# Proseminar Learning Surrogate Models For Fluid Simulation

Dominik Wüst

dominik.wuest@student.kit.edu
Karlsruher Institut für Technologie, 76131 Karlsruhe, Germany
Betreuer: Till Riedel

**Zusammenfassung.** Die akkurate Simulation von physikalischem Verhalten erfordert häufig erheblichen Rechenaufwand. Um diese Berechnungen mit geringerem Rechenaufwand zu approximieren wurde in den letzten Jahrzehnten an einer Vielzahl von Approximationsmodellen geforscht. Insbesondere im Bereich der Strömungssimulationen wurde in den letzten Jahren deutlicher Fortschritt gemacht. Diese Arbeit bietet einen Überblick über den Stand der Forschung im Bereich der Approximation von Computational Fluid Dynamics (CFD) und Large Eddy Simulationen (LES) mittels eingelernter Ersatzmodelle. Es werden Forschungsergebnisse für den Einsatz verschiedener Neuronaler Netze in den jeweiligen Anwendungsfeldern dargestellt und verglichen.

**Schlüsselwörter:** surrogate model · neural network · fluid simulations · CFD · LES · FEM

## 1 Einleitung

Physiksimulationen sind ein alltäglicher Bestandteil in vielen Ingenieursberufen, so finden Simulationen von Kräfte- und Spannungsverteilungen in einzelnen Bauteilen, Gruppen von Teilen und ganzen Maschinen Einsatz in vielen Bereichen der Konstruktion, dem Automobilbau und der Architektur. Für Luft- und Raumfahrt sowie die Automobilindustrie sind die Simulation von Stromlinien und aerodynamischem Verhalten ein zentrales Thema. In diesen Bereichen erweisen sich Simulationen häufig als erheblich günstiger als reale Tests an gefertigten Teilen. Auch in der Film- und Videospielindustrie wird immer mehr Gebrauch gemacht von realistischen Gas- und Flüssigkeitssimulationen.

Dennoch sind dem Einsatz dieser Simulationen Grenzen gesetzt. Für ein akkurates Ergebnis ist je nach Komplexität und Auflösung des Modells ein erheblicher Rechenaufwand von mehreren Stunden bis hin zu Wochen nötig. Besonders für Echtzeitanwendungen wie Videospiele aber auch für die Evaluation eines Entwurfs während der Entwicklung ist solch ein Rechenaufwand nicht möglich. In den letzten Jahrzehnten wurde daher ausgiebig an Approximationsmodellen für diese Simulationen geforscht. Das Ziel ist die Approximation der Simulation mit erheblich kürzerer Rechenzeit. Als Ersatzmodelle haben sich eingelernte Systeme

als vielversprechend erwiesen, da die nötigen Trainingsdaten direkt aus der Simulation generiert werden. Insbesondere Neuronale Netze lassen sich nach einer erfolgreichen (rechenintensiven) Trainingsphase schnell und häufig auch parallel auswerten.

In vielen Bereichen werden unterschiedliche Simulationen verwendet, sowie sehr unterschiedliche Anforderungen an die Simulation und damit auch die Ersatzmodelle gestellt. Hier heben sich die aktuellen Arbeiten zur Approximation von Flüssigkeitssimulationen deutlich von denen zur Finite-Elemente-Methode (FEM) ab: Während im FEM Bereich auf spezialisierte Modelle für die jeweilige Anwendung gesetzt wird, versuchen die Arbeiten zu Flüssigkeitssimulationen die zugrundeliegenden physikalischen Eigenschaften zu generalisieren. Um diese Aussage zu belegen wird in dieser Arbeit ein Überblick über die Forschungsergebnisse der letzten Jahre gegeben. Verschiedene Ansätze und Neuronale Netze werden aufgeführt und die Einschränkungen und Ergebnisse verglichen.

## 2 Physiksimulationen im Überblick

### 2.1 Finite-Elemente-Methode

Für die physikalisch akkurate Modellierung von Kräfte- und Spannungsverteilungen ist es in der Regel notwendig, Differentialgleichungssysteme (DGS) zu lösen. Die vorkommenden Ableitungen beschreiben dabei das Änderungsverhalten einzelner Größen, im Falle von Flüssigkeiten sind z.B häufig die Geschwindigkeit sowie die Beschleunigung in den Gleichungen enthalten. Diese von Ort (und häufig auch Zeit) abhängigen Gleichungen modellieren die Ausbreitung, Übertragung und Erhaltung von Kräften, Impulsen oder Bewegungen innerhalb eines gegebenen Systems. Zusammen mit einigen Gleichungen zur Beschreibung von Randbedingungen bilden diese ein nichtlineares DGS (NL-DGS). [10] Eine direkte Lösung solcher NL-DGS ist in den meisten Fällen nicht möglich. Daher finden numerische Verfahren wie die *Finite-Elemente-Methode* (FEM) hier Anwendung. [2]

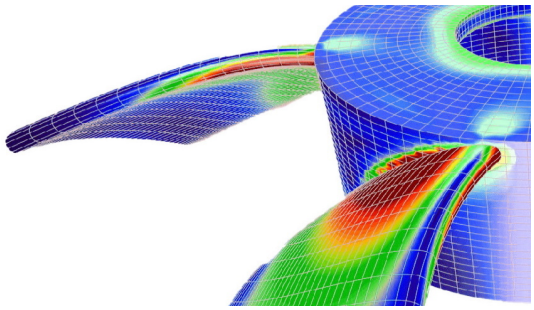

**Abb. 1.** FEM Simulation auf einem dreidimensionalen Körper
**Quelle** `https://www.s-inotec.de/wp-content/uploads/2020/05/FEM_Cut.jpg`

Bei der FEM handelt es sich um ein Verfahren zur Diskretisierung von DGS. Dabei wird der zugrunde liegende Körper (z.B. ein Festkörper oder eine Flüssigkeit) in endlich viele Teilkörper einfacher Form (z.B. Quader, Dreiecke) zerlegt. Abbildung 1 zeigt eine solche Zerlegung in Quader. Die Teilkörper werden als Elemente bezeichnet. Daher kommt der Name „Finite-Elemente-Methode“. Das physikalische Verhalten dieser Elemente kann aufgrund der einfachen Form mit bekannten Ansätzen berechnet werden. Das Verhalten des Gesamtkörpers wird nun durch das Verhalten der Elemente sowie deren Interaktion untereinander nachgebildet. Für die Ansatzfunktionen zur Lösung der DGS auf den Elementen wird dazu Kontinuität an den Übergängen zu anderen Elementen gefordert. So entsteht ein NL-DGS aus den Ansätzen zur Lösung der finiten Elemente sowie den Kontinuitätsbedingungen an den Übergängen. Im Gegensatz zur Ausgangssituation handelt es sich hierbei um ein diskretes System. Für solche Gleichungssysteme existieren numerische Lösungsverfahren, diese liefern eine Approximation der Lösung des Ausgangssystems. Die Genauigkeit der Approximation ist dabei abhängig von der Anzahl der gewählten finiten Elemente. [2]

### 2.2 Fluid Dynamics und die Navier-Stokes Gleichungen

Mit Navier-Stokes Gleichungen sind die Impulsgleichungen für Strömungen gemeint. Dabei handelt es sich um ein nichtlineares Gleichungssystem von partiellen Differentialgleichungen zweiter Ordnung. Die Gleichungen beschreiben die Strömungsrichtungen anhand von Druck und Dichte der umgebenden Flüssigkeit und unter Berücksichtigung der bestehenden Stömungsrichtungen. Eine Lösung dieses NL-DGS beschreibt somit den Strömungsverlauf. Es sei gesagt, dass bisher nicht bewiesen wurde, ob die Navier-Stokes Gleichungen in ihrer allgemeinen Form (eindeutig) lösbar sind. Diese Fragestellung gehört zu den bisher ungelösten Millennium-Problemen der Mathematik. [10]

Computational Fluid Dynamics (CFD) ist ein Überbegriff für numerische Lösungsansätze der Strömungsmechanik. Ausgangspunkt ist ein NL-DGS (in der Regel die Navier-Stokes Gleichungen). Je nach gewählter Flüssigkeit können bereits an den Differentialgleichungen Vereinfachungen vorgenommen werden, z.B. falls die Flüssigkeit nicht kompressierbar ist. Zur Lösung des Gleichungssystems wird eine Zerlegung in statische Gitterzellen (wie die Elemente bei FEM) oder bewegte Partikel vorgenommen. Ein wichtiger Bestandteil der CFD ist die Ermittlung geeigneter Gitterauflösungen bzw. Partikelanzahlen für die Diskretisierung, sodass z.B. auch Turbulenzen darstellbar sind. [11]

### 2.3 Turbulenzen: Large Eddy Simulationen

Wie auch CFD dienen die Large Eddy Simulationen (LES) der Lösung von NL-DGS zur Strömungsmodellierung von Flüssigkeiten und Gasen. Bei der Berechnung der Strömungen mittels CFD können zwar Wirbel und Turbulenzen berechnet werden, jedoch ist hierfür eine sehr feine Gitterauflösung notwendig. Dies hat sehr hohen Rechenaufwand zur Folge. Die LES befasst sich speziell

mit der Berechnung von Turbulenzen und Wirbelstrukturen. Dazu werden die Navier-Stokes Gleichungen bezüglich Ort und Zeit tiefpassgefiltert. So lassen sich die Wirbelstrukturen effizient über bekannte Verfahren wie FEM berechnen. Jedoch sind die durch den Tiefpass abgeschnittenen Informationen nicht irrelevant und müssen durch Feinstrukturmodelle, sogenannte *Subgrid-scale* (SGS) Modelle, approximiert werden. Diese SGS Modelle sind aktueller Gegenstand der Forschung, neben mathematischen und empirischen Modellen werden auch Neuronale Netze angewendet. Ein weiterer Ansatz sind *Reduced Order Modelle* (ROM), welche ein reduziertes NL-DGS lösen.[4]

### 2.4 Motivation für Ersatzmodelle

Bei all den oben vorgestellten Verfahren steht die Lösung von NL-DGS im Mittelpunkt. Dabei haben die Verfahren gemeinsam, dass das Ausgangssystem mit einer begrenzten Gitterauflösung diskretisiert wird. Diese Abtastung liefert zwar ein numerisch lösbares Gleichungssystem, jedoch ist dessen Lösung nur eine Approximation der eigentlich gesuchten Lösung. Die Genauigkeit der Approximation ist dabei abhängig von der gewählten Gitterauflösung. Da auch die Komplexität des konkretisierten Systems abhängig ist von der Gitterauflösung steigt der Rechenaufwand mit zunehmender Genauigkeit der Approximation stark an.

Ein weiteres Problem ist die Lösung des konkretisierten Gleichungssystems an sich: Es handelt sich nach wie vor um ein NL-DGS. Zwar sind für die einzelnen Differentialgleichungen aufgrund der einfachen Form der gewählten Elemente bereits Lösungsansätze bekannt, jedoch sind auch diese unterschiedlich komplex. Hinzu kommt der nicht lineare Zusammenhang der Gleichungen, das NL-DGS kann nur iterativ, z.B. über das Newton-Verfahren gelöst werden. So tragen neben der Gitterauflösung bei der Diskretisierung auch die Form der gewählten Elemente sowie die Konvergenzgeschwindigkeit des iterativen Lösungsverfahrens zur Gesamtlaufzeit bei. Da sich diese Berechnungen nur begrenzt parallelisieren lassen liegt die Rechenzeit selbst auf modernster Hardware nicht selten bei mehreren Tagen für eine Simulation.

Im Folgenden wird der Einsatz von eingelernten Systemen und verschiedenen Neuronalen Netzen zur Approximation dieser Verfahren diskutiert. Es werden verschiedene Ersatzmodelle vorgestellt und deren Laufzeit verglichen. Das Hauptaugenmerk liegt dabei auf den Strömungssimulationen. Es werden erfolgsversprechende Ansätze für die jeweilige Kategorie herausgearbeitet sowie deren Schwachstellen kritisch betrachtet. Insbesondere werden auch Modelle betrachtet, welche in mehreren Kategorien erfolgreich eingesetzt wurden. Die Modelle werden auch bezüglich ihrer Fähigkeit zur Generalisierung verglichen.

## 3 Neuronale Netze

### 3.1 Grundlegender Aufbau von neuronalen Netzen

Im folgenden wird die grundlegende Struktur neuronaler Netze (NN) erläutert. NNs bilden eine Menge von Eingabewerten nichtlinear auf eine Menge von Aus-

gabewerten ab. Sie approximieren damit also eine Funktion

$$f : \mathbb{R}^M \to \mathbb{R}^N$$

Aufgrund der samplingbasierten Trainingsmethoden können NNs auf beliebige Probleme, also insbesondere solche die sich nur schwer oder gar nicht mathematisch beschreiben lassen, eingelernt werden. In den letzten Jahrzehnten konnten die NNs aufgrund von stetig steigender Trainingsdatenmegen, verfügbarer Rechenleistung und Weiterentwicklung in vielen Bereichen beachtliche Fortschritte erzielen.

Das NN besteht dabei aus mehreren hintereinandergeschalteten Ebenen (engl. *Layer*), wobei die Ausgabe eines Layers direkt als Eingabe für das darauf folgende verwendet wird. Ein Layer überführt dabei seinen Eingabevektor $\vec{x} = (x_1, ..., x_K)^\top \in \mathbb{R}^K$ abhängig von festgelegten Gewichtswerten in einen Ausgabevektor $\vec{x'} = (x'_1, ..., x'_L)^\top \in \mathbb{R}^L$. Dabei können Ein- und Ausgabe unterschiedliche Dimensionen haben, häufig werden hier auch zwei- oder dreidimensionale Eingaben wie Bilder oder Volumen verarbeitet. Da für die Layer in der Regel nichtlineare Überführungsfunktionen verwendet werden entsteht durch das Hintereinanderreihen mehrerer Layer ein komplexes System mit vielen Freiheitsgraden. Im Falle eines sogenannten *Fully Connected Layer* (FCLayer) wird die Überführung durch Multiplikation mit einer Gewichtsmatrix und anschließender nichtlinearer Aktivierung vorgenommen. Ein NN bestehend aus hintereinandergeschalteten FCLayern wird als *Multi Layer Perceptron* (MLP) bezeichnet. Im Laufe der Arbeit werden auch Convolutional Layers (ConvLayer), welche die Überführung durch Faltung mit einer Filtermatrix realisieren, sowie weitere Layertypen vorgestellt.

Die unterschiedlichen Layer haben gemeinsam, dass ihre Berechnung abhängig von einer Menge an Gewichten durchgeführt wird. Die Gewichte eines gesamten NNs bestimmen daher, was für eine Funktion realisiert wird. Unter dem Training eines NN versteht man das ermitteln dieser Gewichte. Da in aller Regel kontinuierliche Funktionen vom NN dargestellt werden sollen, ist es nicht möglich, den Wertebereich vollständig abzudecken. Daher wählt man eine Menge von Samples aus, welche repräsentativ für die zu lernende Funktion ist. Im Falle des Ende-zu-Ende Trainingsansatzes sind diese Samples Tupel aus Eingabevektor und Ausgabevektor der zu lernenden Funktion: $(\vec{x} = (x_1, x_2, ..., x_M)^\top, \vec{y} = (y_1, y_2, ..., y_N)^\top)$. Die Tupel können Messwerte aus realen Experimenten oder anderen Datenbeständen, welche die zu lernende Funktion beschreiben, sein. Jedoch ist es in vielen Anwendungegebieten sehr aufwändig, eine ausreichend große Samplemenge zu finden. Grund dafür ist der hohe Bedarf an Trainingsdaten: die Samples repräsentieren nur konkrete Punkte der Funktion, um jedoch eine hinreichend gute Abstraktion des NN zu erreichen ist eine hohe Überdeckung nötig. Anderenfalls werden Bereiche zwischen den Samples häufig falsch interpoliert und es entstehen Abweichungen.

Durch die sogenannte Loss-Funktion wird ein Maß für diese Abweichung festgelegt. Dazu wird der Loss-Funktion das erwartete Ergebnis $\vec{d} = (d_1, d_2, ..., d_N)^\top$ sowie der Ausgabevektor $\vec{y} = (y_1, y_2, ..., y_N)^\top$ des NN übergeben und daraus der

Fehler des NN berechnet. Eine häufig verwendete Loss-Funktionen ist der *Mean-Squared-Error* (MSE):

$$loss(\vec{y}, \vec{d}) = \sum_{i=1}^{N} (y_i - d_i)^2$$

Anhand dieser Loss-Funktion ist für das Training des NN ein Minimierungsproblem gegeben: Finde Gewichte für die Layer des NN, sodass $loss(\vec{y}, \vec{d})$ minimal wird. An dieser Formel wird deutlich, dass bei wenigen Samples die zu Grunde liegende Funktion $f$ auch nur an wenigen Punkten approximiert wird und für die dazwischenliegenden Punkte keine Optimierung vorgenommen wird. Die für das Training verwendeten Loss-Funktionen sowie die verwendeten Algorithmen hängen stark vom Layertyp und den zu lernenden Daten ab und werden daher in den folgenden Kapiteln diskutiert.

Alternativ zum Ende-zu-Ende Training existieren auch unbeaufsichtigte Trainingsmethoden, hierbei werden lediglich Eingabesamples zur Verfügung gestellt. Eine Fehlermetrik basierend auf dieser Eingabe wird dann verwendet, um das NN zu trainieren. Dieser Ansatz wird auch als *Deep Learning* (DL) bezeichnet.

### 3.2 Neuronale Netze als Physiksimulation

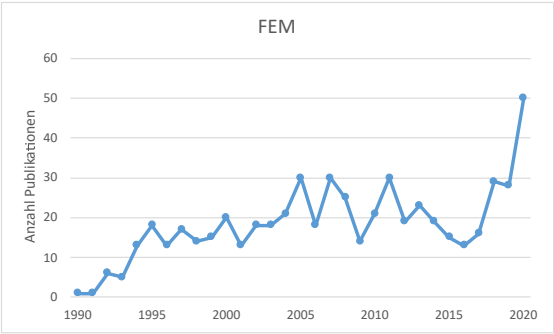


**Abb. 2.** NN Publikationen zu FEM

Die Idee, neuronale Netze als Ersatzmodelle für rechenaufwändige Physiksimulationen zu verwenden kam bereits Anfang der 90er Jahre auf und ist seither Gegenstand der Forschung. Dabei lag das Hauptaugenmerk auf der effizienten Approximation der Finite-Elemente-Methode (FEM). Da FEM ein grundlegendes Verfahren zur Physiksimualtion darstellt erkannte man schon damals den Bedarf an beschleunigten Verfahren. Spezialisierungen wie die CFD Verfahren zur Strömungssimulation wurden erst deutlich später betrachtet. Abbildung 2 zeigt die Anzahl der Veröffentlichungen pro Jahr, welche sich mit der Beschleunigung von FEM durch neuronale Netze beschäftigen. Aus der Abbildung geht das andauernde Forschungsinteresse in diesem Bereich sowie der deutlich steigende

Forschungsaufwand in den letzten drei Jahren hervor. Das steigende Interesse in den letzten Jahren geht einher mit den Fortschritten der Strömungssimulationen, welche in den Folgenden Kapiteln diskutiert werden.

In der Industrie ist es in den meisten Bereichen aus Zeit- und Kostengründen üblich, Entwürfe anhand von Simulationen zu verifizieren, bevor die Produktion und damit die Tests an realen gefertigten Teilen beginnen. Bereits Mitte des 20. Jahrhunderts wurde hierfür FEM eingesetzt. [2] Da bei komplexen Entwürfen die exakte Simulation mittels FEM sehr rechenaufwändig ist, erschließt sich hier ein großer Markt für beschleunigte Approximationsmodelle. Da die Anwendungen und Rahmenbedingungen je nach Indsutriezweig stark variieren unterscheiden sich auch die Anforderungen an die Ersatzmodelle stark. Daher befassen sich die ersten Arbeiten zur Approximation von FEM mit dem Training neuronaler Netze für spezielle Einsatzgebiete. Einige Beispiel hierfür sind:

1. *Hot Rolling Prozess* (Verfahren zur Metallverarbeitung):
   Ein MLP bildet Maschinenparameter wie Druck, Metalltemperatur, etc. auf Materialeigenschaften wie Dichte, innere Spannung, etc. nach der Verarbeitung ab. Die Trainingsdaten werden aus einer FEM Simulation generiert. [15]
2. Lasergesteuerte Biegeprozesse zur Metallverarbeitung:
   Ein MLP bildet Laserenergie, Materialstärke, etc. auf einen Biegewinkel ab. Für das Training kommt die MSE Loss-Funktion zum Einsatz. [1]
3. Strukturelle Schäden von Rohren aufgrund von Korrosion erkennen:
   Dabei werden Position und Größe der Roststellen auf eine Druckobergrenze abgebildet. [16]

Diese spezialisierten Approximationsmodelle sind auch heute ein großer Bestandteil der Forschung, da stetig neue Anforderungen von der Industrie gestellt werden. Hier werden fast ausschließlich MLPs verwendet, diese haben sich in den letzten drei Jahrzehnten als gut geeignet erwiesen. Einer der Gründe ist die effiziente Auswertung der FCLayer: Durch ausnutzen der Parallelisierbarkeit sind die beschriebenen Ansätze meist echtzeitfähig.

Für komplexere Aufgaben kommen in den letzten Jahren auch Deep Learning Ansätze zum Einsatz. Arbeiten wie [6] befassen sich mit dem Training von zusammengesetzten NNs. Diese beinhalten deutlich mehr Layer und benötigen dementsprechend mehr Trainingsdaten. Unüberwachte Trainingsansätze vereinfachen dabei das Erstellen der Datensätze, da keine Ausgabewerte wie beim Ende-zu-Ende Training berechnet werden müssen. [6]

Die Forschung im FEM Bereich ist also sehr anwendungsspezifisch. Dies hat zur Folge, dass für jeden Anwendungsbereich neue NNs entworfen und trainiert werden. Dies spiegelt sich auch im historischen Verlauf der Publikationen wieder: Es finden sich nur wenige Arbeiten, welche sich gegenseitig zitieren, denn Vergleiche oder Weiterentwicklungen sind meist nur im selben Anwendungsgebiet möglich. Im Gegensatz zu den spezialisierten FEM Modellen setzen die Arbeiten im CFD und LES Bereich auf gute Generalisierung des zugrundeliegenden physikalischen Verhaltens. Um diese Aussage zu belegen werden im Folgenden verschiedene Arbeiten vorgestellt und verglichen.

### 3.3 Einsatz von Neuronalen Netzen für Strömungssimulationen

Wir betrachten nun den Einsatz eines NNs für die Strömungssimulation. Dazu befassen wir uns mit der Approximation von gerichtetem Strömungsverhalten im Zweidimensionalen. Solche Strömungen treten unter anderem bei Simulationen und Tests in Windkanälen auf und können mit den Navier-Stokes-Gleichungen modelliert werden. Mit sogenannten CFD-Solvern kann dann auf einem Gitter eine diskrete Lösung ermittelt werden. Die Auflösung dieser Lösung hängt von der gewählten Gitterauflösung ab. Da alle Punkte zwischen den Gitterpunkten interpoliert werden, hängt die Genauigkeit der Lösung also von der Gitterauflösung ab. Gleichzeitig steigt der Rechenaufwand für höhere Auflösungen stark an. Um nun mit weniger Rechenaufwand eine gute Approximation der Strömungen zu erhalten, schlägt [3] den Einsatz von Convolutional Neural Networks (CNNs) zur Approximation des Strömungsfeldes vor. Diese enthalten neben FCLayern auch die in Abschnitt 3.1 angesprochenen ConvLayer und eignen sich daher gut zur Verarbeitung von zweidimensionalen Bildern. Als Eingabe erhält das CNN ein 256x128 Pixel Bild. Dabei sind die Pixelwerte entweder 1, falls sich an der Stelle ein Objekt befindet, oder 0, falls nicht. Für die freien Pixel soll das CNN nun die Strömungsrichtung als Vektor ermitteln und komponentenweise als Ausgabebilder zurückgeben. Das CNN realisiert also eine Funktion

$$f : \{0,1\}^{256\times128\times1} \to \mathbb{R}^{256\times128\times2}$$

Die nötigen Samples können mithilfe von CFD-Solvern erstellt werden. Hierzu werden zufällig gewählte Primitive wie Dreiecke, Kreise, Rechtecke,... an zufällig gewählten Positionen im Bild platziert. Die Ausgabe wird dann mit einem CFD-Solver berechnet. Auf diese Weise wird ein Datensatz von 100.000 Samples für das Training und 10.000 Samples zur Validierung erstellt.

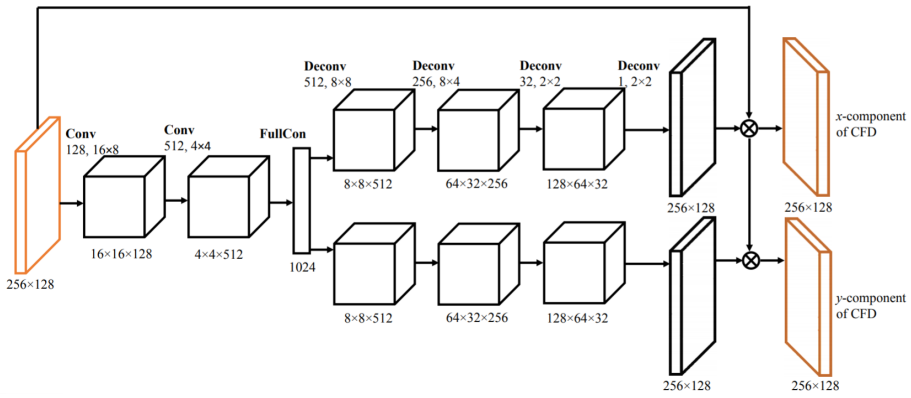


**Abb. 3.** Aufbau des CNN **Quelle** [3]

Bei CNNs wird die Anzahl der benötigten Gewichte im Vergleich zu MLPs stark reduziert, da hier für die ConvLayer nur eine Filtermatrix gespeichert wird. Diese ist in der Regel sehr viel kleiner als die Gewichtsmatrix eines FCLayer. Zudem kann das CNN die zweidimensionale Struktur des Eingabebildes erhalten und nutzen. Das zur Approximation verwendete CNN besteht aus zwei ConvLayers gefolgt von einem FCLayer. Anschließend werden aus dieser Ausgabe die x- und y-Komponente der Stömungsrichtung separat durch drei Deconvolution-Layers (DeconvLayer) extrahiert. Die Ausgabe wird nun noch maskiert, sodass die zugrundeliegende Geometrie erkennbar ist.

Das CNN wird auf den Trainingssamples trainiert und der Trainingsfortschritt anhand der Validierungssamples gemessen. Zeigt sich kein neuer Trainingsfortschritt, so wird das Training beendet. Als Loss-Funktion wird der *Average Relative Error* (ARE) verwendet. Dabei werden Ausgabe des CNN und erwartetes Ergebnis pixelweise verglichen und die Abweichung relativ zur Länge der erwarteten Richtung gesetzt. Anschließend wird der Durchschnitt über alle Pixel gebildet. Diese Fehlermetrik bewirkt, dass die Vektoren mit großer Länge, also die Bereiche mit starker Strömung, den gleichen Einfluss haben wie Bereiche mit schwacher Strömung. In Abbildung 4 ist zu sehen, dass auch die strömungsarmen Bereiche (blau) gut approximiert werden.

Um nun das Ergebnis zu bewerten wird eine Menge von Testsamples verwendet. Hierbei handelt es sich um Querschnitte von verschiedenen Autos, z.B. Jeeps, Sportwagen, etc. Für den Vergleich werden Strömungsfelder mit dem trainierten CNN, einem CPU basierten CFD-Solver sowie einem GPU-beschleunigten CFD-Solver berechnet. Verglichen werden die Abweichung der Ausgabe sowie die nötige Rechenzeit. Abbildung 4 zeigt die Ergebnisse von Validierungsdaten und Testdaten, es wird deutlich, dass das grundlegende Strömungsverhalten vom CNN abgebildet wird. Das CNN ist in der Lage die im Training von den Primitiven gelernten Strömungseigenschaften auf unbekannte Samples zu übertragen. Auch für komplexere Körper wie den Autoquerschnitt der Testdaten liefert das CNN eine annehmbare Approximation. Dies spricht für eine gute Generalisierung des CNN. Im Vergleich zu den in 3.2 beschriebenen Arbeiten ist dieses Modell also unabhängig von konkreten Anwendungen und kann Strömungsverhalten für beliebige Geometrie im Windkanal approximieren.

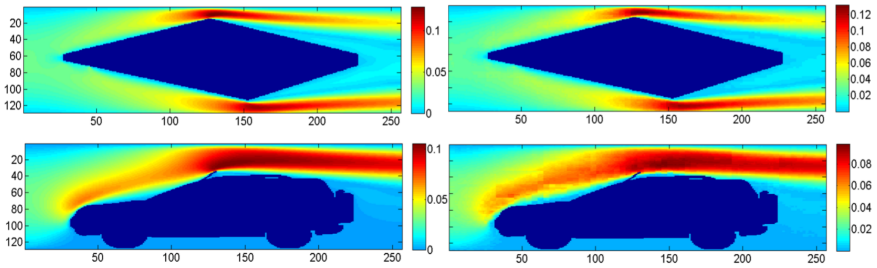


**Abb. 4.** Vergleich CFD-Solver (links) und CNN (rechts) **Quelle** [3]

Neben der Qualität der Ausgabe wird auch die dafür benötigte Rechenleistung verglichen. Während die Dauer der Berechnung mit einem CFD-Solver auf dem CPU bei durchschnittlich 80 Sekunden und auf der GPU bei durchschnittlich 2 Sekunden liegt, benötigt das CNN weniger als 20 Millisekunden. Damit wird eine Visualisierung des Strömungsfeldes in Echtzeit ermöglicht. Die Autoren von [3] stellen auch ein Ansatz für dreidimensionale Modelle vor, dieser liefert vergleichbare Beschleunigungen. [3]

## 4 Approximation von Fluid Dynamics (CFD)

### 4.1 Gitterbasierte Approximation

**Convolutional Neural Networks** Wie in 3.3 beschrieben eignen sich CNNs zur Approximation von Strömungsfeldern auf festen Gittern. Ein Grund hierfür ist die lokale Abhängigkeit von Strömungen und die Erhaltung solcher lokalen Eigenschaften durch die ConvLayer. Die Strömung an einer Stelle hängt stark vom Strömungsverhalten in der nahen Umgebung, also den benachbarten Gitterzellen, ab. [3] zeigt, dass CNNs diese Lokalität sowohl im zweidimensionalen als auch im dreidimensionalen nutzen können. Zudem kann die Größe der Ein- und Ausgabe leicht geändert werden, ohne das Modell neu trainieren zu müssen. Dies ist möglich, da die Filtermatrix auch mit größeren Layern gefaltet werden kann.[17] Neben dem Beispiel aus Abschnitt 3.3 setzen auch die Arbeiten [17] und [19] auf den Einsatz von CNNs um gitterbasierte Ersatzmodelle für CFD zu trainieren.

**Deep Learning** Für den Ende-zu-Ende Trainingsansatz lässt sich mithilfe von CFD-Solvern einfach ein Datensatz erstellen und die so trainierten CNNs ermöglichen eine schnelle Ermittlung eines stabilen Endzustandes (siehe 3.3). So kann z.B. für einen Fahrzeugquerschnitt ein Strömungsfeld bei gegebenen Ausgangsströmungen ermittelt werden. Jedoch sind in den Trainingsdaten keine Informationen über die zeitliche Strömungsänderung zwischen Ausgangs- und Endzustand gegeben. Dies hat zur Folge, dass so trainierte CNNs auch nur den Endzustand approximieren können, jedoch keine zeitliche Entwicklung darstellen. Datensätze, die auch zeitliche Zwischenschritte der Simulation enthalten, existieren zwar, jedoch sind diese für das Ende-zu-Ende Training ungeeignet. Grund dafür ist, dass beim Training von einem Eingabezustand auf einen Ausgabezustand trainiert wird. Dabei ist der Eingabezustand immer aus dem Trainingsdatensatz, jedoch wird nie der errechnete Ausgabezustand als neuer Eingabezustand verwendet. Dies hat zur Folge, dass sich beim Training keine Fehler aufsummieren, denn nach jedem Trainingsschritt wird auf die vom CFD-Solver berechnete Lösung aus dem Datensatz als neue Eingabe zurückgegriffen. Erst beim Test wird dann die Ausgabe das CNNs wieder als Eingabe für den nächsten Zeitschritt verwendet. Hier summieren sich dann die Fehler auf und bereits nach wenigen Zeitschritten weicht das CNN weit von der simulierten Lösung ab. Für dieses Problem erweisen sich Deep Learning (DL) Ansätze als hilfreich: [17] nutzt

unbeaufsichtigtes Lernen und trainiert ein CNN auf nicht klassifizierten Zeitreihendaten. Das CNN minimiert dabei die Richtungsänderung zum vorherigen Zeitschritt, während es gleichzeitig den Druck zwischen den Gitterzellen konstant hält. Aufgrund der guten Parallelisierbarkeit von CNNs bleibt auch dieser Ansatz echtzeitfähig. Die Approximation von Zeitreihen wird auf diese Weise verbessert, jedoch bleibt auch dieser Ansatz nur für wenige Zeitschritte akkurat. [17]

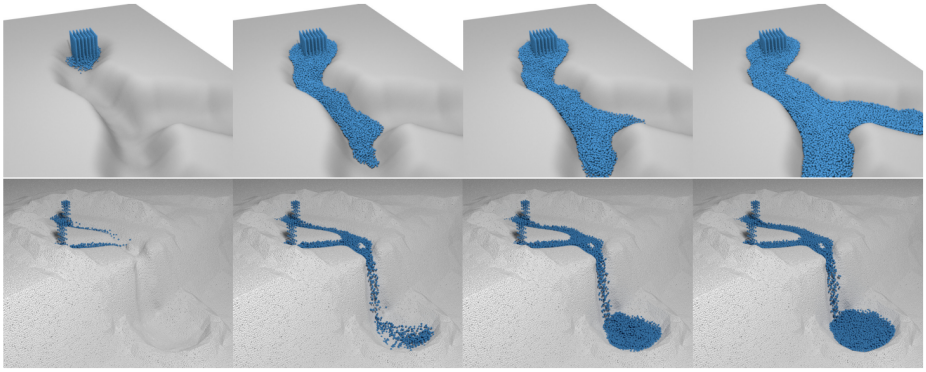

**Abb. 5.** Partikelsimulation mit einem CNN in einer unbekannten Szene **Quelle** [18]

### 4.2 Partikelsimulation

**Convolutional Neural Networks** Wie oben beschrieben sind CNNs in vielen Anwendungsfällen zur Approximation von CFD Simulationen gut geeignet, da sie sehr effizient gespeichert und ausgewertet werden können und lokale Zusammenhänge erhalten. Für gitterbasierte Modelle ist diese Lokalität der Schlüssel zu den guten Ergebnissen: die Gitterzellen bewegen sich nicht und die Lokalität bleibt über alle Simulationsschritte (auch über die Zeit) erhalten. Die Interaktion der Zellen mit ihren lokalen Nachbarn reicht aus, um das gesamte Strömungsverhalten zu berechnen.

Bei partikelbasierten Modellen ist diese Lokalität jedoch nur temporär: über die Zeit hinweg bewegen sich die Partikel. Damit ändert sich auch die lokale Umgebung. [18] präsentiert einen Ansatz, um solche partikelbasierten Simulationen mit CNNs zu realisieren. Die Arbeit verzeichnet Verbesserungen für zeitliche Simulationsverläufe, sowohl was die Rechendauer als auch die Genauigkeit angeht. Interessant ist hier die Loss-Funktion:

$$loss(\vec{y}) = \sum_{i=1}^{N} \phi_i \|\vec{y} - \vec{d}\|^{\gamma}$$

Dabei ist $\phi_i$ invers proportional zur Anzahl der Nachbarn von Partikel $i$. So werden Partikel mit wenigen Nachbarn, also solche die sich an der Oberfläche

oder am Rand befinden, besonders stark gewichtet. Dies verbessert die Genauigkeit bei Interaktionen der Flüssigkeit mit Hindernissen. Das so trainierte CNN verwendet kontinuierliche Faltung um die variablen Nachbarschaftsbeziehungen der Partikel zu berücksichtigen. Auch dieser Ansatz zeigt eine gute Generalisierung und kann durch ändern der Partikeleigenschaften auf unterschiedliche Anwendungen angepasst werden. [18]

**Graph Neural Networks** Eine Alternative zur Darstellung der lokalen Beziehungen der Partikel sind Graph Neural Networks (GNN). Dabei werden die Partikel als Knoten eines Graphen gespeichert und durch Kanten werden Beziehungen zwischen den Partikeln simuliert. Das GNN nutzt dabei die Kanten, um Interaktionen zwischen den Partikeln darzustellen. Um Strömungssimulationen mit GNNs durchzuführen sind folgende Schritte nötig [12]:

1. Weltzustand $X^t$ zum Zeitpunkt $t$ wird in Graphen $G^t = (V^t, E^t)$ kodiert
2. Durch *Message-Passing* entlang der Kanten $E^t$ werden die Interaktionen der Partikel simuliert (mehrere Iterationen)
3. Graph $G^t$ wird in Strömungszustand $Y^t$ dekodiert
4. anhand der Strömungsinformationen $Y^t$ wird ein neuer Weltzustand $X^{t+1}$ berechnet.

Der Weltzustand $X^t$ ist dabei die Menge aller Partikel zum Zeitpunkt $t$, diese haben Eigenschaften wie Position, Geschwindigkeit, etc. Anhand eines MLP wird aus dem Weltzustand ein Graph erstellt. Im zweiten Schritt werden die Interaktionen zwischen den Knoten simuliert. Dazu wird das GNN verwendet. Durch gelerntes *Message-Passing* werden Werte zwischen den Knoten entlang der Kanten ausgetauscht und die Knotenwerte werden über mehrere Iterationen verändert. [14] Anhand eines weiteren MLP werden dann aus dem Graphen Strömungsinformationen extrahiert und damit der neue Weltzustand $X^{t+1}$ berechnet. [12]

In ihrer Arbeit vergleichen die Autoren die *Summed Squared Error* (SSE), *Mean Squared Error* (MSE) sowie *Maximum Mean Discrepancy* (MMD) Loss-Funktionen. Sie stellen fest, dass sich je nach Viskosität der Flüssigkeit und Dauer der Situation verschiedene Loss-Funktionen eignen und schlagen hier weitere Recherche vor. Um schnelle Akkumulation von Fehlern zu vermeiden werden die Trainingseingaben mit zufälligem Rauschen versehen, so lernt das NN kleine Fehler in der Eingabe zu verarbeiten. Das Ergebnis zeigt, dass auf diese Art sehr akkurate Approximationen erzielt werden können. Insbesondere werden Approximationsfehler kaum akkumuliert, somit ist der Ansatz auch für Simulationen über viele Zeitschritte geeignet. Das GNN erreicht nach Angaben der Autoren bei längerer Dauer bessere Approximationen als das oben beschriebene partikelbasierte CNN aus [18]. [12] Jedoch sind insbesondere die Iterationen des Message-Passing auf dem GNN rechenaufwändiger als die Auswertung eines CNN, daher ist das GNN nicht echtzeitfähig und damit im Vergleich zu den CNN-Ansätzen deutlich langsamer. [12]

Während für die gitterbasierten Ansätze verhältnismäßig einfache Loss-Funktionen wie MSE oder SSE verwendet werden können, sind für Partikelsimulationen also speziellere Loss-Funktionen (vgl. Abschnitt 4.2) nötig und die Wahl spielt eine größere Rolle. [12]

# 5 Approximation von Turbulenzen (LES)

## 5.1 Subgrid-scale Modelle durch Multi Layer Perceptron

Wie in Abschnitt 2.3 beschrieben erfordert die Simulation von Wirbelstrukturen mittels CFD eine sehr feine Gitterauflösung. LES beschleunigt diese Simulation, indem die rechenintensiven Navier-Stokes Gleichungen (2.2) tiefpassgefiltert werden. So kann das gefilterte NL-DGS mit bekannten Methoden wie FEM oder Ansätzen der CFD gelöst werden. Die Auswirkungen der vom Filter abgeschnittenen Informationen werden dann von einem sogenannten *Subgrid-scale* (SGS) Modell approximiert. Bei diesen SGS Modellen handelt es sich häufig um empirische Modelle die auf Basis gesammelter Daten die Einflüsse der Subgrid-Größen auf die Simulation abschätzen [4]. Physikbasierte Modelle bringen aufgrund der Komplexität der SGS an Randregionen und im Übergangsbereich des Filters schnell sehr hohen Rechenaufwand mit sich. Daher ist auch hier der Einsatz von Approximationsmodellen Gegenstand der Forschung. Für diese ohnehin datenabhängige Aufgabe eignen sich eingelernte Systeme besonders gut.[13]

Die SGS liefern dabei für jede Gitterzelle und jeden Zeitschritt der Simulation einen Skalarwert (engl. *subgrid scale*) welcher die Einflüsse der abgeschnittenen Frequenzen modelliert. Eingabe der Funktion sind die hochfrequenten diskretisierten Anteile der Navier-Stokes Gleichungen. Im Paper [13] wird vorgeschlagen diese Funktion mit einem MLP zu approximieren. Zur Berechnung der nötigen Trainingssamples wird eine vollständige LES Simulation mit einem *Scale-similarity* Modell [5] als SGS durchgeführt. Dabei zeichnet sich das *Scale-similarity* Modell durch hohe Genauigkeit für die Approximation aus. Für das Ende-zu-Ende Training des MLP wird dann der *Summed squared error* (SSE) verwendet. [13] Das aus MLP besteht dabei aus einem 15 Neuronen großen Eingabelayer und zwei versteckten Layer. Aufgrund der geringen Größe und einfachen Struktur lässt sich das NN auf nur 4500 Samples erfolgversprechend trainieren. Beim Einsatz als SGS Modell für die LES lässt sich eine Zeitersparnis von 20% messen. [13] Allerdings werden neben Approximationsfehlern des NN auch die Schwächen des *Scale-similarity* Modells vom NN übernommen, so zeigt auch das NN ein für SGS Modelle typisches rauschen. Die Autoren schlagen hier unter anderem vor, das NN auf gefilterten Daten zu trainieren, um einen anschließenden Filterschritt während der Berechnung der LES zu sparen.

## 5.2 Reduced Order Modelle und Proper Orthogonal Decomposition

Ein alternativer Ansatz zur Filterung sind sogenannte *Reduced Order Modelle* (ROM). Ziel ist es die wichtigsten Features und physikalischen Eigenschaften

eines hochdimensionalen NL-DGS auf einen Unterraum zu reduzieren. Dieser Unterraum hat dabei eine kleinere Dimension. Für diese Reduktion werden Techniken wie *Proper Orthogonal Decomposition* (POD) eingesetzt. Das so erzeugte ROM wird dann genutzt, um das Strömungsverhalten in den nächsten Iterationen abzuschätzen. Für diese Abschätzung existieren mathematische und physikalische Modelle wie die *Garlekin Projektion*. Wie schon bei den SGS Modellen sind auch die ROM Modelle datengetrieben und eignen sich damit hervorragend zur Approximation mit NNs. [9]

Im Folgenden werden Ansätze zur Approximation des ROM Modells sowie der POD Methode vorgestellt:

**LSTM als Reduced Order Modell** Bei sogenannten *Long short-term Memory* (LSTM) Netzen handelt es sich um *Rekurrente Neuronale Netze* (RNN). Aufgrund der Rekurrenz können im LSTM Informationen gepuffert werden. Dies erweist sich besonders für die Verarbeitung von zeitlichen Signalen, wie z.B. Sprache, als wichtig. Im Falle der oben angesprochenen ROMs besteht die Aufgabe darin, das Strömungsverhalten über mehrere Iterationen hinweg vorherzusagen. Damit ist auch hier eine zeitlicher Zusammenhang zwischen aufeinander folgenden Iterationen gegeben. Die Arbeit [9] befasst sich daher mit dem Einsatz von LSTMs als ROM-Ersatz.

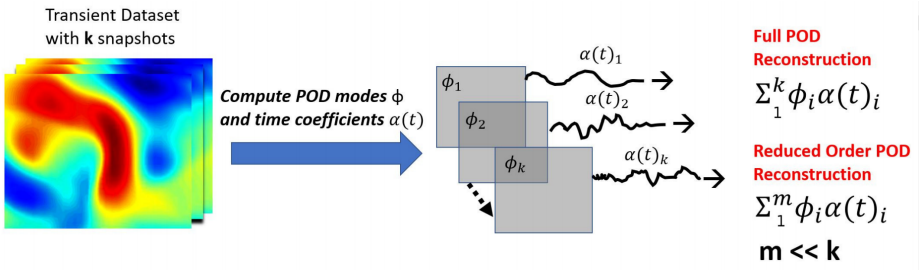


**Abb. 6.** Trainingsdatenreihen für das LSTM **Quelle** [9]

Da LSTMs für zeitliche Verläufe eingesetzt werden, sind auch zeitliche Datenreihen für das Training nötig. Anhand der POD-Methode werden die Schlüsselfeatures ermittelt. Abbildung 6 zeigt die Zerlegung der zweidimensionalen Bildreihen in Features (POD modes $\phi_i$) und zeitliche Koeffizienten ($\alpha(t)_i$). Eine Teilmenge dieser Features mit den jeweiligen Koeffizienten bildet dann das ROM. Die Zerlegung in Schlüsselfeatures kann alternativ auch über ein NN approximiert werden (siehe 5.2). Die Aufgabe des LSTM ist dann die Vorhersage der zeitlichen Koeffizienten $\alpha(t)_i$. Dazu wird für jeden Koeffizienten ein eigenes LSTM$_i$ trainiert. Das LSTM$_i$ wird mit einer zeitlichen Sequenz $(\alpha(t-k)_i, ..., \alpha(t-1)_i, \alpha(t)_i)$ initialisiert und soll diese dann bis zu einem Horizont $t'$ fortsetzen: $(\alpha(t+1)_i, ..., \alpha(t+t')_i)$. Diese Ausgabe wird dann mit der tatsächlichen Koeffizientenfolge vergli-

chen. Dazu wird der *Mean Absolute Scaled Error* (MASE) verwendet. Dieser berücksichtigt den Abstand vom Horizont $t'$, sodass sich Fehler auf den ersten vorhergesagten Zeitschritten stärker auswirken. [9]

Jedoch zeigt das LSTM bei größerem Horizont $t'$, also für Vorhersagen über viele Zeitschritte, Einbrüche der Genauigkeit. Die Autoren merken hier an, dass ihre Arbeit auf einem sehr allgemeinen Datensatz basiert und vermuten, dass bei spezifischeren Anwendungen ein gezielteres Training durchgeführt werden könne. [9] „Eine passende Anwendung für das datengetriebene LSTM-ROM [...] wäre in spezialisierten Flusssteueranwendungen, bei denen das Modell eine eng beschränkte Gruppe von Flussverhalten generalisiert“ [9, Seite 18].

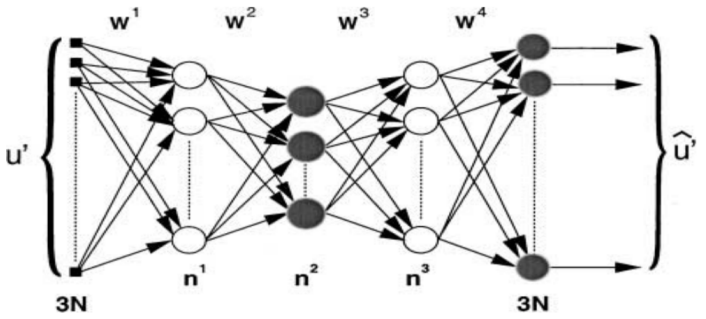


**Abb. 7.** Aufbau des MLP als Autoencoder **Quelle** [8]

**NN-basierter Autoencoder als POD** Grundlage für die Reduced-order Modelle ist die Reduktion des Ausgangsgleichungssystems auf einen Unterraum von Schlüsselfeatures. Ziel hierbei ist es, eine möglichst geringe Dimension des Unterraums zu erreichen und gleichzeitig möglichst wenig Informationen zu verlieren. Meist wird hierfür die POD Methode verwendet. Jedoch erweist sich diese Methode an Wänden (Objektkanten und Oberflächen) als fehleranfällig, da hier entscheidende Features entfernt werden.

Die Autoren von [8] trainieren ein MLP als Autoencoder, d.h. es wird darauf trainiert, die gegebene Eingabe auch wieder genau so auszugeben, jedoch werden in den versteckten Layern weniger Neuronen verwendet als in den Ein- und Ausgabelayern (siehe Abbildung 7). Auf diese Art lernt das Netzwerk, möglichst viele der Informationen aus dem Eingabesignal in einem kleineren Signal zu kodieren und anschließend wieder zu dekodieren. Als Loss-Funktion wurde der *Summed Squared Error* (SSE) verwendet. Dieses Vorgehen ermöglicht das Training ohne aufwändige Datensätze zu verwenden, denn als Zielwert wird die Eingabe selbst verwendet. Nach erfolgreichem Training wird das MLP dann in einen Kodierer und einen Dekodierer aufgeteilt. Mithilfe des Kodierers kann dann eine Featuremenge für das ROM ermittelt werden und nach der Simulation daraus auch wieder das Strömungsverhalten dekodiert werden. Das Ergebnis ist eine

Alternative zu POD, welche in Randregionen die Einflüsse der Oberfläche in den Features kodiert. Da anhand dieser Features dann das Flussverhalten approximiert wird, zeigt der Ansatz gerade für das Flussverhalten an Oberflächen deutliche Vorteile. „Die zusätzliche Rechenzeit [...] im Vergleich zu POD relativiert sich durch den stark parallelen Charakter [des MLP]“ [8] Auch dieser Ansatz ist nicht auf einen konkreten Anwendungsfall zugeschnitten und kann daher in vielen Bereichen eingesetzt werden.

## 6 Fazit

**Parallelisierbarkeit** Über alle vorgestellten Bereiche hinweg zeigt sich die Parallelisierbarkeit der NNs als große Stärke: Während bei den iterativen Lösungsansätzen der FEM- und CFD-Solver Parallelisierung nur bedingt möglich ist und häufig aufwändig zu programmieren und zu verwalten ist, können NNs innerhalb eines Layers fast immer parallel ausgewertet werden. [3] Gleichzeitig bieten aktuelle GPUs eine vergleichbare Anzahl an Recheneinheiten, sodass die Auswertung eines Layers fast vollständig parallel erfolgen kann. Dies hat zur Folge, dass NNs nach abgeschlossenem Training auch auf aktuellen Mittelklasse-Computern sehr effizient ausgewertet werden können. Insbesondere die Speichereffizienz der CNNs trägt hier auch bei beschränktem Hauptspeicher zu den geringen Auswertungszeiten bei und ermöglichen in vielen Fällen sogar Approximationen in Echtzeit. [3,17] Aufwändigere Architekturen wie GNNs oder DNNs benötigen zwar deutlich mehr Rechenzeit, sind jedoch nach wie vor schneller als GPU-basierte FEM- bzw. CFD-Solver. [12,6]

Zusätzlich kann bei NNs die Rechendauer exakt vorhergesagt werden. Bei iterativen Verfahren hingegen spielen Konvergenzgeschwindigkeit und Abbruchkriterien eine große Rolle. Je nach Eingabe kann die Konvergenzgeschwindigkeit variieren und damit auch das Abbruchkriterium wesentlich früher / später erreicht werden. Dies führt zu hohen Latenzen und besonders im Echtzeitbetrieb zu störendem Ruckeln. [17]

**Convolutional Neural Networks** CNNs leisten sowohl bei gitter- als auch bei partikelbasierten Modellierungen der CFD Simulationen gute und vor allem schnelle Arbeit. Unter den fünf am häufigsten zitierten Arbeiten zum Thema CFD Approximation mittels NNs der letzten fünf Jahre sind gleich drei CNN basierte Ansätze. Ein Grund hierfür könnte sein, dass diese Arbeiten alle echtzeitfähige Ergebnisse liefern und daher als Geschwindigkeitsvergleich dienen.

**Generalisierung der Fluid Dynamics Modelle** Neben der effizienten Auswertung tragen die CNNs auch zu einer guten Generalisierung bei. An den gitterbasierten Modellen der CFD Simulationen in Abschnitt 4.1 wird deutlich, dass die CNNs die zugrundeliegenden physikalischen Eigenschaften der Strömungen lernen und nach dem Training erfolgreich ür verschiedene Anwendungen eingesetzt werden können (vgl. Abschnitte 3.3, 4.1). Auch das Training und der Ein-

satz von GNNs ist nicht von einer speziellen Anwendung abhängig sondern generalisiert die physikalische Interaktion von Partikeln mit gegebenen Eigenschaften (vgl. Abschnitt 4.2). Dies ermöglicht auch den Vergleich der Arbeiten untereinander (vlg. Abschnitt 4.2). Ebenso weisen die SGS Modelle aus Abschnitt 5.1 gute Generalisierung auf und können in LES Simulationen unabhängig von deren Anwendung eingesetzt werden. Damit zeigt sich ein grundlegender Unterschied zu den FEM-Approximationsmodellen: Anstatt spezialisierte Modelle für bestimmte Einsatzgebiete zu trainieren werden im Bereich der Strömungssimulationen grundlegende physikalische zusammenhänge gelernt. Diese können dann auf verschiedene Anwendungsgebiete übertragen werden.

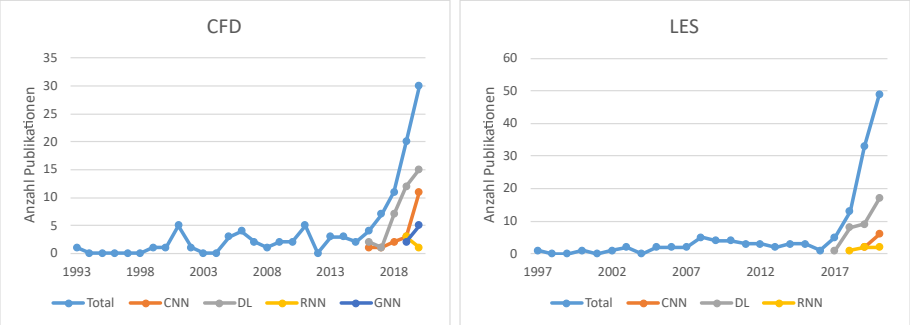


**Abb. 8.** NN Publikationen zu CFD und LES

**Publikationen zur Approximation von CFD und LES** Abbildung 8 zeigt das steigende Forschungsinteresse an NN-basierten Ersatzmodellen. Die ausgiebige Forschung beginnt bei LES und CFD im Jahr 2017. Grundlage für das plötzliche Interesse sind die zuvor erschienene Arbeit [7] im Bereich der Large Eddy Simulationen und das in Abschnitt 3.3 vorgestellte Paper [3] im Bereich der CFD. Beide Arbeiten zeigen exemplarisch das Potential von NNs für die Approximation von Physiksimulationen und wurden in den darauffolgenden Jahren von den meisten hier betrachteten Arbeiten zitiert. Dabei wurden die dort vorgestellten Konzepte erweitert oder für bestimmte Anwendungen verfeinert. Da in diesen beiden Arbeiten die CNNs und DL-Konzepte mit großem Erfolg eingesetzt wurden, ist auch das steigende Forschungsinteresse an beiden zu erklären. Der gesteigerte Forschungsaufwand führt zu Weiterentwicklungen der Netzarchitekturen (vgl. Abschnitt 4.1: CNNs auf Zeitreihendaten) sowie die Entwicklung neuer Architekturen (vgl. Abschnitt 4.2: GNNs für Partikelsimulation, Abschnitt 5.2: LSTMs als Reduced Order Modelle). Da viele dieser Arbeiten wie beschrieben zwar erfolgversprechend, aber nicht ohne Einschränkungen sind, ist auch in den nächsten Jahren mit Weiterentwicklungen dieser Arbeiten zu rechnen.

## Statistiken

Für die Statistiken wurden die Publikationen mithilfe der Software *Publish or Perish* aufgelistet und anhand des Titels kategorisiert. Mehrfach gelistete Arbeiten wurden aussortiert. Auf diese Art wurden die Daten für die Diagramme Abb. 2 und Abb. 8 erhoben. Stand der Daten ist der 17. Januar 2021. Publikationen im Jahr 2021 wurden nicht berücksichtigt.

## Literatur


1. Fetene, B.N., Shufen, R., Dixit, U.S.: Fem-based neural network modeling of laser-assisted bending. Neural Computing and Applications **29**(6), 69–82 (2018)
2. Finite-Elemente-Methode: Finite-elemente-methode — Wikipedia, the free encyclopedia (2020), `https://de.wikipedia.org/w/index.php?title=Finite-Elemente-Methode&oldid=203159760`, [Online; accessed 14.12.2020]
3. Guo, X., Li, W., Iorio, F.: Convolutional neural networks for steady flow approximation. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 481–490 (2016)
4. Large Eddy Simulation: Large eddy simulation — Wikipedia, the free encyclopedia (2017), `https://de.wikipedia.org/w/index.php?title=Large_Eddy_Simulation&oldid=163349669`, [Online; accessed 16.12.2020]
5. Layton, W., Lewandowski, R.: A simple and stable scale-similarity model for large eddy simulation: Energy balance and existence of weak solutions. Applied Mathematics Letters **16**(8), 1205 – 1209 (2003). https://doi.org/https://doi.org/10.1016/S0893-9659(03)90118-2, `http://www.sciencedirect.com/science/article/pii/S0893965903901182`
6. Liang, L., Liu, M., Martin, C., Sun, W.: A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. Journal of The Royal Society Interface **15**(138), 20170844 (2018)
7. Ling, J., Kurzawski, A., Templeton, J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fluid Mechanics **807**, 155–166 (2016). https://doi.org/10.1017/jfm.2016.615
8. Milano, M., Koumoutsakos, P.: Neural network modeling for near wall turbulent flow. Journal of Computational Physics **182**(1), 1–26 (2002)
9. Mohan, A.T., Gaitonde, D.V.: A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks. arXiv preprint arXiv:1804.09269 (2018)
10. Navier-Stokes-Gleichungen: Navier-stokes-gleichungen — Wikipedia, the free encyclopedia (2020), `https://de.wikipedia.org/w/index.php?title=Navier-Stokes-Gleichungen&oldid=205576698`, [Online; accessed 12.12.2020]
11. Numerische Strömungsmechanik: Numerische strömungsmechanik — Wikipedia, the free encyclopedia (2020), `https://de.wikipedia.org/w/index.php?title=Numerische_Strmungsmechanik&oldid=197025811`, [Online; accessed 15.12.2020]
12. Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P.W.: Learning to simulate complex physics with graph networks. arXiv preprint arXiv:2002.09405 (2020)
13. Sarghini, F., De Felice, G., Santini, S.: Neural networks based subgrid scale modeling in large eddy simulations. Computers & fluids **32**(1), 97–108 (2003)

14. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2008)
15. Shahani, A., Setayeshi, S., Nodamaie, S., Asadi, M., Rezaie, S.: Prediction of influence parameters on the hot rolling process using finite element method and neural network. Journal of materials processing technology **209**(4), 1920–1935 (2009)
16. Silva, R., Guerreiro, J., Loula, A.: A study of pipe interacting corrosion defects using the fem and neural networks. Advances in Engineering Software **38**(11-12), 868–875 (2007)
17. Tompson, J., Schlachter, K., Sprechmann, P., Perlin, K.: Accelerating eulerian fluid simulation with convolutional networks. In: International Conference on Machine Learning. pp. 3424–3433. PMLR (2017)
18. Ummenhofer, B., Prantl, L., Thuerey, N., Koltun, V.: Lagrangian fluid simulation with continuous convolutions. In: International Conference on Learning Representations (2019)
19. Xiao, X., Zhou, Y., Wang, H., Yang, X.: A novel cnn-based poisson solver for fluid simulation. IEEE Transactions on Visualization and Computer Graphics **26**(3), 1454–1465 (2020). https://doi.org/10.1109/TVCG.2018.2873375

## Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst, und weder ganz oder in Teilen als Prüfungsleistung vorgelegt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die benutzten Werken im Wortlaut oder dem Sinn nach entnommen sind, habe ich durch Quellenangaben kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen sowie für Quellen aus dem Internet.

| | |
|---|---|
| Name: | Dominik Wüst |
| Titel der Arbeit: | Proseminar Learning Surrogate Models<br>For Fluid Simulation |
| Ort, Datum: | ---------- |
| Unterschrift: | ---------- |