Accessibility** → Original models cost $10,000+ monthly for GPU infrastructure and cloud computing. Distilled models can run on $500-1000 hardware, making AI accessible to startups and researchers
- **Environmental Impact** → Large models consume 1000+ kWh daily, equivalent to powering 100 homes. Distilled models reduce energy usage by 80-90%, significantly lowering carbon footprint

HOW IT WORKS?



1. **Train the Teacher Model:**

- Start with a complex model, known as the teacher, that is trained on a large dataset. The teacher model aims to achieve high accuracy and learns rich feature representations.

2. **Generate Soft Targets:**

- The teacher model outputs probabilities for each class instead of traditional hard labels (0 or 1). These probabilities indicate the teacher's confidence and provide information about class similarities. Temperature scaling is applied to the softmax function to control the output distribution.
    - Higher temperature values create softer distributions, revealing inter-class relationships.

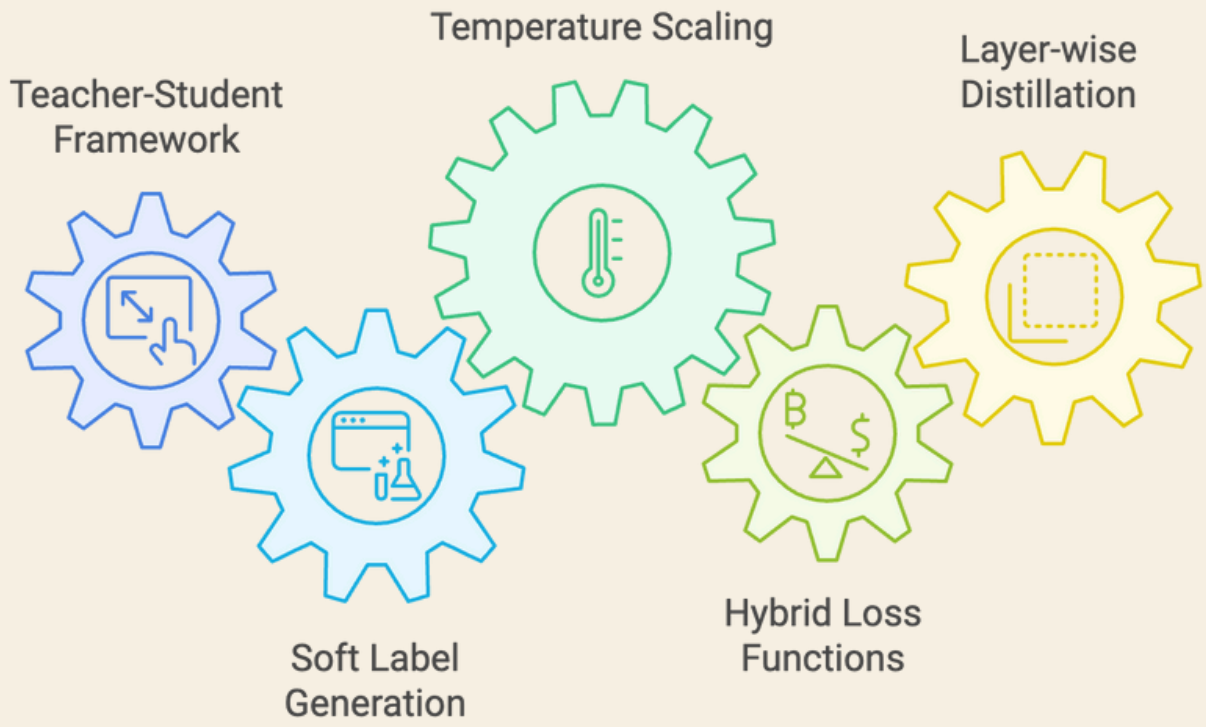
3. **Train the Student Model:**

- The goal is to train a smaller, simpler model (the student) using the soft targets from the teacher. A specialized loss function combines the student's predictions with the teacher's soft targets. This loss function accounts for both:
    - Distillation Loss: How well the student matches the teacher's outputs.
    - Standard Loss: The student's accuracy against the true labels.
- The final loss function is a weighted sum of these two components, allowing the student to learn effectively.

4. **Visual Representation:**

- Use a flow diagram to illustrate the steps from the teacher model to soft targets and then to the student model. Show how different temperature settings affect the probability distributions to enhance understanding.

KEY TECHNIQUES IN LLM DISTILLATION



- **Teacher-Student Framework**

This foundational approach involves training a complex, high-capacity LLM (the teacher) and using its outputs to guide a smaller model (the student). The student learns from the teacher's predictions, allowing it to generalize effectively while maintaining performance.

- **Soft Label Generation**

Soft labels are generated by the teacher model instead of using traditional hard labels. This richer information about the model's confidence helps the student learn more effectively by providing insights into class similarities.

- **Temperature Scaling**

By increasing the temperature, the probability distribution becomes smoother, highlighting relationships between classes and offering the student model a more nuanced understanding of the teacher’s decision-making.

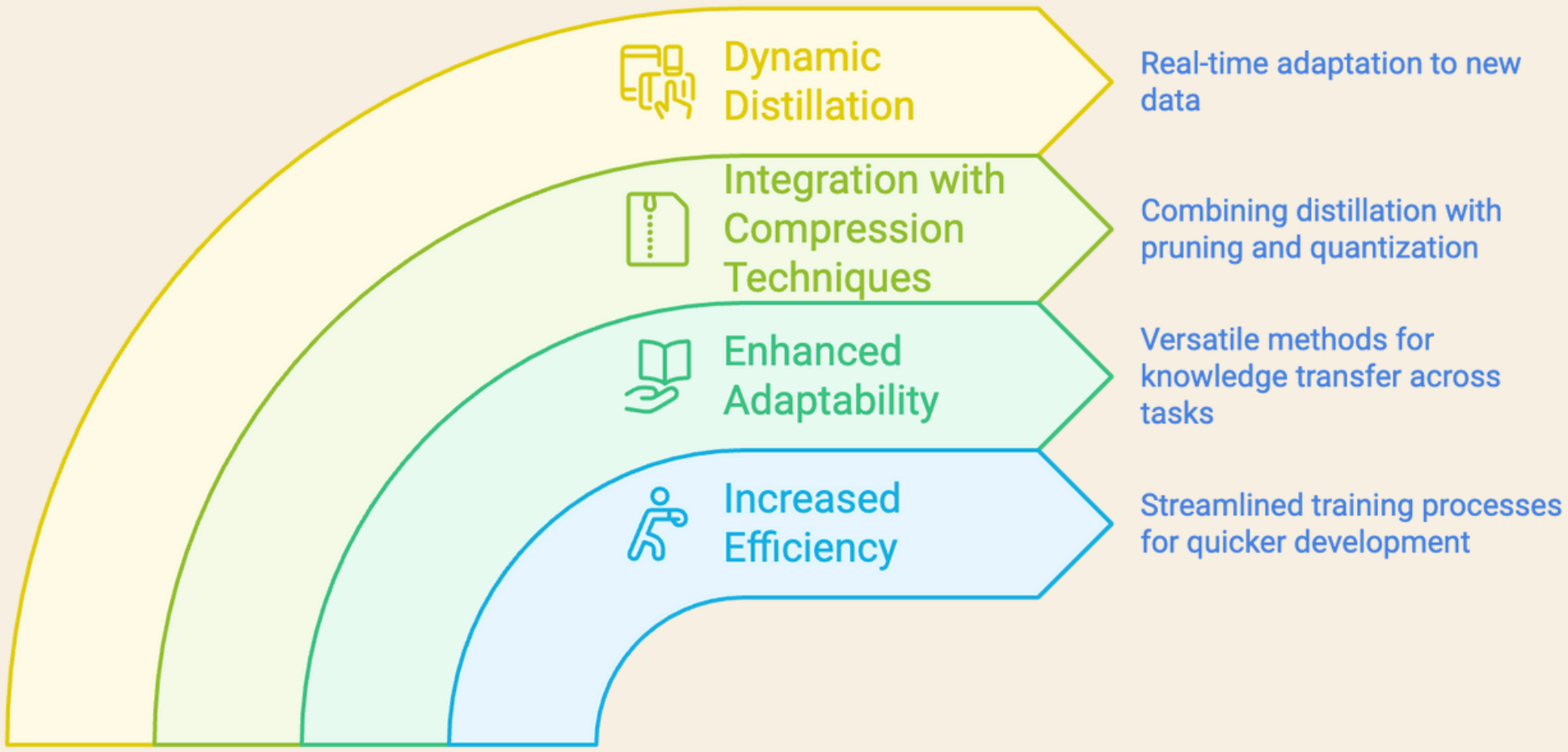
- **Hybrid Loss Functions**

The loss function for the student model often combines distillation loss (from the teacher’s outputs) with standard classification loss (against true labels). This hybrid approach encourages the student to learn from both the teacher's knowledge and the actual data, balancing generalization and accuracy.

- **Layer-wise Distillation**

Knowledge can be transferred from multiple layers of the teacher model to the student model, not just from the final output. By matching intermediate representations, the student gains a deeper understanding of the teacher’s internal processes, leading to improved learning outcomes.

FUTURE OF KNOWLEDGE DISTILLATION



- **Increased Efficiency in Model Training**

Future advancements in knowledge distillation will focus on streamlining the training process, allowing for quicker and more resource-efficient model development. This will make it easier to create smaller models that maintain high performance without extensive computational demands.

- **Enhanced Adaptability and Transferability**

Emerging techniques will aim to make distillation methods more versatile, enabling models to easily transfer knowledge across different tasks and domains. This adaptability will reduce the need for retraining and facilitate the application of AI models in diverse scenarios.

- **Integration with Other Compression Techniques**

Knowledge distillation is expected to be combined with other model compression strategies, such as pruning and quantization. This integration will create even smaller, more efficient models that maintain their effectiveness, pushing the limits of model scalability.

- **Dynamic and Online Distillation**

The development of dynamic and online distillation methods will allow models to adapt in real-time to new data. This continuous learning capability will make AI systems more resilient and responsive to changing environments and user needs.

IMPLEMENTATION

1. **Import Packages**

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
```

2. Create Distiller Class

```python
class Distiller(keras.Model):
    def __init__(self, student, teacher):
        super().__init__()
        self.teacher = teacher
        self.student = student

    def compile(
        self,
        optimizer,
        metrics,
        student_loss_fn,
        distillation_loss_fn,
        alpha=0.1,
        temperature=3,
    ):
        super().compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.distillation_loss_fn = distillation_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    def train_step(self, data):
        x, y = data

        # Forward pass of teacher
        teacher_predictions = self.teacher(x, training=False)

        with tf.GradientTape() as tape:
            student_predictions = self.student(x, training=True)
            student_loss = self.student_loss_fn(y, student_predictions)
            distillation_loss = (
                self.distillation_loss_fn(
                    tf.nn.softmax(teacher_predictions / self.temperature, axis=1),
                    tf.nn.softmax(student_predictions / self.temperature, axis=1),
                )
                * self.temperature**2
            )

            loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss
```

```
        # Compute gradients
        trainable_vars = self.student.trainable_variables
        gradients = tape.gradient(loss, trainable_vars)

        # Update weights
        self.optimizer.apply_gradients(zip(gradients, trainable_vars))

        # Update the metrics configured in `compile()`.
        self.compiled_metrics.update_state(y, student_predictions)

        # Return a dict of performance
        results = {m.name: m.result() for m in self.metrics}
        results.update(
            {"student_loss": student_loss, "distillation_loss": distillation_loss}
        )
        return results

    def test_step(self, data):
        # Unpack the data
        x, y = data

        # Compute predictions
        y_prediction = self.student(x, training=False)

        # Calculate the loss
        student_loss = self.student_loss_fn(y, y_prediction)

        # Update the metrics.
        self.compiled_metrics.update_state(y, y_prediction)

        # Return a dict of performance
        results = {m.name: m.result() for m in self.metrics}
        results.update({"student_loss": student_loss})
        return results
```

3. Create student and teacher models

```
# Create the teacher
teacher = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(256, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
        layers.Conv2D(512, (3, 3), strides=(2, 2), padding="same"),
        layers.Flatten(),
        layers.Dense(10),
    ],
    name="teacher",
)

# Create the student
student = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(16, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
        layers.Conv2D(32, (3, 3), strides=(2, 2), padding="same"),
        layers.Flatten(),
        layers.Dense(10),
    ],
    name="student",
)

# Clone student for later comparison
student_scratch = keras.models.clone_model(student)
```

4.Prepare the dataset

```
# Prepare the train and test dataset.
batch_size = 64
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize data
x_train = x_train.astype("float32") / 255.0
x_train = np.reshape(x_train, (-1, 28, 28, 1))

x_test = x_test.astype("float32") / 255.0
x_test = np.reshape(x_test, (-1, 28, 28, 1))
```

5. Train the teacher

```
# Train teacher as usual
teacher.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)

# Train and evaluate teacher on data.
teacher.fit(x_train, y_train, epochs=10)
teacher.evaluate(x_test, y_test)
```

6. Distill teacher to student

```
# Initialize and compile distiller
distiller = Distiller(student=student, teacher=teacher)
distiller.compile(
    optimizer=keras.optimizers.Adam(),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
    student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    distillation_loss_fn=keras.losses.KLDivergence(),
    alpha=0.1,
    temperature=10,
)

# Distill teacher to student
distiller.fit(x_train, y_train, epochs=10)
```

```
student.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)
student.evaluate(x_test, y_test)
```

7. Train student from scratch for comparison

```
# Train student as doen usually
student_scratch.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)

# Train and evaluate student trained from scratch.
student_scratch.fit(x_train, y_train, epochs=10)
student_scratch.evaluate(x_test, y_test)
```

Expected Output:

```
[0.05867832154035568, 0.9822999835014343]
```

Follow for more AI/ML posts

SAVE **LIKE** **SHARE**