M16 - Fine-Tuning

Stand: 05.2025

Fine-Tuning ist eine bewährte Technik, um ein vortrainiertes KI-Modell gezielt auf spezifische Aufgaben oder Datensätze anzupassen. Dabei werden die bereits gelernten Strukturen und Muster eines bestehenden Modells genutzt und weiterverfeinert. Dieser Prozess spart nicht nur Rechenressourcen und Zeit, sondern führt auch bei kleineren Datenmengen zu beeindruckenden Ergebnissen.

In der Praxis bedeutet Fine-Tuning, dass Entwickler:innen lediglich einen Bruchteil der Rechenleistung und Datenmenge benötigen, die für das Training eines Modells von Grund auf erforderlich wären. Gleichzeitig wird die Möglichkeit geschaffen, eigene oder sensible Daten gezielt einzusetzen. Fine-Tuning ist somit nicht nur ein technisches Werkzeug, sondern auch ein strategischer Ansatz zur Effizienzsteigerung, Individualisierung und Qualitätssicherung von KI-Anwendungen.

Zudem ist Fine-Tuning ein zentraler Bestandteil des sogenannten Model Optimization Workflows. Dabei wird die Modellleistung durch eine Kombination aus **Evals**, **Prompt Engineering** und **Fine-Tuning** in einem iterativen Zyklus optimiert. Ziel ist es, die Qualität der Modellantworten durch Feedback und gezielte Anpassung kontinuierlich zu verbessern. Dieser sogenannte Optimierungs-Flywheel ermöglicht es, systematisch bessere Prompts, Trainingsdaten und daraus resultierende Modelle zu entwickeln.

1 Fine-Tuning-Ansätze

1.1 Transfer Learning

- **Grundprinzip**: Ein vortrainiertes Modell wird als Ausgangspunkt verwendet. Die allgemeinen Merkmale der frühen Schichten bleiben erhalten.
- **Vorgehen**: Die letzten Schichten werden ersetzt oder angepasst, um spezifische Aufgaben zu lösen.
- **Einsatzgebiete**: Bildklassifikation, Verarbeitung natürlicher Sprache (NLP), Computer Vision.
- **Vorteile**: Schneller Einstieg, geringer Datenbedarf, bewährte Basismodelle.
- **Vergleich zum Pre-Training**: Während das Pre-Training ein Modell mit großen Datenmengen (oft Billionen von Tokens) trainiert, um allgemeine Sprachmuster zu lernen, benötigt Fine-Tuning nur einen Bruchteil der Daten und Rechenkapazität für eine spezifische Aufgabe.

1.2 Parameter-effizientes Fine-Tuning (PEFT)

- **Prinzip**: Anpassung nur weniger Parameter, während das Basismodell unverändert bleibt.
- **Methoden**:
  - **LoRA (Low-Rank Adaptation)**: Kompakte Matrizen für effiziente Gewichtsänderung. Reduziert den Rechenaufwand erheblich durch Low-Rank-Approximation der Gewichtsänderungen.
  - **QLoRA**: Eine Erweiterung von LoRA, die Gewichtsparameter auf 4-Bit-Präzision quantisiert, wodurch der Speicherbedarf weiter reduziert wird.
  - **DoRA (Weight-Decomposed Low-Rank Adaptation)**: Zerlegt Gewichte in Größen- und Richtungskomponenten für präzisere Updates bei gleichbleibender Effizienz.
  - **Adapter**: Zusätzliche Module zwischen bestehenden Schichten.
  - **Prompt Tuning**: Anpassung durch trainierbare Prompts.
- **Vorteile**: Spart Ressourcen, wiederverwendbar, ideal für verschiedene Aufgaben.
- **Anwendungsbereich**: Besonders geeignet für ressourcenbeschränkte Umgebungen und für mehrere Spezialisierungsaufgaben mit demselben Basismodell.

1.3 Instruction Fine-Tuning

- **Ziel**: Das Modell lernt, auf klare, natürlichsprachliche Anweisungen zu reagieren.
- **Daten**: Input-Output-Paare mit expliziten Instruktionen.
- **Anwendungen**: Sprachassistenten, automatisierte Kommunikation, LLM-basierte Tools.
- **Formatbeispiel**: Oft in diesem Format: "###Human: $< InputQuery >$ ###Assistant: $< GeneratedOutput >$"

1.4 Supervised Fine-Tuning (SFT)

- **Ansatz**: Feinabstimmung mit handverlesenen, hochwertigen Beispielen.
- **Zweck**: Optimierung für bestimmte Anforderungen oder Zielgruppen.
- **Typisch kombiniert mit**: Reinforcement Learning from Human Feedback (RLHF).
- **Datenerfordernisse**: Benötigt mindestens 10 Beispiele, empfohlen sind 50-100 qualitativ hochwertige Demonstrationen. Die Qualität der Daten ist entscheidender als die Menge.
- **Prozess**: Besteht aus Datenvorbereitung, Upload der Trainingsdaten, Erstellung eines Fine-Tuning-Jobs und anschließender Evaluierung des Modells.

2 Weitere Ansätze OpenAI

2.1 Direct Preference Optimization (DPO)

- **Vorgehen**: Training mit präferierten und abgelehnten Antwortpaaren.
- **Nutzen**: Verfeinert Nuancen wie Stil, Tonalität, Ausdruck.
- **Funktionsweise**: Ein effizienterer Weg als RLHF, um Modelle an menschliche Präferenzen anzupassen. Jedes Beispiel im Datensatz enthält einen Prompt, eine bevorzugte Ausgabe und eine nicht-bevorzugte Ausgabe.
- **Beta-Parameter**: Kann zwischen 0 und 2 konfiguriert werden, um zu steuern, wie streng das neue Modell am vorherigen Verhalten festhält versus sich an den neuen Präferenzen orientiert.

2.2 Reinforcement Fine-Tuning (RFT)

- **Prinzip**: Modell wird nicht mit festen Zielantworten, sondern anhand von Bewertungssignalen (Grader) trainiert.
- **Vorteile**: Besonders geeignet für komplexe, mehrdeutige Aufgaben.
- **Grader-Konzept**: RFT verwendet Grader, die die Modellantworten bewerten und ein numerisches Signal (zwischen 0 und 1) zurückgeben. Diese können als String-Check, Text-Similarity oder Model-Grader konfiguriert werden.
- **Anwendungsbereich**: Besonders effektiv bei Aufgaben, bei denen Experten in der Domäne sich über die richtigen Antworten einig sind und die Aufgabe eindeutig bewertbar ist.
- **Unterstützung**: Aktuell nur für reasoning-Modelle wie o4-mini verfügbar.

2.3 Vision Fine-Tuning

- **Zweck**: Anpassung von Modellen mit visuellen Eingaben (z. B. Bilder).
- **Anwendungen**: Bildklassifikation, visuelle Beschreibungen, Objektlokalisierung.
- **Technische Hinweise**: Unterstützt Base64-Bilder oder URLs, max. 10 Bilder pro Beispiel.
- **Besonderheiten**:
  - Bilder müssen im JPEG-, PNG- oder WEBP-Format vorliegen
  - Maximalgröße pro Bild: 10 MB
  - Bilder mit Menschen, Gesichtern, Kindern oder CAPTCHAs werden aus Datenschutzgründen ausgeschlossen
  - Der Detail-Parameter kann auf "low" gesetzt werden, um Trainingskosten zu reduzieren

2.4 Modell-Distillation

- **Konzept**: Nutzung der Ausgaben eines großen Modells, um ein kleineres Modell zu trainieren, das ähnliche Leistung für eine spezifische Aufgabe erzielt.
- **Vorteile**: Reduziert Kosten und Latenz erheblich, da kleinere Modelle effizienter sind.
- **Prozess**:

1. Speichern qualitativ hochwertiger Ausgaben eines großen Modells mit dem Parameter `store: true`
2. Evaluierung der gespeicherten Antworten mit dem großen und kleinen Modell
3. Auswahl relevanter Antworten als Trainingsdaten für das kleine Modell
4. Fine-Tuning des kleinen Modells mit diesen Beispielen
5. Evaluierung des fine-getuned kleineren Modells

- **Anwendungsbereich**: Besonders nützlich, wenn ein spezifischer, begrenzter Aufgabenbereich abgedeckt werden soll.

3 Fine-Tuning-Pipeline für LLMs

3.1 Datenvorbereitung

- Datensammlung aus verschiedenen Quellen
- Vorverarbeitung und Formatierung (z.B. JSONL-Format)
- Umgang mit unausgeglichenen Daten (Oversampling, Undersampling)
- Datensatzaufteilung (Training/Validierung/Test)

3.2 Modellinitialisierung

- Auswahl eines geeigneten vortrainierten Modells
- Einrichtung der Umgebung und Installation der Abhängigkeiten
- Laden des Modells in den Speicher

3.3 Trainingsumgebung

- Konfiguration von Hardwareressourcen (GPU/TPU)
- Definition von Hyperparametern (Lernrate, Batch-Größe, Epochen)
- Initialisierung von Optimierern und Verlustfunktionen

3.4 Fine-Tuning-Prozess

- Auswahl der Fine-Tuning-Technik (Voll, PEFT, etc.)
- Durchführung des Trainings mit regelmäßigen Validierungen
- Überwachung von Metriken und Verlustfunktionen

3.5 Evaluierung und Validierung

- Aufsetzen von Evaluierungsmetriken
- Analyse der Trainingsverlaufskurve
- Überwachung und Interpretation der Ergebnisse

3.6 Deployment

- Export des fine-getuned Modells
- Einrichtung der Infrastruktur
- API-Entwicklung für die Modellinteraktion

3.7 Monitoring und Wartung

- Kontinuierliche Überwachung der Modellleistung
- Aktualisierung des LLM-Wissens bei Bedarf
- Wiederholte Feinabstimmung bei veränderter Datenlage

4 Schlüsselkomponenten der Modelloptimierung

4.1 Evaluierungen (Evals)

- **Nutzen**: Systematische Tests zur Bewertung von Modellantworten.
- **Formate**: Multiple Choice, Klassifikation, Stringvergleich etc.
- **Grader-Typen**:
  - **String-Check-Grader**: Einfache String-Operationen (gleich, ungleich, enthält)
  - **Text-Similarity-Grader**: Bewertung der Ähnlichkeit zwischen Modellantwort und Referenz
  - **Model-Grader**: Nutzung eines separaten Modells zur Bewertung der Ausgaben
  - **Python-Grader**: Ausführung von Python-Code zur Bewertung
  - **Multi-Grader**: Kombination mehrerer Grader für komplexe Bewertungskriterien
- **Integrierter Prozess**: Evals sollten vor dem Fine-Tuning erstellt werden, um eine Baseline zu etablieren und den Fortschritt zu messen.

4.2 Prompt Engineering

- **Ziele**: Maximale Modellleistung ohne Training.
- **Methoden**: Klare Instruktionen, Kontextbereitstellung, Few-Shot-Beispiele.
- **Zusammenspiel mit Fine-Tuning**: Prompt Engineering kann Fine-Tuning ergänzen oder in manchen Fällen sogar ersetzen.
- **Beispiel**: Die Prompt-Konstruktion mit relevanten Beispielen (Few-Shot-Learning) kann die Leistung signifikant verbessern, ohne das Modell neu zu trainieren.

5 Best Practices

5.1 Datenstrategie

1. **Datenqualität schlägt Datenmenge**
   - Beginnen Sie mit 50-100 hochwertigen Beispielen
   - Verwenden Sie realistische Daten aus der Zielanwendung

- Stellen Sie sicher, dass die Daten repräsentativ für die Aufgabe sind

2. **Vielfältige und realistische Beispiele wählen**
   - Decken Sie verschiedene Szenarien, Formulierungen und Nuancen ab
   - Vermeiden Sie starke Verzerrungen in den Trainingsdaten
   - Berücksichtigen Sie auch Randfall-Szenarien
3. **Konsistente Formatierung (z. B. JSONL)**
   - Verwenden Sie das korrekte Format für Ihre Fine-Tuning-Methode
   - Stellen Sie sicher, dass jede Zeile ein vollständiges JSON-Objekt enthält
   - Validieren Sie Ihr Datenformat vor dem Training

5.2 Trainingsstrategie

4. **Schrittweises Auftauen von Schichten**
   - Beginnen Sie mit dem Training der obersten Schichten
   - Tauen Sie schrittweise tiefere Schichten auf
   - Dies führt zu stabilerem Training und verhindert Overfitting
5. **Kleine Lernraten zur Stabilisierung**
   - Verwenden Sie Lernraten zwischen 1e-4 und 2e-4 für stabile Konvergenz
   - Ein Lernraten-Schedule mit Warmup und linearem Abfall kann hilfreich sein
   - Experimentieren Sie mit verschiedenen Batch-Größen
6. **Regelmäßige Evaluierung und Monitoring**
   - Setzen Sie vor dem Fine-Tuning Evaluierungen (Evals) auf
   - Überwachen Sie Trainings- und Validierungsmetriken
   - Implementieren Sie Early Stopping, um Overfitting zu vermeiden

5.3 Technische Exzellenz

7. **Verwendung von Checkpoints und Modellversionierung**
   - Speichern Sie regelmäßig Zwischenstände (alle 5-8 Epochen)
   - Vergleichen Sie die Leistung verschiedener Checkpoint-Modelle
   - Behalten Sie die Versionshistorie bei, um Regressionen zu erkennen
8. **Hyperparameter systematisch optimieren**
   - Nutzen Sie automatisierte Hyperparameter-Optimierung (Random Search, Grid Search, Bayesian)
   - Fokussieren Sie auf Lernrate, Batch-Größe und Epochenanzahl
   - Dokumentieren Sie die Ergebnisse verschiedener Konfigurationen
9. **Tools wie TensorBoard, W&B, MLflow einsetzen**
   - Visualisieren Sie Trainingsmetriken in Echtzeit
   - Verfolgen Sie Experimente und deren Ergebnisse
   - Vergleichen Sie verschiedene Trainingsläufe untereinander

5.4 Sicherheit und Effizienz

10. **Datenschutzkonformität beachten**
    - Anonymisieren Sie sensible Daten vor dem Training
    - Beachten Sie rechtliche Anforderungen beim Training mit personenbezogenen Daten
    - Implementieren Sie Sicherheitsprüfungen für Modelleingaben und -ausgaben
11. **Kosten im Blick behalten (Token-Effizienz)**
    - Überwachen Sie den Token-Verbrauch während des Trainings
    - Optimieren Sie Prompts für kürzere Ausgaben, wo möglich
    - Verwenden Sie Modell-Distillation für häufig genutzte Aufgaben

5.5 Spezifische Techniken für erweiterte Anwendungen

12. **Multi-Task Learning**
    - Trainieren Sie das Modell für mehrere verwandte Aufgaben gleichzeitig
    - Verwenden Sie spezifische Adapter für verschiedene Aufgaben
    - Kombinieren Sie bei Bedarf mehrere Adapter für komplexe Anwendungen
13. **Modell-Quantisierung**
    - Reduzieren Sie die Präzision der Modellparameter (z.B. von 32-bit auf 8-bit)
    - Verwenden Sie QLoRA für effizientes Training mit quantisierten Modellen
    - Testen Sie die Leistung quantisierter Modelle gründlich
14. **Multimodale Integration**
    - Bei Vision Fine-Tuning: Achten Sie auf Bildqualität und -größe
    - Verwenden Sie den "detail"-Parameter, um Trainingskosten zu optimieren
    - Testen Sie verschiedene Kombinationen von Text- und Bildeingaben

6 Herausforderungen & Perspektiven

6.1 Skalierbarkeit

- **Rechnerische Ressourcen**: Fine-Tuning großer Modelle erfordert erhebliche Rechen- und Speicherkapazitäten
- **Memory-Effizienz**: Techniken wie LoRA, QLoRA und Half Fine-Tuning adressieren diese Herausforderungen
- **Zukünftige Entwicklungen**: Co-Design von Hardware und Algorithmen speziell für LLM-Training

6.2 Ethische Überlegungen

- **Bias und Fairness**: Trainingsdaten können Verzerrungen enthalten, die sich auf das Modell übertragen

- **Datenschutz**: Umgang mit sensiblen oder proprietären Daten während des Fine-Tunings
- **Transparenz und Nachvollziehbarkeit**: Dokumentation des Fine-Tuning-Prozesses und seiner Auswirkungen

6.3 Integration mit neuen Technologien

- **Internet of Things (IoT)**: LLMs können IoT-Daten in Echtzeit analysieren und Entscheidungen optimieren
- **Edge Computing**: Fine-Tuned Modelle können direkt auf Edge-Geräten eingesetzt werden
- **Federated Learning**: Training über verteilte Datenquellen ohne zentrale Datenspeicherung

7 Fazit

Fazit

Fine-Tuning ist mehr als nur eine technische Maßnahme – es ist ein strategischer Hebel zur Anpassung von KI-Systemen an konkrete Anforderungen, Zielgruppen und Kontexte. Die Kombination aus fundierter Datenbasis, passenden Methoden und iterativer Evaluation bildet das Fundament für erfolgreiche KI-Projekte.

Mit Plattformen wie OpenAI lassen sich diese Prozesse effizient gestalten – von der Vorbereitung über das Training bis zur Bewertung. Erweiterte Verfahren wie DPO, RFT und Vision Fine-Tuning ermöglichen zusätzlich eine feinkörnige Kontrolle und Weiterentwicklung leistungsfähiger Modelle.

Die siebenstufige Fine-Tuning-Pipeline bietet einen strukturierten Ansatz vom Datenmanagement bis zur kontinuierlichen Überwachung, während parameter-effiziente Methoden die Ressourcennutzung optimieren. Durch den gezielten Einsatz dieser Techniken können Entwickler leistungsfähige KI-Lösungen erstellen, die genau auf ihre spezifischen Anforderungen zugeschnitten sind.