Künstliche Intelligenz
für Business Analytics

Grundlagen, Architekturen und Anwendungen

6 Operationalisierung von KI-Systemen

Patrick Baier

Während sich die bisherigen Kapitel darauf fokussiert haben, wie KI-Systeme zur Lösung von Problemstellungen aus verschiedenen Anwendungsbereichen genutzt werden können, richtet dieses Kapitel den Fokus auf deren Operationalisierung. Allgemein bezeichnet Operationalisierung den Prozess, ein entwickeltes KI-Modell oder -System aus der Forschungs- oder Entwicklungsphase in den realen Betrieb zu überführen, wo es in seinem vorgesehenen Anwendungskontext eingesetzt wird.

6.1 Vom Prototyp zum Go-live

Der Ausgangspunkt für die Operationalisierung von KI-Systemen ist in der Regel das Ende der Entwicklungsphase. In dieser Phase wurde durch iterative Anpassungen im Training versucht, die Vorhersagequalität des Modells zu steigern. Der nächste Schritt besteht darin, das trainierte Modell in den Realbetrieb (den sogenannten *Live-Betrieb*) zu überführen, wo es zur Wertschöpfung des Unternehmens beiträgt.

Um die damit einhergehenden Herausforderungen von Beginn an anschaulich darzustellen, wird nachfolgend eine Beispielanwendung eingeführt, auf die in regelmäßigen Abständen in den weiteren Ausführungen dieses Kapitels zurückgegriffen wird.

6.1.1 Beispielanwendung

Unser Beispielunternehmen operiert im Finanzbereich und gibt an seine Kunden Kreditkarten aus, mit denen diese weltweit und online bezahlen können. Um Transaktionen mit gestohlenen Kreditkartendaten möglichst frühzeitig zu erkennen, nutzt das Unternehmen ein KI-Modell auf Basis des maschinellen Lernens (*ML-Modell*), das innerhalb weniger Sekunden jede Kundentransaktion überprüft, um festzustellen, ob es sich um eine normale Transaktion oder um einen Betrugsversuch (d. h., die Karte bzw. die notwendigen Daten wurden gestohlen) handelt. Das Modell wurde auf vergangenen Transaktionen der Kunden trainiert, bei denen im Nachhinein bekannt war, ob Betrug vorlag oder nicht (es handelt sich daher um ein klassisches ML-Modell aus dem Bereich *Supervised Learning*). Mögliche Eingabewerte (sogenannte *Features*) für die Vorhersage eines solchen Modells wären z. B. Ort der Transaktion, Betrag der Transaktion, Trans-

aktionshistorie des Kunden und der zur Transaktion zugehörige Händler, der das Geld empfängt. Auf Basis dieser Features liefert das Modell für eine gegebene Kreditkartentransaktion einen Wert zwischen 0 und 1 zurück, der angibt, wie hoch die Wahrscheinlichkeit ist, dass es sich bei dieser Transaktion um Betrug handelt. Liegt dieser über einem zuvor definierten Schwellenwert, wird die Transaktion automatisch storniert.

Derartige ML-Modelle spielen in der Praxis von Kreditkartentransaktionen tatsächlich eine bedeutende Rolle und sind durch eine Reihe von besonderen Herausforderungen gekennzeichnet:

1. Ein Vorhersagefehler des Modells hat eine sofortige, negative, finanzielle Konsequenz für das Unternehmen. Das Stornieren einer normalen Transaktion führt zum Verlust der Transaktionsgebühr (und verärgert den Kunden). Für eine nicht erkannte betrügerische Transaktion muss das Unternehmen ggf. haften.
2. Das Modell muss auch für eine sehr große Anzahl an zeitlich möglicherweise nah aufeinanderfolgenden Transaktionen verlässlich eine Antwort liefern.
3. Die Vorhersage des Modells muss kurz vor Abschluss der Transaktion relativ schnell (d. h. innerhalb weniger Sekunden) zur Verfügung stehen.

ML-Modelle werden in der Regel in der Programmiersprache *Python* mithilfe entsprechender ML-Bibliotheken entwickelt. Die Spezifikation des Modells und das anschließende Training auf den Trainingsdaten sind als iterativer Prozess gestaltet. Dabei werden die Modelleigenschaften (z. B. die Wahl der Features oder Belegung der Hyperparameter) so lange optimiert, bis das Modell auf den separaten Testdaten eine möglichst hohe Vorhersagegenauigkeit erreicht. In der bereits eingeführten Beispielanwendung wäre dies z. B. der Fall, wenn das Modell hinreichend genau zwischen Betrug und Nicht-Betrug bei einer Menge gegebener Transaktionen unterscheidet. Wir werden nun im Folgenden beispielhaft darstellen, wie ein solches ML-Modell nach dem Training in eine operative Live-Umgebung überführt werden kann.

Im einfachsten Fall wird das Modell lokal auf einem Rechner (ggf. mit GPU-Unterstützung) trainiert. Nach erfolgreichem Training wird die Struktur des Modells zusammen mit den gelernten Parametern als Datei auf der Festplatte gespeichert. Um das Modell nun im Live-Betrieb einzusetzen (d. h., um reale Kreditkartentransaktionen zu überprüfen), muss es für andere IT-Systeme erreichbar sein. Am einfachsten geschieht dies, indem es als Teil einer Webapplikation (*Web-App*) geladen wird, die über das Netzwerk mittels HTTP-Protokoll von anderen Systemen angefragt werden kann. Da sich die Funktionalität dieser App lediglich auf das Beantworten von Anfragen mithilfe des ML-Modells beschränkt, bezeichnet man die Web-App in diesem Kontext auch oft als *Microservice*. Allgemein ist ein Microservice ein kleiner, in sich geschlossener Softwaredienst, der eine ganz bestimmte Funktion innerhalb einer größeren Anwendung übernimmt.

Bei einer neuen, zu überprüfenden Kreditkartentransaktion würde ein vorgeschalteter Microservice nun alle nötigen Feature-Daten für die Transaktion zusammenstellen. Dazu werden typischerweise verschiedene Datenbanktabellen angefragt,

die Daten zur Transaktion und zur Transaktionshistorie des Kunden enthalten. Die Daten werden vom gleichen Service entsprechend vorverarbeitet, bis sie das gleiche Format haben wie die Daten, mit denen das ML-Modell trainiert und getestet wurde. Anschließend werden die Daten gebündelt an die Web-App des ML-Modells gesendet. Diese ruft das ML-Modell intern auf und liefert dessen Antwort, die Betrugswahrscheinlichkeit der Transaktion, an das aufrufende System zurück. Auf Basis dieser Wahrscheinlichkeit wird nun entschieden, ob die Transaktion storniert wird oder nicht.

Dieser einfache Ansatz reicht aus, um ein ML-Modell schnell in den Produktiveinsatz zu bringen (und wird sicherlich in dem einen oder anderen KI-Start-up auch direkt so umgesetzt). Er genügt jedoch nicht, um eine große Anzahl an Modellen für eine große Menge an Anfragen zuverlässig bereitzustellen. Daher entwickeln wir im Folgenden den beschriebenen Prozess weiter, und nähern uns den heute gültigen Best Practices bei der Operationalisierung von ML-Modellen an.

6.1.2 Automatisierung der Modellerstellung

Ein Problem des beschriebenen Ansatzes ist die fehlende Automatisierung und die damit verbundene Fehleranfälligkeit sowie die mangelnde Reproduzierbarkeit des trainierten Modells, da dieses direkt vom Entwickler gespeichert wird. Daher ist es sinnvoll, die folgenden Schritte des oben genannten Ablaufs zu automatisieren:

- Wenn ein Entwickler ein neues ML-Modell trainiert, das er in den Live-Betrieb bringen möchte, dann speichert er nicht das Modell, sondern lediglich dessen Spezifikation. Die Spezifikation enthält alle wesentlichen Informationen, die beim Modelltraining genutzt wurden, wie Quelle der Trainings- und Testdaten, verwendete Features, Art des Modells, die gewählten Hyperparameter usw. Diese Spezifikation lädt der Entwickler anschließend in ein zentrales Code-Repository hoch (z. B. GitHub, GitLab, BitBucket).
- Das Training des eigentlichen Modells, das später im Live-Betrieb eingesetzt werden soll, übernimmt ein zentraler, leistungsfähiger Rechner (sogenannter Trainingsserver), der auf das zuvor genannte Code-Repository zugreifen kann. Zum eigentlichen Training des Modells startet der Entwickler einen Job auf dem Trainingsserver. Dieser lädt dann die jeweilige Modellspezifikation und die darin spezifizierten Trainingsdaten, woraufhin das Modell trainiert.
- Nach dem erfolgreichen Training speichert der Trainingsserver das Modell in einen speziellen *Model-Storage* zusammen mit der für das Training verwendeten Spezifikation.

Die beschriebenen Automatisierungsschritte ermöglichen es, das Training des Modells vom jeweiligen Entwickler zu entkoppeln, und gewährleisten so eine einfache Reproduzierbarkeit des Modells. Durch die Speicherung der Modellspezifikation entsteht zudem Transparenz darüber, welche Modelle im Model-Storage abgelegt sind. Abbildung 6.1 liefert einen Überblick über den beschriebenen Prozess.

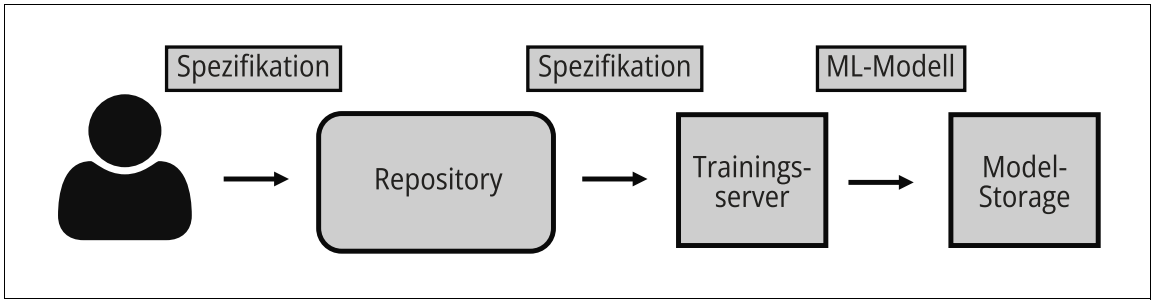

Abb. 6.1 *Automatisierung des Modelltrainings*

6.1.3 Deployment von ML-Modellen

Eine Schwachstelle des beschriebenen Deployment-Szenarios besteht darin, dass das gespeicherte Modell im Rahmen der Web-App nur auf einem einzigen Server verfügbar ist und somit nicht mit der Anzahl der eingehenden Anfragen skaliert. Dadurch kann es bei einer hohen Anzahl an Anfragen zur Überlastung und somit zu starken Verzögerungen in der Antwortzeit der Web-App kommen. Dies ist in vielen Fällen (wie z. B. im Kreditkartenszenario) nicht akzeptabel.

Die gängige Lösung hierfür besteht darin, das Modell mithilfe einer Microservices-Plattform (z. B. *Kubernetes*) über mehrere Server automatisch zu skalieren. Dafür muss die Web-App in eine sogenannte Container-Virtualisierung (z. B. *Docker*) überführt werden, die es ermöglicht, die Web-App auf einem beliebigen Server zu starten. Die Microservices-Plattform sorgt dafür, dass immer eine ausreichende Zahl an Web-Apps gleichzeitig im Live-Betrieb läuft, um alle ankommenden Anfragen unmittelbar zu beantworten. Jede dieser Web-Apps lädt beim Start das neueste Modell aus dem Model-Storage und beantwortet eine Teilmenge der zu bearbeitenden Anfragen.

6.2 Monitoring von KI-Systemen

Um ein KI-System im Live-Betrieb erfolgreich einzusetzen, ist eine dauerhafte Überwachung des Systemverhaltens nach dem Deployment unumgänglich. Beim sogenannten *Monitoring* wird fortwährend überprüft, ob die Eingabedaten für das ML-Modell und dessen Vorhersagen sich in einem erwartbaren Rahmen bewegen. Wenn dies nicht der Fall ist, liegt möglicherweise ein Fehler vor, sodass die Vorhersagen des ML-Modells sehr viel ungenauer ausfallen können als erwartet.

Monitoring ist eine Disziplin, die bei herkömmlichen IT-Systemen schon seit Jahrzenten praktiziert wird und dementsprechend durch ausgereifte Monitoring-Tools unterstützt wird. Ziel des Monitorings ist es, alle wichtigen Systemeigenschaften zu überwachen und bei einer festgestellten Abweichung vom Sollzustand eine Alarmnachricht auszulösen. Diese wird typischerweise an einen verantwortlichen Systemadministrator gesendet, der sich das Problem daraufhin anschaut und versucht, es zu beheben.

Herkömmliches IT-Monitoring umfasst in der Regel die Überwachung von Hardwareressourcen, auf denen die zu überwachende Anwendung läuft (z. B. CPU-Auslas-

tung oder Speicherbelegung der Serverinstanzen), sowie die Sammlung von Daten zu wichtigen Metriken der Anwendungen (z. B. Anfragen pro Minute oder Antwortzeit des Service). Während diese beiden Dimensionen auch für KI-Systeme relevant sind, kommt beim Live-Betrieb von ML-Modell noch eine zusätzliche Komponente hinzu: die Qualität der Input-Features und der Vorhersagen des Modells. Um dies zu veranschaulichen, greifen wir im folgenden Abschnitt wieder auf das Beispiel der Betrugserkennung bei Kreditkartentransaktionen zurück.

6.2.1 Vom Fehler zur Katastrophe

Nehmen wir an, das ML-Modell zur Erkennung von Kreditkartenbetrug läuft im Live-Betrieb im Rahmen der Web-App auf einem Server und gibt für jede Transaktion eine Betrugswahrscheinlichkeit zurück. Eines der Features, anhand derer das Modell seine Vorhersage trifft, ist die Höhe des Geldbetrags der jeweiligen Transaktion. Das Modell wurde auf historischen Transaktionen trainiert, bei denen dieser Betrag immer in Euro vorlag. Da das Modell aber auch Transaktionen aus anderen Zahlungsräumen verarbeiten soll, rechnet im Live-Betrieb ein vorgeschalteter Microservice jede Fremdwährung in Euro um, bevor die Transaktion an das ML-Modell weitergeleitet wird. Wenn wir jetzt weiter annehmen, dass dieser vorgeschaltete Service temporär fehlerhaft arbeitet und daher falsche Beträge bei der Umrechnung liefert, dann kann das unter Umständen dazu führen, dass die Vorhersagen unseres Modells komplett unzuverlässig werden. Beispielsweise werden Betrugsfälle gar nicht mehr erkannt oder fast jede normale Transaktion als Betrug erkannt und storniert.

Das beschriebene Beispiel zeigt, wie ein einfacher Fehler in einem vorgeschalteten Service durch ein ML-Modell kaskadiert und zu einem gewaltigen monetären Schaden führen kann, wenn dieser über einen längeren Zeitraum unentdeckt bleibt. Die besondere Problematik in diesem Beispiel liegt darin, dass herkömmliches IT-Monitoring aufseiten des ML-Modells diese Art von Fehler nicht erkannt hätte, da das Problem nicht auf der Hardwareseite, sondern in der abweichenden Verteilung der Eingangsdaten des Modells liegt. Eine solche Abweichung kann jedoch die Vorhersagequalität eines ML-Modells stark negativ beeinflussen. Angenommen, dass das Modell dadurch keinen Betrug mehr erkennen würde, wäre der Schaden sogar erst sichtbar, wenn sich Kunden aufgrund von Unregelmäßigkeiten bei ihrer Kreditkarte beim Unternehmen beschweren.

6.2.2 Die Rolle der Datenverteilung

Wie in den vorangehenden Grundlagenkapiteln beschrieben, werden ML-Modelle auf einer Menge von Trainingsdaten trainiert. In unserer Beispielanwendung sind dies historische Kreditkartentransaktionen, bei denen im Nachhinein bekannt ist, ob sie Betrug waren oder nicht. Auf einer zweiten Menge an historischen Daten wird das Modell nach dem Training in der Regel getestet und anhand einer Metrik (z. B. *F1-Score*) dessen Performance evaluiert. Wenn diese als gut genug bewertet wird, geht das Modell ins Deployment für den Live-Betrieb. Eine wichtige und oft

übersehene Eigenschaft von ML-Systemen in diesem Kontext ist, dass ein Modell im Live-Betrieb nur dann eine ähnlich gute Performance wie in der Testphase liefert, wenn die Datenverteilung der Inputdaten des Modells in der Live-Umgebung ähnlich zu der Datenverteilung ist, mit der das Modell trainiert wurde (siehe z. B. [Koh et al. 2021]).

Ist dies nicht der Fall, spricht man von einem *Distribution Shift*. Abbildung 6.2 zeigt zwei Histogramme für dasselbe Feature (links zur Trainingszeit, rechts im Live-Betrieb), bei denen ein solcher Distribution Shift erkennbar ist. Ein Beispiel für einen gravierenden Distribution Shift wäre ein Modell, das im Training auf Geldbeträge in Cent trainiert wurde, in der Live-Umgebung jedoch Beträge als ganze Eurobeträge zugespielt bekommt.

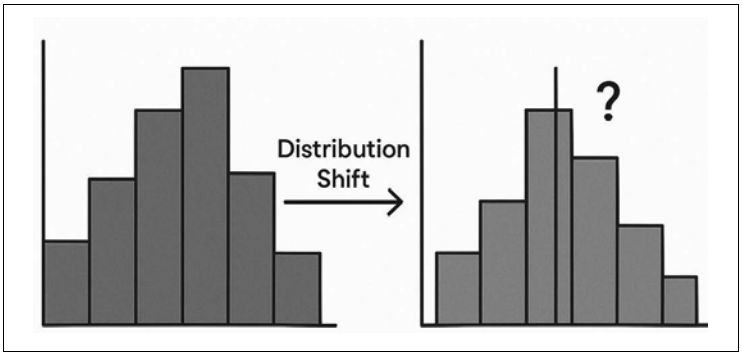

Abb. 6.2 *Änderung der Datenverteilung*

Die Gründe für Distribution Shifts zwischen Live-Betrieb und Trainingsdaten können vielfältiger Natur sein. Im Folgenden sollen einige beispielhaft aufgezählt werden:

- **Mangelnde Kommunikation**
  Es herrscht mangelnde Kommunikation zwischen den unterschiedlichen Entwicklungsteams. Im Beispiel mit den Eurobeträgen können fehlende Absprachen zwischen dem Team, das das Modell trainiert, und dem Team, das das Modell im Live-Betrieb mit Eingabewerten versorgt, zu einer unterschiedlichen Dimensionierung der Eingabewerte führen.
- **Fehlerhafter vorgeschalteter Service**
  Ein vorgeschalteter Service, der das Modell im Live-Betrieb mit Eingabewerten versorgt, arbeitet fehlerhaft. Mögliche Ursachen sind in diesem Kontext vielfältig und reichen von Fehlern in der Programmlogik (sogenannte *Bugs*) über zu lange Antwortzeiten des Service wegen Überlast bis zu Hardware- oder Netzwerkproblemen.
- **Natürliche Verschiebungen**
  Ein oft wenig beachteter Punkt sind natürliche Verschiebungen in der Verteilung von Daten, die im Allgemeinen ein großes Problem für ML-Modelle darstellen. Dabei unterscheidet man bei der Veränderung der Verteilungen zwischen wiederkehrenden Shifts (sogenannten saisonalen Effekten), stetigen Drifts und spontanen Drifts:

- **Saisonale Effekte** sind Veränderungen in der Datenverteilung, die sich mit einem gleichbleibenden zeitlichen Abstand wiederholen. So beeinflusst z. B. der Wochentag die Anzahl der Transaktionen, die ein Kunde mit einer Kreditkarte tätigt. Um ein ML-Modell dagegen robust zu machen, muss beim Training sichergestellt werden, dass der Zeitraum der Trainingsdaten groß genug ist, um alle potenziellen saisonalen Effekte möglichst mehrfach zu beinhalten.
- **Stetige Drifts** zeichnen sich durch eine fortwährende Änderung der Datenverteilung im Zeitverlauf aus. So werden z. B. die Geldbeträge der Kreditkartentransaktionen im Laufe der Zeit aufgrund von Inflation immer größer. Um ein Modell robust gegenüber dieser Art von Veränderung zu machen, müssen Features, bei denen ein stetiger Drift vorliegt, auf einen einheitlichen Wert normalisiert werden – und zwar im Training und im Live-Betrieb. Bei Geldbeträgen legt man sich beispielsweise auf ein Referenzdatum fest und normalisiert alle davor und danach liegenden Beträge mittels der jeweiligen Inflationsrate auf eine einheitliche Kaufkraft.
- Ein **spontaner Drift** hingegen ändert die Verteilung eines Input-Features schlagartig. Mit diesem Szenario können die meisten ML-Modelle nur schwer bis gar nicht umgehen. Ein bekanntes Beispiel stammt aus der Zeit der Coronakrise: Beim Schätzen von Lieferzeiten konnten die ML-Modelle vieler Lieferdienste nicht mehr damit umgehen, dass das Verkehrsaufkommen auf der Straße sich innerhalb kürzester Zeit drastisch änderte. Es gibt zwar erste Ansätze aus der aktuellen Forschung, um mit solchen Situationen in gewissen Szenarien umzugehen [Dong et al. 2024], eine allgemeine Lösung hierfür ist allerdings momentan noch nicht in Sicht.

Ähnlich wie bei den Eingabedaten eines Modells kann auch die Verteilung der Vorhersagen des Modells überwacht und mit denen zur Trainingszeit verglichen werden, um Unregelmäßigkeiten zu erkennen. So kann z. B. die Anzahl der vorhergesagten Betrugsfälle eines Live-Modells mit der Anzahl der Betrugsfälle in einem vergleichbaren Zeitraum aus den Trainingsdaten verglichen werden.

6.2.3 ML-Monitoring in der Praxis

In der Praxis besteht die Aufgabe des ML-Monitorings darin, die Verteilung der Input-Features und der Vorhersage des Modells fortwährend im Live-Betrieb zu überwachen und mit der Verteilung zur Trainingszeit zu vergleichen. Bei einer zu großen Abweichung bei einer der Verteilungen muss, ähnlich wie beim klassischen IT-Monitoring, ein verantwortlicher Administrator alarmiert werden, der sich das Problem anschaut.

Technisch kann ein solches System beispielhaft, wie folgt, realisiert werden: Jeder Eingabewert für das Modell wird im Live-Betrieb zuerst in einer Zeitreihendatenbank (z. B. *InfluxDB*) gespeichert, bevor dieser vom Modell verarbeitet wird. In einem festgelegten Zeitintervall (z. B. alle 10 Minuten) werden die zuletzt gesammelten Daten aus der Datenbank extrahiert und in Form einer empirischen Verteilung als Histo-

gramm aggregiert. Dieses wird anschließend mit der empirischen Verteilung der Trainingsdaten verglichen (ähnlich zu Abb. 6.2). Um den Unterschied der beiden Histogramme quantitativ zu messen, eignen sich mathematische Verfahren wie z. B. die *Wasserstein-Metrik*. Liegt der errechnete Wert über einem zuvor festgelegten Grenzwert, wird ein Alarm in Form einer E-Mail an einen Systemadministrator ausgelöst.

Beim Vergleich der beiden Verteilungen gibt es mehrere statistische Herausforderungen, da der Zeitraum der beiden aggregierten Datensätze in der Regel unterschiedlich groß ist und auch hier saisonale Effekte eine Rolle spielen können. Da eine Diskussion hier den Rahmen sprengen würde, verweisen wir an dieser Stelle auf die Arbeit von Baier und Dragiev [Baier & Dragiev 2021], die sich mit dem Design und den Herausforderungen eines solchen Systems im Detail beschäftigen.

Zur Umsetzung eines ML-Monitoring-Systems gibt es auf dem Markt mittlerweile eine Reihe von Frameworks, die das beschriebene System auf ähnliche Art und Weise implementieren und somit eine ML-Monitoring-Lösung out of the box anbieten. Der bekannteste Anbieter in diesem Kontext ist *Evidently*, der Teile seines Monitoring-Tools auch als Open-Source-Version anbietet. Neben einer ganzen Reihe weiterer Anbieter offerieren auch die großen Cloud-Anbieter eigene ML-Monitoring-Lösungen im Rahmen ihrer jeweiligen Frameworks (z. B. *Amazon SageMaker Model Monitor*).

6.3 ML-Operations

Zum Abschluss dieses Kapitels wollen wir noch kurz auf das derzeit oft diskutierte Thema *ML-Operations* (MLOps) eingehen. MLOps ist ein Konzept, das Methoden des DevOps (Development and Operations) aus der allgemeinen Softwareentwicklung auf den Bereich des maschinellen Lernens überträgt. Das Ziel besteht darin, ML-Modelle effizient, reproduzierbar und skalierbar zu entwickeln, bereitzustellen und zu betreiben. Der aufmerksame Leser wird feststellen, dass die bisherigen Betrachtungen in diesem Kapitel auf denselben Zweck abzielen. Deployment und Monitoring sind in der Tat mitunter die wichtigsten Bestandteile des MLOps-Konzepts, weshalb sie in diesem Kapitel anhand eines Beispiels ausführlich dargestellt wurden. Im Rahmen von MLOps gibt es aber noch eine Reihe anderer Themen, die der Vollständigkeit halber im Folgenden noch kurz erläutert werden:

- **Datenmanagement**
  MLOps befasst sich mit Konzepten, mit denen Trainingsdaten für ML-Modelle versioniert, validiert und in einem standardisierten Format bereitgestellt werden. Das Ziel ist es, jederzeit nachvollziehen zu können, mit welchen Daten ein Modell trainiert wurde.
- **Modellentwicklung**
  Bei der iterativen, experimentellen Entwicklung eines neuen ML-Modells helfen MLOps-Konzepte dabei, Experimente zu dokumentieren, zu vergleichen und zu reproduzieren. Tools wie *MLflow* oder *Weights & Biases* unterstützen beim Tracking und Vergleich der Ergebnisse.

- **Modell-Retraining**
  Wenn die Performance eines Modells im Live-Betrieb mit der Zeit sinkt, muss das Modell in der Regel auf frischeren Daten neu trainiert werden – idealerweise automatisch. MLOps beschäftigt sich mit der Fragestellung, wie dieser Prozess reibungslos und nachvollziehbar abläuft.

Aktuell gibt es am Markt kein einzelnes Tool, das alle MLOps-Teilbereiche vollständig abdeckt. Große Cloud-Plattformen wie *Amazon SageMaker*, *Google Vertex AI* oder *Azure ML* bieten eine breite End-to-End-Abdeckung, sind jedoch oft stark an das jeweilige Ökosystem gebunden. Open-Source-Tools wie *MLflow*, *Kubeflow*, *Feast* oder *Evidently* fokussieren sich auf einzelne Bereiche wie Tracking, Deployment oder Monitoring und lassen sich flexibel kombinieren. Viele Unternehmen bauen daher eine modulare MLOps-Toolchain, die genau auf ihre Infrastruktur und Bedürfnisse zugeschnitten ist. Dabei entsteht oft ein Trade-off zwischen Benutzerfreundlichkeit, Automatisierung und technischer Kontrolle. Die Auswahl der richtigen Tools hängt stark vom Reifegrad des Teams, von der IT-Umgebung, regulatorischen Anforderungen und dem gewünschten Automatisierungsgrad ab.

6.4 Zusammenfassung

Wie in diesem Kapitel anschaulich dargestellt, ist die Operationalisierung von KI-Systemen ein zentraler Aspekt, um nachhaltig Wertschöpfungsprozesse in Unternehmen mittels KI zu etablieren. Die Herausforderungen, die damit einhergehen, sind nicht zu unterschätzen und übersteigen in vielen Fällen sogar die Komplexität und den Aufwand, die mit der Entwicklung des eigentlichen KI-Modells verbunden sind. Für Unternehmen ist es daher wichtig, bereits zu Beginn eines KI-Projektes den Aufwand für die Operationalisierung bei der Planung des Projektes zu berücksichtigen. Zudem ist es aus eigener Erfahrung sinnvoll, das Projektteam mit erfahrenen Softwareentwicklern, die Kenntnisse in den Bereichen MLOps oder DevOps mitbringen, zu ergänzen. So ist von Anfang an sichergestellt, dass der späteren Operationalisierung des KI-Systems schon bei dessen Entwicklung Rechnung getragen wird.

18 KI-Bohrer: KI-gesteuerte Lärmreduzierung für urbane Geothermiebohrungen

Daniel Ladwig · Martin Spitznagel · Jan Vaillant · Klaus Dorer · Janis Keuper

Dieses Kapitel beschreibt das Projekt, Geothermiebohrungen mithilfe von künstlicher Intelligenz zu unterstützen. Das Ziel ist die Reduzierung der Lärmemissionen und Steigerung der Betriebseffizienz, um die Wirtschaftlichkeit zu fördern und eine größere Akzeptanz von Geothermieprojekten in Städten zu erreichen. Hierzu werden generative Modelle mit Deep Reinforcement Learning kombiniert, um den Bohrprozess durch Vorhersagen und Vorschläge zu verbessern.

18.1 Einleitung

Die urbane geothermische Energieerzeugung wird zunehmend als zentraler Bestandteil zur Erreichung der globalen Klimaziele anerkannt. Der Einsatz der Geothermie-Technologie in städtischen Gebieten ist jedoch nicht unproblematisch, insbesondere wegen der erheblichen Lärmbelästigung, die bei den Bohrarbeiten entsteht. Die derzeitigen Methoden zur Minderung dieses Problems sind weitgehend manuell und oft ungeeignet, um den Lärmpegel innerhalb der gesetzlichen städtischen Grenzen zu halten. In dicht besiedelten Gebieten können kontinuierliche Tiefbohrungen, die für die Gewinnung geothermischer Energie erforderlich sind, die örtliche Bevölkerung erheblich beeinträchtigen. Gesetzliche Vorschriften begrenzen den Lärmpegel in der Nacht oft auf 35 dB, was angesichts des 24/7-Betriebs dieser Projekte eine große Herausforderung darstellt. Derzeitige Lösungen, wie die zeitliche Verlagerung des Betriebs und physische Barrieren, bieten nur begrenzte Abhilfe.

Im Rahmen dieses Kapitels wird eine Anwendung der künstlichen Intelligenz vorgestellt, um die Beschränkungen traditioneller Techniken zur Geräuschreduzierung bei Geothermiebohrungen zu überwinden. Durch die Integration von *Deep Reinforcement Learning* (DRL) mit generativen neuronalen Netzwerkmodellen werden dynamisch Bohrparameter auf der Grundlage von kontinuierlichem Feedback vorgeschlagen. Das System verwendet zwei Modelle, die anhand von realen Daten trainiert wurden: eines zur Vorhersage von Lärmemissionen und das andere zur Vorhersage von Bohrszenarien. Ein DRL-Modell nutzt diese Simulationen, um optimale Bohrstrategien zu erlernen, die den Lärm minimieren und gleichzeitig die Bohreffizienz erhalten. Die Leistung des Systems soll durch die Anwendung in der

Praxis weiter verfeinert werden, um seine Effektivität an verschiedenen städtischen Geothermie-Standorten zu gewährleisten.

Das Projekt ist in drei verschiedene Phasen gegliedert, die in Abbildung 18.1 dargestellt sind:

- Testmodus
- Simulation
- Live-Betrieb

Im *Testmodus* simulieren generative Modelle die Schallausbreitung, das Verhalten der Bohranlage und sagen den Bohrfortschritt anhand bestimmter Zustände und Befehle voraus. In der *Simulationsphase* nutzt ein DRL-Modell die Eingaben der generativen Modelle, um die Steuerung der simulierten Bohranlage durch Wiederholungen zu erlernen und Betriebsempfehlungen zu geben, die dann anhand realer Szenarien überprüft werden. Im *Live-Betrieb* schließlich fungiert das DRL-System als Assistent, der während des Bohrvorgangs strategische Steuerungsänderungen empfiehlt. Das Bedienpersonal behält dabei die Kontrolle, überwacht den Prozess und bewertet die von der KI empfohlenen Befehle. Das System wird kontinuierlich verbessert, indem es aus aktuellen Daten und menschlichem Feedback lernt.

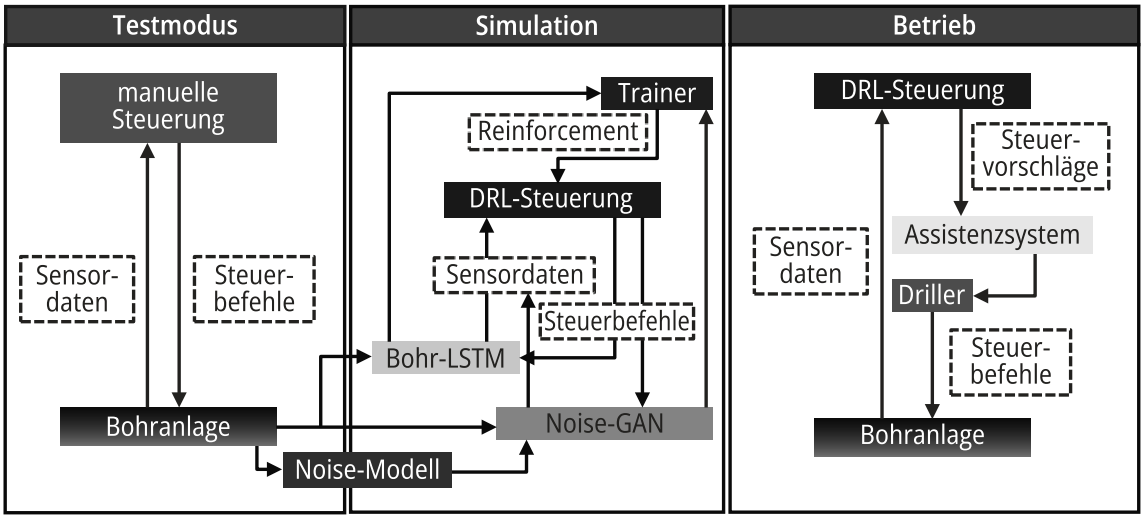

Abb. 18.1 *Architektur der KI-Modelle während der Entwicklungsphasen: Testmodus, Simulation und Live-Betrieb*

18.2 Forschungsstand: generative Modelle und Deep Reinforcement Learning

Dieser Abschnitt bündelt Beiträge zu generativen Modellen mit Physikbezug und deren Einsatz für DRL-Policies sowie realitätsnahe Simulationen. Die Integration physikalischer Prinzipien in generative Modelle ist ein schnell voranschreitendes Feld. Modelle wie PUGAN [Cong et al. 2023] und FEM-GAN [Argilaga 2023] haben erfolgreich Generative Adversarial Networks (GANs) mit physikalischer Modellierung

kombiniert und so die Leistung in Umgebungen mit komplexen physikalischen Gesetzen verbessert. Physikalisch geführte GANs haben die Effizienz und Präzision in Bereichen wie der Fluiddynamik und der Identifizierung von Struktursystemen durch die Einbeziehung physikbasierter Verlustfunktionen und Simulationen deutlich verbessert [Kim et al. 2018; Yu & Liu 2024]. Darüber hinaus hat das maschinelle Lernen erhebliche Fortschritte beim Verstehen und Vorhersagen physikalischer Interaktionen gemacht. Dies zeigen z. B. Modelle, die die Dynamik von Blocktürmen über das einfache Auswendiglernen hinaus erfassen [Lerer et al. 2016], oder Anwendungen wie die Sturzerkennung durch Verfolgung von Körperteilen [Li et al. 2016] und die Erzeugung physikalisch plausibler menschlicher Animationen [Yuan et al. 2023]. Darüber hinaus haben physikgesteuerte KI-Ansätze, einschließlich Grey-Box-Modelle, physikalische Gesetze effektiv in das Modelltraining integriert und so die Leistung und Zuverlässigkeit verbessert [Daw et al. 2021; Gao et al. 2023].

Die Integration von generativen Modellen mit Deep Reinforcement Learning wurde erstmals von Sun et al. [Sun et al. 2024] anhand einer Fallstudie im Bereich der Nah-/Fernfeldkommunikation untersucht. In dieser Arbeit wird erwähnt, wie generative KI das Deep Reinforcement Learning verbessern könnte. Die Simulation von DRL-Umgebungen mit generativen Modellen anhand realer Szenarien könnte weitere Anwendungen für DRL erschließen.

18.3 Assistenzsystem

Die Vorhersagen, Prognosen und Empfehlungen der entwickelten Modelle werden dem Bohrpersonal in einer ständig aktualisierten Benutzeroberfläche des Assistenzsystems präsentiert. Abbildung 18.2 zeigt eine Visualisierung der Anwendung. Die Schnittstelle ist in drei Bereiche unterteilt:

- Statusbereich (oben)
- Analysebereich (Mitte)
- Empfehlungsbereich (unten)

Im Statusbereich werden die beiden wichtigsten Indikatoren angezeigt: die Durchdringungsrate (Rate of Penetration, ROP) und der Schallpegel. Dies ermöglicht einen schnellen Überblick über den aktuellen Zustand. Der Analysebereich zeigt die historischen, die aktuellen und die prognostizierten Trends der Überwachungsvariablen an. Diese Datendiagramme helfen bei der Erklärung einer gegebenen Empfehlung zur Anpassung der aktuellen Maschinensteuerungsparameter und unterstützen den Maschinenbediener bei der Entscheidung, die Empfehlung anzuwenden oder abzulehnen. Insbesondere zeigt es die Prognose des Maschinenverhaltens mit und ohne Annahme der Empfehlung. Der Empfehlungsbereich zeigt neue Steuerungsempfehlungen des DRL-Agenten an und ermöglicht auch die Abgabe von Feedback zu deren Qualität. Eine zentrale Middleware ruft die Maschinendaten ab, bereitet sie für die verschiedenen Modelle auf, wartet die Ergebnisse ab und sendet synchron Updates an die Benutzeroberfläche.

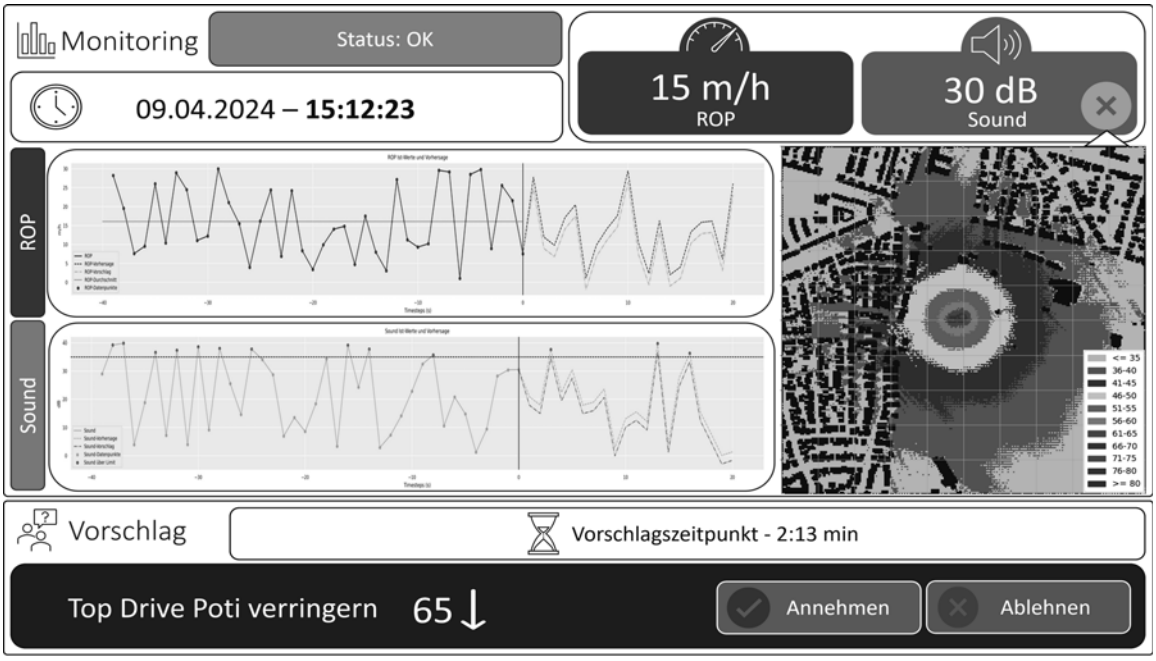

Abb. 18.2 *Benutzeroberfläche des Assistenzsystems*

18.4 Drill-LSTM-Modell: Vorhersage von Maschinenzuständen für optimierte Bohrprozesse

Die präzise Vorhersage von Maschinenzuständen ist für eine fundierte Entscheidungsfindung bei Bohrungen unerlässlich. Das Drill-LSTM-Modell dient als grundlegende Komponente für die Simulation des Maschinenverhaltens und ermöglicht es dem DRL-Agenten, die Ergebnisse verschiedener Steuerungsmaßnahmen zu verstehen und vorherzusehen. In diesem Projekt wurde ein Sequenz-zu-Sequenz-Long-Shortterm-Memory-(LSTM-)Netzwerk verwendet, um zukünftige Maschinenzustände auf der Grundlage von definierten Aktionsmerkmalen vorherzusagen. Der Datensatz enthält etwa 700 Merkmale, die während des gesamten Bohrvorgangs gesammelt wurden. Der Trainingsdatensatz umfasst Bohrvorgänge über einen Monat, während der Testdatensatz eine Woche abdeckt. Im Rahmen der Datenvorverarbeitung wurde das Aufzeichnungsintervall auf ein einheitliches 60-Sekunden-Intervall standardisiert und sämtliche Merkmale wurden normalisiert. Für die Vorhersageaufgabe wurde das Modell so trainiert, dass es den nächsten Zeitschritt auf der Grundlage eines definierten Verlaufsfensters vorhersagt, wobei sich die aktuellen Experimente auf die Vorhersage von einem Schritt konzentrieren. Das Modell wurde für 50 Epochen mit der Verlustfunktion des mittleren quadratischen Fehlers (engl. *Mean Squared Error*, MSE) trainiert.

Abbildung 18.3 veranschaulicht die Vorhersageleistung des Drill-LSTM-Modells anhand von vier anonymisierten Maschinenmerkmalen. Das Modell erfasst erfolgreich allgemeine Trends. Diese Ergebnisse bilden eine Grundlage für künftige mehrstufige Vorhersagen, die architektonische Verbesserungen erfordern, um die Vorhersagegenauigkeit über längere Zeiträume aufrechtzuerhalten. Der Hauptzweck des Vorhersagemodells besteht darin, als Maschinenmodell für das Assistenzsystem zu dienen, das die Vorhersage zukünftiger Maschinenzustände auf der Grundlage potenzieller Benut-

zeraktionen ermöglicht. Die wahren Maschinenparameterwerte (Linien) werden mit den vorhergesagten Werten (Strichlinien) über den Testdatensatz verglichen.

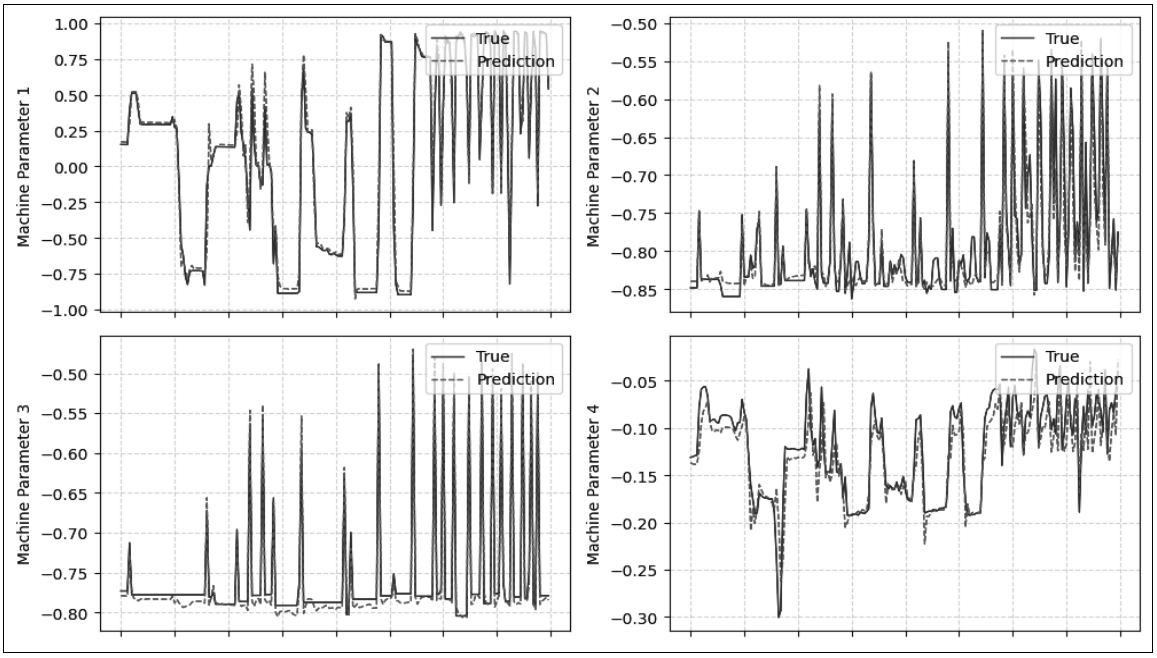

Abb. 18.3 *Vorhersageleistung des Sequenz-zu-Sequenz-LSTM-Modells für vier Maschinenmerkmale*

18.5 Sound-GAN: Echtzeitprognose der Schallausbreitung für Lärmminimierung

Das Verständnis und die Vorhersage der Schallausbreitung sind für die Minderung der Lärmbelästigung bei Bohrarbeiten von entscheidender Bedeutung. Das Sound-GAN-Framework nutzt generative Modelle, um die Schallverteilung effizient zu simulieren, und liefert Echtzeit-Feedback, das der DRL-Agent nutzen kann, um dem menschlichen Bedienpersonal lärmmindernde Maßnahmen zu empfehlen.

Ein Vergleich der Leistungsfähigkeit dieses modellbasierten Ansatzes gegenüber einer Simulation bei der Verarbeitung eines einzelnen Daten-Samples wird in Tabelle 18.1 dargestellt. Die Tabelle verdeutlicht insbesondere, wie die benötigte Rechenzeit bei komplexeren Schallquellen erheblich zunehmen kann.

Model Condition	Mean Runtime (s)	Std. Dev. (s)
UNet	0.0126	0.0012
GAN	0.0095	0.0012
Diffusion	4.1560	0.0061
Simulation Single Source – 3rd Order Reflections	186.2295	16.8491
Simulation Machine Setup – Complex Source	540.0000	-

Tab. 18.1 *Leistungsvergleich in Sekunden für die Verarbeitung eines Standortes durch generative Modelle (UNet, GAN, Diffusion) im Vergleich zur Simulationsmethode mit einfacher und komplexerer Schallquelle*

Als Beispiel dient ein Testaufbau für eine komplexe Quelle mit 28 beschreibenden Schallquellen, bei dem die Simulationszeit deutlich ansteigt. Es ist jedoch wichtig anzumerken, dass dieser Vergleich aus theoretischer Sicht möglicherweise keine gänzlich faire Bewertung liefert, da bei der Erstellung der Analyse keine zusätzlichen Anstrengungen unternommen wurden, um die herkömmlichen Simulationscodes speziell für eine schnellere GPU-Ausführung zu optimieren, was die gemessenen Leistungsunterschiede beeinflussen könnte.

Simulation vs. Generative AI

Aufbauend auf früheren Arbeiten, in denen die Effektivität generativer Modelle zur Vorhersage der Schallausbreitung evaluiert wurde [Spitznagel & Keuper 2024], wurden über 15.000 Datenproben mit dem Noise Modelling v4 Framework [Bocher et al. 2019] generiert, das den CNOSSOS-EU-Standards entspricht [Kephalopoulos et al. 2012]. Jede Probe wurde durch eindeutige Bohrungsparameter definiert. Diese Simulationsdaten dienen als Grundlage für drei verschiedene generative Bild-zu-Bild-Modelle: GANs (auf Basis von [Isola et al. 2016]), UNets [Ronneberger et al. 2015] und DDPM-Diffusionsmodelle (Denoising Diffusion Probabilistic Models, [Ho et al. 2020]). Generative Modelle übertreffen herkömmliche Schallausbreitungssimulationen hinsichtlich der Verarbeitungsgeschwindigkeit erheblich und erreichen eine bis zu 50.000-fache Verbesserung der Laufzeit bei einem mittleren absoluten Fehler von 0,55 dB in ihren Vorhersagen.

Simulationsaufbau

Der Datensatz für dieses Projekt wurde mit dem Noise Modelling Framework v4 [Bocher et al. 2019] erstellt. Es wurden systematisch fünf wichtige Maschinenparameter geändert, die jeweils verschiedene Komponenten repräsentieren, die die Lärmverteilung um eine stationäre Bohrmaschine beeinflussen. Die Parameter wurden anonymisiert, wobei die Belastungswerte von 1,0, der maximalen dB-Leistung einer Komponente, bis 0,5 reichten, was eine lineare Dämpfung von -20 dB anzeigt. Jede Simulation basierte auf dem festen Standort der Bohrmaschine und den anfänglichen Lärmmessungen.

Der endgültige Datensatz umfasst über 15.000 Datenpunkte, von denen jeder eine einzigartige Kombination von Maschineneinstellungen darstellt. Dabei bleibt die Geräuschquelle über alle Simulationen hinweg stationär. Abbildung 18.4 zeigt einen Vergleich der Schallausbreitungskarten unter verschiedenen Betriebszuständen. Die linke Karte (a) stellt niedrige und die rechte Karte (b) hohe Lasteinstellungen dar, die beide farblich nach dB-Pegeln codiert sind. Mithilfe dieses Simulationsaufbaus kann das Modell erfassen, wie sich unterschiedliche Maschinenlasten auf die räumliche Verteilung des Lärms auswirken.

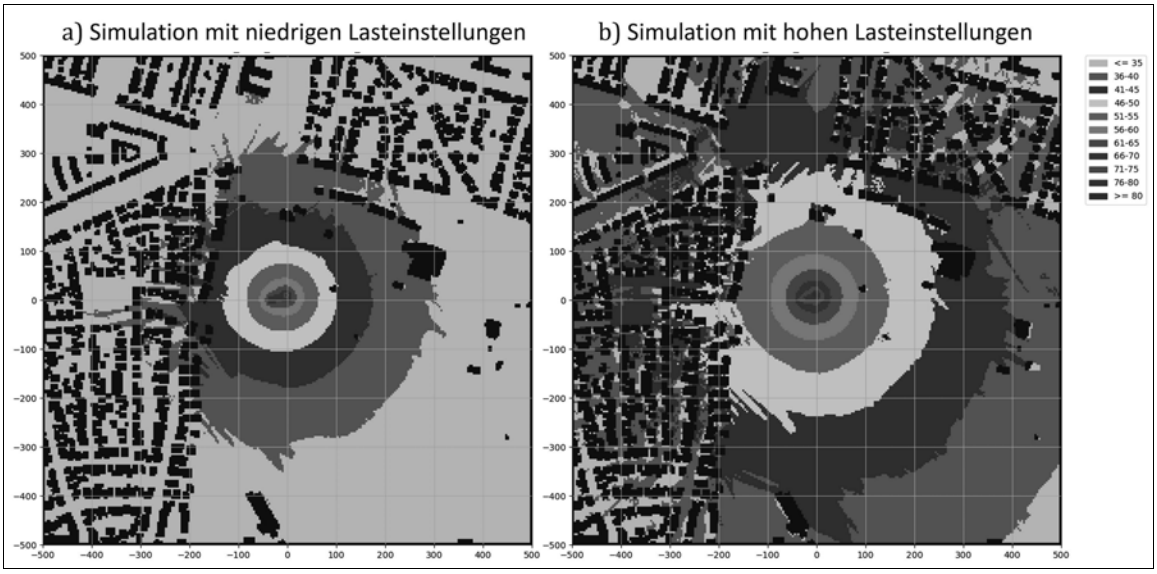

Abb. 18.4 *Vergleich der simulierten Schallausbreitungskarten bei verschiedenen Maschineneinstellungen*

Ergebnisse der Soundvorhersage

Basierend auf diesem Simulationsaufbau wurde die Vorhersage der Schallausbreitung anhand einer festen Graustufen-OpenStreetMap (OSM) untersucht, bei der Gebäude durch schwarze Pixel und Freiflächen durch weiße Pixel dargestellt werden. Die Aufgabe wurde als eine bedingte Bild-zu-Bild-Übersetzung formuliert, bei der jedes Modell das gleiche OSM-Bild als Eingabe erhielt und das Ziel darin bestand, die interpolierte Schallverteilungskarte auf der Grundlage der eingestellten Maschinenparameter vorherzusagen.

Jede Modellarchitektur wurde für 50 Epochen trainiert, wobei 20 % des Datensatzes zum Testen zurückgehalten wurden. Die UNet- und Diffusionsmodelle wurden mit der Verlustfunktion des mittleren quadratischen Fehlers (MSE) trainiert, während das GAN eine Kombination aus Binary-Cross-Entropy-(BCE-)Loss und L1-Loss verwendete. Die Bewertung der Modelle basierte auf zwei Schlüsselmetriken: Mean Absolute Error (MAE), der die durchschnittliche Größe der Vorhersagefehler quantifiziert, und Weighted Mean Absolute Percentage Error (wMAPE), der falsche Vorhersagen hinter und innerhalb von Gebäuden bestraft.

Model	MAE	wMAPE	LoS MAE	NLoS MAE	LoS wMAPE	NLoS wMAPE
UNet	0.70	12.78	0.58	0.85	5.32	21.87
GAN	0.48	3.42	0.32	0.68	1.93	5.24
DDPM	1.19	24.94	1.07	1.35	14.23	37.98

Tab. 18.2 *Bewertung aller Architekturen bei LoS- und NLoS-Aufgaben unter Verwendung von MAE- und wMAPE-Metriken*

Die Ergebnisse der Bewertung sind in Tabelle 18.2 zusammengefasst, die die Leistung jedes Modells sowohl für Line-of-Sight- (LoS) als auch für Non-Line-of-Sight-Regio-

nen (NLoS) zeigt. Das GAN-Modell übertraf durchweg die anderen Architekturen und erzielte sowohl unter LoS- als auch unter NLoS-Bedingungen die niedrigsten MAE- und wMAPE-Werte. Um die Leistung des GAN-Modells weiter zu analysieren, wurde eine Heatmap des wMAPE über den gesamten Testdatensatz erstellt, wie in Abbildung 18.5 dargestellt. Diese Heatmap bietet eine pixelweise Visualisierung des Vorhersagefehlers. Die höchsten Fehler treten in NLoS-Bereichen auf.

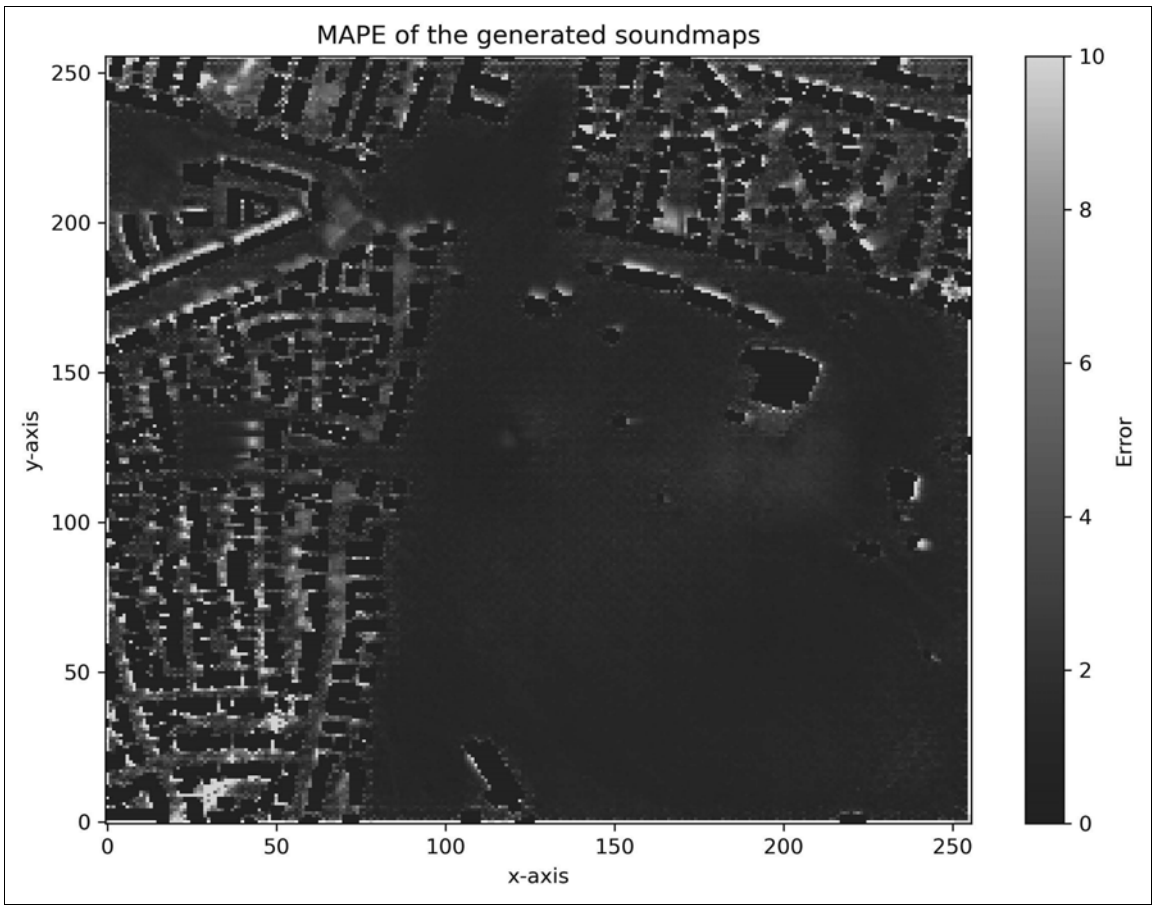

Abb. 18.5 *Heatmap des mittleren absoluten prozentualen Fehlers (MAPE) über den gesamten Datensatz für die UNet-Vorhersage*

Zusätzlich zur quantitativen Analyse bietet Abbildung 18.6 einen visuellen Vergleich der von allen drei Modellen vorhergesagten Schallverteilungen für einen einzelnen Datenpunkt. Das GAN-Modell zeigt die genaueste räumliche Verteilung des Schalls. Im Gegensatz dazu weist das Diffusionsmodell größere Abweichungen mit einer sichtbaren Übervorhersage auf, insbesondere an Gebäudekanten und in verdeckten Räumen.

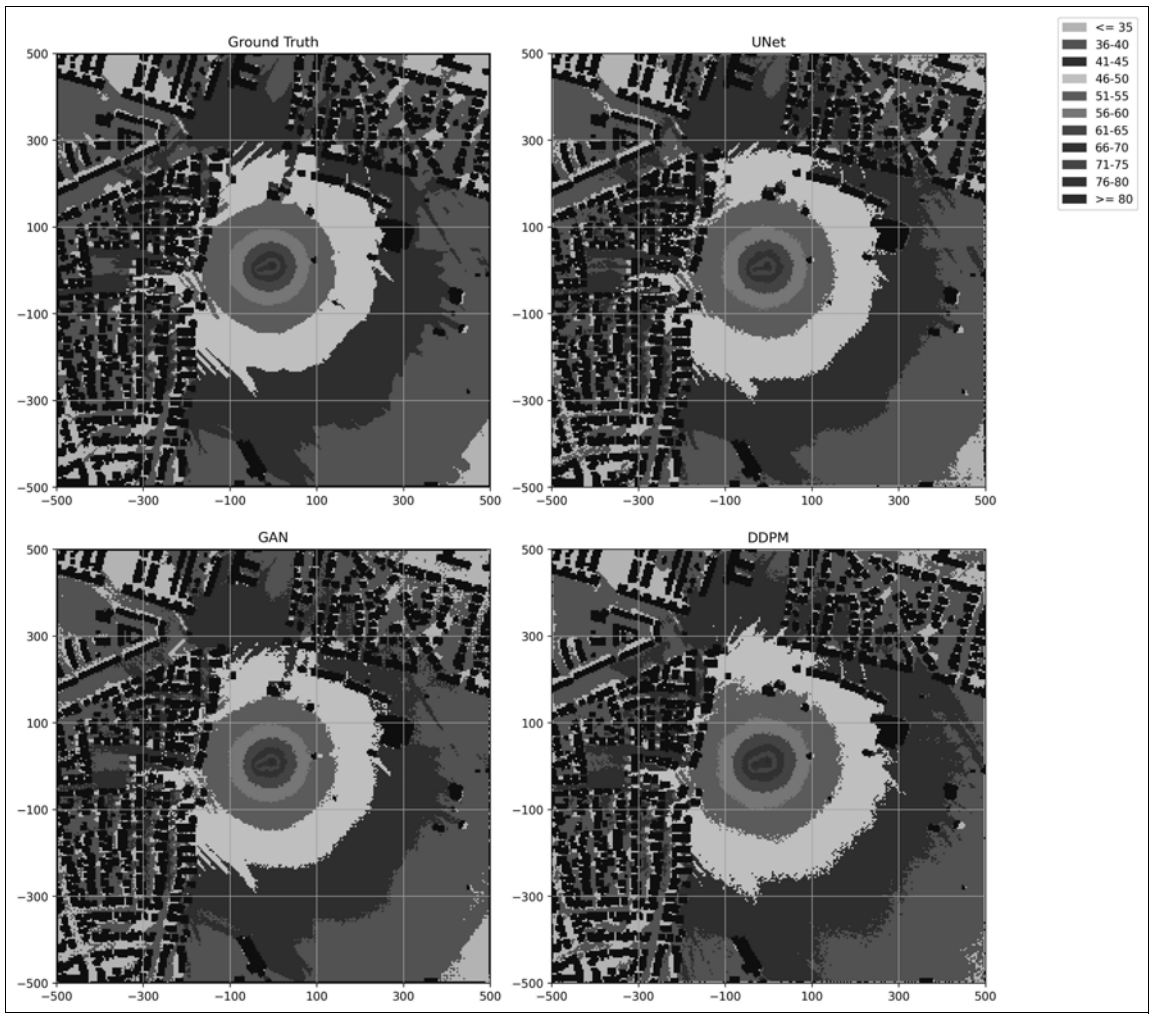

Abb. 18.6 *Vergleich der tatsächlichen mit der vorhergesagten Schallausbreitung für einen einzelnen Datenpunkt mit drei Modellen: UNet, GAN und DDPM*

18.6 DRL-Agent: intelligente Steuerungsempfehlung für lärmoptimierte Bohrprozesse

Die Empfehlung optimaler Steuerungsanpassungen für Geothermie-Bohranlagen erfordert eine effektive Steuerungsstrategie. In diesem Kontext wird Deep Reinforcement Learning eingesetzt, um einen Agenten zu trainieren, der diese Aufgabe erfüllt. Da das Training an der realen Bohrmaschine keine sinnvolle Option ist, wird eine Simulation der Maschine (DRILL-LSTM) entwickelt, um das notwendige Maschinenverhalten für das Training zu emulieren. Um den entstehenden Lärm eines Maschinenzustandes nach Steuerungsänderungen an der simulierten Maschine überwachen und darauf reagieren zu können, wird ein Modell namens NOISE-GAN erstellt. Diese beiden Modelle bilden die Voraussetzung für die Entwicklung eines DRL-Agenten. Die Kombination von generativen Modellen mit Deep Reinforcement Learning ist vielversprechend, insbesondere für Bereiche, in denen das Training eines Agenten an einer realen Maschine nicht möglich ist.

Die zweite entscheidende Herausforderung, die es zu lösen gilt, besteht darin, Steuerungsänderungen zu empfehlen, die von menschlichen Bedienern verarbeitet werden können. Ein DRL-Agent arbeitet typischerweise mit hoher Frequenz und hat direkten Zugang zur Umgebung, um bei Bedarf schnell auf Änderungen reagieren zu können. Da die Steuerung der Bohrmaschine immer noch vom menschlichen Bediener abhängt, der die endgültige Entscheidung trifft, müssen die Bohrempfehlungen besondere Anforderungen erfüllen:

- Die Empfehlungen müssen so prägnant wie möglich sein, um den Bohrer nicht zu stören oder abzulenken, aber mit der größtmöglichen Wirkung innerhalb eines vernünftigen Zeitrahmens.
- Aktuelle Empfehlungen müssen kontinuierlich auf ihre Relevanz überwacht werden, damit sie bei Überschreitung eines kritischen Zeitfensters oder bei Änderungen des Maschinenzustands, die sie obsolet machen würden, verworfen oder aktualisiert werden können.

Die Herausforderung besteht also darin, Empfehlungen für Menschen auf der Grundlage eines hochfrequent beobachtenden DRL-Agenten zu erstellen.

18.7 Zusammenfassung

In diesem Kapitel wurde gezeigt, dass durch den Einsatz generativer KI-Modelle die Vorhersage der Schallausbreitung um einen Faktor 50.000 beschleunigt werden kann. Diese signifikante Performanzsteigerung ermöglicht das effiziente Training eines Deep-Reinforcement-Learning-Modells, das Bohrprozesse in urbanen Gebieten hinsichtlich Geschwindigkeit und Lärmemission optimiert. Die so verringerten Schallemissionen von Geothermiebohrungen können die Toleranz und Akzeptanz in der Bevölkerung gegenüber solchen Projekten in Städten erhöhen. Dies trägt zur wirtschaftlichen Tragfähigkeit solcher Projekte bei und zu ihrem potenziellen Einfluss auf die Erreichung der Klimaziele. Die Integration von KI zur Lärmkontrolle bei Geothermiebohrungen stellt eine transformative Lösung für eine der dringendsten Herausforderungen bei städtischen Projekten für erneuerbare Energien dar. Dies verspricht nicht nur eine wirksame Reduzierung der Lärmbelästigung, sondern auch eine Optimierung der Bohrarbeiten. Die Fortschritte in der generativen KI eröffnen Wege zum Einsatz von Deep Reinforcement Learning für Anwendungsfelder, in denen ein direktes Training in der realen Umgebung nicht möglich ist.

Danksagung

Das Projekt »KI-Bohrer« wird durch Förderung des Bundesministeriums für Forschung, Technologie und Raumfahrt (BMFTR) im Programm »Forschung an Fachhochschulen in Kooperation mit Unternehmen (FH-Kooperativ)« finanziell unterstützt (Förderung 13FH525KX1).